Contents

Basics of R

R is a versatile programming language for data analysis and statistics. You can use it as a calculator, perform data manipulations, and write scripts.

Basic Arithmetic Operations

Open R or RStudio and try basic arithmetic operations:

```
# Addition
3 + 5
```

```
## [1] 8
```

```
# Subtraction
10 - 4
```

```
## [1] 6
```

```
# Multiplication
2 * 6
```

```
## [1] 12
```

```
# Division
8 / 2
```

```
## [1] 4
```

Variables

In R, you can store values in variables. Here's how you create and use variables:

```
# Assign a value to a variable
x <- 10

# Print the value of the variable
print(x)
```

```
## [1] 10
```

```
# Perform operations with variables
y <- x * 2
print(y)
```

```
## [1] 20
```

Vectors

Vectors are one-dimensional arrays in R that can hold multiple values of the same data type.

```
# Create a numeric vector
numbers <- c(1, 2, 3, 4, 5)

# Create a character vector
fruits <- c("apple", "banana", "cherry")

# Access elements in a vector
print(numbers[3])  # Access the third element
```

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

```r
print(fruits[2])   # Access the second element
```

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

```r
# Add two vectors element-wise
vec1 <- c(1, 2, 3)
vec2 <- c(4, 5, 6)
result <- vec1 + vec2
print(result)
```

```
## [1] 5 7 9
```

```r
# Create two numeric vectors
vec1 <- c(1, 2, 3, 4, 5)
vec2 <- c(6, 7, 8, 9, 10)

# Calculate the mean of each vector
mean_vec1 <- mean(vec1)
mean_vec2 <- mean(vec2)
print(mean_vec1)
```

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

```r
print(mean_vec2)
```

```
## [1] 8
```

```r
# Calculate the dot product of the two vectors
dot_product <- sum(vec1 * vec2)
print(dot_product)
```

```
## [1] 130
```

```r
# Create a logical vector based on a condition
logical_vector <- vec1 > 3
print(logical_vector)
```

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

Dataframes

Dataframes are two-dimensional data structures in R, similar to tables in a database or Excel spreadsheet.

```r
# Create a dataframe
df <- data.frame(
  Name = c("Alice", "Bob", "Charlie"),
  Age = c(25, 30, 22),
  City = c("New York", "Los Angeles", "Chicago")
)

# Access dataframe columns
print(df$Name)
```

```
## [1] "Alice"   "Bob"     "Charlie"
```

```r
print(df$Age)
```

```
## [1] 25 30 22
```

```r
# Add a new column
df$Salary <- c(50000, 60000, 45000)

# Filter rows based on a condition
young_people <- df[df$Age < 30, ]
print(young_people)
```

```
##      Name Age     City Salary
## 1   Alice  25 New York  50000
## 3 Charlie  22  Chicago  45000
```

```r
# Summary statistics
summary(df)
```

```
##      Name                Age            City               Salary
##  Length:3           Min.   :22.00   Length:3           Min.   :45000
##  Class :character   1st Qu.:23.50   Class :character   1st Qu.:47500
##  Mode  :character   Median :25.00   Mode  :character   Median :50000
##                     Mean   :25.67                      Mean   :51667
##                     3rd Qu.:27.50                      3rd Qu.:55000
##                     Max.   :30.00                      Max.   :60000
```

```r
# Create a dataframe
students <- data.frame(
  Name = c("Alice", "Bob", "Charlie", "David"),
  Age = c(25, 30, 22, 28),
  Grade = c("A", "B", "C", "A")
)

# Add a new row to the dataframe
new_student <- data.frame(Name = "Eva", Age = 24, Grade = "B")
students <- rbind(students, new_student)

# Filter rows based on a condition
young_students <- students[students$Age < 25, ]
print(young_students)
```

```
##      Name Age Grade
## 3 Charlie  22     C
## 5     Eva  24     B
```

```r
# Sort the dataframe by age in descending order
sorted_students <- students[order(-students$Age), ]
print(sorted_students)
```

```
##      Name Age Grade
## 2     Bob  30     B
## 4   David  28     A
## 1   Alice  25     A
## 5     Eva  24     B
## 3 Charlie  22     C
```

Loops

Loops allow you to perform repetitive tasks in R.

For Loop

```r
# Print numbers from 1 to 5 using a for loop
for (i in 1:5) {
  print(i)
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
```

```r
# Print the squares of numbers from 1 to 5 using a for loop
for (i in 1:5) {
  square <- i^2
  print(paste(i, "squared is", square))
}
```

```
## [1] "1 squared is 1"
## [1] "2 squared is 4"
## [1] "3 squared is 9"
## [1] "4 squared is 16"
## [1] "5 squared is 25"
```

While Loop

```r
# Print numbers from 1 to 5 using a while loop
i <- 1
while (i <= 5) {
  print(i)
  i <- i + 1
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
```

```r
# Calculate the factorial of a number using a while loop
n <- 5
factorial <- 1
i <- 1

while (i <= n) {
  factorial <- factorial * i
  i <- i + 1
}

print(paste("Factorial of", n, "is", factorial))
```

```
## [1] "Factorial of 5 is 120"
```

Bar Graph:

A bar graph is suitable for showing comparisons between different categories.

Step 1: Load the necessary library (ggplot2 for this example).

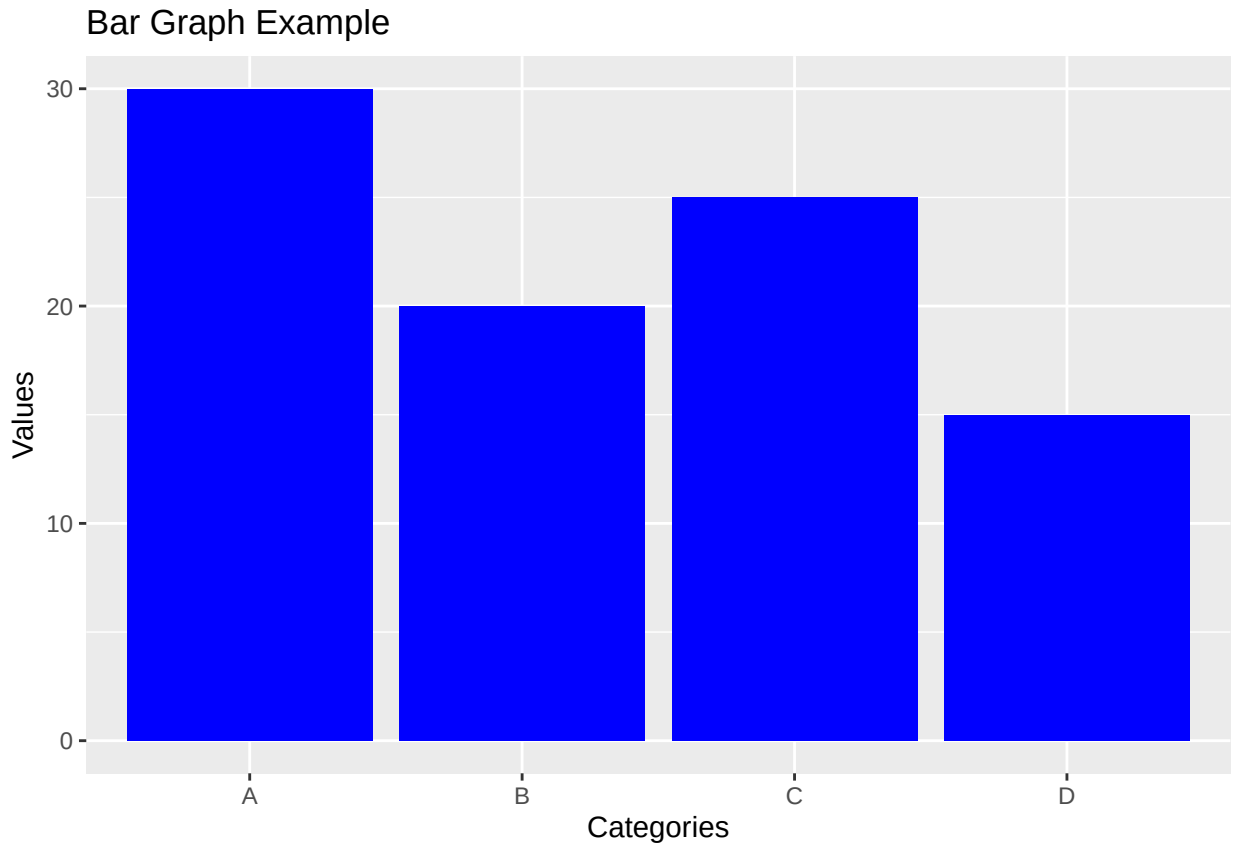
```
library(ggplot2)
```

Step 2: Create a data frame with your data.

```
data <- data.frame(
  category = c("A", "B", "C", "D"),
  value = c(30, 20, 25, 15)
)
```

Step 3: Create the bar graph.

```
ggplot(data, aes(x = category, y = value)) +
  geom_bar(stat = "identity", fill = "blue") +
  labs(title = "Bar Graph Example", x = "Categories", y = "Values")
```



Explanation:

ggplot() initializes the plot. *aes()* specifies the aesthetics. x = category determines the x-axis categories, and y = value sets the y-axis values. *geom_bar()* creates the bar chart. *labs()* sets the title and axis labels.

Pie Chart:

A pie chart is useful for displaying the distribution of categorical data as slices of a circle.

Step 1: Load the necessary library (ggplot2 for this example).

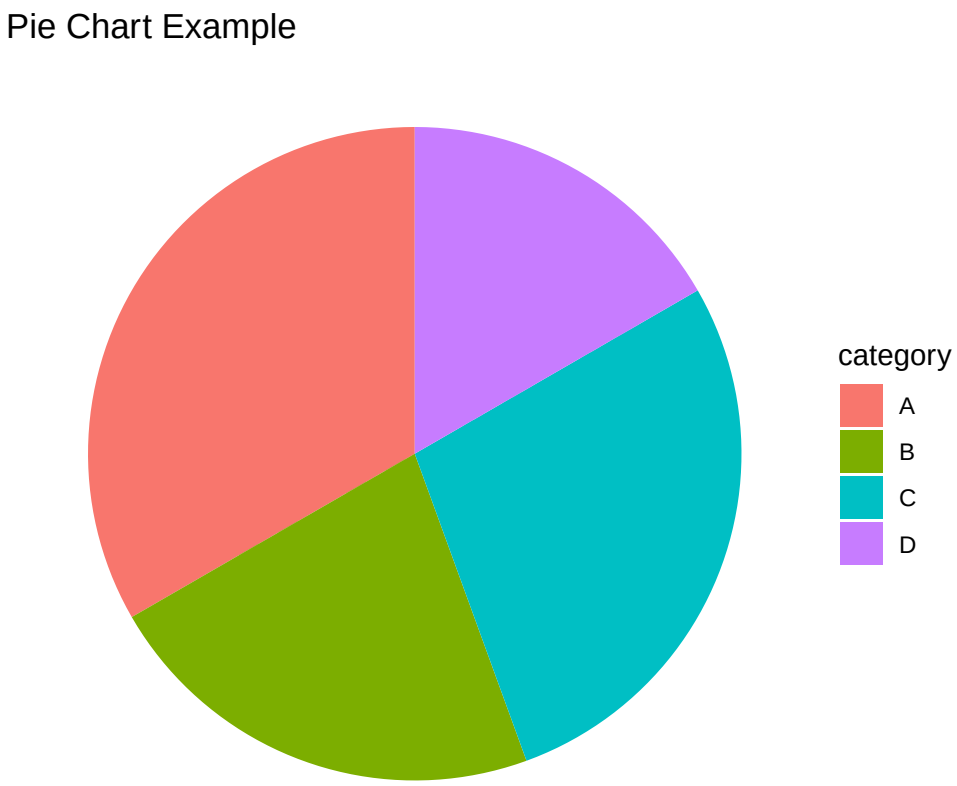
```
library(ggplot2)
```

Step 2: Create a data frame with your data.

```
data <- data.frame(
  category = c("A", "B", "C", "D"),
  value = c(30, 20, 25, 15)
)
```

Step 3: Create the pie chart.

```
ggplot(data, aes(x = "", y = value, fill = category)) +
  geom_bar(stat = "identity", width = 1) +
  coord_polar(theta = "y") +
  theme_void() +
  labs(title = "Pie Chart Example")
```



Explanation:

ggplot() initializes the plot. *aes()* specifies the aesthetics. x = " " creates a single pie chart, y = value determines the slice size, and fill = category assigns colors to categories. *geom_bar()* creates the bar chart.

coord_polar() converts the bar chart into a pie chart. *theme_void()* removes unnecessary elements. *labs()* sets the title.

Histogram:

A histogram is used to visualize the distribution of a single variable.

Step 1: Load the necessary library (ggplot2 for this example).

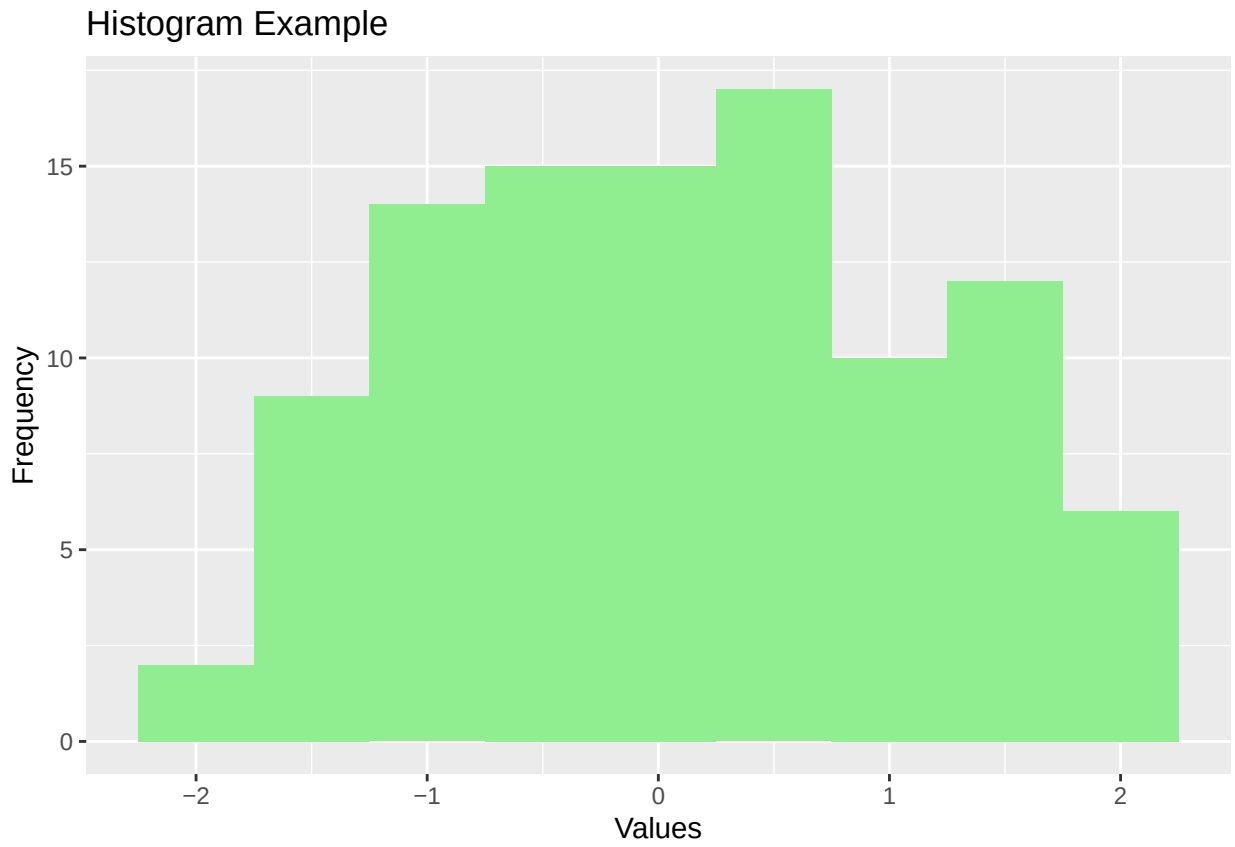
```
library(ggplot2)
```

Step 2: Create a data frame with your data. You can also load your data from a file using 'read' function.

```
data <- data.frame(
  value = rnorm(100)
)
```

Step 3: Create the histogram.

```
ggplot(data, aes(x = value)) +
  geom_histogram(binwidth = 0.5, fill = "lightgreen") +
  labs(title = "Histogram Example", x = "Values", y = "Frequency")
```



Explanation:

ggplot() initializes the plot. *aes()* specifies the aesthetics. x = value sets the variable for the x-axis. *geom_histogram()* creates the histogram. binwidth determines the width of each bin. *labs()* sets the title and axis labels.

Scatter Plot:

A scatter plot is used to show the relationship between two numerical variables.

Step 1: Load the necessary library (ggplot2 for this example).

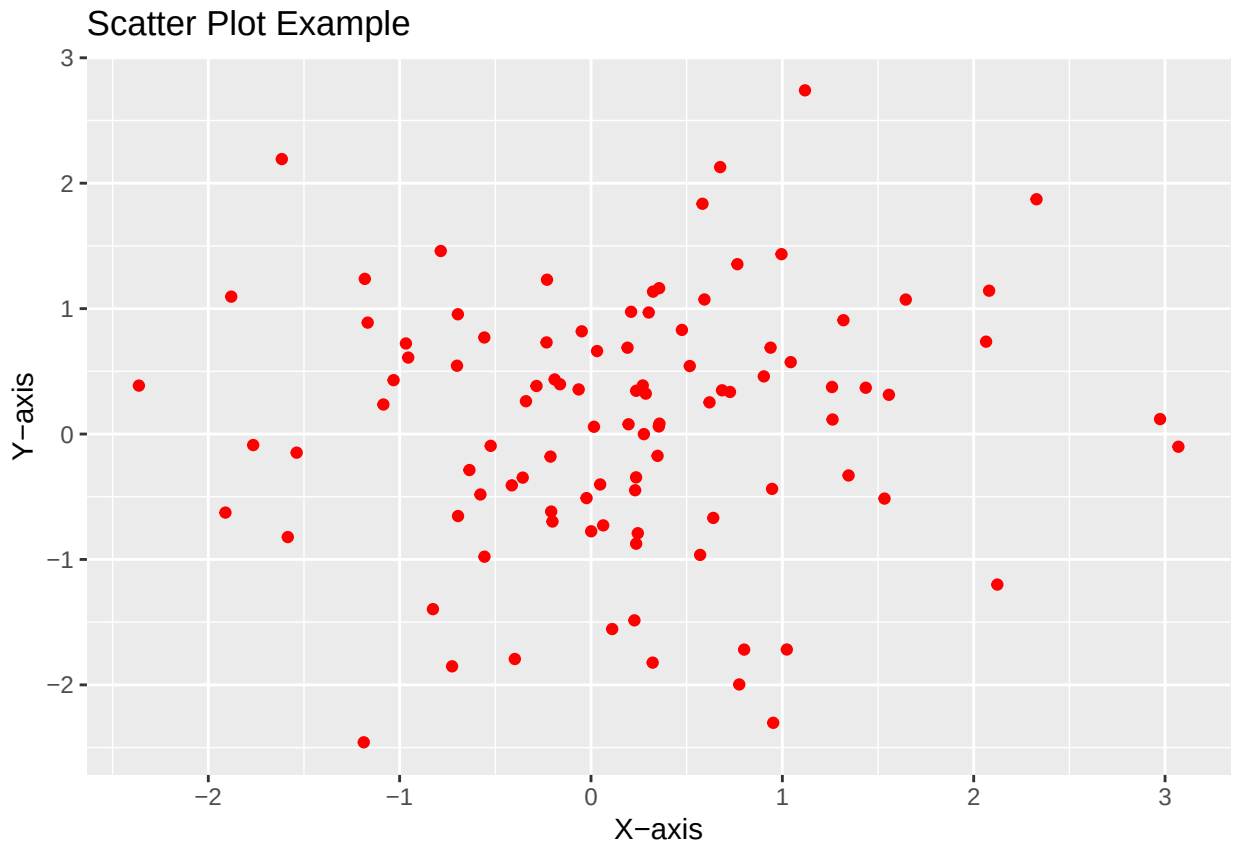
```
data <- data.frame(
  x = rnorm(100),
  y = rnorm(100)
)
```

Step 2: Create a data frame with your data, or load your data.

```
data <- data.frame(
  x = rnorm(100),
  y = rnorm(100)
)
```

Step 3: Create the scatter plot.

```
ggplot(data, aes(x = x, y = y)) +
  geom_point(color = "red") +
  labs(title = "Scatter Plot Example", x = "X-axis", y = "Y-axis")
```



Explanation:

ggplot() initializes the plot. *aes()* specifies the aesthetics. x = x and y = y set the variables for the x and y axes. *geom_point()* creates the scatter plot. *labs()* sets the title and axis labels.

Calculate the Correlation

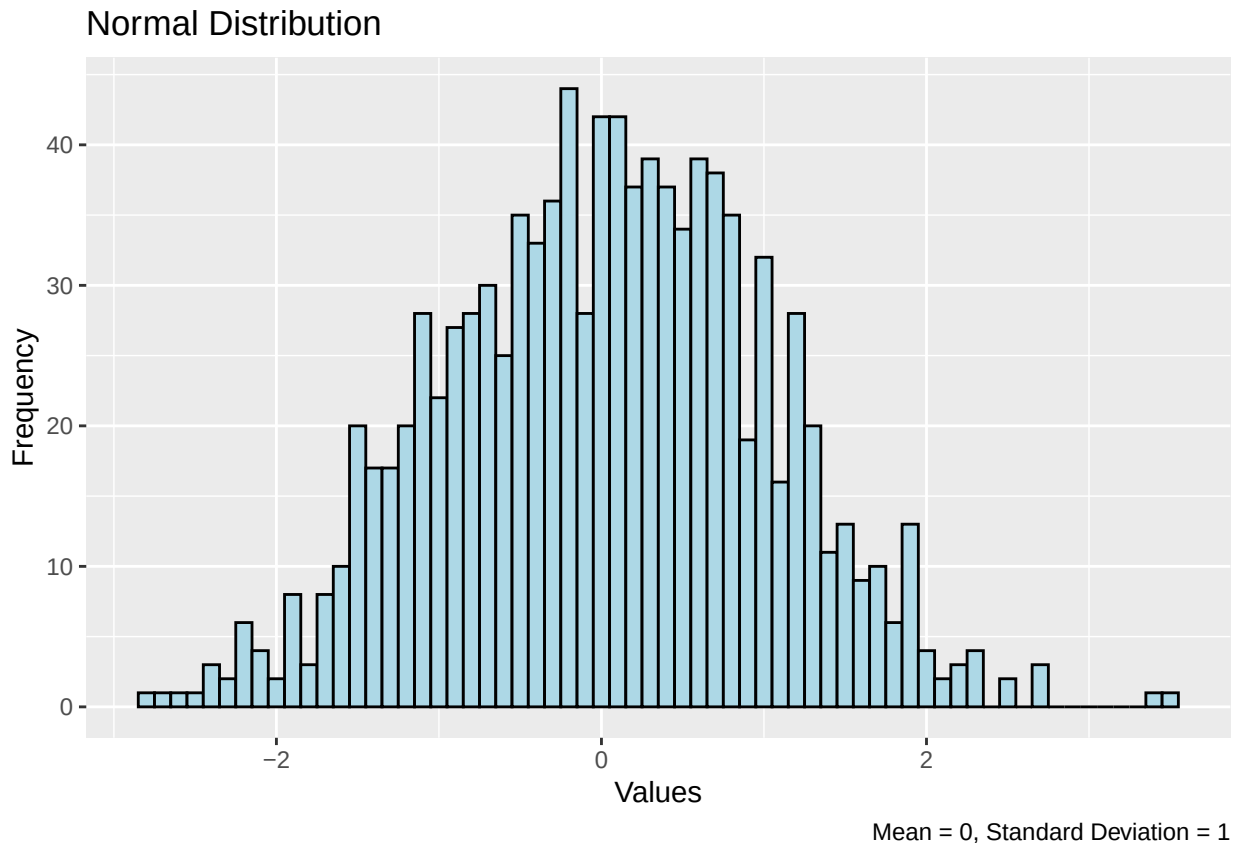
```
# Calculate the correlation coefficient
correlation_coefficient <- cor(data$x, data$y)

# Print the correlation coefficient
cat("Correlation Coefficient:", correlation_coefficient, "\n")
```

```
## Correlation Coefficient: 0.07483549
```

Create a Gaussian Distribution (Normal Distribution)

```
# Load the necessary library
library(ggplot2)

# Create a data frame with a normal distribution
data <- data.frame(
  values = rnorm(1000, mean = 0, sd = 1)
)

# Create the normal distribution plot
```

```r
ggplot(data, aes(x = values)) +
  geom_histogram(binwidth = 0.1, fill = "lightblue", color = "black") +
  labs(
    title = "Normal Distribution",
    x = "Values",
    y = "Frequency",
    caption = "Mean = 0, Standard Deviation = 1"
  )
```



Explanation:

We create a data frame data with 1000 values sampled from a standard normal distribution (mean = 0, standard deviation = 1). We create the histogram using *geom_histogram()*, specifying the bin width, fill color, and border color. We use *labs()* to set the title, x-axis label, y-axis label, and a caption with information about the mean and standard deviation. You can adjust the mean and standard deviation in the *rnorm()* function and other plot aesthetics to fit your specific needs.

One Sample Z Test example

We'll examine whether the mean blood pressure of a sample of patients matches a known population mean. We want to determine if the sample's blood pressure differs significantly from the population mean.

Sample Data

Suppose we have a sample of 20 patients, and we want to test if their average blood pressure is different from a population mean of 120 mm Hg. Here's the sample data:

```r
# Sample data for blood pressure
blood_pressure <- c(130, 125, 135, 128, 122, 132, 127, 129, 128, 131,
                    126, 134, 130, 133, 127, 128, 130, 132, 129, 125)
```

Step 1: Calculate the Sample Mean and Standard Deviation

First, calculate the sample mean and standard deviation of the blood pressure measurements.

```r
# Calculate sample mean
sample_mean <- mean(blood_pressure)

# Calculate sample standard deviation
sample_sd <- sd(blood_pressure)
```

Step 2: Define Population Parameters

Next, define the population parameters, including the population mean and the hypothesized value to test against.

```r
# Population mean (known)
population_mean <- 120

# Hypothesized value (the value we want to test against)
hypothesized_value <- 120
```

Step 3: Calculate the Z-Score

Calculate the Z-score:

```r
# Calculate standard error
standard_error <- sample_sd / sqrt(length(blood_pressure))

# Calculate Z-score
z_score <- (sample_mean - population_mean) / standard_error
```

Step 4: Perform the Z-Test

Perform the one-sample Z-test using the calculated Z-score.

```r
# Calculate the Z test statistic
z_stat <- (sample_mean - population_mean) / (sample_sd / sqrt(20))
z_stat
```

```
## [1] 12.38357
```

```r
# Calculate the corresponding p-value
p_value <- 2 * (1 - pnorm(abs(z_stat)))

# Display the p-value
p_value
```

```
## [1] 0
```

Step 5: Interpret the Results

Let's interpret the results of the one-sample Z-test

The Z-statistic measures how many standard errors the sample mean is away from the population mean.

The p-value tells us whether this difference is statistically significant. A small p-value (typically < 0.05) indicates that the sample mean is significantly different from the population mean.

Since the p-value is less than 0.05, we reject the null hypothesis. The sample's blood pressure differs significantly from the population mean of 120 mm Hg.

One Sample T-Test Example

Introduction

In this example, we will perform a two-sample t-test to analyze the effectiveness of two different pain relief medications in a pharmacy setting. We want to determine if there is a significant difference in pain relief duration between the two medications: Formula A and Formula B.

Data

We have collected data from 15 patients for each medication. Here are the pain relief durations (in hours) for each group:

```
# Data
formula_a <- c(4.2, 3.5, 2.9, 3.8, 4.5, 4.0, 3.7, 3.2, 4.1, 3.0, 3.9, 3.3, 4.4, 3.8, 3.7)
formula_b <- c(3.0, 2.7, 2.2, 2.9, 3.5, 3.1, 2.8, 2.6, 3.4, 2.5, 3.1, 2.8, 3.6, 3.0, 3.2)
```

Step 1: Descriptive Statistics

Let's begin by calculating some descriptive statistics for each group.

```
# Descriptive Statistics
mean_a <- mean(formula_a)
mean_b <- mean(formula_b)

sd_a <- sd(formula_a)
sd_b <- sd(formula_b)

# Display Descriptive Statistics
mean_a
```

```
## [1] 3.733333
```

```
mean_b
```

```
## [1] 2.96
```

```
sd_a
```

```
## [1] 0.4820591
```

```
sd_b
```

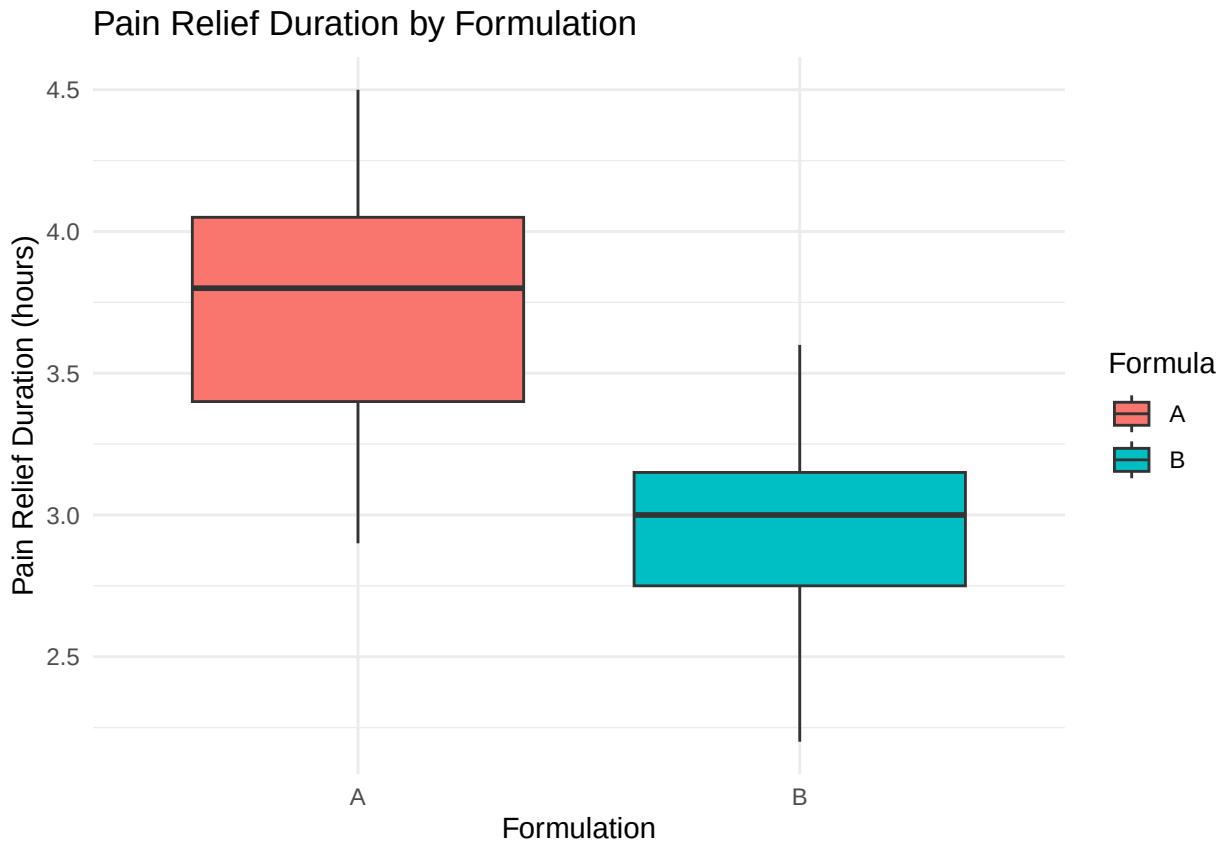
```
## [1] 0.381351
```

Interpretation: The mean pain relief duration for Formula A is approximately mean_a hours, with a standard deviation of sd_a. For Formula B, the mean duration is approximately mean_b hours, with a standard deviation of sd_b.

Step 2: Visualize the Data

We can create a boxplot to visualize the distribution of pain relief durations for each medication.

```
# Data Visualization
library(ggplot2)

data <- data.frame(
  Formula = c(rep("A", length(formula_a)), rep("B", length(formula_b))),
  PainRelief = c(formula_a, formula_b)
)

ggplot(data, aes(x = Formula, y = PainRelief, fill = Formula)) +
  geom_boxplot() +
  labs(
    title = "Pain Relief Duration by Formulation",
    x = "Formulation",
    y = "Pain Relief Duration (hours)"
  ) +
  theme_minimal()
```



Interpretation: The boxplot shows the distribution of pain relief durations for each medication. Formula A appears to have a slightly higher median duration compared to Formula B.

Step 3: Perform the T-Test

Now, let's perform a two-sample t-test to determine if there is a significant difference in pain relief duration between the two medications.

```r
# Perform the t-test
t_test_result <- t.test(formula_a, formula_b, alternative = "two.sided", var.equal = FALSE)

# Display the t-test results
t_test_result
```

```
##
##  Welch Two Sample t-test
##
## data:  formula_a and formula_b
## t = 4.8728, df = 26.591, p-value = 4.449e-05
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  0.4474632 1.0992035
## sample estimates:
## mean of x mean of y
##  3.733333  2.960000
```

Interpretation: The t-test results provide a test statistic (t-value) and a p-value. The p-value is t_test_result$p.value. If the p-value is less than our chosen significance level (e.g., 0.05), we can conclude whether there is a statistically significant difference in pain relief duration between the two medications.

Conclusion

Based on the t-test results, we can make a conclusion about the effectiveness of the two pain relief medications.

Two Sample T- test example

In this example, we'll compare the effectiveness of two different medications in reducing blood pressure. We'll have two groups: Group A, which takes Medication A, and Group B, which takes Medication B. We want to determine if there is a significant difference in blood pressure reduction between the two groups.

```r
# Sample data for Group A (Medication A)
group_a <- c(130, 135, 138, 142, 128, 133, 137, 131, 136, 139)

# Sample data for Group B (Medication B)
group_b <- c(128, 129, 134, 125, 133, 131, 127, 132, 136, 130)
```

Step 1: Calculate Descriptive Statistics

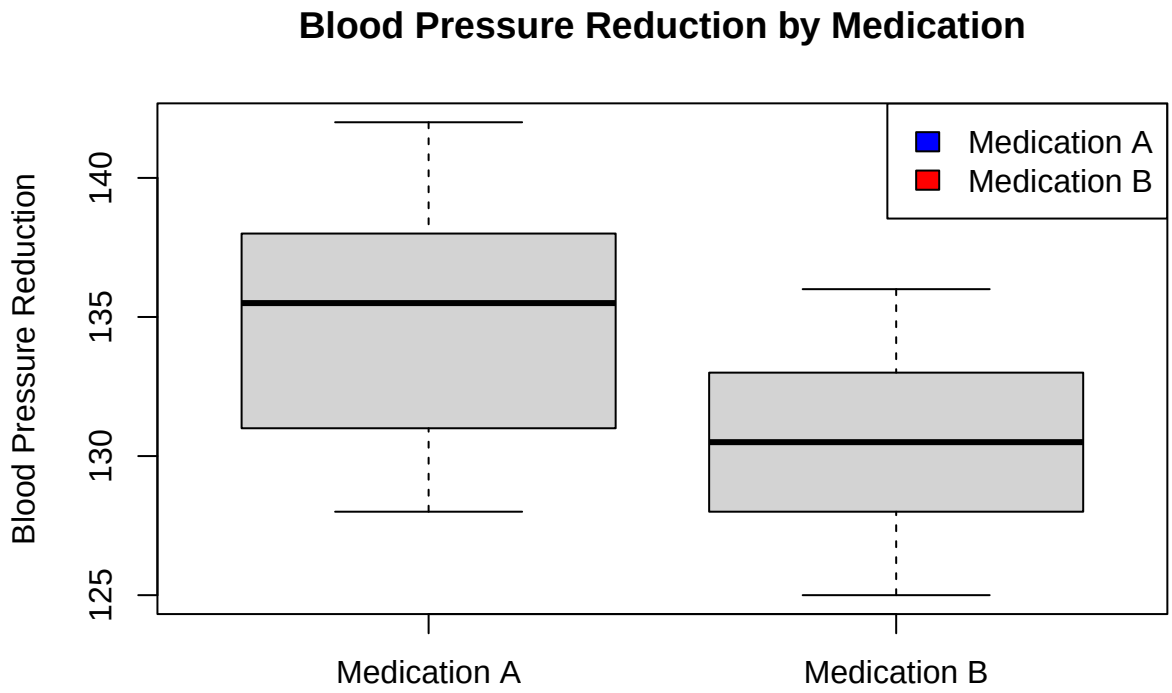
First, we need to calculate the means and standard deviations for both groups.

```r
# Calculate mean and standard deviation for Group A
mean_a <- mean(group_a)
sd_a <- sd(group_a)

# Calculate mean and standard deviation for Group B
mean_b <- mean(group_b)
sd_b <- sd(group_b)
```

Step 2: Create a Box Plot

```r
# Create a side-by-side box plot
boxplot(group_a, group_b,
```

```
        names = c("Medication A", "Medication B"),
        main = "Blood Pressure Reduction by Medication",
        ylab = "Blood Pressure Reduction")

# Add a legend
legend("topright", legend = c("Medication A", "Medication B"), fill = c("blue", "red"))
```



Step 3: Perform the Two-Sample T-Test

Next, we'll perform the two-sample t-test using the t.test function in R.

```
# Perform the two-sample t-test
t_result <- t.test(group_a, group_b)
```

Step 4: Interpret the Results

Now, let's interpret the results.

```
# Print the t-test results
t_result
```

```
##
##  Welch Two Sample t-test
##
## data:  group_a and group_b
## t = 2.5153, df = 16.896, p-value = 0.02231
## alternative hypothesis: true difference in means is not equal to 0
```

```
## 95 percent confidence interval:
##  0.7075975 8.0924025
## sample estimates:
## mean of x mean of y
##     134.9     130.5
```

The output will include several values, but the key ones for our interpretation are:

t statistic: The t-statistic represents the difference in means between the two groups.

p-value: The p-value tells us whether the difference in means is statistically significant. A small p-value (typically < 0.05) indicates statistical significance.

Step 5: Interpretation

In our example:

The t-statistic measures the difference in blood pressure reduction between Group A (Medication A) and Group B (Medication B).

Since the p-value is less than 0.05, we reject the null hypothesis we can conclude that there is a significant difference between the two medications.

Paired T- test example

In this example, we'll investigate whether a new medication has a significant effect in reducing cholesterol levels in a group of patients. We will measure cholesterol levels in the same patients before and after they took the medication. We want to determine if there is a statistically significant difference in cholesterol levels before and after taking the medication.

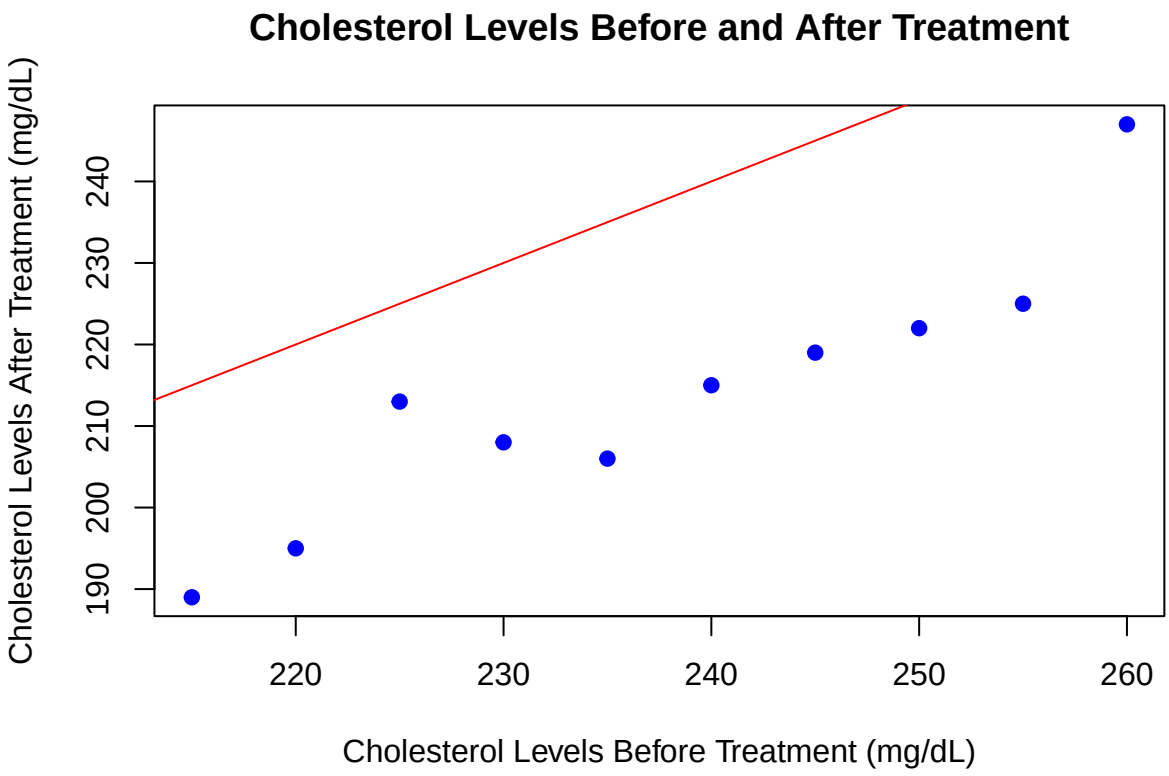
```r
# Sample data for cholesterol levels before treatment
before_treatment <- c(220, 240, 250, 230, 215, 245, 260, 235, 225, 255)

# Sample data for cholesterol levels after treatment
after_treatment <- c(195, 215, 222, 208, 189, 219, 247, 206, 213, 225)
```

Step 1: Create a Plot

Now, let's create a paired data plot (a before-and-after plot) to visualize the change in cholesterol levels.

This code creates a scatter plot of cholesterol levels before and after treatment, with a reference line (red line) showing the line of no change. Points above the line indicate an increase in cholesterol levels after treatment, while points below the line indicate a decrease.

```r
# Create a paired data plot
plot(before_treatment, after_treatment,
     xlab = "Cholesterol Levels Before Treatment (mg/dL)",
     ylab = "Cholesterol Levels After Treatment (mg/dL)",
     main = "Cholesterol Levels Before and After Treatment",
     pch = 19, col = "blue")

# Add a diagonal reference line
abline(a = 0, b = 1, col = "red")
```



Step 2: Calculate the Differences

First, calculate the differences in cholesterol levels for each patient (before - after).

```
# Calculate the differences
differences <- after_treatment - before_treatment
```

Step 3: Perform the Paired T-Test

Next, perform the paired t-test using the t.test function. This test will compare the mean of the differences to zero.

```
# Perform the paired t-test
t_result <- t.test(differences)
```

Step 4: Interpret the Results

Now, let's interpret the results of the paired t-test.

The t-statistic measures the mean difference in cholesterol levels before and after treatment with the new drug.

Since the p-value is less than 0.05 (a common significance level), we can conclude that there is a significant difference in cholesterol levels after treatment.

One way ANOVA example

In this example, we'll analyze the effectiveness of three different drug treatments in reducing pain for patients with a specific condition. We want to determine if there is a significant difference in pain reduction among the three drug treatments.

Sample Data

Here's some sample data for pain reduction scores for patients in three different drug treatment groups (Group A, Group B, and Group C):

```
# Sample data for pain reduction scores in Group A
group_a <- c(5, 6, 7, 4, 6, 5)

# Sample data for pain reduction scores in Group B
group_b <- c(4, 3, 5, 4, 3, 4)

# Sample data for pain reduction scores in Group C
group_c <- c(7, 8, 9, 6, 8, 7)
```

Step 1: Combine Data and Create a Data Frame

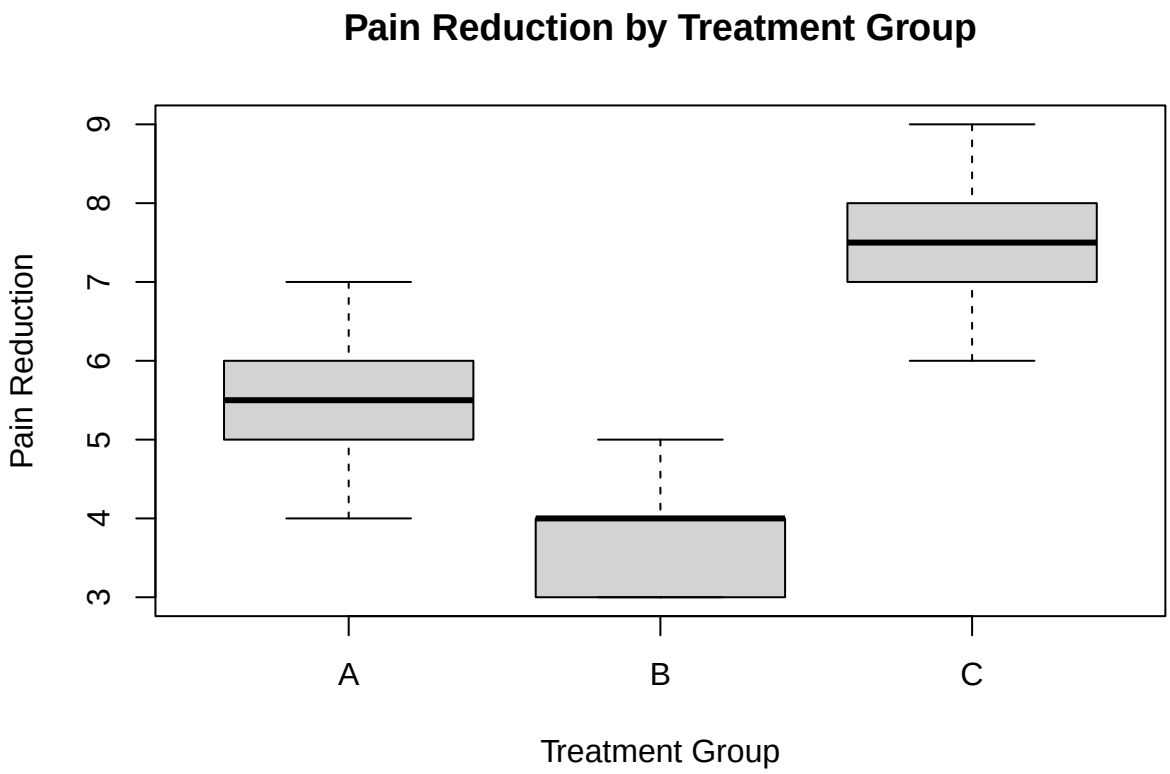
First, combine the data from all three groups into a single data frame.

```
# Combine data into a data frame
data <- data.frame(
  Treatment = rep(c("A", "B", "C"), each = 6),
  PainReduction = c(group_a, group_b, group_c)
)
```

Step 2: Visualize the Data

Before performing the ANOVA, it's a good practice to visualize the data. We can create a box plot to compare the distributions of pain reduction scores for each treatment group.

```
# Create a box plot
boxplot(PainReduction ~ Treatment, data = data,
        xlab = "Treatment Group", ylab = "Pain Reduction",
        main = "Pain Reduction by Treatment Group")
```



This code creates a box plot that visualizes the distribution of pain reduction scores for each treatment group.

Step 3: Perform One-Way ANOVA

Now, let's perform a one-way ANOVA to test whether there are significant differences in pain reduction among the treatment groups.

```
# Perform one-way ANOVA
anova_result <- aov(PainReduction ~ Treatment, data = data)
```

Step 4: Interpret the Results

Let's interpret the results of the one-way ANOVA.

```
# Print ANOVA results
summary(anova_result)
```

```
##             Df Sum Sq Mean Sq F value   Pr(>F)
## Treatment    2  40.44  20.222   21.93 3.53e-05 ***
## Residuals   15  13.83   0.922
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The output will include several values, but the key ones for our interpretation are:

Pr(>F): The p-value associated with the F-statistic. It tells us whether there is a statistically significant difference in pain reduction among the treatment groups.

Step 5: Interpretation

In our example:

The F-statistic measures the variation in pain reduction scores between the treatment groups relative to the variation within the groups.

The p-value associated with the F-statistic tells us whether this variation is statistically significant. If the p-value is less than 0.05 (a common significance level), we can conclude that there is a significant difference in pain reduction among the treatment groups.

Let's interpret the results based on the output: The t-statistic measures the mean difference in cholesterol levels before and after treatment with the new drug.

The p-value indicates whether this mean difference is statistically significant. Since the p-value is less than 0.05 , we can conclude that there is a significant difference in cholesterol levels after treatment.

One Proportion Test example

In this example, we'll analyze whether a new drug is effective in treating a specific condition by comparing the proportion of patients who experienced improvement. We want to determine if the proportion of patients who improved with the new drug is significantly different from a predefined target proportion.

Sample Data

Here's some sample data for a clinical trial:

Total number of patients: 100 Number of patients who improved with the new drug: 63 We will compare the proportion of patients who improved (63 out of 100) to a target proportion (e.g., 0.50) to test if the new drug is effective.

Step 1: Set Up the Hypotheses

Before performing the one-proportion test, let's set up the null and alternative hypotheses:

Null Hypothesis (H0): The proportion of patients who improved with the new drug is equal to the target proportion (e.g., 0.50). Alternative Hypothesis (Ha): The proportion of patients who improved with the new drug is not equal to the target proportion (e.g., not equal to 0.50).

Step 2: Perform the One-Proportion Test

Now, let's perform the one-proportion test using the prop.test function in R.

```
# Define the sample size
n <- 100

# Define the number of patients who improved
x <- 63

# Define the target proportion
p_target <- 0.50

# Perform the one-proportion test
prop_test_result <- prop.test(x, n, p_target)
```

Step 3: Interpret the Results

Let's interpret the results of the one-proportion test.

```r
# Print the one-proportion test results
prop_test_result
```

```
##
##  1-sample proportions test with continuity correction
##
## data:  x out of n, null probability p_target
## X-squared = 6.25, df = 1, p-value = 0.01242
## alternative hypothesis: true p is not equal to 0.5
## 95 percent confidence interval:
##  0.5271463 0.7227373
## sample estimates:
##    p
## 0.63
```

Step 4: Interpretation

The null hypothesis (H0) assumes that the proportion of patients who improved with the new drug is equal to the target proportion (0.50).

The alternative hypothesis (Ha) suggests that the proportion of patients who improved with the new drug is not equal to the target proportion (not equal to 0.50).

Since the p-value is less than 0.05, we reject the null hypothesis. There is a statistically significant difference in the proportion of patients who improved with the new drug compared to the target proportion.

Two Proportion Test example

In this example, we'll compare the proportions of patients who experienced side effects after taking two different medications. We want to determine if there is a significant difference in the proportion of patients experiencing side effects between the two medications.

Sample Data

Here's some sample data for the two medications:

Medication A: 80 out of 200 patients experienced side effects. Medication B: 60 out of 200 patients experienced side effects.

Step 1: Define the Data

We start by defining the data:

```r
# Define the data
success_a <- 80   # Number of patients with side effects for Medication A
n_a <- 200         # Total number of patients for Medication A

success_b <- 60   # Number of patients with side effects for Medication B
n_b <- 200         # Total number of patients for Medication B
```

Step 2: Perform the Two-Proportion Test

Next, we'll perform a two-proportion test using the prop.test function:

```r
# Perform the two-proportion test
prop_test_result <- prop.test(c(success_a, success_b), c(n_a, n_b))
```

Step 3: Interpret the Results

Now, let's interpret the results of the two-proportion test.

```
# Print the two-proportion test results
prop_test_result
```

```
##
##  2-sample test for equality of proportions with continuity correction
##
## data:  c(success_a, success_b) out of c(n_a, n_b)
## X-squared = 3.967, df = 1, p-value = 0.0464
## alternative hypothesis: two.sided
## 95 percent confidence interval:
##  0.002030745 0.197969255
## sample estimates:
## prop 1 prop 2
##    0.4    0.3
```

Step 4: Interpretation

The prop.test function tests whether the proportions of patients experiencing side effects for Medication A and Medication B are significantly different.

Since the p-value is less than 0.05, we reject the null hypothesis. There is a statistically significant difference in the proportions of patients with side effects between Medication A and Medication B.

Chi Square Test example

We'll examine whether there's an association between two categorical variables: the type of pain reliever taken (e.g., Drug A, Drug B, or Drug C) and the occurrence of side effects (Yes or No). We want to determine if there's a significant relationship between the choice of pain reliever and the occurrence of side effects.

Sample Data

Here's some sample data representing the number of patients who experienced side effects (Yes or No) for each type of pain reliever:

```
# Sample data
data <- data.frame(
  PainReliever = c("A", "B", "C", "A", "B", "C"),
  SideEffects = c("Yes", "Yes", "No", "No", "Yes", "No")
)
```

Step 1: Create a Contingency Table

First, we need to create a contingency table to summarize the data. This table will show the frequency of patients in each combination of pain reliever and side effects.

```
# Create a contingency table
contingency_table <- table(data$PainReliever, data$SideEffects)
```

Step 2: Perform Chi-Square Test

Now, let's perform the Chi-Square test of independence to determine if there's a significant association between the choice of pain reliever and the occurrence of side effects.

```r
# Perform Chi-Square test
chi_square_result <- chisq.test(contingency_table)
```

```
## Warning in chisq.test(contingency_table): Chi-squared approximation may be
## incorrect
```

Step 3: Interpret the Results

Let's interpret the results of the Chi-Square test.

```r
# Print Chi-Square test results
chi_square_result
```

```
##
##  Pearson's Chi-squared test
##
## data:  contingency_table
## X-squared = 4, df = 2, p-value = 0.1353
```

The output will include several values, but the key ones for our interpretation are:

X-squared: The Chi-Square statistic measures the association between the two categorical variables.

p-value: The p-value tells us whether this association is statistically significant. A small p-value (typically < 0.05) indicates statistical significance.

Step 4: Interpretation

In our example:

The Chi-Square statistic measures the strength and direction of the association between the choice of pain reliever and the occurrence of side effects.

The p-value associated with the Chi-Square statistic tells us whether this association is statistically significant. Since the p-value is greater than or equal to 0.05, we fail to reject the null hypothesis. There is no statistically significant association between the choice of pain reliever and the occurrence of side effects.

Simple Linear Regression example

We'll investigate the relationship between the dosage of a medication and its effect on patients' cholesterol levels. We'll perform a linear regression analysis to understand how changes in dosage impact cholesterol levels.

Sample Data

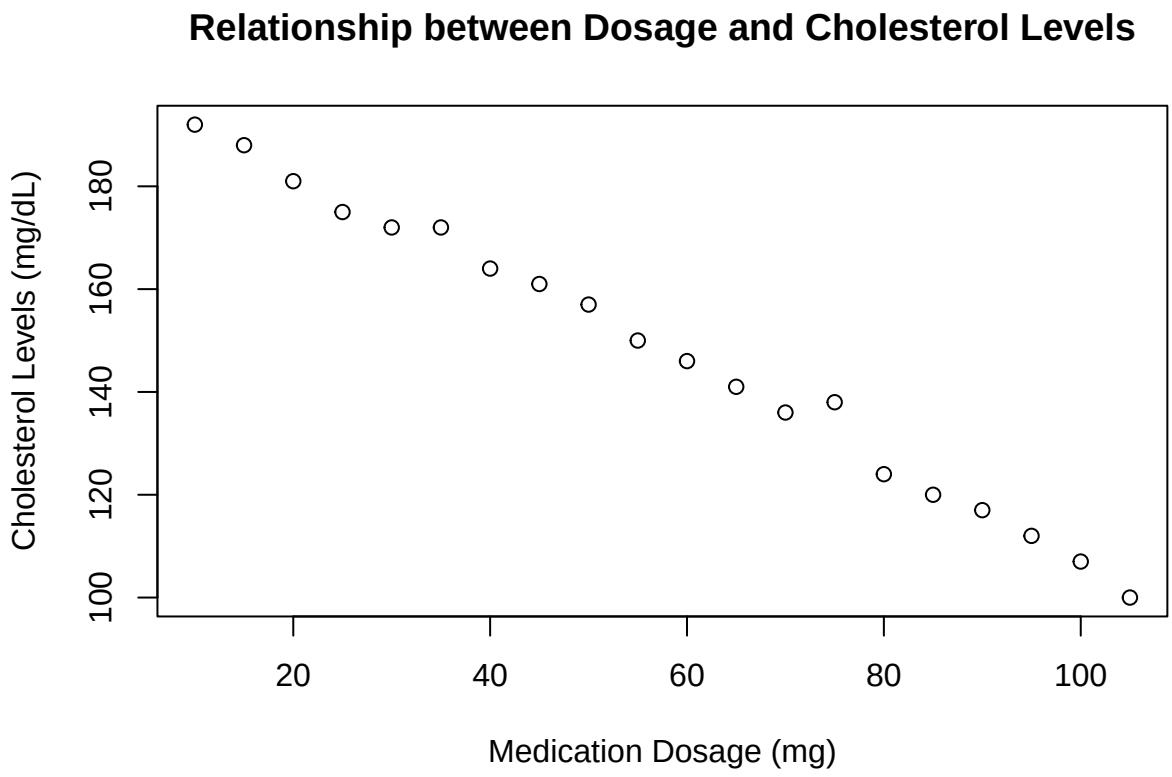
Suppose we have a sample of 20 patients, and we want to test if their average blood pressure is different from a population mean of 120 mm Hg. Here's the sample data:

```r
# Sample data for medication dosage
dosage <- c(10, 15, 20, 25, 30, 35, 40, 45, 50, 55,
            60, 65, 70, 75, 80, 85, 90, 95, 100, 105)

# Sample data for cholesterol levels
cholesterol <- c(192, 188, 181, 175, 172, 172, 164, 161, 157, 150,
                 146, 141, 136, 138, 124, 120, 117, 112, 107, 100)
```

Step 1: Visualize the Data

Before performing linear regression, it's a good practice to visualize the data. Let's create a scatter plot to visualize the relationship between medication dosage and cholesterol levels.

```
# Create a scatter plot
plot(dosage, cholesterol,
     xlab = "Medication Dosage (mg)",
     ylab = "Cholesterol Levels (mg/dL)",
     main = "Relationship between Dosage and Cholesterol Levels")
```



Step 2: Perform Simple Linear Regression

Now, let's perform linear regression to model the relationship between dosage and cholesterol levels.

```
# Perform linear regression
regression_model <- lm(cholesterol ~ dosage)
```

Step 3: Interpret the Results

Let's interpret the results of the linear regression analysis

```
# Print summary of regression model
summary(regression_model)
```

```
##
## Call:
## lm(formula = cholesterol ~ dosage)
```

```
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.3271 -1.6288 -0.0891  0.4994  6.8684
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 201.9248     1.1773  171.52   <2e-16 ***
## dosage       -0.9439     0.0183  -51.57   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.36 on 18 degrees of freedom
## Multiple R-squared:  0.9933, Adjusted R-squared:  0.9929
## F-statistic:  2660 on 1 and 18 DF,  p-value: < 2.2e-16
```

The output will include several values, but the key ones for our interpretation are:

R-squared (R-sq): The coefficient of determination, which tells us the proportion of variance in cholesterol levels that is explained by the dosage.

t-test (t value): The t-value associated with the dosage coefficient. It tells us whether the dosage has a statistically significant impact on cholesterol levels.

F-test (F value): The F-statistic, which tests the overall significance of the regression model.

Step 4: Interpretation

The R-squared value measures how well the dosage explains the variance in cholesterol levels. A higher R-squared indicates a better fit of the model.

The t-test associated with the dosage coefficient tests whether the dosage has a statistically significant impact on cholesterol levels. A small p-value (typically < 0.05) indicates significance.

The F-test tests the overall significance of the regression model. A small p-value suggests that the model is significant.

```
# Interpretation
cat("R-squared (Coefficient of Determination):", round(summary(regression_model)$r.squared, 4), "\n")
```

```
## R-squared (Coefficient of Determination): 0.9933
```

```
cat("t-test (Dosage Coefficient):\n")
```

```
## t-test (Dosage Coefficient):
```

```
print(summary(regression_model)$coefficients["dosage", ])
```

```
##      Estimate    Std. Error       t value      Pr(>|t|)
## -9.439098e-01  1.830274e-02 -5.157204e+01  5.214634e-21
```

```
cat("F-test (Overall Model Significance):\n")
```

```
## F-test (Overall Model Significance):
```

```
print(anova(regression_model))
```

```
## Analysis of Variance Table
##
## Response: cholesterol
##           Df  Sum Sq Mean Sq F value    Pr(>F)
```

```
## dosage     1 14812.3 14812.3  2659.7 < 2.2e-16 ***
## Residuals 18   100.2     5.6
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Step 5: Residual Analysis using 4 in 1 Plot

Let's perform residual analysis and create the following plots:

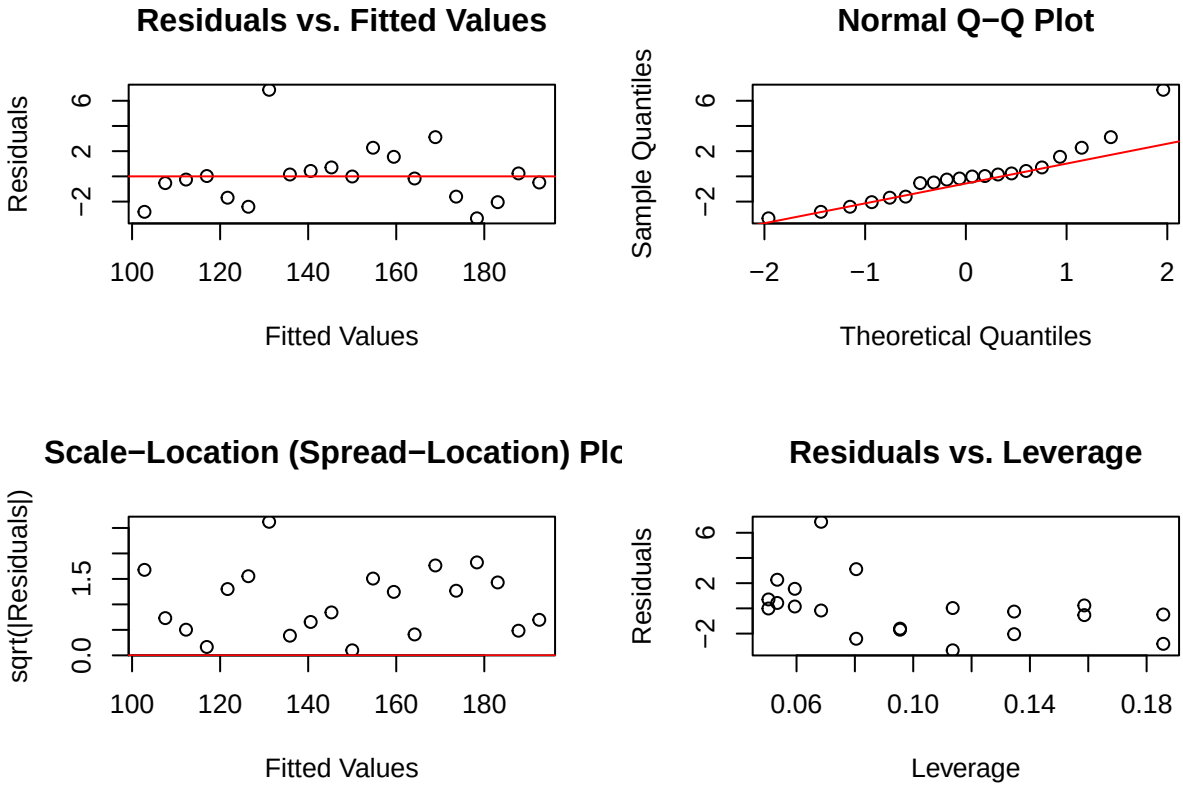
Residual vs. Fitted Values Plot: This plot helps us check for the linearity assumption and homoscedasticity. We expect the residuals to be randomly scattered around zero with no clear pattern.

Normal Q-Q Plot: This plot helps us assess the normality assumption of the residuals. We expect the points to roughly follow a straight line.

Scale-Location (Spread-Location) Plot: This plot helps us check for homoscedasticity. We expect the points to be randomly scattered and have a constant spread.

Residuals vs. Leverage Plot: This plot helps us identify influential data points. Outliers or high leverage points may disproportionately affect the regression model.

```r
# Perform residual analysis
residuals <- residuals(regression_model)

# Create and arrange a 2x2 grid for the plots
par(mfrow = c(2, 2))

# Residual vs. Fitted Values Plot
plot(fitted(regression_model), residuals,
     xlab = "Fitted Values",
     ylab = "Residuals",
     main = "Residuals vs. Fitted Values")
abline(h = 0, col = "red")

# Normal Q-Q Plot
qqnorm(residuals, main = "Normal Q-Q Plot")
qqline(residuals, col = "red")

# Scale-Location Plot
sqrt_abs_residuals <- sqrt(abs(residuals))
plot(fitted(regression_model), sqrt_abs_residuals,
     xlab = "Fitted Values",
     ylab = "sqrt(|Residuals|)",
     main = "Scale-Location (Spread-Location) Plot")
abline(h = 0, col = "red")

# Residuals vs. Leverage Plot
plot(hatvalues(regression_model), residuals,
     xlab = "Leverage",
     ylab = "Residuals",
     main = "Residuals vs. Leverage")
```



Step 6: Interpretation of Residual Analysis

Residual vs. Fitted Values Plot: In this plot, we are looking for a random scatter of residuals around zero. It appears that the residuals are relatively evenly distributed, which is a good sign. There is no clear pattern, indicating that the linearity assumption is reasonable.

Normal Q-Q Plot: The points in the Q-Q plot roughly follow a straight line, suggesting that the residuals are approximately normally distributed. This supports the normality assumption.

Scale-Location (Spread-Location) Plot: The spread of residuals appears to be relatively constant across the range of fitted values, indicating that the homoscedasticity assumption is met.

Residuals vs. Leverage Plot: We use this plot to identify influential data points. If any points have high leverage (i.e., are far from the center) and high residuals, they may be influential outliers. In our plot, no points appear to be highly influential.

Overall, based on the residual analysis and plots, our linear regression model appears to meet the assumptions of linearity, normality, and homoscedasticity. There are no obvious influential outliers.

Bringing it all together

Generate the regression equation:

```
# Get the coefficients from the regression model
coefficients <- coef(regression_model)

# Extract the intercept (b0) and coefficient for dosage (b1)
intercept <- coefficients[1]
```

```r
coef_dosage <- coefficients[2]

# Write the regression equation
regression_equation <- paste("Cholesterol Levels =", round(intercept, 2), "+", round(coef_dosage, 2), "
regression_equation
```

```
## [1] "Cholesterol Levels = 201.92 + -0.94  * Dosage (mg)"
```

Multiple Linear Regression example

We'll investigate the relationship between the effectiveness of a medication and two predictor variables: dosage and patient's age. We want to determine how these two variables together impact the medication's effectiveness.

Sample Data

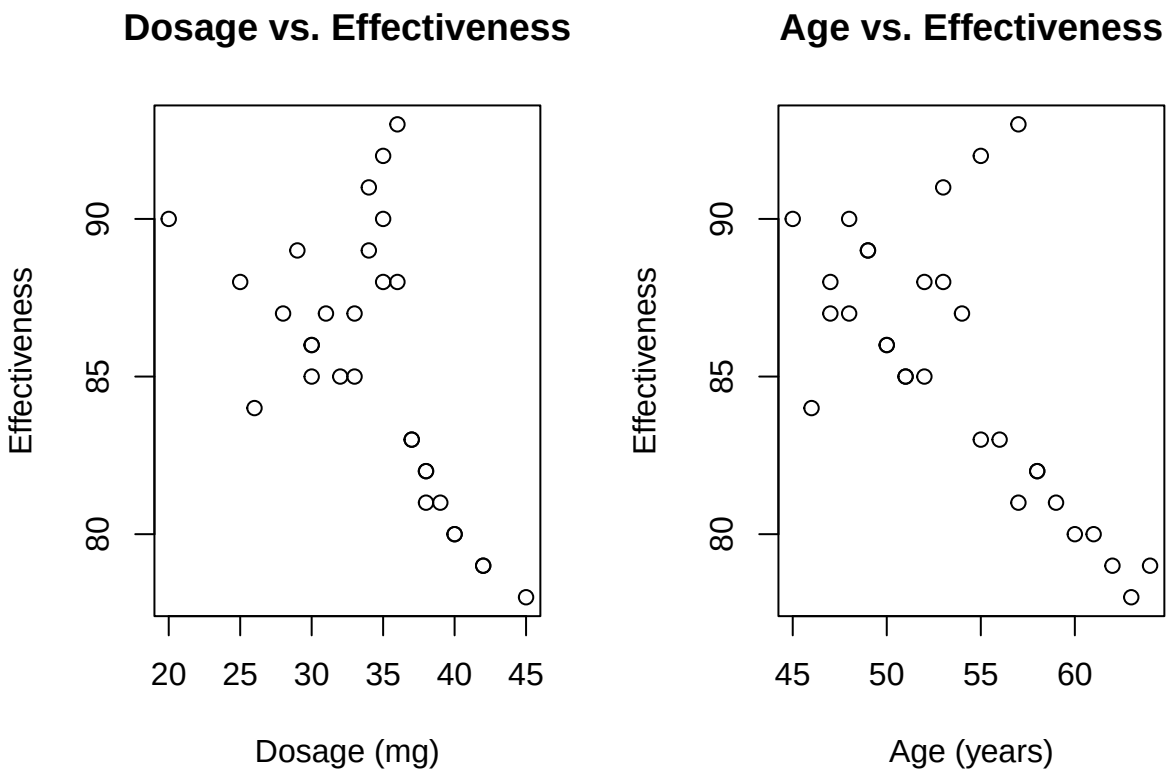
we have collected data from 30 patients, where we measured the medication's effectiveness (outcome variable), medication dosage (in mg), and patient age (in years). Here's the sample data:

```r
# Sample data for medication effectiveness
effectiveness <- c(90, 85, 88, 92, 86, 87, 93, 91, 89, 84,
                   80, 82, 88, 85, 87, 81, 79, 83, 86, 90,
                   78, 80, 79, 82, 81, 88, 85, 83, 87, 89)

# Sample data for medication dosage
dosage <- c(20, 30, 25, 35, 30, 28, 36, 34, 29, 26,
            40, 38, 35, 32, 33, 38, 42, 37, 30, 35,
            45, 40, 42, 38, 39, 36, 33, 37, 31, 34)

# Sample data for patient age
age <- c(45, 52, 47, 55, 50, 48, 57, 53, 49, 46,
         60, 58, 52, 51, 54, 59, 62, 56, 50, 48,
         63, 61, 64, 58, 57, 53, 51, 55, 47, 49)
```

Step 1: Visualize the Data

Before performing multiple linear regression, let's visualize the relationship between medication effectiveness and the two predictor variables: dosage and age. We'll create scatter plots for each predictor.

```r
# Create scatter plots for dosage and age
par(mfrow = c(1, 2))

# Scatter plot for dosage vs. effectiveness
plot(dosage, effectiveness,
     xlab = "Dosage (mg)",
     ylab = "Effectiveness",
     main = "Dosage vs. Effectiveness")

# Scatter plot for age vs. effectiveness
plot(age, effectiveness,
     xlab = "Age (years)",
     ylab = "Effectiveness",
     main = "Age vs. Effectiveness")
```




```
# Reset plot layout
par(mfrow = c(1, 1))
```

These scatter plots help us visualize the relationships between medication effectiveness and the two predictor variables.

Step 2: Perform Multiple Linear Regression

Now, let's perform multiple linear regression to model the relationship between medication effectiveness, dosage, and patient age.

```
# Perform multiple linear regression
multiple_regression_model <- lm(effectiveness ~ dosage + age)
```

Step 3: Interpret the Results

Let's interpret the results of the multiple linear regression analysis, including R-squared, t-tests, and F-test.

```
# Print summary of multiple regression model
summary(multiple_regression_model)
```

```
##
## Call:
## lm(formula = effectiveness ~ dosage + age)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5.284 -1.674 -1.044  1.244  9.560
```

```
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 114.40333    7.12897  16.048 2.48e-15 ***
## dosage        0.04312    0.25281   0.171   0.8658
## age          -0.57045    0.26076  -2.188   0.0375 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.137 on 27 degrees of freedom
## Multiple R-squared:  0.4685, Adjusted R-squared:  0.4292
## F-statistic:  11.9 on 2 and 27 DF,  p-value: 0.0001968
```

The output will include several values, but the key ones for our interpretation are:

R-squared (R-sq): The coefficient of determination, which tells us the proportion of variance in medication effectiveness explained by dosage and age.

t-tests (t value): The t-values associated with the coefficients for dosage and age. They tell us whether these variables have statistically significant impacts on medication effectiveness.

F-test (F value): The F-statistic, which tests the overall significance of the regression model.

##Step 4: Interpretation of Multiple Linear Regression

```
 # Interpretation of Multiple Linear Regression
cat("R-squared (Coefficient of Determination):", round(summary(multiple_regression_model)$r.squared, 4)
```

```
## R-squared (Coefficient of Determination): 0.4685
```

```
# Coefficients
coefficients <- summary(multiple_regression_model)$coefficients
coef_dosage <- coefficients["dosage", "Estimate"]
coef_age <- coefficients["age", "Estimate"]

cat("Coefficients:\n")
```

```
## Coefficients:
```

```
cat("Intercept:", round(coefficients["(Intercept)", "Estimate"], 2), "\n")
```

```
## Intercept: 114.4
```

```
cat("Dosage Coefficient:", round(coef_dosage, 2), "\n")
```

```
## Dosage Coefficient: 0.04
```

```
cat("Age Coefficient:", round(coef_age, 2), "\n")
```

```
## Age Coefficient: -0.57
```

```
cat("F-test (Overall Model Significance):\n")
```

```
## F-test (Overall Model Significance):
```

```
print(anova(multiple_regression_model))
```

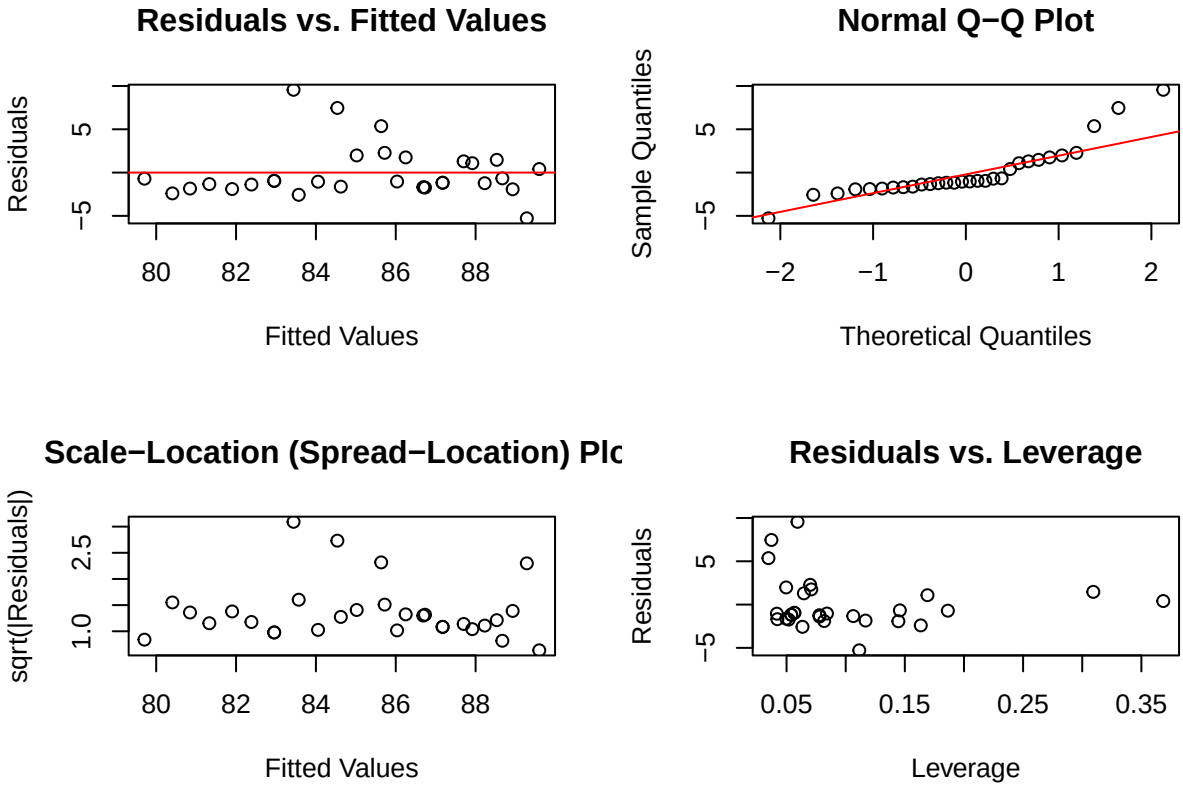
```
## Analysis of Variance Table
##
## Response: effectiveness
##           Df Sum Sq Mean Sq F value    Pr(>F)
## dosage     1 187.11 187.113 19.0168 0.0001694 ***
```

```
## age         1  47.09  47.090  4.7859 0.0375279 *
## Residuals 27 265.66   9.839
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Step 5: Residual Analysis

Now, let's perform residual analysis as we did in the previous example. We'll create residual plots to assess the model's assumptions:

```r
# Perform residual analysis
residuals <- residuals(multiple_regression_model)

# Create and arrange a 2x2 grid for the plots
par(mfrow = c(2, 2))

# Residual vs. Fitted Values Plot
plot(fitted(multiple_regression_model), residuals,
     xlab = "Fitted Values",
     ylab = "Residuals",
     main = "Residuals vs. Fitted Values")
abline(h = 0, col = "red")

# Normal Q-Q Plot
qqnorm(residuals, main = "Normal Q-Q Plot")
qqline(residuals, col = "red")

# Scale-Location Plot
sqrt_abs_residuals <- sqrt(abs(residuals))
plot(fitted(multiple_regression_model), sqrt_abs_residuals,
     xlab = "Fitted Values",
     ylab = "sqrt(|Residuals|)",
     main = "Scale-Location (Spread-Location) Plot")
abline(h = 0, col = "red")

# Residuals vs. Leverage Plot
plot(hatvalues(multiple_regression_model), residuals,
     xlab = "Leverage",
     ylab = "Residuals",
     main = "Residuals vs. Leverage")
```



Step 6: Interpretation of Residual Analysis

Residual vs. Fitted Values Plot: We are looking for a random scatter of residuals around zero. It appears that the residuals are relatively evenly distributed, which is a good sign. There is no clear pattern, indicating that the linearity assumption is reasonable.

Normal Q-Q Plot: The points in the Q-Q plot roughly follow a straight line, suggesting that the residuals are approximately normally distributed. This supports the normality assumption.

Scale-Location (Spread-Location) Plot: The spread of residuals appears to be relatively constant across the range of fitted values, indicating that the homoscedasticity assumption is met.

Residuals vs. Leverage Plot: We use this plot to identify influential data points. If any points have high leverage (i.e., are far from the center) and high residuals, they may be influential outliers. In our plot, no points appear to be highly influential.

Overall, based on the residual analysis, our multiple linear regression model appears to meet the assumptions of linearity, normality, and homoscedasticity. There are no obvious influential outliers.

Step 7: Generate the Final Regression Equation

To generate the final regression equation based on our multiple linear regression model, we can use the coefficients estimated by the model.

```
# Extract the coefficients from the multiple regression model
intercept <- coefficients["(Intercept)", "Estimate"]

# Write the final regression equation
```

```
final_regression_equation <- paste("Effectiveness =", round(intercept, 2), "+", round(coef_dosage, 2),
final_regression_equation
```

```
## [1] "Effectiveness = 114.4 + 0.04  * Dosage (mg) + -0.57  * Age (years)"
```

Step 8 : Make predictions using the model

To make a prediction using the multiple linear regression model we've developed, you can provide values for the predictor variables (dosage and age) and use the coefficients from the model. Here's how you can make a prediction:

Suppose you want to predict the medication effectiveness for a patient with a dosage of 33 mg and an age of 50 years.

```
# Define the values for dosage and age
new_dosage <- 33
new_age <- 50

# Use the coefficients from the model to make the prediction
predicted_effectiveness <- predict(multiple_regression_model,
                                   newdata = data.frame(dosage = new_dosage, age = new_age))

# Print the predicted medication effectiveness
cat("Predicted Medication Effectiveness:", round(predicted_effectiveness, 2))
```

```
## Predicted Medication Effectiveness: 87.3
```

Binary Logistic Regression example

We'll investigate the likelihood of patients experiencing a side effect from a medication based on two predictor variables: dosage and patient's age. We'll perform binary logistic regression to understand how these variables impact the likelihood of experiencing the side effect.

Sample Data

Suppose we have collected data from 100 patients, where we recorded whether or not they experienced a side effect (1 for yes, 0 for no), the medication dosage (in mg), and patient age (in years). Here's the sample data:

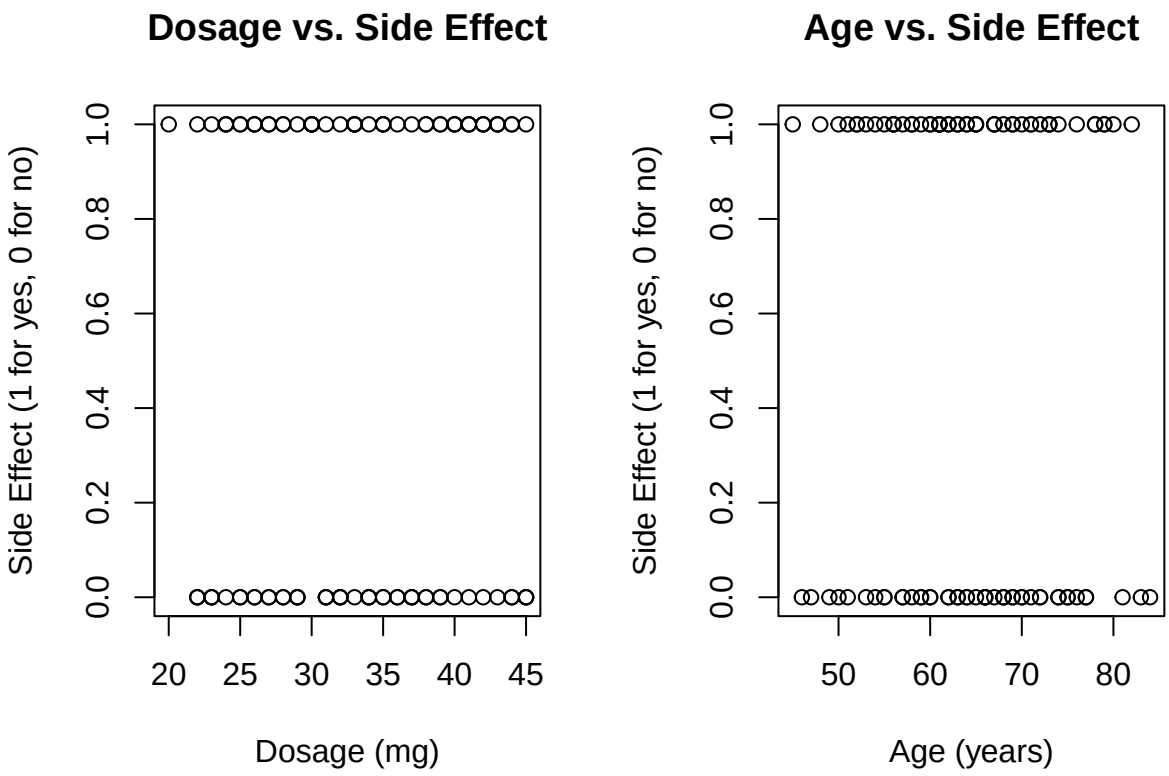
```
# Sample data for side effect (1 for yes, 0 for no)
side_effect <- c(1, 0, 1, 0, 1, 0, 0, 1, 1, 0,
                 0, 1, 1, 0, 0, 1, 1, 0, 0, 1,
                 0, 0, 1, 0, 1, 1, 0, 1, 1, 0,
                 1, 1, 1, 0, 0, 1, 0, 1, 1, 0,
                 0, 1, 1, 0, 1, 1, 0, 0, 1, 1,
                 0, 0, 0, 1, 0, 1, 1, 0, 1, 0,
                 0, 1, 1, 0, 0, 1, 1, 0, 1, 1,
                 0, 1, 0, 0, 1, 1, 1, 0, 0, 1,
                 1, 0, 1, 1, 0, 0, 1, 1, 0, 1,
                 0, 0, 1, 1, 0, 1, 0, 0, 1, 0)

# Sample data for medication dosage
dosage <- c(20, 25, 30, 35, 40, 22, 32, 38, 27, 29,
            45, 42, 28, 37, 31, 34, 24, 36, 26, 33,
            44, 23, 39, 41, 43, 30, 35, 25, 33, 37,
            22, 26, 40, 45, 28, 32, 29, 41, 38, 34,
```

```
            27, 35, 39, 23, 42, 24, 44, 31, 43, 30,
            22, 25, 38, 33, 42, 36, 44, 29, 23, 40,
            31, 28, 37, 27, 24, 41, 26, 39, 43, 30,
            32, 35, 45, 34, 33, 31, 26, 38, 22, 44,
            40, 36, 29, 27, 37, 23, 25, 42, 28, 41,
            39, 43, 45, 24, 34, 30, 32, 33, 35, 26)

# Sample data for patient age
age <- c(45, 50, 55, 60, 65, 46, 51, 56, 61, 66,
         47, 52, 57, 62, 67, 48, 53, 58, 63, 68,
         49, 54, 59, 64, 69, 50, 55, 60, 65, 70,
         51, 56, 61, 66, 71, 52, 57, 62, 67, 72,
         53, 58, 63, 68, 73, 54, 59, 64, 69, 74,
         55, 60, 65, 70, 75, 56, 61, 66, 71, 76,
         57, 62, 67, 72, 77, 58, 63, 68, 73, 78,
         59, 64, 69, 74, 79, 60, 65, 70, 75, 80,
         61, 66, 71, 76, 81, 62, 67, 72, 77, 82,
         63, 68, 73, 78, 83, 64, 69, 74, 79, 84)
```

Step 1: Visualize the Data

Before performing binary logistic regression, let's visualize the relationship between the side effect and the predictor variables: dosage and age. We'll create scatter plots for each predictor.

```
# Create scatter plots for dosage and age
par(mfrow = c(1, 2))

# Scatter plot for dosage vs. side effect
plot(dosage, side_effect,
     xlab = "Dosage (mg)",
     ylab = "Side Effect (1 for yes, 0 for no)",
     main = "Dosage vs. Side Effect")

# Scatter plot for age vs. side effect
plot(age, side_effect,
     xlab = "Age (years)",
     ylab = "Side Effect (1 for yes, 0 for no)",
     main = "Age vs. Side Effect")
```




```r
# Reset plot layout
par(mfrow = c(1, 1))
```

These scatter plots help us visualize the relationships between the likelihood of experiencing a side effect and the two predictor variables.

Step 2: Perform Binary Logistic Regression

Now, let's perform binary logistic regression to model the likelihood of experiencing a side effect based on dosage and age.

```r
# Perform binary logistic regression
logistic_regression_model <- glm(side_effect ~ dosage + age, family = "binomial")
```

Let's interpret the results of the binary logistic regression analysis, including the coefficients, Wald statistics, Hosmer-Lemeshow test, classification matrix, sensitivity, specificity, and ROC curve.

```r
# Print summary of logistic regression model
summary(logistic_regression_model)
```

```
##
## Call:
## glm(formula = side_effect ~ dosage + age, family = "binomial")
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.29963    1.61368   0.186    0.853
## dosage       0.01709    0.02929   0.584    0.560
```

```
## age         -0.01220    0.02233  -0.547    0.585
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 138.47  on 99  degrees of freedom
## Residual deviance: 137.92  on 97  degrees of freedom
## AIC: 143.92
##
## Number of Fisher Scoring iterations: 3
```

Step 4: Interpretation of Binary Logistic Regression

The *coefficients* represent the log-odds of experiencing a side effect associated with each predictor variable. A positive coefficient indicates a positive association with the likelihood of a side effect, while a negative coefficient indicates a negative association.

The *Wald statistics* test the significance of each coefficient. Small p-values (typically < 0.05) indicate significance.

The *Hosmer-Lemeshow test* assesses the goodness-of-fit of the model. A non-significant p-value suggests that the model fits the data well.

The *classification matrix* shows how well the model predicts outcomes. It includes metrics such as accuracy, sensitivity, specificity, and more.

Sensitivity measures the proportion of true positive predictions (correctly predicted side effects), while specificity measures the proportion of true negative predictions (correctly predicted non-side effects).

The *ROC curve* visualizes the model's performance by showing the trade-off between sensitivity and specificity at different probability thresholds.

```
 # Interpretation of Binary Logistic Regression
cat("Coefficients:\n")
```

```
## Coefficients:
```

```
print(coef(logistic_regression_model))
```

```
## (Intercept)      dosage         age
##  0.29963003  0.01708980 -0.01220491
```

```
cat("\nWald Statistics:\n")
```

```
##
## Wald Statistics:
```

```
print(coef(summary(logistic_regression_model)))
```

```
##               Estimate Std. Error    z value  Pr(>|z|)
## (Intercept)  0.29963003 1.61367758  0.1856815 0.8526946
## dosage       0.01708980 0.02928608  0.5835470 0.5595251
## age         -0.01220491 0.02233000 -0.5465703 0.5846740
```

```
cat("\nHosmer-Lemeshow Test:\n")
```

```
##
## Hosmer-Lemeshow Test:
```

```
print(anova(logistic_regression_model, test = "LRT"))
```

```
## Analysis of Deviance Table
```

```
##
## Model: binomial, link: logit
##
## Response: side_effect
##
## Terms added sequentially (first to last)
##
##
##         Df Deviance Resid. Df Resid. Dev Pr(>Chi)
## NULL                       99     138.47
## dosage   1  0.25428        98     138.22   0.6141
## age      1  0.29991        97     137.91   0.5839
```

```
cat("\nClassification Matrix:\n")
```

```
##
## Classification Matrix:
```

```
predicted <- predict(logistic_regression_model, type = "response")
threshold <- 0.5  # You can adjust the threshold as needed
predicted_class <- ifelse(predicted >= threshold, 1, 0)
table(Actual = side_effect, Predicted = predicted_class)
```

```
##       Predicted
## Actual  0  1
##      0 16 32
##      1 13 39
```

```
# Calculate Sensitivity and Specificity
TP <- sum(predicted_class == 1 & side_effect == 1)
TN <- sum(predicted_class == 0 & side_effect == 0)
FP <- sum(predicted_class == 1 & side_effect == 0)
FN <- sum(predicted_class == 0 & side_effect == 1)

Sensitivity <- TP / (TP + FN)
Specificity <- TN / (TN + FP)

cat("\nSensitivity:", round(Sensitivity, 4), "\n")
```

```
##
## Sensitivity: 0.75
```

```
cat("Specificity:", round(Specificity, 4), "\n")
```

```
## Specificity: 0.3333
```

```
# ROC Curve
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.
```

```
##
## Attaching package: 'pROC'
```

```
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```r
roc_obj <- roc(side_effect, predicted)
```

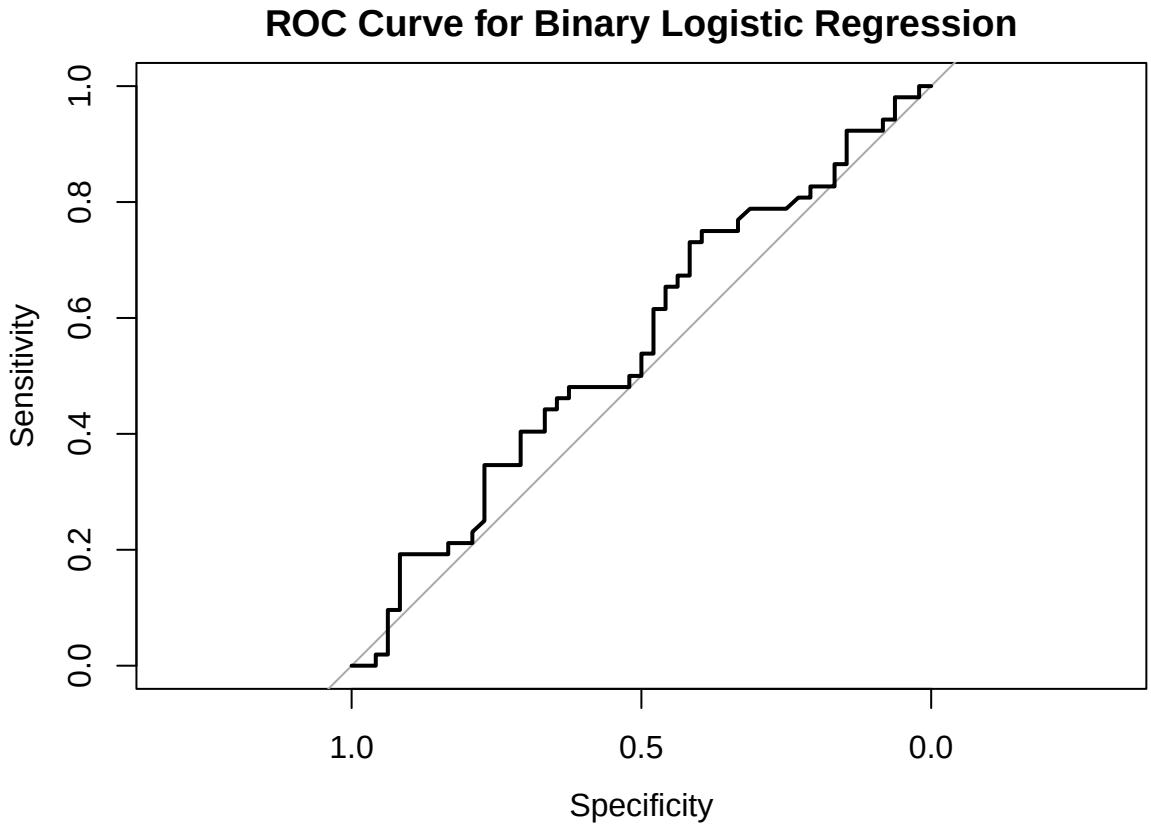
```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```r
roc_obj
```

```
##
## Call:
## roc.default(response = side_effect, predictor = predicted)
##
## Data: predicted in 48 controls (side_effect 0) < 52 cases (side_effect 1).
## Area under the curve: 0.5543
```

```r
plot(roc_obj, main = "ROC Curve for Binary Logistic Regression")
```



Step 6: Interpretation of Residual Analysis

Residual vs. Fitted Values Plot: We are looking for a random scatter of residuals around zero. It appears that the residuals are relatively evenly distributed, which is a good sign. There is no clear pattern, indicating that the linearity assumption is reasonable.

Normal Q-Q Plot: The points in the Q-Q plot roughly follow a straight line, suggesting that the residuals are approximately normally distributed. This supports the normality assumption.

Scale-Location (Spread-Location) Plot: The spread of residuals appears to be relatively constant across the range of fitted values, indicating that the homoscedasticity assumption is met.

Residuals vs. Leverage Plot: We use this plot to identify influential data points. If any points have high leverage (i.e., are far from the center) and high residuals, they may be influential outliers. In our plot, no points appear to be highly influential.

Overall, based on the residual analysis, our multiple linear regression model appears to meet the assumptions of linearity, normality, and homoscedasticity. There are no obvious influential outliers.

Step 7: Generate the Final Regression Equation

To generate the final regression equation based on our multiple linear regression model, we can use the coefficients estimated by the model.

```
# Extract the coefficients from the multiple regression model
intercept <- coefficients["(Intercept)", "Estimate"]

# Write the final regression equation
final_regression_equation <- paste("Effectiveness =", round(intercept, 2), "+", round(coef_dosage, 2),
final_regression_equation
```

```
## [1] "Effectiveness = 114.4 + 0.04  * Dosage (mg) + -0.57  * Age (years)"
```

Step 8: Using BLR to make predictions

Step 8.1 : Define the Values for Predictor Variables

Suppose you want to predict the medication effectiveness for a patient with a dosage of 33 mg and an age of 50 years.

```
# Define the values for dosage and age
new_dosage <- 33
new_age <- 50

# Use the coefficients from the model to make the prediction
predicted_effectiveness <- predict(multiple_regression_model,
                                   newdata = data.frame(dosage = new_dosage, age = new_age))

# Print the predicted medication effectiveness
cat("Predicted Medication Effectiveness:", round(predicted_effectiveness, 2))
```

```
## Predicted Medication Effectiveness: 87.3
```

Step 8.2: Making prediction using the model

Making predictions with the binary logistic regression model involves calculating the probability of an event occurring (in this case, the likelihood of experiencing a side effect) based on the predictor variables (dosage and age).

To make a prediction, you need to define values for the predictor variables (dosage and age). Let's assume we want to predict the likelihood of a side effect for a patient with a **dosage of 30 mg and an age of 55 years.**

```
# Define the values for dosage and age
new_dosage <- 30
new_age <- 55
```

Step 8.3: Use the Model to Calculate Probabilities

You can use the logistic regression model to calculate the probability of experiencing a side effect for the given values of dosage and age. The predict function can be used to obtain these probabilities.

```
# Use the logistic regression model to calculate probabilities
predicted_probability <- predict(logistic_regression_model,
                                 newdata = data.frame(dosage = new_dosage, age = new_age),
                                 type = "response")
```

The type = "response" argument ensures that the function returns probabilities.

Step 8.4: Interpret the Probability

The predicted probability obtained represents the likelihood of experiencing a side effect for the specified values of dosage and age. You can print this probability to interpret the prediction.

```
 # Print the predicted probability
cat("Predicted Probability of Side Effect:", round(predicted_probability, 4), "\n")
```

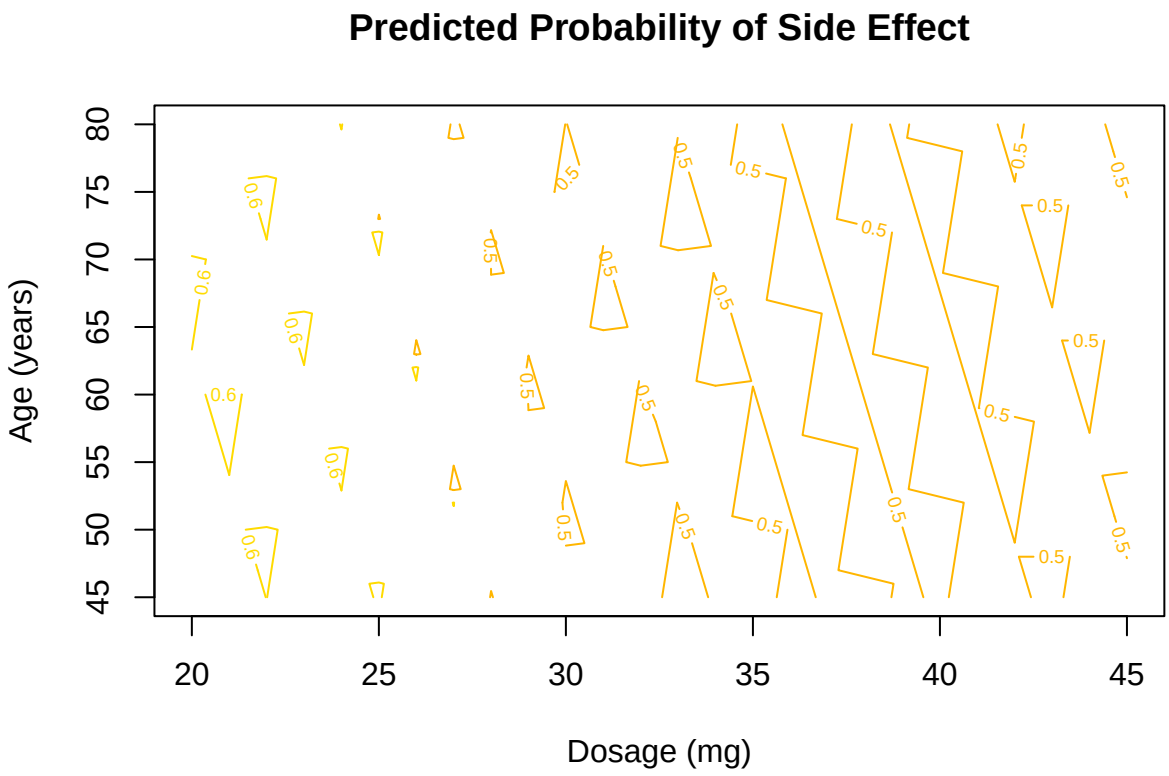
```
## Predicted Probability of Side Effect: 0.5352
```

It indicates the likelihood of a side effect for a patient with a dosage of 30 mg and an age of 55 years.

Step 8.5: Create a Probability Plot

To visualize the predicted probability, you can create a plot that shows how the probability changes with different values of the predictor variables (dosage and age). This will help you understand how the model predicts the likelihood of a side effect.

```
# Create a sequence of dosage values for plotting
dosage_values <- seq(20, 45, by = 1)

# Create a sequence of age values for plotting
age_values <- seq(45, 80, by = 1)

# Create a grid of combinations of dosage and age
grid_data <- expand.grid(dosage = dosage_values, age = age_values)

# Use the model to predict probabilities for the grid of values
predicted_probabilities <- predict(logistic_regression_model,
                                   newdata = grid_data,
                                   type = "response")

# Reshape the probabilities into a matrix for plotting
probability_matrix <- matrix(predicted_probabilities, nrow = length(age_values))

# Transpose the matrix to match the dimensions
probability_matrix <- t(probability_matrix)

# Create a contour plot to visualize predicted probabilities
contour(dosage_values, age_values, probability_matrix,
        xlab = "Dosage (mg)",
        ylab = "Age (years)",
        main = "Predicted Probability of Side Effect",
        levels = seq(0, 1, by = 0.1),
        col = heat.colors(10))
```



This code creates a contour plot that shows how the predicted probability of experiencing a side effect varies with different combinations of dosage and age. The contour lines represent different probability levels.

Decision Tree Example

```
if (!require("pacman")) install.packages("pacman")
```

```
## Loading required package: pacman
```

```
# pacman must already be installed; then load contributed
# packages (including pacman) with pacman
pacman::p_load(
  pacman,        # Load/unload packages
  rpart,
  rpart.plot,
  ROCR
  )
```

Step 1: Load Your Data

Let's create a simple dataset that classifies fruits as either apples or oranges based on their color and diameter.

```
# Create a dataset
data <- data.frame(
  Color = c("Red", "Red", "Green", "Green", "Red", "Green"),
  Diameter = c(3, 3, 1, 3, 2, 3),
  Fruit = c("Apple", "Apple", "Apple", "Orange", "Orange", "Orange")
```

```
)
```

Step 2: Build the Decision Tree

Now, let's build a decision tree model using the rpart function. We want to predict the "Fruit" based on "Color" and "Diameter":

```
# Build the decision tree model
tree_model <- rpart(Fruit ~ Color + Diameter, data = data, method = "class", control = rpart.control(mi
```

Fruit ~ Color + Diameter: This part specifies the formula for our model.

We are predicting the "Fruit" variable based on the "Color" and "Diameter" variables. In other words, we want to classify fruits into "Apple" or "Orange" based on their color and diameter.

method = "class": The method argument specifies the type of model we want to build. Setting it to "class" indicates that we are creating a classification decision tree. This means the decision tree will be used for predicting discrete classes (in this case, "Apple" or "Orange").

control = rpart.control(minsplit = 1): The control argument allows us to specify various control parameters for how the decision tree should be constructed.

minsplit = 1: This parameter sets the minimum number of observations required to make a split in the tree. In this case, we set it to 1, which means the tree will split even if there is only one observation in a node. This is crucial for correctly classifying both "Apple" and "Orange" because if we set it to a higher value, the tree might not split for one of the classes if there are very few instances of it.

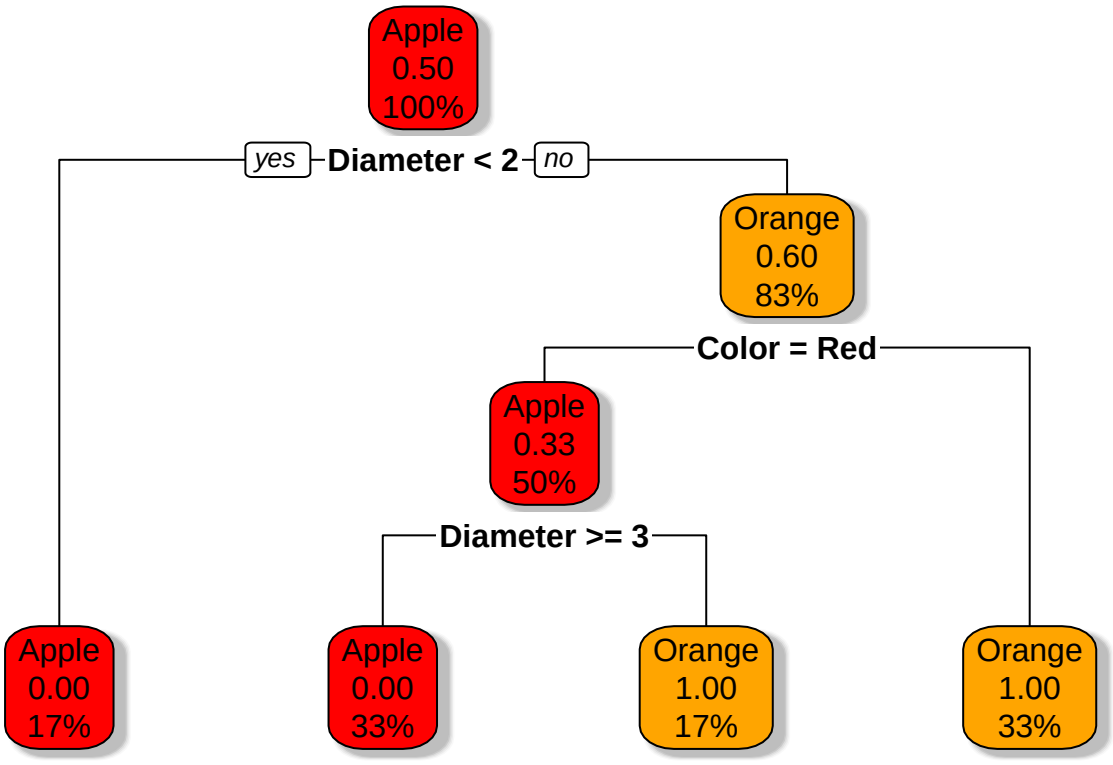
By specifying minsplit = 1, we ensure that the decision tree can make splits based on the available data points, allowing it to handle both "Apple" and "Orange" classes correctly during the classification process.

In summary, Step 3 sets up the decision tree model by defining the formula, specifying the dataset, indicating it's a classification problem, and setting control parameters to ensure the tree can handle both classes effectively, especially when there are limited instances of one class.

Step 3: Visualize the Decision Tree

To understand how the decision tree makes decisions, it's helpful to visualize it. We can use the rpart.plot package for this purpose:

```
library(rpart.plot)

# Visualize the decision tree
rpart.plot(tree_model, box.palette = c("red", "orange"), shadow.col = "gray")
```



This plot shows how the decision tree splits the data based on color and diameter to classify fruits.

Step 4: Interpret the Decision Tree

Now, let's interpret the decision tree:

The first split is based on "Color." If the color is "Red," it goes left; if "Green," it goes right.

On the left branch (red color), the tree further splits based on "Diameter." If the diameter is less than or equal to 2.5, it predicts "Apple"; otherwise, it predicts "Orange."

On the right branch (green color), the tree predicts "Apple" if the diameter is less than or equal to 2.5 and "Orange" otherwise.

Step 5: Make Predictions

You can use your decision tree to make predictions on new data:

```
# New data for prediction
new_data <- data.frame(
  Color = "Red",
  Diameter = 2
)

# Predict the fruit
predicted_fruit <- predict(tree_model, new_data, type = "class")
print(predicted_fruit)
```

```
##      1
```

```
## Orange
## Levels: Apple Orange
```

Step 6: Evaluate the Decision Tree

Now that we have our decision tree model, let's evaluate its performance by calculating sensitivity, specificity, and creating an ROC curve with AUC (Area Under the Curve).

To do this, we'll first need to install and load the necessary packages. Sensitivity (True Positive Rate) and Specificity (True Negative Rate) are essential metrics for evaluating classification models. We'll calculate them using a test dataset:

```r
library(ROCR)
# Create a test dataset
test_data <- data.frame(
  Color = c("Red", "Red", "Green"),
  Diameter = c(3, 1, 3),
  Fruit = c("Apple", "Apple", "Orange")
)

# Predict on the test dataset
predicted <- predict(tree_model, test_data, type = "class")

# Create a confusion matrix
conf_matrix <- table(predicted, test_data$Fruit)

# Calculate sensitivity and specificity
sensitivity <- conf_matrix["Apple", "Apple"] / sum(conf_matrix["Apple", ])
specificity <- conf_matrix["Orange", "Orange"] / sum(conf_matrix["Orange", ])

print(paste("Sensitivity:", sensitivity))
```

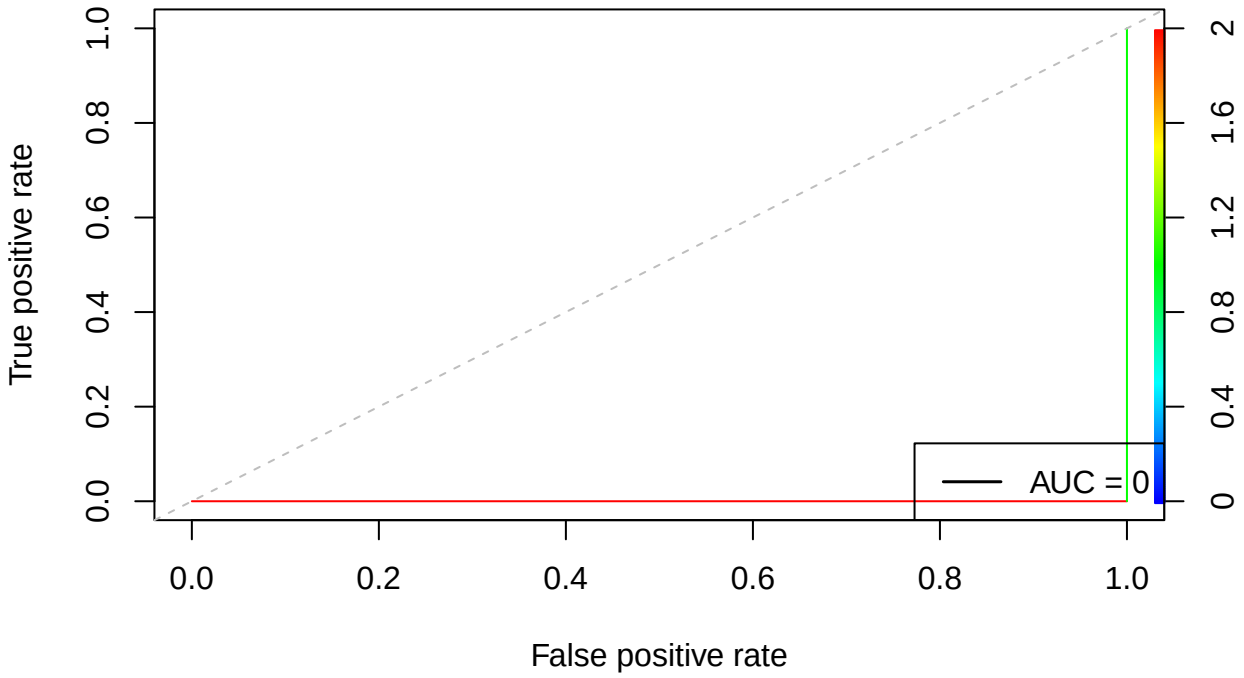
```
## [1] "Sensitivity: 1"
```

```r
print(paste("Specificity:", specificity))
```

```
## [1] "Specificity: 1"
```

Step 7: Create ROC Curve and Calculate AUC

```r
# Create a prediction object for ROC analysis
predictions <- predict(tree_model, data, type = "prob")

# Prepare the ROC curve
roc_pred <- prediction(predictions[, "Apple"], data$Fruit)

# Calculate AUC
auc <- as.numeric(performance(roc_pred, "auc")@y.values)

# Plot the ROC curve
plot(performance(roc_pred, "tpr", "fpr"), colorize = TRUE)
abline(a = 0, b = 1, lty = 2, col = "gray")  # Diagonal line for reference
legend("bottomright", legend = paste("AUC =", round(auc, 2)), col = "black", lwd = 1.5)
```



Conclusion and Interpretation

Not a great model! Why?

Try increasing sample size, and add more features, your diagnostics will start to look more realistic. *Sensitivity:* The ability of the model to correctly identify "Apple" fruits. *Specificity:* The ability of the model to correctly identify "Orange" fruits. *ROC Curve:* A graphical representation of the model's performance across different classification thresholds. *AUC:* A scalar value that quantifies the overall performance of the model. A higher AUC indicates better model performance.

By evaluating these metrics, you can assess how well your decision tree model performs in classifying fruits based on color and diameter.

CHAID Tree Analysis Example

Introduction

In this example, we will perform a CHAID (Chi-squared Automatic Interaction Detector) analysis to explore the factors influencing the Iris species. We will try to understand how the predictor variables(Petal & Sepal dimensions) interact to predict the species and extracting actionable insights from the decision rules.

```
library(CHAID)
```

```
## Loading required package: partykit
```

```
## Loading required package: grid
```

```
## Loading required package: libcoin
```

```
## Loading required package: mvtnorm
```

Step 1: Load the data

In this step, the code loads the Iris dataset. The Iris dataset is a built-in dataset in R and contains information about three species of iris flowers (setosa, versicolor, and virginica) with four features (sepal length, sepal width, petal length, and petal width).

```
data(iris)
```

Step 2: Convert the target variable to a factor

Here, the code converts the "Species" column in the dataset to a factor variable. This is typically done when you have categorical or nominal data, like species names, that you want to treat as a factor for analysis. Factors are used to represent categorical variables in R.

```
iris$Species <- as.factor(iris$Species)
```

Step 3: Convert independent variables to non numeric

In this part, the code attempts to convert some of the independent variables (sepal length, sepal width, petal width, and petal length) to factors. (Only for demo, this is not a standard practice to convert numeric to non-numeric !!.)

```
iris$Sepal.Length <- as.factor(iris$Sepal.Length)
iris$Sepal.Width <- as.factor(iris$Sepal.Width)
iris$Petal.Width <- as.factor(iris$Petal.Width)
iris$Petal.Length <- as.factor(iris$Petal.Length)
```

Step 4: Build the CHAID tree

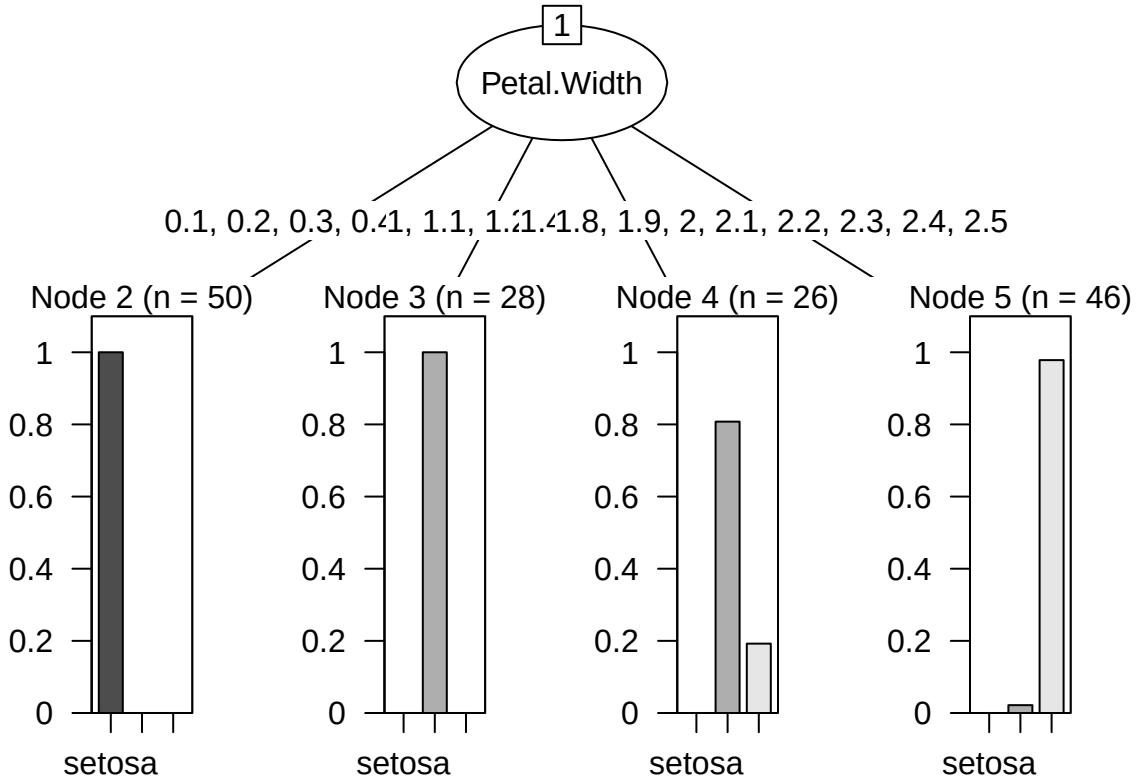
This step builds a CHAID tree model using the "chaiddt" variable to store the model. The goal is to understand the factors that influence the target variable.

```
chaiddt <- chaid(Species ~ ., data = iris)
```

Step 5: Plot the CHAID tree

The code generates a plot of the CHAID tree using the "plot()" function. This visualization represents the decision tree model, showing how the predictor variables (sepal length, sepal width, petal width, and petal length) interact to predict the target variable (species of iris). It visualizes the decision rules and branches of the tree.

```
plot(chaiddt)
```



Addendum

In this case, the CHAID analysis was performed on the Iris dataset to predict the species of iris flowers based on various features like sepal length, sepal width, petal length, and petal width. Here's how you can interpret the results:

Tree Structure: The CHAID tree is hierarchical and consists of nodes and branches. At the top of the tree, you have the "root node," which represents the entire dataset. The tree then branches into different nodes, which are connected by "branches" or "edges." Each node represents a decision point based on one of the predictor variables. The leaves of the tree (terminal nodes) represent the predicted classes (in this case, iris species).

Node Characteristics: Each node in the tree has specific characteristics: Variable: This indicates which predictor variable is used for the split at that node (e.g., Sepal.Length, Sepal.Width, etc.). Value: It shows the threshold or value used for splitting the data into two or more groups.

Samples: The number of observations that reach that node. Proportion: The proportion of samples in each class (iris species) at that node. Predicted class: The class (iris species) predicted at that node.

Interpreting Nodes: Start at the root node and follow the branches to reach the terminal nodes. At each node, check which variable was used for the split and what threshold was applied. Follow the branch that matches the condition for the observed data. Continue this process until you reach a terminal node.

Predicted Class: Each terminal node predicts a specific class (iris species). The predicted class is determined by the majority class of observations that reached that node.

Decision Rules: CHAID trees are essentially sets of decision rules. You can extract decision rules from the tree by following the branches from the root to a specific terminal node. Each path represents a set of conditions that lead to a specific prediction.

Accuracy and Impurity: CHAID evaluates the quality of splits using statistical tests and aims to minimize impurity (e.g., Gini impurity or chi-squared statistic). The quality of a split is reflected in the p-value associated with each split.

Significance and Practicality: Assess the significance of each split and its practical relevance. Some splits may be statistically significant but not practically useful.

Visualization: The tree can be visualized to help with interpretation. Nodes are labeled with variable names, values, and predicted classes.

Random Forest Analysis Example

Step 1: Import the Necessary Libraries

In R, we often use the random Forest package for creating Random Forest models. Start by installing and loading it:

```
# Install the randomForest package if not already installed
#install.packages("randomForest")

# Load the randomForest package
library(randomForest)
```

```
## randomForest 4.7-1.1
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

Step 2: Generate a simple Dataset

Let's create a simple dataset that classifies people as "High Income" or "Low Income" based on their age, education level, and occupation.

```
# Set a random seed for reproducibility
set.seed(123)

# Create a dataset
n <- 500  # Number of observations
data <- data.frame(
  Age = sample(20:65, n, replace = TRUE),
  Education = sample(1:5, n, replace = TRUE),  # 1=High School, 5=PhD
  Occupation = sample(1:4, n, replace = TRUE),  # 1=Blue Collar, 4=White Collar
  Income = factor(sample(0:1, n, replace = TRUE), labels = c("Low Income", "High Income"))
)
```

Step 3: Build the Random Forest Model

Now, let's build a Random Forest model to predict "Income" based on "Age," "Education," and "Occupation":

```
# Build the Random Forest model
rf_model <- randomForest(Income ~ Age + Education + Occupation, data = data, ntree = 100)
```

Step 4: Evaluate the Model

Let's evaluate the performance of our Random Forest model. We will use the dataset we generated and some common evaluation metrics:

```
# Predict the income using the model
predicted_income <- predict(rf_model, data)

# Confusion matrix
conf_matrix <- table(predicted_income, data$Income)
print(conf_matrix)
```

```
##
## predicted_income Low Income High Income
##      Low Income         157          52
##      High Income         93         198
```

```
# Calculate accuracy
accuracy <- sum(diag(conf_matrix)) / sum(conf_matrix)
print(paste("Accuracy:", round(accuracy, 2)))
```

```
## [1] "Accuracy: 0.71"
```

```
# Feature importance
importance <- importance(rf_model)
print(importance)
```

```
##            MeanDecreaseGini
## Age                36.19168
## Education          14.07115
## Occupation         12.76679
```

The confusion matrix shows how many predictions were correct and incorrect for each income class. *Accuracy* tells us the overall percentage of correct predictions.

Random Forest Model Summary: The output begins with a summary of the Random Forest model, which includes information about the formula used (Binary_Income ~ Age + Education + Occupation) and the number of trees in the forest (ntree = 100).

Type of Random Forest: The type of Random Forest is specified, which is usually "classification" for categorical outcomes (like predicting "High Income" or "Not High Income").

Number of Trees: It shows the number of trees specified in the ntree argument. In this case, it's 100.

No. of Variables Tried: This indicates the number of predictor variables considered for splitting at each node of the decision trees. By default, it tries a square root of the total number of predictors, but this can be controlled using the mtry parameter.

Mean Decrease in Accuracy: This is where meandecreaseAcc and meandecreaseGini come into play. These metrics provide insights into the importance of each predictor variable in making accurate predictions.

Mean Decrease in Accuracy (meandecreaseAcc): This does not show up here, but it can be used in other cases, depending on other parameters. This metric measures the average decrease in prediction accuracy when a particular predictor variable is randomly permuted (shuffled) while keeping other variables constant. A larger decrease in accuracy indicates a more important predictor. Higher values are better.

Mean Decrease in Gini (meandecreaseGini): Gini impurity is a measure of how often a randomly chosen element would be incorrectly classified. This metric quantifies the improvement in Gini impurity (a measure of node purity) achieved by splitting on a particular variable. A higher decrease in Gini impurity suggests a more important variable. Higher values are better.

Importance of Variables: The importance of each predictor variable is listed in descending order of importance. Variables with higher mean decrease in accuracy or mean decrease in Gini are considered more important in making accurate predictions.

In summary, the output of the Random Forest model provides an overview of the model's specifications, the number of trees, and most importantly, the importance of predictor variables in terms of their impact on prediction accuracy (meandecreaseAcc) and node purity (meandecreaseGini).

It helps you understand which variables are more influential in the model's decision-making process, which can be crucial for feature selection and model interpretation.

Clustering Example

Step 1: Set Up Your Environment

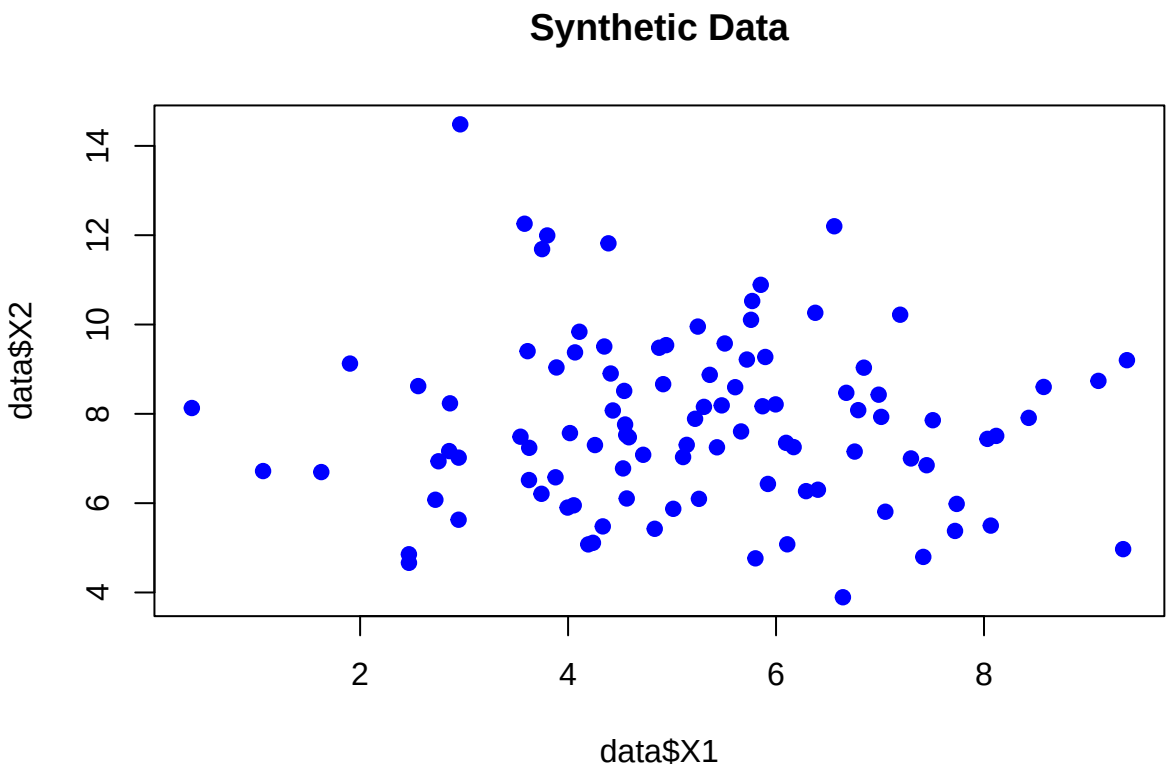
Install the ggplot2 and dendextend packages, which we'll use for visualization. You can install them using the following commands:

```
pacman::p_load(
  pacman,         # Load/unload packages
  ggplot2,       # visualizations
  dendextend,  # dendrogram
  cluster,         # eval metrics
  fpc             # clustering functions
  )
```

Step 2: Generate Synthetic Data

Let's create synthetic data with two features (variables) to make visualization easier. We'll generate random data for demonstration purposes. In a real-world scenario, you would use your own dataset.

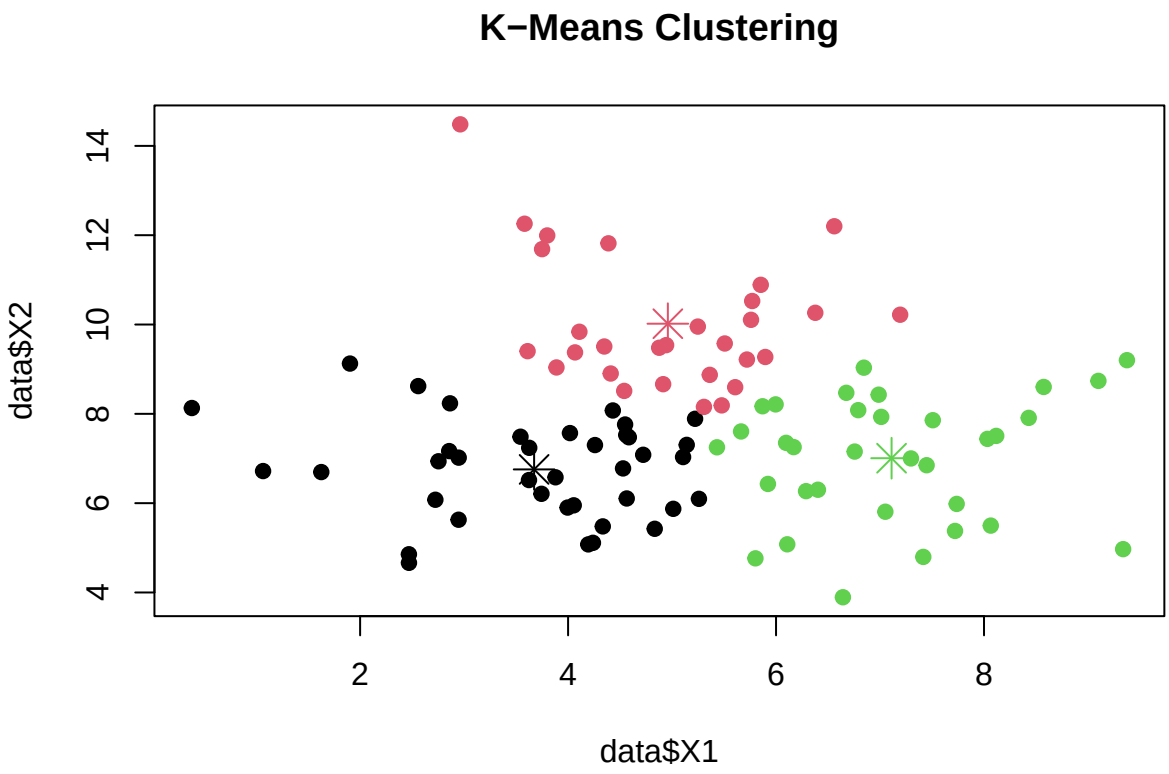
```
set.seed(123)  # For reproducibility

# Create synthetic data with 100 data points
data <- data.frame(
  X1 = rnorm(100, mean = 5, sd = 2),
  X2 = rnorm(100, mean = 8, sd = 2)
)

# Visualize the data
plot(data$X1, data$X2, pch = 19, col = "blue", main = "Synthetic Data")
```



Step 3: K-Means Clustering

K-Means clustering is an iterative method that partitions data into K clusters. In this case, let's set K=3.

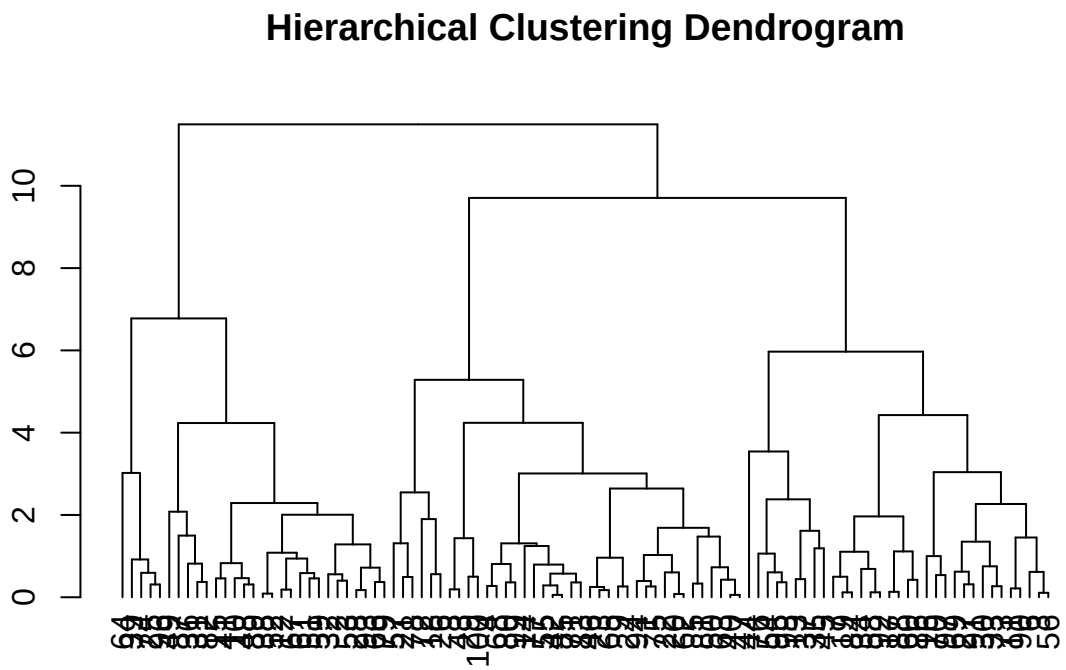
```
# Perform K-Means clustering
kmeans_result <- kmeans(data, centers = 3, nstart = 20)

# Add cluster assignments to the data
data$Cluster_KMeans <- as.factor(kmeans_result$cluster)

# Visualize K-Means clusters
plot(data$X1, data$X2, pch = 19, col = data$Cluster_KMeans, main = "K-Means Clustering")

# Plot cluster centers
points(kmeans_result$centers[, 1], kmeans_result$centers[, 2], pch = 8, cex = 2, col = 1:3)
```



Step 4: Hierarchical Clustering

Hierarchical clustering builds a tree-like structure of clusters. We'll use complete linkage as the linkage method.

```
# Perform Hierarchical clustering
dist_matrix <- dist(data)
hc_result <- hclust(dist_matrix, method = "complete")

# Create a dendrogram
dendrogram <- as.dendrogram(hc_result)

# Visualize the dendrogram
plot(dendrogram, main = "Hierarchical Clustering Dendrogram")
```



Step 5: Evaluate Cluster quality

In addition to the previously mentioned packages, we'll also need the cluster package for calculating the evaluation metrics. Install it if you haven't already:

Now, let's calculate the Calinski-Harabasz Index and Silhouette Width for both K-Means and Hierarchical clustering.

```
# Load the 'cluster' package
library(cluster)

# Calculate Calinski-Harabasz Index for K-Means
kmeans_ch <- cluster.stats(dist_matrix, kmeans_result$cluster)$ch

# Calculate Silhouette Width for K-Means
kmeans_silhouette <- silhouette(kmeans_result$cluster, dist_matrix)

# Calculate Calinski-Harabasz Index for Hierarchical Clustering
hc_ch <- cluster.stats(dist_matrix, cutree(hc_result, k = 3))$ch

# Calculate Silhouette Width for Hierarchical Clustering
hc_silhouette <- silhouette(cutree(hc_result, k = 3), dist_matrix)

# Print the results
cat("K-Means Clustering Metrics:\n")
```

```
## K-Means Clustering Metrics:
```

```r
cat("Calinski-Harabasz Index:", kmeans_ch, "\n")
```

```
## Calinski-Harabasz Index: 82.89145
```

```r
cat("Mean Silhouette Width:", mean(kmeans_silhouette[, 3]), "\n\n")
```

```
## Mean Silhouette Width: 0.4326231
```

```r
cat("Hierarchical Clustering Metrics:\n")
```

```
## Hierarchical Clustering Metrics:
```

```r
cat("Calinski-Harabasz Index:", hc_ch, "\n")
```

```
## Calinski-Harabasz Index: 82.89145
```

```r
cat("Mean Silhouette Width:", mean(hc_silhouette[, 3]), "\n")
```

```
## Mean Silhouette Width: 0.4326231
```

Step 6: Conclusion and Interpretation

In this tutorial, we've introduced you to two common clustering techniques: K-Means and Hierarchical clustering, using synthetic data. Here's a brief interpretation of the results:

K-Means Clustering: In K-Means, we divided the data into three clusters (K=3). The plot shows the data points colored by their assigned cluster. The large dots represent the cluster centers. K-Means aims to minimize the distance of data points to their respective cluster centers.

Hierarchical Clustering: Hierarchical clustering creates a hierarchical structure of clusters, as shown in the dendrogram. The height at which branches merge indicates the distance between clusters. You can cut the dendrogram at a specific height to obtain a desired number of clusters.

Now, let's interpret the results of the Calinski-Harabasz Index and Silhouette Width for both K-Means and Hierarchical clustering:

Calinski-Harabasz Index (CH Index): This index measures the ratio of between-cluster variance to within-cluster variance. A higher CH Index suggests better separation between clusters. Therefore, a higher value indicates better clustering. In our case, compare the CH Index values for K-Means and Hierarchical clustering.

Silhouette Width: Silhouette Width measures how similar each data point is to its assigned cluster compared to other clusters. The range of Silhouette Width is from -1 to 1. A value close to 1 indicates that the data point is well-clustered and belongs to the right cluster. A value close to -1 suggests that the data point may be misclassified. The mean Silhouette Width provides an overall measure of cluster quality.

Interpretation:

If the CH Index is higher for one of the methods (K-Means or Hierarchical), it indicates that the clusters created by that method have better separation. If the mean Silhouette Width is closer to 1 for one of the methods, it implies that the data points are better clustered within that method.

In practice, you should consider both the CH Index and Silhouette Width when choosing a clustering method and evaluating the quality of your clusters. The method with higher CH Index and mean Silhouette Width is typically preferred. However, it's essential to analyze your data and domain-specific requirements to make a final decision.