STATISTICAL COMPUTING AND R PROGRAMMING

MODULE-05

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.

Regression: The linear regression is defined as the analyze the relationship between the independent and dependent variable is called regression.

In statistics, regression refers to a statistical method used to examine the relationship between one or more independent variables and a dependent variable.

It aims to model and understand how changes in the independent variables are associated with changes in the dependent variable.

Regression analysis helps in:

- Predicting or estimating the value of the dependent variable based on the values of one or more independent variables.
- Understanding the strength and nature of the relationship between variables.
- Making inferences about the population based on sample data.

There are various types of regression models,

1. **Simple Linear Regression:** Examining the relationship between two variables, typically one independent variable and one dependent variable.
2. **Multiple Linear Regression:** Exploring the relationship between a dependent variable and multiple independent variables.
3. **Logistic Regression:** Used when the dependent variable is categorical, predicting the probability of an event occurring.

NOTE:

1) Regression analysis involves fitting a regression model to the data, assessing the models goodness of fit, evaluating the significance of variables, and using the model for prediction or inference.
2) It's widely used across numerous fields to understand relationships, make predictions, and inform decision-making based on data.
3) **The dependent variable(Y):**It is the outcome variable or response variable. That want to be predict.

4) **The Independent variable(X):** The independent variable is predictor variable that influence the dependent variable.

An Example of a Linear Relationship:

Let's consider an example illustrating a linear relationship between two variables, such as the relationship between a person's age and their reaction time.

Variables:

Dependent Variable (Y): Reaction time (in milliseconds) response variable - this is what we want to predict or understand.

Independent Variable (X): Age of the person (in years) - this is the predictor variable.

Assumption:

We assume that younger individuals tend to have faster reaction times, and as people age, their reaction times might increase.

Data Collection:

Gather data on reaction times and ages from individuals across different age groups.

Analysis:

After collecting the data, plot the reaction times against ages on a scatter plot.

If a linear relationship exists, the points might tend to form a pattern where, on average, as age increases, reaction time also tends to increase or decrease in a linear manner.

Linear Relationship:

A positive linear relationship would imply that as age increases, reaction time increases.

A negative linear relationship would suggest that as age increases, reaction time decreases.

Modeling:

Use linear regression to fit a line to the data and quantify the relationship between age and reaction time.

By studying this relationship, we can determine if there's a correlation between age and reaction time, and if so, whether it's linear, helping us understand how one variable might affect the other. This example demonstrates how linear relationships between variables can be explored and modelled in statistics.

Simple Linear Regression:

Definition: Simple linear regression is a statistical method used to model the relationship between two continuous variables.

OR

A Simple Linear regression predict a one dependent and one independent variable.

It assumes that there is a linear relationship between the predictor variable (often denoted as *X*) and the outcome variable (often denoted as *Y*).

The simple linear regression model can be represented as:

$$Y=\beta 0+\beta 1\cdot X+\varepsilon$$

Where:

- *Y* is the dependent variable (outcome or response variable).
- *X* is the independent variable (predictor or explanatory variable).
- $\beta 0$ is the intercept (the value of *Y* when *X* is 0).
- $\beta 1$ is the slope (the change in *Y* for a one-unit change in *X*).
- ε represents the error term (the difference between the predicted and actual values).

The goal of simple linear regression is to estimate the values of $\beta 0$ and $\beta 1$ that minimize the sum of squared differences between the observed *Y* values and the values predicted by the model for given *X* values.

Applications:

Simple linear regression is a foundational statistical technique used to establish relationships between two quantitative variables.

1.**Economics and Finance:** In finance, it's used to model the relationship between variables like interest rates and stock prices or GDP and unemployment rates. Economists might employ it to analyze the effect of inflation on consumer spending.

2.**Market Research:** Simple linear regression helps in analyzing the impact of advertising expenditure on sales. It assists in understanding how changes in advertising spending might affect product sales.

3.**Healthcare and Medicine:** Medical researchers often use simple linear regression to study the relationship between a drug dosage and its effectiveness or to examine the impact of certain lifestyle factors (like diet, exercise) on health indicators.

4.**Education:** Simple linear regression can be applied in educational research to investigate the relationship between study hours and exam scores or to predict student performance based on various factors.

5.**Sports Analytics:** Analysts might use it to study the relationship between the number of hours of practice and athletic performance or to predict team performance based on player statistics.

SIMPLE LINEAR REGRESSION FEATURES IN STATISTICS

1. **One Dependent Variable:** It involves predicting or explaining the behavior of a single dependent variable based on changes in one independent variable.
2. **Straight-line Relationship:** The relationship between the independent and dependent variables is assumed to be linear. The regression equation is represented by a straight line: $Y = \beta_0 + \beta_1 X + \varepsilon$, where Y is the dependent variable, X is the independent variable, β_0 is the intercept, β_1 is the slope, and ε is the error term.
3. **Minimization of Residuals:** The regression line is determined by minimizing the sum of squared differences (residuals) between the observed values and the values predicted by the model.
4. **Statistical Inference:** It allows for statistical hypothesis testing to determine the significance of the relationship between variables. This includes testing whether the slope of the regression line is significantly different from zero and assessing the overall goodness-of-fit of the model.
5. **Assumptions:** Simple linear regression assumes that there is a linear relationship between the variables, residuals are normally distributed, residuals have constant variance (homoscedasticity), and there is independence among observations.
6. **Interpretation of Coefficients:** The slope coefficient (β_1) represents the change in the dependent variable for a one-unit change in the independent variable. The intercept (β_0) represents the value of the dependent variable when the independent variable is zero (if such an interpretation is meaningful in the context).
7. **Prediction:** It allows for predicting the value of the dependent variable based on a given value of the independent variable, within the range of the observed data.

The goal of simple linear regression is to estimate the values of $\beta 0$ and $\beta 1$ that minimize the sum of squared differences between the observed Y values and the values predicted by the model for given X values.

$Y == \beta_0 + \beta_1 X + \varepsilon$

The formula to estimate the slope ($\beta 1$) is: The intercept(β_0)

$$\beta_1 = \sum_{i=1}^{n} \frac{(x_i - \bar{x})(y_i - \bar{y})}{(x_i - \bar{x})^2} \qquad \beta_0 = \bar{Y} - \beta_0 \, . \bar{X}$$

And once the slope ($\beta 1$) is calculated, the intercept ($\beta 0$) can be found using the formula:

Where:

n is the number of data points.

$\bar{X}$ and $\bar{Y}$ are the means of X and Y respectively.

Estimating the Intercept and Slope Parameters

- The goal is to use your data to estimate the regression parameters, yielding the estimates $\hat{\beta}0$ and $\hat{\beta}1$. this is referred to as *fitting* the linear model.
- In this case, the data comprise *n* pairs of observations for each individual. The fit- ted model of interest concerns the mean response value, denoted $\hat{y}$, for a specific value of the predictor, *x*, and is written as follows:

$$\hat{y} = \hat{\beta}0 + \hat{\beta}1.\,x$$

oR

$$E[Y] = \hat{\beta}0 + \hat{\beta}1\,.x$$

OR

$$E[Y|X = x] = \hat{\beta}0 + \hat{\beta}1\,.x$$

Alternative notation such as E[*Y*] or E[*Y*|*X* = *x*] is used on the left side of to emphasize the fact that the model gives the mean (that is, the expected value) of the response. For compactness, many simply use something like $\hat{y}$, as shown here.

Let your *n* observed data pairs be denoted x_i and y_i for the predictor and response variables, respectively; $i = 1, \ldots, n$. Then, the parameter esti- mates for the simple linear regression function are

$\hat{\beta}1 = \rho_{xy}\frac{s_y}{s_x}$ **and** $\beta_0 = \bar{Y} - \beta_1\,.\bar{X}$

$$\beta_1 = \sum_{i=1}^{n} \frac{(x_i - \bar{x})(y_i - \bar{y})}{(x_i - \bar{x})^2}$$

Problems:

X={2,3,5,7,9} y={4,5,7,10,15}

Step1:

To calculate mean $\bar{X} = \frac{\Sigma x}{n} = \frac{26}{5}$=5.2

$$\bar{Y} = \frac{\Sigma Y}{n} = \frac{41}{5}=8.2$$

Step2:To calculate Slope β_1

$$\beta_1 = \sum_{i=1}^{n} \frac{(x_i-\bar{x})(y_i-\bar{y})}{(x_i-\bar{x})^2} = \frac{3.4}{34.64}=0.098$$

1. $(x_i - \bar{x})(y_i - \bar{y})$=(2-5.2)(4-8.2)+(3-5.2)(5-8.2)+……=3.4

2. $(x_i - \bar{x})^2$=((2-5.2)^2)+((3-5.2)^2)+…=34.64

Step3:To calculate Intercept (β_0)

$$\beta_0 = \bar{Y} - \beta_1 . \bar{X}$$

=8.2-0.098*5.2

=7.69

Step4:Write the linear Equation.

To substitute the value β_0 and β_1 into the equation

$$\hat{y} = \hat{\beta}0 + \beta 1. x$$

$$Y=7.69+0.098x$$

To Write in R program

The syntax lm function:

lm(formula, data)

S.No	Parameters	Description
1.	Formula	It is a symbol that presents the relationship between x and y.
2.	Data	It is a vector on which we will apply the formula.

Example:

```
#Creating input vector for lm() function
x <- c(2,3,5,7,9)
y <- c(4,5,7,10,15)

# Applying the lm() function.
relationship_model<- lm(y~x)

#Printing the coefficient
print(relationship_model)
```

Output:

```
Call:
lm(formula = y ~ x)

Coefficients:
(Intercept)   slope x
     7.67      0.098
```

The predict() Function

Now, we will predict the weight of new persons with the help of the predict() function. There is the following syntax of predict function:

Syntax:

predict(object, newdata)

S.No	Parameter	Description
1.	object	It is the formula that we have already created using the lm() function.
2.	Newdata	It is the vector that contains the new value for the predictor variable.

Source Code:

```
#Creating input vector for lm() function
x <- c(2,3,5,7,9)
y <- c(4,5,7,10,15)

# Applying the lm() function
relationship_model<- lm(y~x)

# Finding the weight of a person with height 9.
z <- data.frame(x = 9)
predict_result<- predict(relationship_model,z)
print(predict_result)
```

Output:

```
       1
13.96951
```

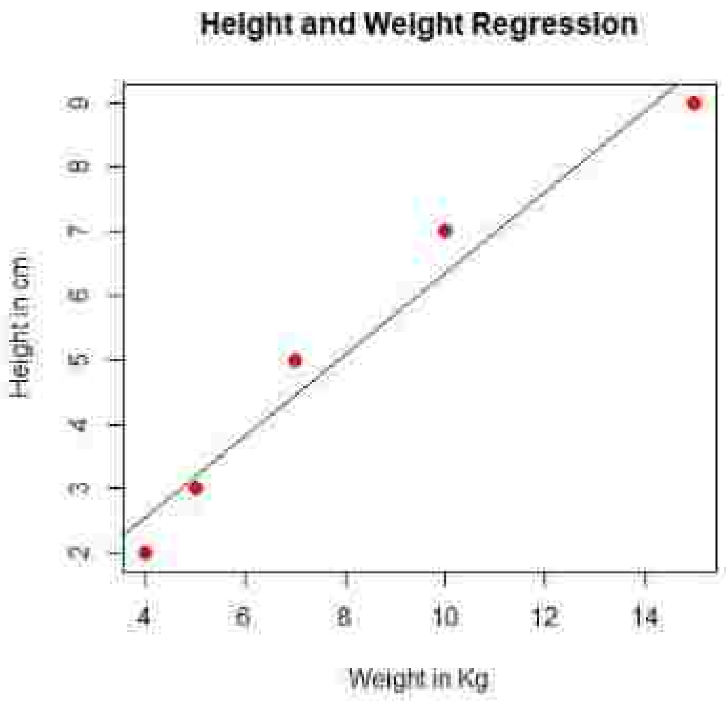
Plotting Regression

Now, we plot out prediction results with the help of the plot() function. This function takes parameter x and y as an input vector and many more arguments.

plot(y,x,col = "red",main = "Height and Weight Regression",abline(lm(x~y)),cex = 1.3,pch = 16,xlab = "Weight in Kg",ylab = "Height in cm")

Saving the

file. dev.off()



Multiple linear regression:

The Multiple linear regression are the Multiple relationship between independent and dependent two or more variable.

OR

A Multiple Linear regression predict a one dependent and Tow-more independent variable.

- It is the extension of the simple linear regression. The multiple linear regression model is used to determine a mathematical relationship among several random variable.
- Multilinear regression, also known as multiple linear regression, is a statistical method used to analyze the relationship between multiple independent variables and a dependent variable.

- It extends simple linear regression, which involves one independent variable, to incorporate several predictors.
- The goal of multiple linear regression is to create a model that best fits the relationship between these variables, allowing prediction or understanding of how changes in the independent variables affects the dependent variable.

EXAMPLE:

let's consider a real-life example of multiple linear regression. Imagine you're trying to predict house prices based on various factors such as square footage, number of bedrooms, and distance from the city center.

Solution: In the case of multiple linear regression for predicting house prices:

Dependent Variable: House price.

Independent Variables (Predictors):

1. **Square Footage:** The size of the house in square feet.
2. **Number of Bedrooms:** The count of bedrooms in the house.
3. **Distance from City Center:** How far the house is from the city center.

Using multiple linear regression, a model can be created based on historical data of houses sold. This model will estimate the house price by considering all these factors simultaneously. For instance, the model might suggest that for every additional square foot, the price increases by a certain amount, or that being closer to the city center correlates with higher prices. This information helps in making predictions about house prices for new properties based on their features.

Applications/Uses of Multiple Linear Regression:

Multiple linear regression finds applications across various fields:

1. **Economics:** Predicting factors influencing GDP growth, inflation rates, or unemployment based on multiple economic indicators.
2. **Marketing:** Determining how various marketing channels, pricing strategies, and demographics affect sales figures.
3. **Healthcare:** Analyzing how factors like age, lifestyle, and medical history impact the prevalence of certain diseases or predicting patient outcomes based on multiple health parameters.
4. **Finance:** Predicting stock prices or asset values based on various financial indicators

such as interest rates, market indices, and company performance metrics.

5. **Environmental Science:** Predicting air or water quality based on factors like pollution levels, weather conditions, and geographical variables.

Extending the simple model to a multiple model

To determine the value of a continuous response variable Y given the values of $p > 1$ independent explanatory variables $X_1, X_2, \ldots, X_p$. The overarching model is defined as

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p + \epsilon$$

Here:

- Y is the dependent variable
- X_1, X_2, X_3 are the independent variables (square footage, number of bedrooms, distance from the city center).
- β_0 is the intercept (constant term).
- $\beta_1, \beta_2, \beta_3$ are the coefficients that represent the impact of each independent variable on the dependent variable.
- ϵ represents the error term.

Example, the estimated equation might look like:

House Price=β0+β1×Square Footage+β2×Number of Bedrooms+β3×Distance from City Center

Each coefficient (β) signifies the change in the house price for a unit change in the respective independent variable, assuming the other variables remain constant.

NOTE:

- where $\beta_0, \ldots, \beta_p$ are the regression coefficients and, as before, you assume independent, normally distributed residuals $\epsilon \sim N(0, \sigma)$ around the mean.
- In practice, you have n data records; each record provides values for each of the predictors X_j; $j = \{1, \ldots, p\}$.
- The model to be fitted is given in terms of the mean response, conditional upon a particular realization of the set of explanatory variables

$$\hat{y} = E[Y \mid X_1 = x_1, X_2 = x_2, \ldots, X_p = x_p] = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 + \ldots + \hat{\beta}_p x_p,$$

where the $\hat{\beta}_j$s represent estimates of the regression coefficients.

Estimating in Matrix Form

The computations involved in minimizing this squared distance are made much easier by a *matrix representation* of the data. When dealing with *n* multivariate observations

$$Y = X\beta + \epsilon$$

where Y and ϵ denote $n \times 1$ column matrices such the

$$y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ \vdots \\ yn \end{bmatrix} \quad € = \begin{bmatrix} €_1 \\ €_2 \\ \vdots \\ \vdots \\ €n \end{bmatrix} \quad \beta = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \vdots \\ \beta n \end{bmatrix}$$

$$x = \begin{bmatrix} 1 & X_{1,1} & X_{p,1} \\ 1 & X_{1,2} & X_{p,2} \\ 1 & X_{1,n} & X_{p,n} \end{bmatrix}$$

$=(x^T * X)^{-1} x^T * Y$

NOTE:

- The symbol represents matrix multiplication, the superscript ^{T} rep- resents the transpose, and $^{-1}$ represents the inverse when applied to matrices
- Extending the size of $\boldsymbol{\beta}$ and $\boldsymbol{X}$ (note the leading column of 1s in $\boldsymbol{X}$) to create structures of size $p + 1$ (as opposed to just the number of predictors p) allows for the estimation of the overall intercept β_0.

Terminology:

1. **lurking variable:** A lurking variable influences the response, another predictor, or both, but goes unmeasured (or is not included) in a predictive model
2. **nuisance or extraneous variable** :A nuisance or extraneous variable is a predictor of secondary or no interest that has the potential to confound relationships between other variables and so affect your estimates of the other regression coefficients.
3. **Multiple Linear Regression:** A statistical method to model the relationship between multiple independent variables and a dependent variable.

4. **Dependent Variable:** Also known as the response variable, it's the variable being predicted or explained by the independent variables in the model.
5. **Independent Variables/Predictors:** Variables used to predict or explain the variation in the dependent variable.
6. **Coefficients:** These represent the weights or slopes assigned to each independent variable, indicating the strength and direction of their influence on the dependent variable. In the regression equation $Y=\beta_0+\beta_1X_1+\beta_2X_2+\ldots+\beta_n X_n+\epsilon$, coefficients $(\beta_0,\beta_1,\beta_2,\ldots,\beta_n)$ represent the slopes of the independent variables. These coefficients quantify the relationship between each independent variable and the dependent variable.
7. **Intercept:** The constant term in the regression equation that accounts for the value of the dependent variable when all independent variables are zero.
8. **Residuals:** The differences between the observed values of the dependent variable and the values predicted by the regression model.
9. **Multicollinearity:** The phenomenon where independent variables in a regression model are highly correlated, which can affect the model's stability and interpretability.
10. **Adjusted R-squared:** A metric that represents the proportion of variation in the dependent variable explained by the independent variables, adjusted for the number of predictors in the model.

A step-by-step to performing multiple linear regression using statistical methods:

Step 1: Formulate the Hypothesis

Define the research question and hypothesize the relationship between the dependent variable and multiple independent variables.

Step 2: Data Collection and Cleaning

collect relevant data for the dependent and independent variables. Clean the data by handling missing values, outliers, and ensuring the data types are appropriate.

Step 3: Explore the Data

Perform exploratory data analysis (EDA) to understand distributions, correlations, and relationships between variables using techniques like scatter plots, correlation matrices, etc.

Step 4: Model Specification

Formulate the multiple linear regression model:

$Y=\beta_0+\beta_1X_1+\beta_2X_2+\ldots+\beta_nX_n+\epsilon$

Where:

Y is the dependent variable.

X1,X2,….,Xn are the independent variables.

β0,β1,....,βn are the coefficients.

ϵ is the error term.

Step 5: Estimate Coefficients

Use statistical methods (such as the least squares method) to estimate the coefficients that minimize the sum of squared differences between predicted and actual values.

Step 6: Assess Model Fit

Evaluate the goodness of fit of the model by examining metrics like R-squared, adjusted R-squared, p-values of coefficients, and residuals analysis.

Step 7: Interpret Results

Interpret the coefficients to understand the relationship between the independent variables and the dependent variable.

Step 8: Validate the Model

Validate the models performance on a separate dataset or using cross-validation techniques to ensure its reliability and generalizability.

Step 9: Make Predictions

Use the validated model to make predictions on new or unseen data based on the established relationships.

SYNTAX FOR MULPLE LINEAR REGRESSION IN R

```
lm(formula,data)
```

S.No	Parameters	Description
1.	Formula	It is a symbol that presents the relationship between x and y.
2.	Data	It is a vector on which we will apply the formula.

Multiple linear regression in R using statistical methods step by step:

Step 1: Load the Required Libraries :

Before you begin, make sure to load the necessary R libraries. For linear regression, you'll typically use the `lm()` function, which is in the base R package.

```
# Load required libraries
library(stats)                    # for basic statistics functions
```

Step 2: Prepare Your Data:

A dataset with the variables you want to use for the regression analysis. Ensure your data is in a format that R can work with, like a dataframe.

Let's assume you have a dataframe named **data** with your variables: **dependent_var**, **independent_var1**, **independent_var2**, etc.

Step 3: Fit the Multiple Linear Regression Model

Use the **lm**() function to fit the multiple linear regression model. The basic syntax for the **lm**() function is:

```
 # Fit the multiple linear regression model
model <- lm(dependent_var ~ independent_var1 + independent_var2 + ..., data = your_data)
```

Replace `dependent_var`, `independent_var1`, `independent_var2`, etc., with the actual column names from your dataset.

Step 4: Analyze the Model

Once you've fitted the model, you can examine various aspects of it using different functions:

```
 # print of the regression model
print(model)
```

Example:

```
 # Load required libraries
 library(stats)
 # Creating a dataframe (replace this with your dataset)
 data <- data.frame(
  dependent_var = c(10, 15, 20, 25, 30),
  independent_var1 = c(3, 6, 7, 9, 11),
  independent_var2 = c(5, 8, 11, 14, 17)
 )
 # Fit the multiple linear regression model
 model <- lm(dependent_var ~ independent_var1 + independent_var2, data = data)
 # Summary of the regression model
print(model)
```

OUTPUT:Call:

```
lm(formula = dependent_var ~ independent_var1 + independent_var2, data = data)
Coefficients:
  (Intercept)  independent_var1  independent_var2
```

1.667 0.000 1.667

This code creates a dataset and performs multiple linear regression using **lm()**, where **Y** is regressed on **X1**, **X2**, and **X3**. The **summary()** function provides detailed information about the regression results, including coefficients, p-values, R-squared, etc.

Replace **Y**, **X1**, **X2**, **X3**, and the sample dataset with your actual data and variable names to apply multiple linear regression to your specific case in R.

Difference between Simple linear Regression and Multiple linear Regression

Simple linear Regression	Multiple linear Regression
1. A Simple Linear regression predict a one dependent and one independent variable.	1. A Multiple Linear regression predict a one dependent and Tow-more independent variable.
2. $Y = \beta_0 + \beta_1 X_1 + \beta_2 + \epsilon$	2. $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p + \epsilon$

Write a program to create an any application of Linear Regression in multivariate context forpredictive purpose.

Source code:

```
# Creating sample data
set.seed(42)
bedrooms <- sample(1:5, 100, replace = TRUE)
house_size <- rnorm(100, mean = 1500, sd = 300)
house_price <- 50000 + 20000 * bedrooms + 100 * house_size + rnorm(100, mean = 0, sd =
5000)

data <- data.frame(bedrooms, house_size, house_price)

# Fitting a multiple linear model
model <- lm(house_price ~ bedrooms + house_size, data = data)

# Creating new data for prediction
```

```
new_data <- data.frame(bedrooms = c(3, 4, 2), house_size = c(1600, 1800, 2000))


# Predicting house_price using the model
new_predictions <- predict(model, newdata = new_data)
new_predictions
```

OUTPUT:

```
       1        2        3
127690.4 147650.5 107250.9
```

- Sample data for **bedrooms**, **house_size**, and **house_price** is created.
- A multiple linear regression model is fitted (**model**) using **lm()** with **house_price** as the dependent variable and **bedrooms** and **house_size** as independent variables.
- New data (**new_data**) with values for **bedrooms** and **house_size** for three houses is created for which we want to predict **house_price**.
- The **predict()** function is used to predict **house_price** for the new data based on the model.
- The output provides predicted house prices for the three houses specified in the **new_data**.

```
# Creating sample data
set.seed(42)
bedrooms <- sample(1:5, 100, replace = TRUE)
house_size <- rnorm(100, mean = 1500, sd = 300)
house_price <- 50000 + 20000 * bedrooms + 100 * house_size + rnorm(100, mean = 0, sd =
5000)


data <- data.frame(bedrooms, house_size, house_price)


# Fitting a multiple linear model
model <- lm(house_price ~ bedrooms + house_size, data = data)
```

```
# Finding confidence intervals for coefficients
conf_intervals <- confint(model)
conf_intervals
```

OUTPUT:

```
              2.5 %     97.5 %
(Intercept) 48610.926 51343.900
bedrooms    18251.833 21500.128
house_size     82.262   115.796
```

- Generates sample data for **bedrooms**, **house_size**, and **house_price**.
- Fits a multiple linear regression model (**model**) using **lm()** with **house_price** as the dependent variable and **bedrooms** and **house_size** as independent variables.
- Uses the **confint()** function to compute confidence intervals for the coefficients in the fitted model.
- The **conf_intervals** output will provide the lower and upper bounds of the confidence intervals for each coefficient in the multiple linear regression model.
- Each row corresponds to a coefficient in the regression model.
- Columns '2.5 %' and '97.5 %' represent the lower and upper bounds of the 95% confidence interval for each coefficient, respectively.
- For instance, the confidence interval for the **bedrooms** coefficient ranges from approximately 18251.833 to 21500.128. This interval provides a range within which we can be reasonably confident that the true value of the coefficient lies.

LINEAR MODEL SELECTION AND DIAGNOSTICS

- Linear model selection and diagnostics involve choosing the appropriate linear model for a given dataset and assessing its performance.
- Linear model selection methods are a critical part of statistical analysis with a wide range of applications, particularly in high-dimensional data analysis, where the number of variables is much larger than the sample size.
- In linear model selection are the select the best model based on the criteria.
- The Linear selection model is also called as the statistical model

The linear selection models are selected or balancing few factors.

1. **Goodness-of-fit**
2. **Complexity**
1. **Goodness-of-fit:** Goodness-of-fit refers to the goal of obtaining a model that best represents the relationships between the response and the predictor (or pre- dictors).
2. **Complexity:** Complexity describes how complicated a model is; this is always tied to the number of terms in the model that require estimation—the inclusion of more predictors and additional functions (such as polynomial transforma tions and interactions) leads to a more complex model.

OR

To represents the complicated model selection from various theoretical and practical challenges

Principle of Parsimony:
Statisticians refer to the balancing act between goodness-of-fit and complexity as the *principle of parsimony*,
where the goal of the associated *model selection* is to find a model that's as simple as possible (in other words, with relatively low complexity), without sacrificing too much goodness-of-fit.

LINEAR MODEL SELECTION:

- In statistics, several model selection algorithms help in choosing the best-fitting model among a set of candidate models.
- A model selection algorithm is to sift through your available explanatory variables

1. **Forward Selection:** Begins with an empty model and iteratively adds predictors that most improve the model fit until a stopping criterion is met.
2. **Backward Elimination:** Starts with a model containing all predictors and removes the least significant ones iteratively until a stopping criterion is reached.
3. **Stepwise Selection:** Combines forward and backward selection methods, allowing both addition and removal of predictors based on certain criteria.
4. **Best Subset Selection:** Fits all possible combinations of predictors and selects the model that best fits the data based on a chosen criterion (e.g., AIC, BIC).

The Partial F-Test:
The partial F-test is probably the most direct way to compare several differ- ent models. It looks at two or more nested models, where the smaller, less complex model is a reduced

version of the bigger, more complex model. Formally, let's say you've fitted two linear regressions models as follows:

$$\hat{y}_{\mathbf{redu}} = \hat{\beta}0 + \hat{\beta}1\ x_1 + \hat{\beta}2\ x_2 + \ldots + \hat{\beta}p\ x_p$$

$$\hat{y}_{\mathbf{full}} = \hat{\beta}0 + \hat{\beta}1\ x_1 + \hat{\beta}2\ x_2 + \ldots + \hat{\beta}p\ x_p + \ldots + \hat{\beta}q\ x_q$$

Here, the reduced model, predicting $\hat{y}_{\mathrm{redu}}$, has p predictors, plus one intercept. The full model, predicting $\hat{y}_{\mathrm{full}}$, has q predictor terms.

provide a statistically significant improvement in goodness-of-fit. The partial F-test addresses these hypotheses:

$$H_0 : \beta_{p+1} = \beta_{p+2} = \ldots = \beta_q = 0$$

$$H_A : \text{At least one of the } \beta_j \neq 0 \text{ (for } j = p, \ldots, q)$$

ForwardSelection:

Forward selection is a model selection technique in linear regression where predictors are incrementally added to the model based on statistical criteria until no further improvement is observed

Example:

```
# Sample data
set.seed(123)
data <- data.frame(
  outcome = rnorm(100),
  predictor1 = rnorm(100),
  predictor2 = rnorm(100),
  predictor3 = rnorm(100)
)

# Fit full linear regression model
full_model <- lm(outcome ~ ., data = data)

# Perform forward selection using stepAIC
library(MASS)
forward_model <- stepAIC(full_model, direction = "forward")

# Summary of the selected model
summary(forward_model)
```

OUTPUT:

```
Call:
lm(formula = outcome ~ predictor1 + predictor2 + predictor3,
    data = data)

Residuals:
     Min       1Q   Median       3Q      Max
-2.35541 -0.58837 -0.08408  0.55592  2.31302

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.09952    0.09309   1.069    0.288
predictor1  -0.04102    0.09547  -0.430    0.668
predictor2  -0.12493    0.09719  -1.285    0.202
predictor3  -0.04219    0.08892  -0.474    0.636

Residual standard error: 0.9172 on 96 degrees of freedom
Multiple R-squared:  0.02106,	Adjusted R-squared:  -0.00953
F-statistic: 0.6885 on 3 and 96 DF,  p-value: 0.5613
```

- Creates a sample dataset with an outcome variable (**outcome**) and three predictor vari ables (**predictor1**, **predictor2**, **predictor3**).
- Fits a full linear regression model (**full_model**) using all predictors.
- Utilizes the **stepAIC** function from the **MASS** package to perform forward selection based on the Akaike Information Criterion (AIC). The **direction = "forward"** argum ent specifies forward selection.
- Displays the summary of the selected model resulting from the forward selection proc ess.

Backward Elimination:

Backward selection is a model selection technique in linear regression where predictors are iteratively removed from the model based on statistical criteria until no further improvement is observed.

Let's consider a dataset named **data** with variables *X*1, *X*2, *X*3, and the response variable *Y*.

Steps:

Data Preparation:

Assume you have a dataset named **data** containing variables *X*1, *X*2,*X*3, and the response variable *Y*.

Backward Selection Implementation:

```
# Sample data
set.seed(123)
data <- data.frame(
  outcome  = rnorm(100),
  predictor1  = rnorm(100),
  predictor2  = rnorm(100),
  predictor3  = rnorm(100)
)
```

```
# Fit full linear regression model
full_model <- lm(outcome ~ ., data = data)

# Perform forward selection using stepAIC
library(MASS)
backward_model <- stepAIC(full_model, direction = "backward")
summary(backward_model )
```

OUTPUT:

```
Call
lm(formula = outcome ~ 1, data = data)

Residuals:
    Min       1Q   Median      3Q     Max
-2.39957 -0.58426 -0.02865  0.60141  2.09693

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.09041    0.09128    0.99    0.324

Residual standard error: 0.9128 on 99 degrees of freedom
```

3. Stepwise selection

Stepwise selection is a technique in linear regression that combines forward and backward selection methods to iteratively add or remove predictors based on statistical criteria.

- One of the most famous criterion measures is known as Akaike'sInformation Criterion(AIC). To have noticed this value as one of the columns in the output of add1 and drop1.For a given linear model, AIC is calculated as follows:

$AIC = -2 \times L + 2 \times (p + 2)$

Here, L is a measure of goodness-of-fit named the log-likelihood, and p is the number of regression parameters in the model, excluding the overall intercept

Source code:

```
# Sample data
set.seed(123)
data <- data.frame(
  outcome  = rnorm(100),
  predictor1 = rnorm(100),
  predictor2 = rnorm(100),
  predictor3 = rnorm(100)
)

# Fit full linear regression model
stepwise_model <- lm(outcome ~ ., data = data)

# Perform forward selection using stepAIC
library(MASS)
backward_model <- stepAIC(full_model, direction = "both")
summary(backward_model )
```

OUTPUT:

```
Call:
lm(formula = outcome ~ 1, data = data)

Residuals:
     Min       1Q   Median       3Q      Max
-2.39957 -0.58426 -0.02865  0.60141  2.09693

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.09041    0.09128    0.99    0.324

Residual standard error: 0.9128 on 99 degrees of freedom
```

Linear Residual Diagnostics

In linear model selection, diagnostics play a crucial role in assessing the model's performance and identifying potential issues or violations of model assumptions.

Some common diagnostics include:

1. **Residual Analysis:** Examining residuals (difference between observed and predicted values) to ensure they follow a normal distribution, have constant variance, and show no clear patterns in their plot against predicted values.
2. **Influence and Outlier Detection:** Identifying influential data points or outliers that might significantly affect the model fit. Tools like Cook's distance, leverage, and studentized residuals help in detecting these.
3. **Multicollinearity:** Assessing the correlation between predictors to identify multicollinearity, which can affect the stability and interpretation of coefficients.
4. **Homoscedasticity:** Checking for constant variance in residuals across different levels of predictors. A plot of residuals against fitted values can help diagnose this.
5. **Normality of Residuals:** Verifying whether residuals approximately follow a normal distribution through techniques like QQ-plots or statistical tests.

ADVANCED GRAPHICS:

1. Many users are first drawn to R because ofits impressive graphical flexibility and the ease with which you can control and tailor_the resulting visuals.
2. Advanced plot customization in statistics involves fine-tuning visual elements of graphs and charts to convey data effectively.
3. 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.
4. The advanced graph in statistics visual representation of the data.
5. Advanced graph customization allows you to tailor every aspect of your visualizations.
6. This can involve adjusting axis limits, changing tick marks, customizing grid lines, modifying line styles and marker shapes, incorporating annotations, adjusting legends, applying color palettes, and even creating interactive or 3D plots.

<u>**Characterstics of advanced graphs**</u>

1) **Color and Style:**

   Color Palettes: Choose meaningful color schemes for different data elements. Use tools like color gradients or categorical color palettes.

   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.

Characteristics of Plot customization:

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

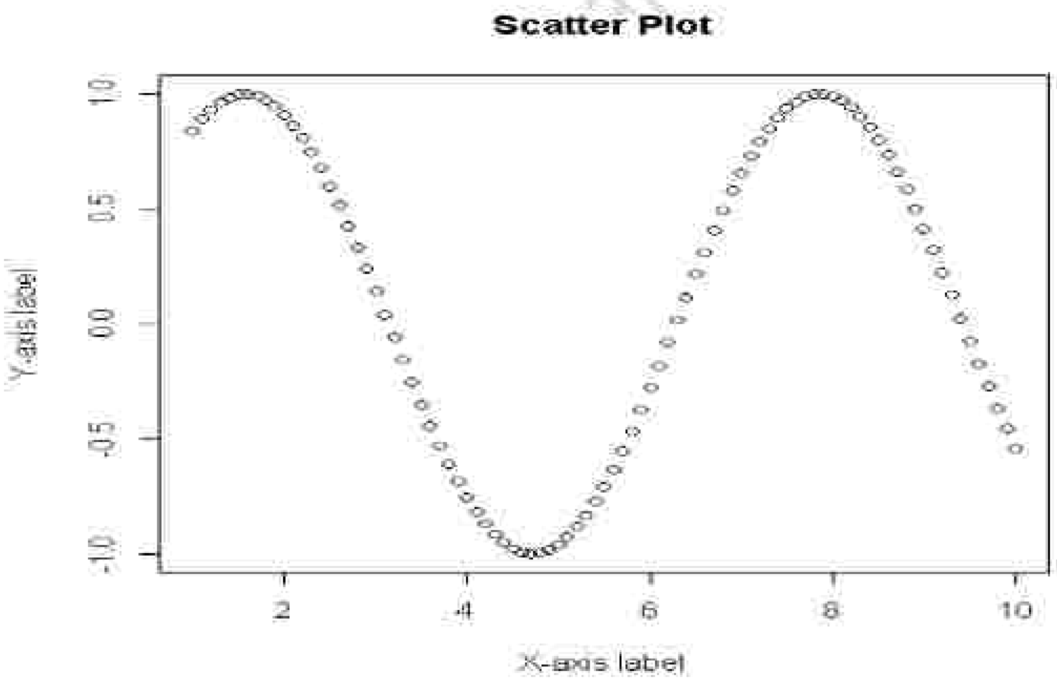
1.Titles and Labels:

- **main:** Main title of the plot.
- **xlab and ylab:** Labels for the x and y-axes.

  Example:

  ```
  plot(x, y, main="Scatter Plot", xlab="X-axis label", ylab="Y-axis label")
  ```

  output:



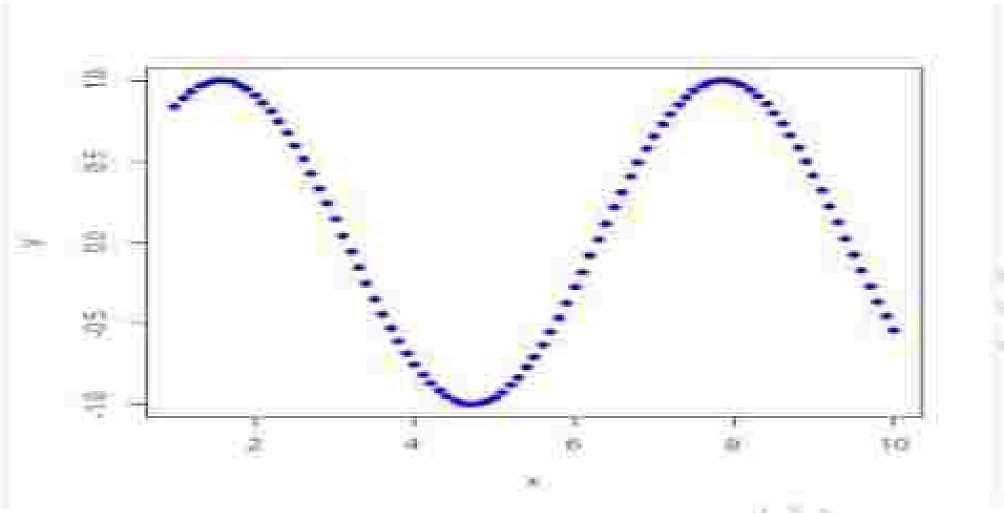
2.Colors and Symbols:

col: Color of the points or lines.
pch: Symbol to be used for plotting points.

Example:

```
plot(x, y, col="blue", pch=16)
```

Output:

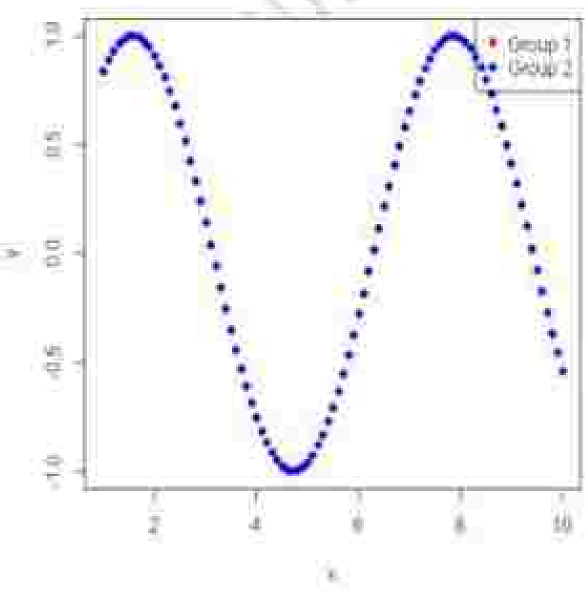


3.Adding a Legend:

legend: Adding a legend to the plot.
Example:

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

output:



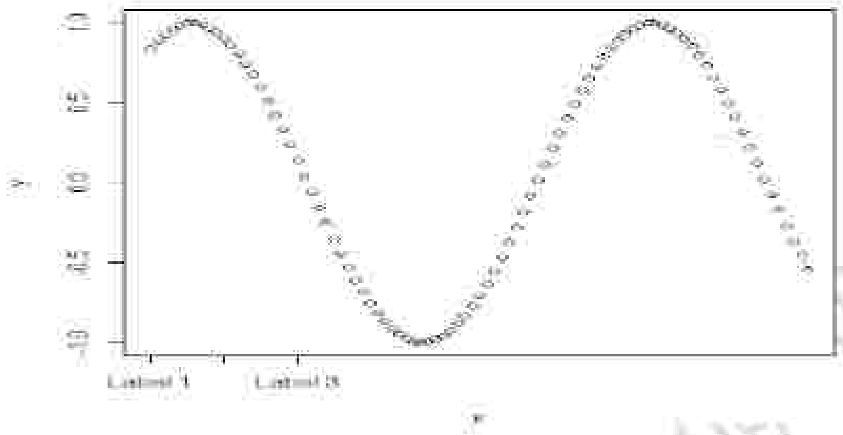
4.Axis Customization:

axis: Customize the axis ticks and labels.

Example:

```
plot(x, y, xaxt="n")  # Turn off x-axis
axis(1, at=c(1, 2, 3), labels=c("Label 1", "Label 2", "Label 3"))
```

output:



5.Lines:

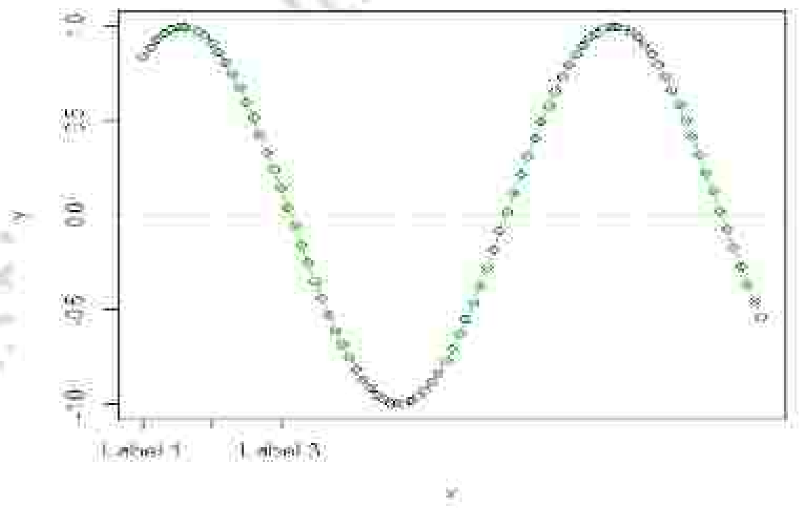
lines: Add lines to the plot.

abline: Add a straight line.

Example:

```
lines(x, y, col="green")  abline(h=0, v=0,
col="red", lty=2)
```

output:



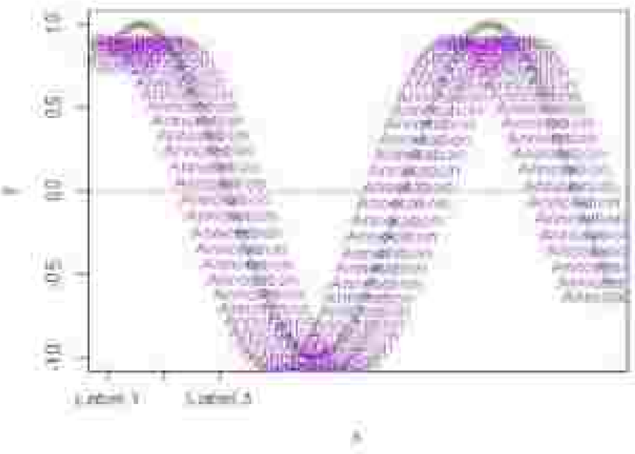
6.Annotations:

text: Add text annotations to the plot.

Example:

```
text(x, y, labels="Annotation", pos=1, col="purple")
```

output:



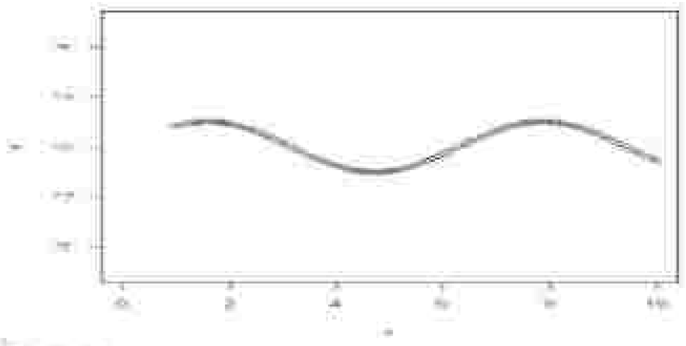
7.Plot Range:

xlim and ylim: Set the range of the x and y-axes.

Example:

```
plot(x, y, xlim=c(0, 10), ylim=c(-5, 5))
```

output:



8.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

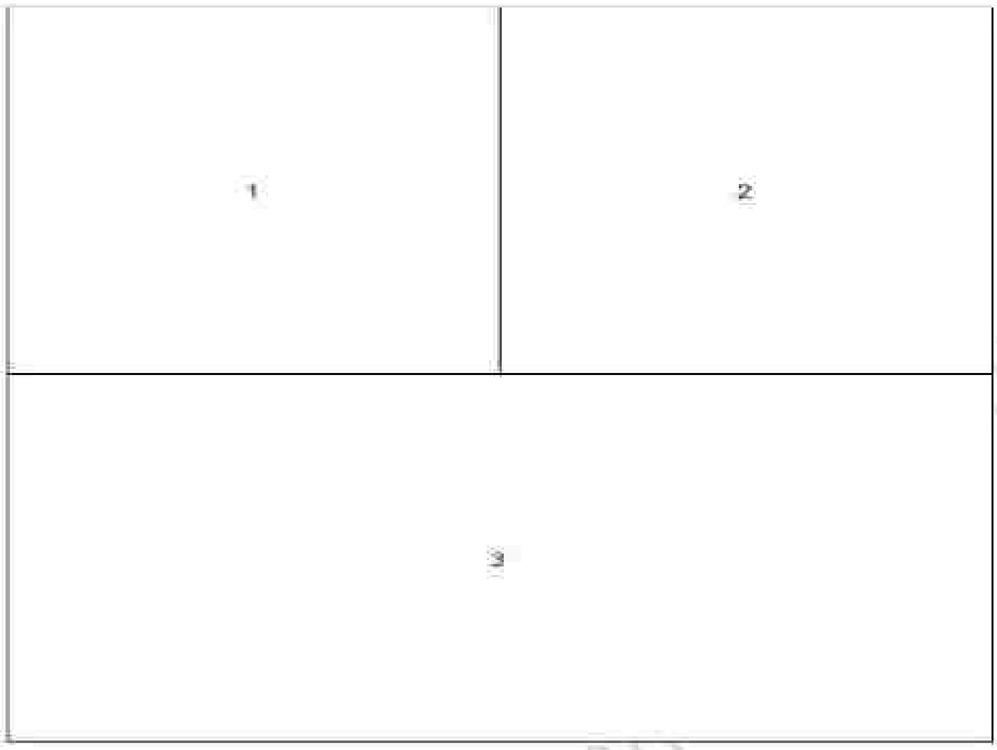
Example: lay. Mat <- matrix(c(1,3,2,3),2,2)

```
lay. Mat

      [,1] [,2]
[1,]    1    2
[2,]    3    3
```

layout(mat=lay. Mat)

```
layout. show(n=max(lay.  Mat))
```

Output:



Plotting Regions and Margins

Definition: Regions are areas of a Minitab graph that can be resized, relocated, and formatted with color, fill patterns, edge line types, and more.

- 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.
- 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.
- The margins of a plot can be customized with the **mar** graphical parameter. These graphical parameters are vectors of the form **c(bottom, left, top, right)**, where each element represent the margin of each side of the plot (in margin lines or in inches). Note that you can access the current values with **par("mar").**

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

1. **plot region:** The *plot region* is all you've dealt with so far. This is where your actual plot appears and where you'll usually be drawing your points, lines, text, and so on. The plot region uses the *user*

coordinate system, which reflects the value and scale of the horizontal and vertical axes.

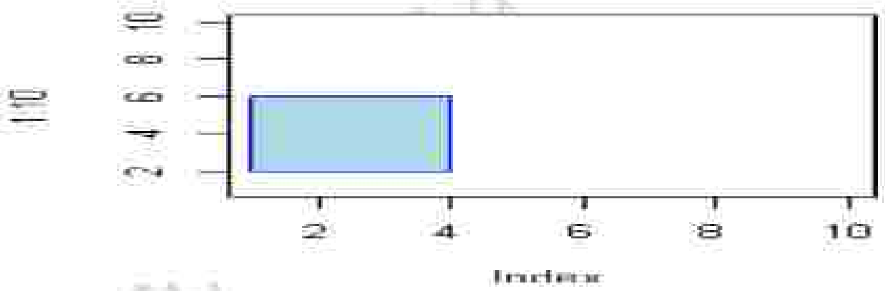
2. **Figure region**: *The figure region* is the area that contains the space for your axes, their labels, and any titles. These spaces are also referred to as the *figure margins*.
3. **outer region** :The *outer region*, also referred to as the *outer margins*, is additional space around the figure region that is not included by default but can be spec- ified if it's needed

1.Plotting Regions: Use rect to draw rectangles or regions on a plot.

Example:

```
 # Draw a rectangle from (1, 2) to (4, 6)
plot(1:10, type="n") # Create an empty
plotrect(1, 2, 4, 6, col="lightblue", border="blue")
```

output:



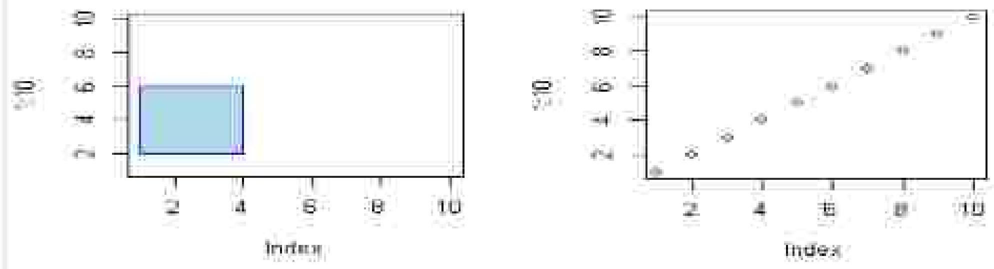
2.Plotting Margins:: Margin is the space between the chart border and the canvas border. You can set the chart margins on any one of the chart's four sides.

Adjust the margins using par(mar=...). The mar parameter specifies the margin sizes in inches (bottom, left, top, right).

Example:

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

output:

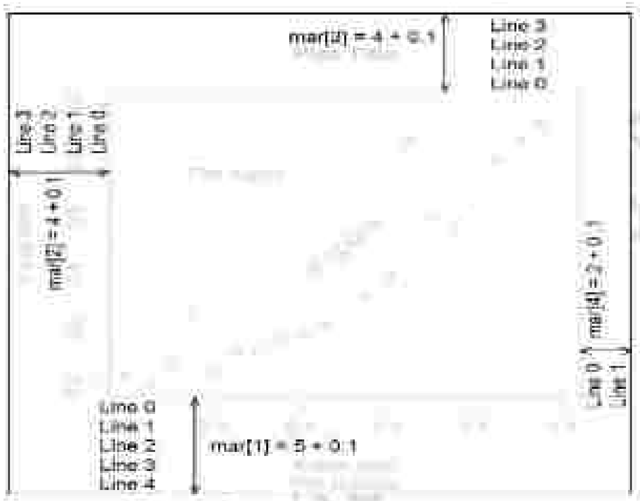


Special built in function using plot regions and margins.

1.mar (margin lines)

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

Note that the 0.1 is just for adding an extra space to fit everything inside the plot area.



2.Outer margin(oma)

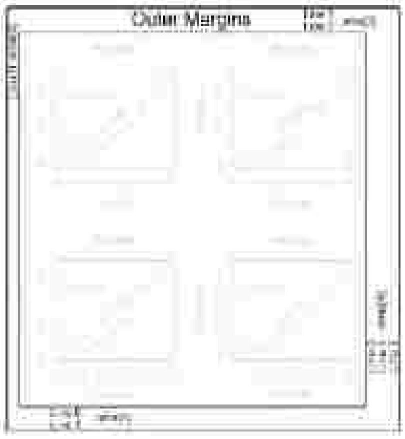
- The margin in margin lines with **oma**. The outer margins are specially useful for adding text to a combination of plots.
- To access the current outer margins with par("oma").

In this example we set

par(oma = c(2, 1, 2, 3) + 0.1).

so two lines are displayed below the plot, one on the left, two on the top and three on the right.

Output:



Example:

par(oma=c(1,4,3,2),mar=

4:7) R> plot(1:10)

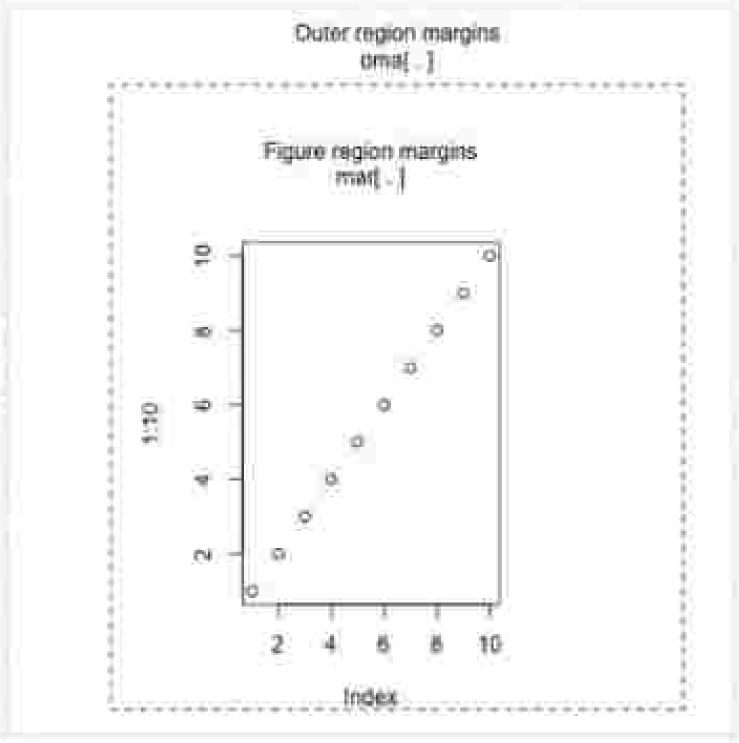
R>

box("figure",lty=2)

R>

box("outer",lty=3)

Output:



Note: The graphical parameters **oma (outer margin)** and **mar** (figure margin) are used to control these regions and magins like **mfrow**, they are initialized through a call to par before you begin to draw any new plot.

3.Default spacing:default figure margin settings with a call to par in R

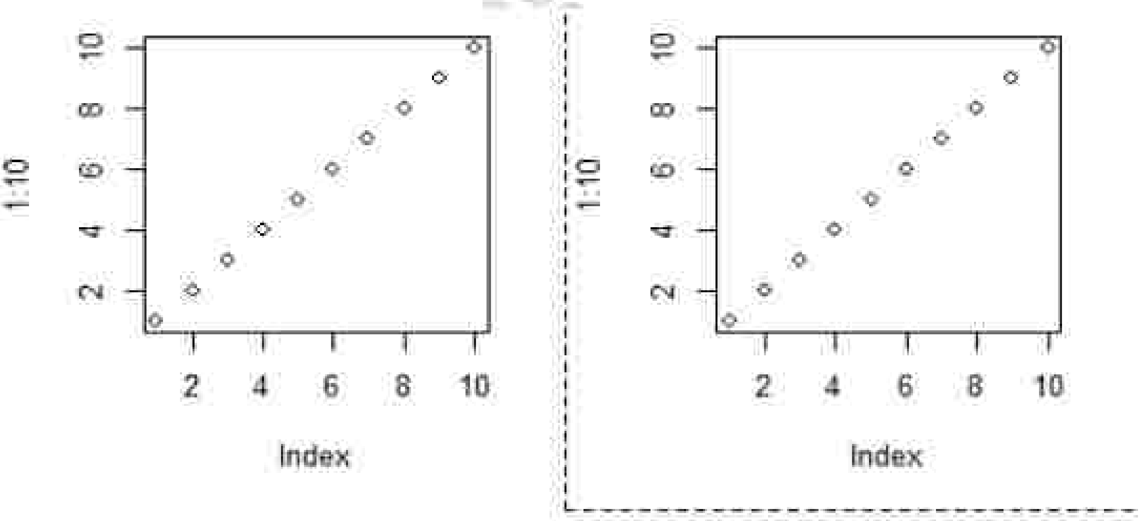
Example:

```
1.par()$oma

[1] 0 0 0 0


2. par()$mar

[1] 5.1 4.1 4.1 2.1
```

here that oma=c(0, 0, 0, 0)—there is no outer margin set by default. The default figure margin space is mar=c(5.1, 4.1, 4.1, 2.1)—in other words, 5.1 lines of text on the bottom, 4.1 on the left and top, and 2.1 on the right.

3.consider the image on the left of Figure 23-4, created in a fresh graphics device with the following:

```
plot(1:10)
box(which="figure",lty=2)
```



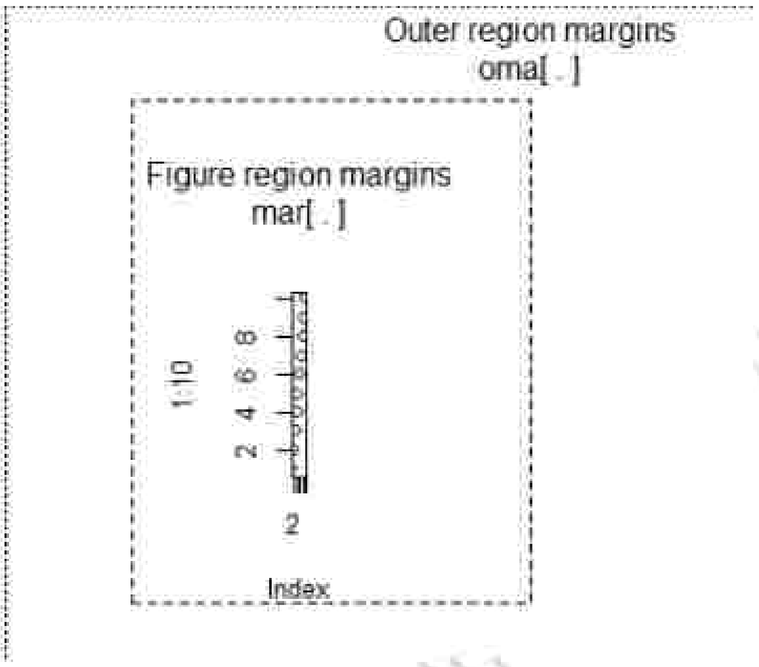
4.Custom Spacing: Let's produce the same plot but with tailored outer margins so that the bottom, left, top, and right areas are one, four, three, and two lines, respec tively, and the figure margins are four, five, six, and seven lines.

Example:

```
par(oma=c(1,4,3,2),mar=4:7)
 plot(1:10)
```

```
box("figure",lty=2)
box("outer",lty=3)

mtext("Figure region margins nmar[ . ]",line=2)
mtext("Outer region margins noma[ . ]",line=0.5,outer=TRUE)
```

Output:



provide the text you want written in a character string as the first argument, and the argument line instructs how many lines of space away from the inside border the text should appear.

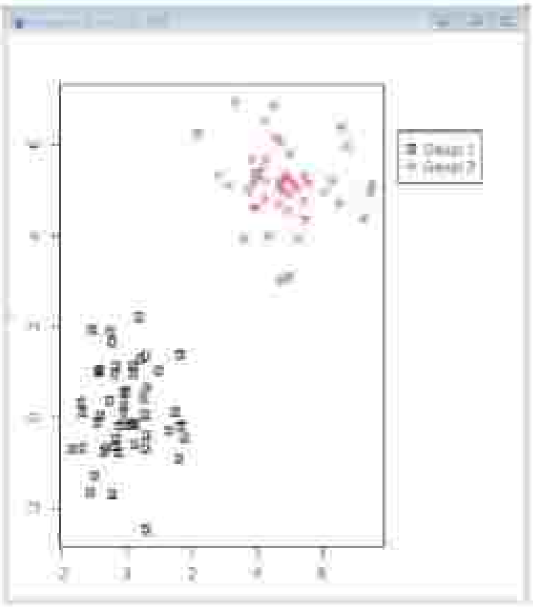
5.Clipping:
Controlling *clipping* allows you to draw in or add elements to the mar- gin regions with reference to the user coordinates of the plot

Example:

```
# create margin around plot
par(mar = c(3, 3, 3, 8), xpd = TRUE)

# Draw scatter plot
plot(sample_data$x, sample_data$y, col = sample_data$group)

# Draw legend
```

legend("topright", inset = c(-0.3, 0.1), legend = c("Group 1","Group 2"), pch = c(11,12), col = 1:2)

Output:



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
- Under typical circumstances, R can read mouse clicks you make inside the device.

Features of Point-and-Click Coordinate Interaction:

1. Retrieving Coordinates Silently
2. Visualizing Selected Coordinates
3.Ad Hoc Annotation

1. Retrieving Coordinates Silently:

- The locator command allows you to find and return user coordinates.
- To see how it works, first execute a call to plot(1,1) to bring up a simple plot with a single point in the middle.
- To use locator, you simply execute the function (with no arguments for default behavior), which will "hang" the console, without returning you to the prompt.

Example:

```
plot(1,1)
locator()
$x
[1] 0.8275456 1.1737525 1.1440526 0.8201909
$y
[1] 1.1581795 1.1534442 0.9003221 0.8630254
```

2.Visualizing Selected Coordinates

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

Example:

```
plot(1,1)

list <- locator(type="o",pch=4,lty=2,lwd=3,col="red",xpd=TRUE) R>

list

$x

[1] 0.5013189 0.6267149 0.7384407 0.7172250 1.0386740 1.2765699

[7] 1.4711542 1.2352573 1.2220592 0.8583484 1.0483300 1.0091491


$y

[1] 0.6966016 0.9941945 0.9636752 1.2819852 1.2766579 1.4891270

[7] 1.2439071 0.9630832 0.7625887 0.7541716 0.6394519 0.9618461
```

- Drawing using locator requires you to specify the plot type,
- Selecting type="o" (as opposed to the silent default, type="n") is what produces the overpotted points and lines in Figure 23-6. For just points, use type="p"; for just lines, use type="l".

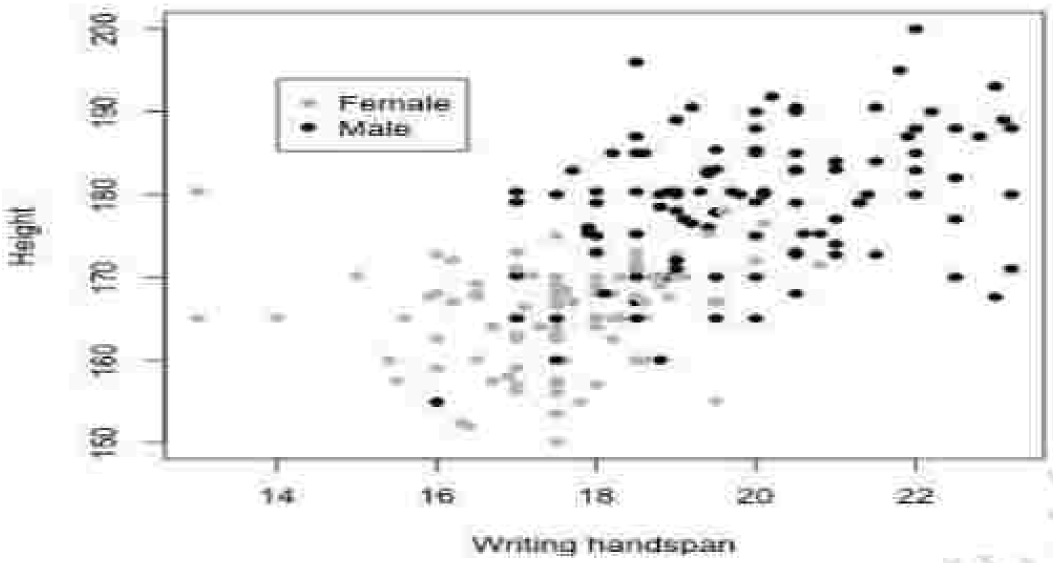
2. Ad Hoc Annotation

   The locator function also allows you to place ad hoc annotations, such as legends, on your plot—remember, since locator returns valid R user coordinates, these results can directly form the positional argument of most standard annotation functions.

- **Example:**

  ```
  plot(survey$Height~survey$Wr.Hnd,pch=16,col=c("gray","black")[as.numeric(survey$
  Sex)], xlab="Writing handspan",ylab="Height")
  ```
- `legend(locator(n=1),legend=levels(survey$Sex),pch=16,col=c("gray","black"))`

OUTPUT:



CUstomizing Traditional R Plots:

- customize traditional plots extensively for instance, in a scatter plot, you can adjust the colors, sizes, shapes of points, add labels, change axes appearance, titles, legends, and much more using various functions within the ggplot2 package or even base R graphics functions.
- 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

1. Graphical Parameters for Style and Suppression
2. Customizing Boxes
3. Customizing Axes

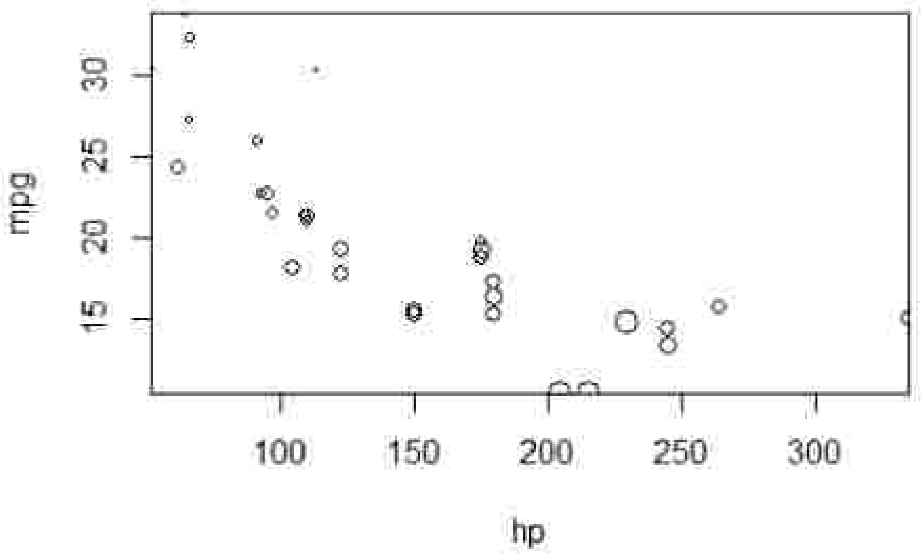
1) Graphical Parameters for Style and Suppression.

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.

Example:

```
hp <- mtcars$hp

mpg <- mtcars$mpg

wtcex <- mtcars$wt/mean(mtcars$wt)

plot(hp,mpg,cex=wtcex)

plot(hp,mpg,cex=wtcex,xaxs="i",yaxs="i")
```

Output:



- let's plot MPG against horsepower (from the ready-to-use mtcars data set) and set each plotted point to be sized proportionally to the weight of each car.
- **This plot is almost the same as the default, but note now that there's no padding space at the end of the axes; the most extreme data points sit right on the axes.**

2.Customizing Boxes:

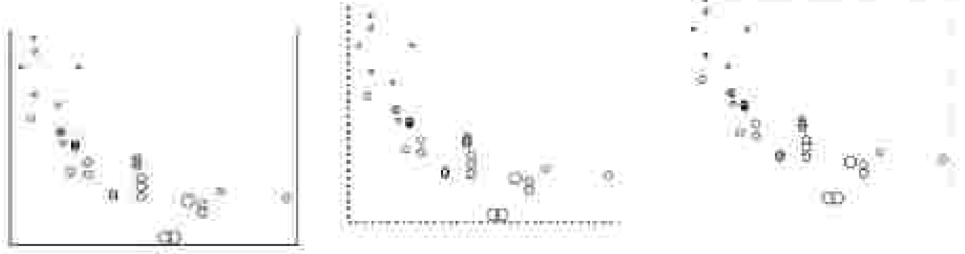
To add a box specific to the current plot region in the active graphics device, you use box and specify its type with bty.

box(bty="u")

Example:

box(bty="]",lty=2,col="gray")

Output:



Various box configurations added to the mtcars scatterplot

- The bty argument is supplied a single character: "o" (default), "l", "7", "c", "u", "]", or "n". The help file entry for bty

- The resulting box boundaries will follow the appearance of the corresponding uppercase letter, with the exception of "n"

3.Customizing Axes

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

The first argument it takes is side, provided with a single integer: 1 (bottom), 2 (left), 3 (top), or 4 (right).

Syntax:

axis(side, at=NULL, labels=TRUE)

Parameters:

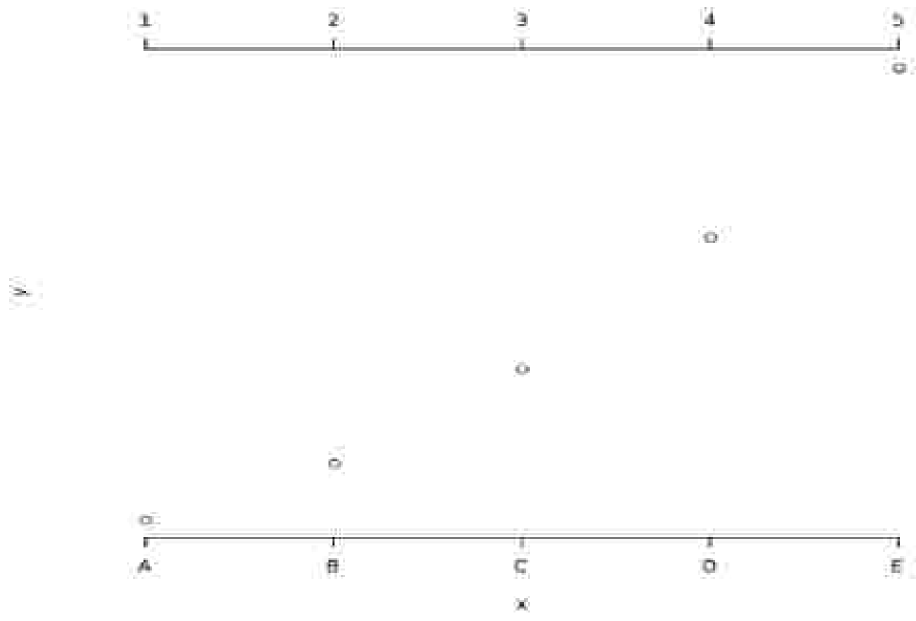
side: It defines the side of the plot the axis is to be drawn on possible values such as below, left, above, and right.

at: Point to draw tick marks

Example:

```
x <- 1:5; y = x * x
plot(x, y, axes =
FALSE)
# Calling the axis() function
axis(side =1, at = 1:5, labels =
LETTERS[1:5]) axis(3)
```

Output: Output:



Specialized Text and Label Notation:

Immediately accessible tools for controlling fonts and displaying special notation, such as Greek symbols and mathematical expressions.

Features of Specialized Text and Label Notation:

1.Font:

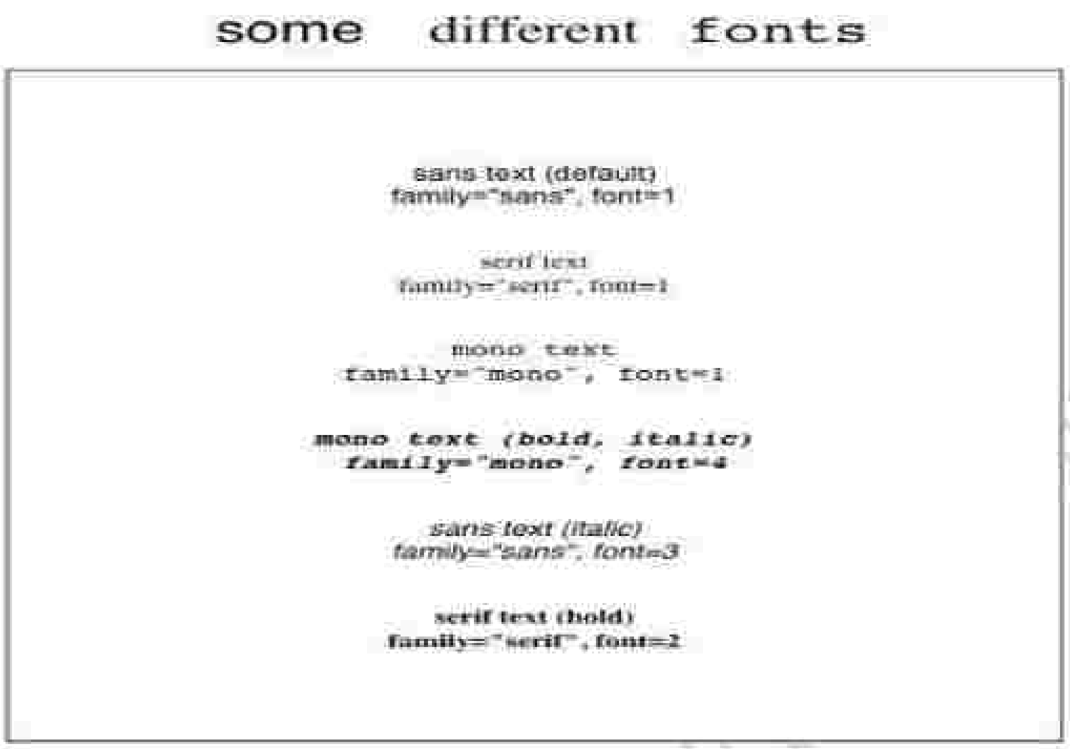
The displayed font is controlled by two graphical parameters: family for the specific font family and font, an integer selector for controlling bold and italic typeface.

- Available fonts depend on both your operating system and the graphics
- Three generic families— "sans" the default, "serif", and "mono"— are always available
- These are paired with the four possible values of font—1 (normal text, default), 2 (bold), 3 (italic), and 4 (bold and italic).

Example:

```
par(mar=c(3,3,3,3))
plot(1,1,type="n",xlim=c(-1,1),ylim=c(0,7),xaxt="n",yaxt="n",ann=FALSE)
text(0,6,label="sans  text (default)\nfamily=\"sans\",  font=1")
text(0,5,label="serif  text\nfamily=\"serif\",  font=1", family="serif",font=1)
text(0,4,label="mono  text\nfamily=\"mono\",  font=1", family="mono",font=1)
text(0,3,label="mono  text (bold,  italic)\nfamily=\"mono\",  font=4", family="mono",font=4)
```

OUTPUT:



some different fonts

sans text (default)
family="sans", font=1

serif text
family="serif", font=1

mono text
family="mono", font=1

mono text (bold, italic)
family="mono", font=4

sans text (italic)
family="sans", font=3

serif text (bold)
family="serif", font=2

Displaying font styles through use of the family and font graphical parameters

2. Greek Symbols

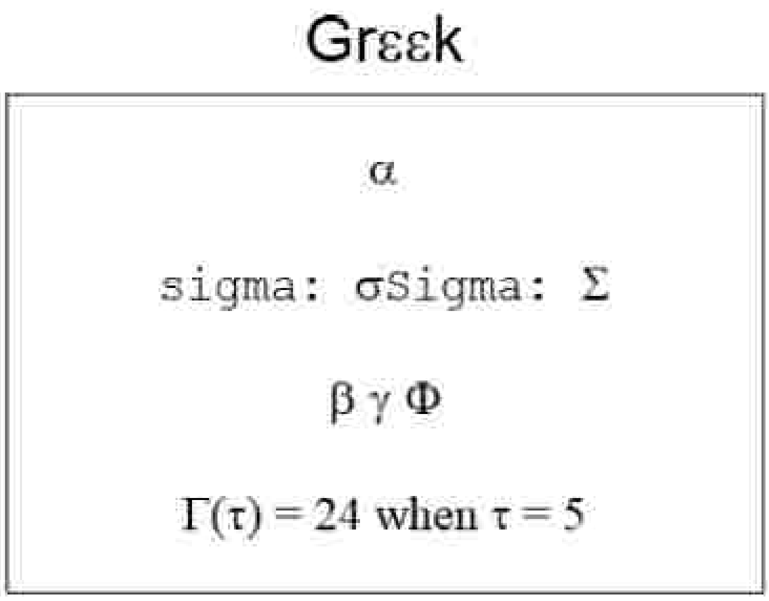
For statistically or mathematically technical plots, annotation may occasionally require Greek symbols or mathematical markup.

- display these using the expression function.

Example:

```
par(mar=c(3,3,3,3))
plot(1,1,type="n",xlim=c(-1,1),ylim=c(0.5,4.5),xaxt="n",yaxt="n", ann=FALSE)
text(0,4,label=expression(alpha),cex=1.5)
text(0,3,label=expression(paste("sigma: ",sigma,"Sigma: ",Sigma)),
family="mono",cex=1.5)
 text(0,2,label=expression(paste(beta," ",gamma," ",Phi)),cex=1.5)
text(0,1,label=expression(paste(Gamma,"(",tau,") = 24 when ",tau," = 5")),
family="serif",cex=1.5)
title(main=expression(paste("Gr",epsilon,epsilon,"k")),cex.main=2)
```

Output:



3.Mathematical Expressions:

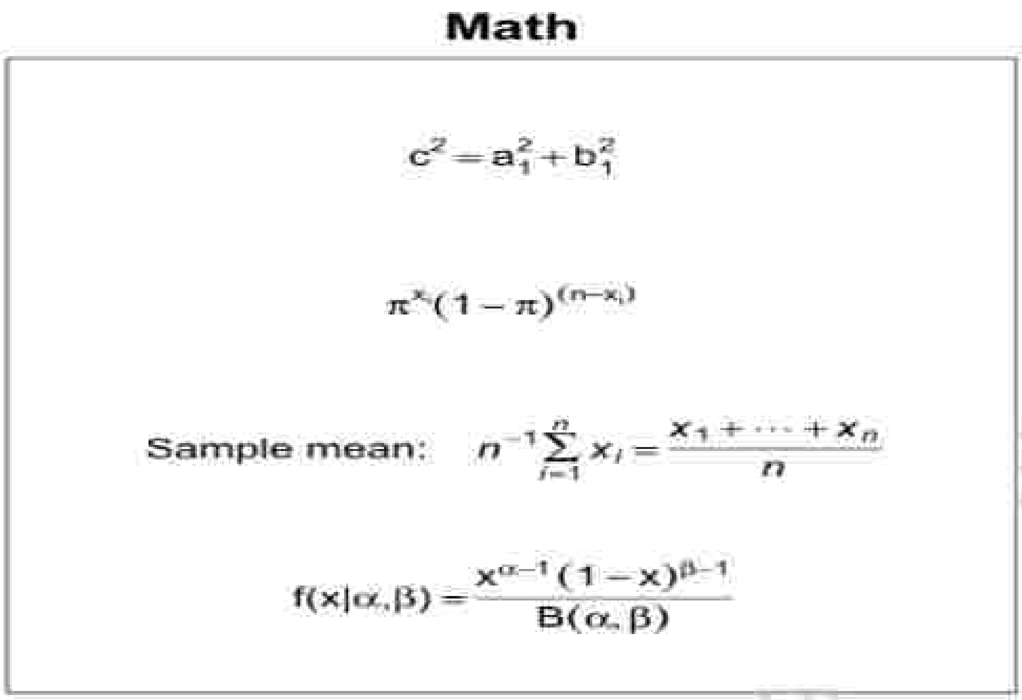
Formatting entire mathematical expressions to appear in R plots is a bit more complicated and is reminiscent of using markup languages

To create the image, I first defined four expression objects as follows:

Example:

```
expr1 <- expression(c^2==a[1]^2+b[1]^2)
expr2 <- expression(paste(pi^{x[i]},(1-pi)^(n-x[i])))
expr3 <- expression(paste("Sample mean:",italic(n)^{-
1},sum(italic(x)[italic(i)], italic(i)==1, italic(n))
==frac(italic(x)[1]+...+italic(x)[italic(n)], italic(n))))
expr4 <- expression(paste("f(x","|",alpha,",",beta,")"==frac(x^{alpha-1}~(1-x)^{beta-1},
B(alpha,beta))))
```

Output:



Examples of typesetting mathematical expressions in R plots

Defining colors and plotting in higher dimensions:

Colors: The colors using various methods, such as specifying named colors, hexadecimal color codes, RGB values, HSL values, among others. The choice of color representation depends on your preference and the context of your visualization.

1.Named Colors:

R provides a range of named colors you can use directly. Some common ones include:

- Red: **"red"**
- Blue: **"blue"**
- Green: **"green"**
- Yellow: **"yellow"**
- Black: **"black"**
- White: **"white"**

2.Hexadecimal Color Codes:

Hexadecimal color codes are a common way to define colors. 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"**

2. Green: **"#00FF00"**
3. Blue: **"#0000FF"**
4. Orange: **"#FFA500"**
5. Purple: **"#800080"**

3.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.

1. Red: **rgb(1, 0, 0)**
2. Green: **rgb(0, 1, 0)**
3. Blue: **rgb(0, 0, 1)**
4. Yellow: **rgb(1, 1, 0)**
5. Purple: **rgb(0.5, 0, 0.5**

4.HSL Values

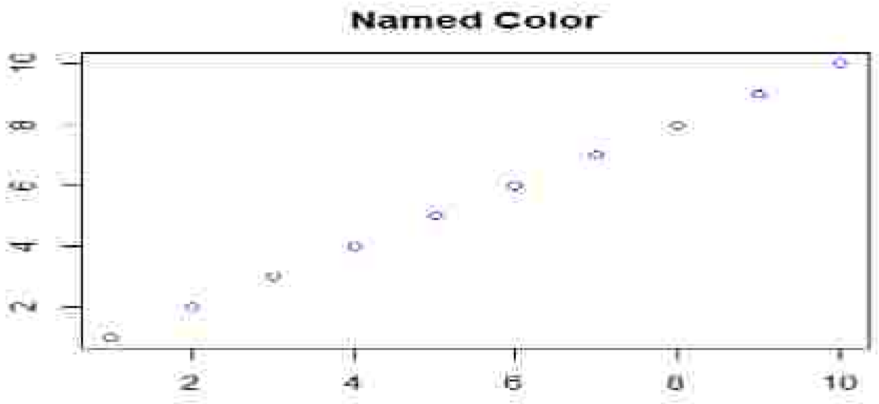
HSL (Hue, Saturation, Lightness) values can also be used to define colors. The hue value represents the color (0-360), saturation represents intensity (0-1), and lightness represents the brightness (0-1). For example:

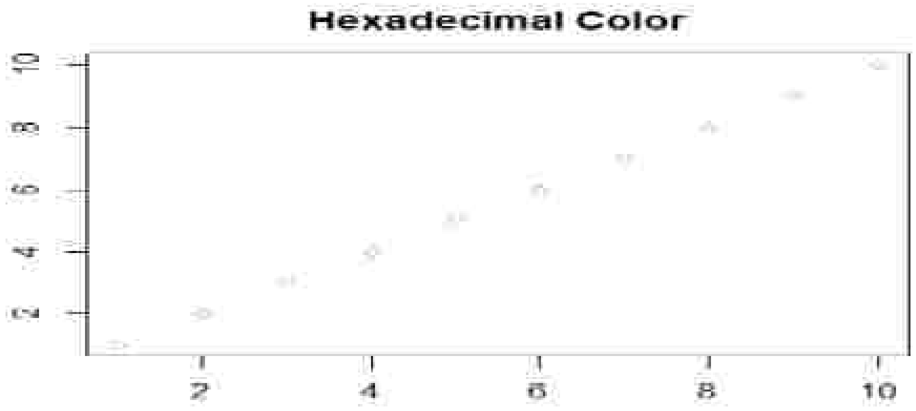
1. Red: **hsl(0, 1, 0.5)**
2. Green: **hsl(120, 1, 0.5)**
3. Blue: **hsl(240, 1, 0.5)**
4. Yellow: **hsl(60, 1, 0.5)**
5. Purple: **hsl(300, 1, 0.5)**

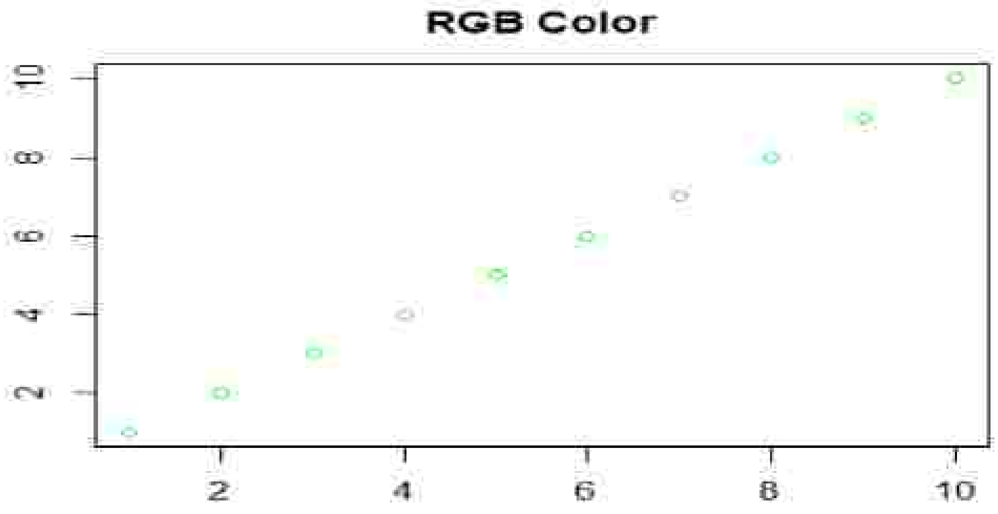
Example:

```
# Using different color representations in a plot

my_color <- "red"
  plot(1:10, col=my_color, pch=16)
plot(1:10, col = "blue", main = "Named Color")

  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 = rgb(0, 1, 0), main = "RGB Color")


plot(1:10, col = hsl(240, 1, 0.5), main = "HSL Color")
```

OUTPUT:







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:**

1.Red-Green-Blue Hexadecimal Color Codes:

- To specifying colors in plots, your instruction to R so far has been given either in the form of an integer value from 1 to 8 or as a character
- One of the most common methods of color specification is to specify different *saturations* or *intensities* of three primaries—red, green, and blue (RGB)

Example:

```
1.palette()
[1] "black"  "red"   "green3"  "blue"  "cyan"   "magenta"
[7] "yellow"   "gray"
```

The col argument lets you select one of eight colors when you supply it an integer from 1 to 8

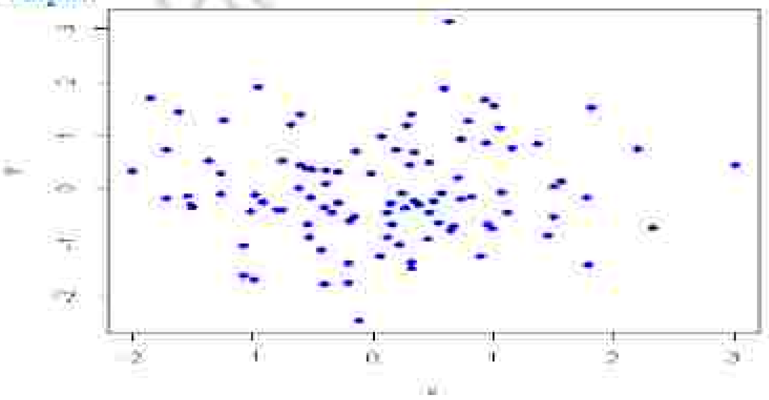
```
2. rgb(t(col2rgb(c("black","green3","pink"))),maxColorValue=255)
[1] "#000000" "#00CD00" "#FFC0CB"
```

2.2D 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.

Scatter Plot:

```
Using basic plot()
function. x <-
rnorm(100)
y <- rnorm(100)
plot(x, y, col="blue", pch=16)
```

output:

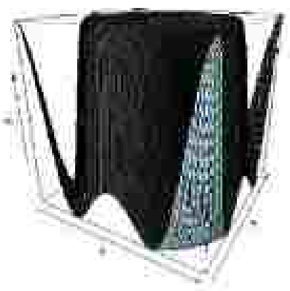


3.Surface Plot: Surface plots are diagrams of three-dimensional data. Rather than showing the individual data points, surface plots show a functional relationship between a designated dependent variable (Y), and two independent variables (X and Z). The plot is a companion plot to the contour plot.

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)
```

output:



REPRESENTING AND USING COLOR:

- One of the most common methods of colour 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 colour.
- To express these values in (R, G, B) order, the result is commonly referred to as a triplet. For example, (0;0;0) represents pure black, (255;255;255) represents pure white, and (0;255;0) is full green.

1.Palette(): Apalette in Rit is simply a built in finction holds the few colours.

```
palette()
[1] "black"  "red"   "green3"  "blue"   "cyan"   "magenta"  "yellow"
[8] "gray
```

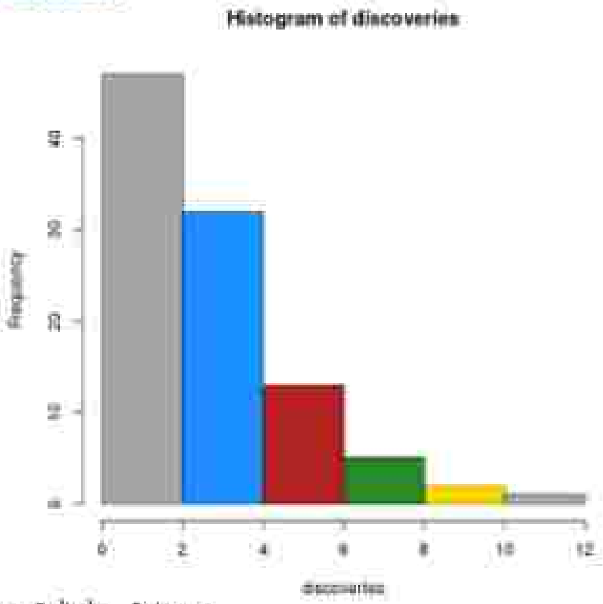
2.Defining your own palettes

If you want to make your own palette, you can just create your own vector of colours.

Example:

```
colors <- c("#A7A7A7", "dodgerblue", "firebrick", "forestgreen", "gold")

hist(discoveries, col = colors)
```

OUPUT:



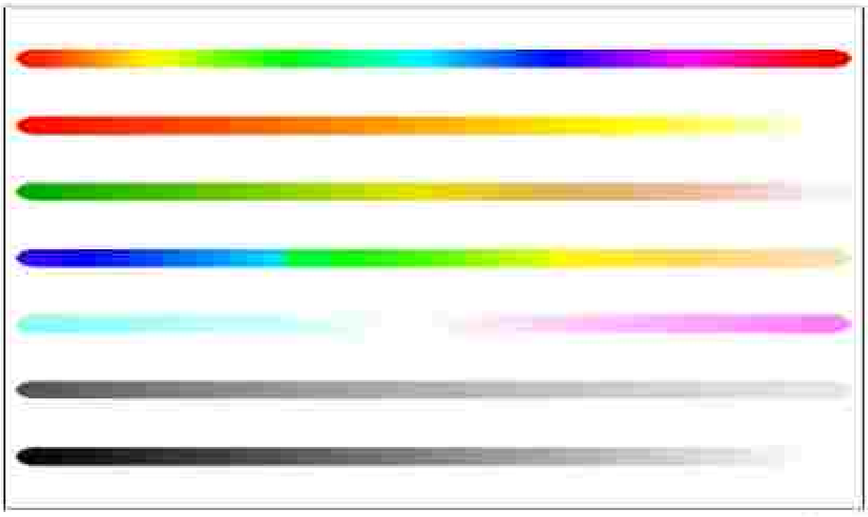
2. Built-in Palettes

- 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 installa- tion. These are defined by the functions rainbow, heat.colors, terrain.colors, topo.colors, cm.colors, gray.colors, and gray.

Example:

```
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))
dev.new(width=8,height=3)
par(mar=c(1,8,1,1))
plot(1,1,xlim=c(1,N),ylim=c(0.5,7.5),type="n",xaxt="n",yaxt="n",ann=FALSE)
points(rep(1:N,7),rep(7:1,each=N),pch=19,cex=3,col=c(rbow,heat,terr,topo,cm,gry1,gry2))
```

OUTPUT:



Showcasing the color ranges of the built-in palettes, with default limits used in gray.colors.

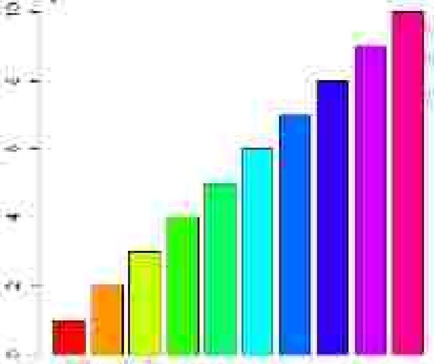
3.Color Palettes:

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

Example:

```
# Example using a color palette
colors <- rainbow(10)  # Generates 10 colors of the
rainbow barplot(1:10, col=colors)
```

output:



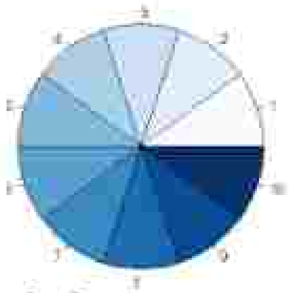
4. Color Functions:

R has functions for creating color gradients or interpolating colors.

Example:

```
library(RColorBrewer)
colors <- colorRampPalette(brewer.pal(9,
"Blues"))(10) pie(rep(1, 10), col=colors)
```

output:

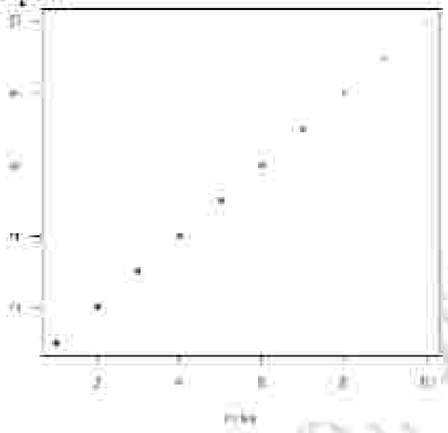


5.Color Scales:

R supports various color scales, like viridis, magma, etc.
Example:

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

output:



<u>3D SCATTER PLOTS.:</u>

creating 3D scatterplots, which allow you to plot raw observations based on three continuous variables at once, as opposed to only two in a conventional 2D scatterplot

- 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:

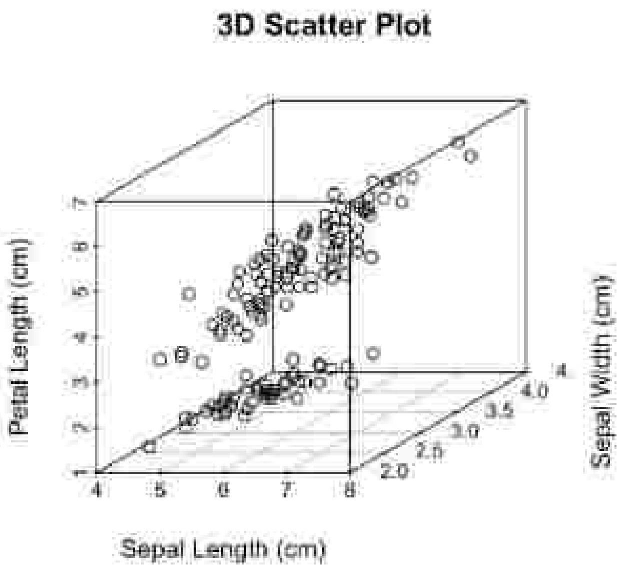
scatterplot3d(x, y=NULL, z=NULL)

x, y, z are the coordinates of points to be plotted.

Example:

scatterplot3d(iris[,1:3], main="3D Scatter Plot", xlab = "Sepal Length(cm)", ylab = "Sepal Width (cm)", zlab = "Petal Length (cm)")

Output:



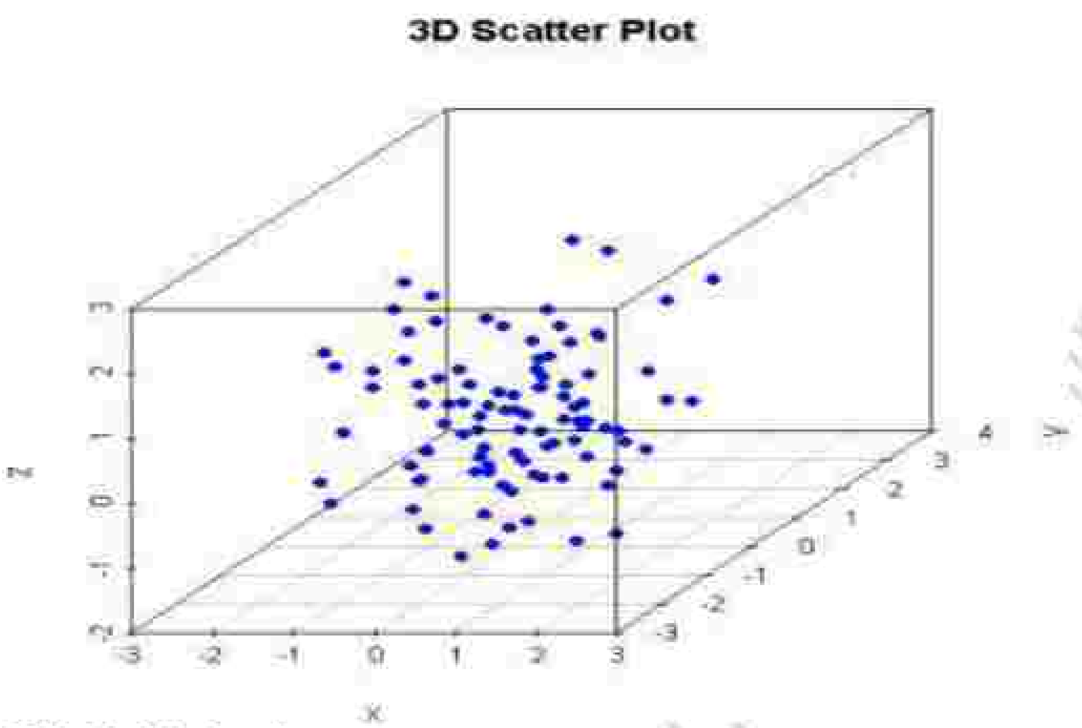
<u>Packages Using 3D scatterplot</u>

- Creating 3D scatter plots in R can be achieved using different packages.
- One popular package for this purpose is scatterplot3d. Here's a basic example of how to create a 3Dscatter plot using this package:

Example:

```
#Install and load the scatterplot3d package
install.packages("scatterplot3d")
library(scatterplot3d)
# Generate some random data set.seed(123)
x <- rnorm(100)
y <- rnorm(100)
z <- rnorm(100)

# Create a 3D scatter plot
scatterplot3d(x, y, z, color="blue", pch=16, main="3D Scatter Plot")
```

OUTPUT:



Fifth Module Questions:

I. Two marks Questions:

1.What is simple linear regression.

2.Define plotting regions and margins.

3.What is 3D scatter plots.

4.Define regions and margins.

5.what are all the specialized text

II. Three marks Questions:

1. define simple linear regression with example.
2. What is prediction? Explain prediction /confidence interval
3. Explain plot customization.
4. Discuss specialized text and label notation.
5. Explain 3D scatter plots

III. Five marks Questions:

1.what is simple linear regression in detail.

2.Explain plot customization.

3.Defining colours and plotting in higher dimensions.

4. write a r program to create an any application of linear regression in multivariate context for predictive purpose.

5.Explain representing and using colour.

IV. Ten marks Questions:

1.Discuss simple linear regression.

2.Briefly explain Multi linear regression.

3.Explain plotting regions and margins.

4.Explain customizing traditional R plots.

5.Explain 3D scatter plot in details.