R Basics

A variable in programming or statistics is a symbolic name associated with a value. It acts as a container for storing data that can be manipulated or analyzed during computation or analysis.

Types of Variables

Numerical (Quantitative):

1. **Continuous**:Can take any value within a range (e.g., height, weight).
2. **Discrete**:Takes specific,countable values (e.g.number of cars,age in years).

```
# Numerical Variables
age <- 25              # Continuous
class(age)
```

```
## [1] "numeric"
```

```
num_cars <- 3          # Discrete
class(num_cars)
```

```
## [1] "numeric"
```

Categorical (Qualitative):

1. **Nominal**:Categories without a logical order (e.g., gender, color).
2. **Ordinal**:Categories with a logical order (e.g. education level, customer satisfaction ratings).
3. **Binary**:Variables with only two possible values (e.g., yes/no, 0/1, true /false).
4. **Text**:Variables that store strings of characters (e.g names, addresses).
5. **Date/Time**:Variables representing dates or times (e.g."2025-01-08","12:30 PM").

```
# Categorical Variables
gender <- "Male"        # Nominal
class(gender)
```

```
## [1] "character"
```

```
education <- "Graduate" # Ordinal
class(education)
```

```
## [1] "character"
```

```r
# Binary Variable
is_employed <- TRUE     # Binary
class(is_employed)
```

```
## [1] "logical"
```

```r
# Text Variable
name <- "Alice"
class(name)
```

```
## [1] "character"
```

```r
# Date/Time Variable
date_of_birth <- as.Date("1990-01-01")
class(date_of_birth)
```

```
## [1] "Date"
```

```r
# for complex variable
k<-3+5i
class(k)
```

```
## [1] "complex"
```

Based on Usage in Analysis:

1. **Independent Variable**:The variable that is manipulated or controlled in an experiment (predictor or input).
2. **Dependent Variable**:The variable being measured or studied (response or output).
3. **Control Variable**:Kept constant to prevent from influencing the results.

```r
# Created a dataframe of data with independent,dependent and control variable.
data <- data.frame(
  Temperature = c(30, 32, 35, 28, 25),        # Independent Variable (Predictor)
  Sales = c(200, 220, 250, 180, 150),         # Dependent Variable (Response)
  Advertising = c(1000, 1200, 1500, 800, 700) # Control Variable
)

# Linear model including independent and control variables
model <- lm(Sales ~ Temperature + Advertising, data = data)

# Display the model summary
summary(model)
```

```
## Warning in summary.lm(model): essentially perfect fit: summary may be
## unreliable
```

```
##
## Call:
```

```
## lm(formula = Sales ~ Temperature + Advertising, data = data)
##
## Residuals:
##          1          2          3          4          5
##  1.476e-13 -5.497e-14 -1.076e-14 -8.187e-14 -7.321e-28
##
## Coefficients:
##               Estimate Std. Error    t value Pr(>|t|)
## (Intercept) -1.000e+02  1.574e-12 -6.352e+13   <2e-16 ***
## Temperature  1.000e+01  8.724e-14  1.146e+14   <2e-16 ***
## Advertising -5.719e-16  1.035e-15 -5.520e-01    0.636
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.257e-13 on 2 degrees of freedom
## Multiple R-squared:      1,  Adjusted R-squared:      1
## F-statistic: 1.834e+29 on 2 and 2 DF,  p-value: < 2.2e-16
```

Based on Scope (in programming)

1. **Global Variable**:Accessible throughout the entire program.
2. **Local Variable**:Accessible only within the function or block in which it is defined.

```r
# Global Variable
global_var <- "I am a global variable"

# Function to demonstrate local and global variables
example_function <- function() {
  # Local Variable
  local_var <- "I am a local variable"

  # Accessing the global variable
  print(global_var)

  # Accessing the local variable
  print(local_var)
}

# Call the function
example_function()
```

```
## [1] "I am a global variable"
## [1] "I am a local variable"
```

```r
# Try accessing the local variable outside the function
# print(local_var)  # This will give an error because local_var is not defined globally
```

Storing value in the variable

```r
s<-"adil";s
```

```
## [1] "adil"
```

```r
t<-"usman";t
```

```
## [1] "usman"
```

```r
u<-"osama";u
```

```
## [1] "osama"
```

```r
x<-4;x
```

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

```r
y<-TRUE;y
```

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

```r
z<-3+2i;z
```

```
## [1] 3+2i
```

arithmetic operators

```r
x<-2
y<-3

x+y           # addition
```

```
## [1] 5
```

```r
x-y           # subtraction
```

```
## [1] -1
```

```r
x/y           # division
```

```
## [1] 0.6666667
```

```r
x*y           # multiplication
```

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

relational operators

```r
x>y
```

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

```r
x<y
```

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

```r
x==y
```

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

Data Structure

```r
# Vector that is a homogeneous single dim data structure
g<-c(1,2,3,4,5)
g
```

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

```r
z<-c("k","z")
z
```

```
## [1] "k" "z"
```

```r
f<-c(T,F,T,F)
f
```

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

```r
# heterogeneous vector
v<-c(1,2,3,4,5,F,T,F)
v
```

```
## [1] 1 2 3 4 5 0 1 0
```

extract vector for the vector

```r
f[1:4]
```

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

```r
f[-4]
```

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

```r
f[3]
```

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

now we create a list of the different elements

```r
x<-list(12,"khawar",TRUE)
x
```

```
## [[1]]
## [1] 12
##
## [[2]]
## [1] "khawar"
##
## [[3]]
## [1] TRUE
```

```r
class(x[1])
```

```
## [1] "list"
```

```r
class(x[2])
```

```
## [1] "list"
```

```r
class(x[3])
```

```
## [1] "list"
```

Now create list of vectors

```r
y<-list(c(1,2,3,4,5),c("k","z"),c(1,2,3,4,5,F,T,F))
y
```

```
## [[1]]
## [1] 1 2 3 4 5
##
## [[2]]
## [1] "k" "z"
##
## [[3]]
## [1] 1 2 3 4 5 0 1 0
```

```r
y[[2]][2]
```

```
## [1] "z"
```

```r
y[[3]][3]
```

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

now we create the data frame in the r

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

```
##      Name Age        City
## 1   Alice  25    New York
## 2     Bob  30 Los Angeles
## 3 Charlie  22     Chicago
```

```r
df$Name
```

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

```r
df$City
```

```
## [1] "New York"    "Los Angeles" "Chicago"
```

some built-in function in the r studio

```r
data("iris")          # call the built in iris dataset.
View(iris)            # view the data set.
str(iris)             # see the strcture of iris data set.
```

```
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

```r
head(iris)            # see the first few rows of data.
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```r
tail(iris)            # see the last few rows of data.
```

```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
## 145          6.7         3.3          5.7         2.5 virginica
## 146          6.7         3.0          5.2         2.3 virginica
## 147          6.3         2.5          5.0         1.9 virginica
## 148          6.5         3.0          5.2         2.0 virginica
## 149          6.2         3.4          5.4         2.3 virginica
## 150          5.9         3.0          5.1         1.8 virginica
```

```r
summary(iris)         # see the basic statistics of data colomns.
```

```
##   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width
##  Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300
##  Median :5.800   Median :3.000   Median :4.350   Median :1.300
##  Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
##  3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
##  Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
##        Species
##  setosa    :50
##  versicolor:50
##  virginica :50
##
##
##
```

```r
dim(iris)             # show no. of rows and colomns of data.
```

```
## [1] 150   5
```

```r
nrow(iris)            # no. of rows of data.
```

```
## [1] 150
```

```r
ncol(iris)            # no. of colomns of data.
```

```
## [1] 5
```

```r
table(iris$Species)   # use to check the frequency of the species
```

```
##
##     setosa versicolor  virginica
##         50         50         50
```

user define function in r

```r
# Define a function called 'square' that takes one argument 'x'
square <- function(x) {
  result <- x^2
  return(result)
}

# Call the 'square' function with an argument and store the result in a variable
output <- square(5)

# Print the result
print(output)
```

```
## [1] 25
```

factor and data frame

```r
x<-factor(c("zohan","dark","light"))
x
```

```
## [1] zohan dark  light
## Levels: dark light zohan
```

```r
x1<-factor(c("zohan","dark","light","zohan","dark","light"))
x1
```

```
## [1] zohan dark  light zohan dark  light
## Levels: dark light zohan
```

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

```
##      Name Age        City
## 1   Alice  25    New York
## 2     Bob  30 Los Angeles
## 3 Charlie  22     Chicago
```

```
df$Name
```

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

```
df$City
```

```
## [1] "New York"    "Los Angeles" "Chicago"
```

now we see how to call the data in repositry in xlsx or csv format.

```
# calling data in excel format
library(readxl)
df<-read_excel("data/kashti.xlsx")  # call the kashti data set.
head(df[,1:8])                      # we call the first few colomns of df
```

```
## # A tibble: 6 x 8
##   survived pclass sex      age sibsp parch  fare embarked
##      <dbl>  <dbl> <chr>  <dbl> <dbl> <dbl> <dbl> <chr>
## 1        0      3 male      22     1     0  7.25 S
## 2        1      1 female    38     1     0 71.3  C
## 3        1      3 female    26     0     0  7.92 S
## 4        1      1 female    35     1     0 53.1  S
## 5        0      3 male      35     0     0  8.05 S
## 6        0      3 male      NA     0     0  8.46 Q
```

```
# now we call the data in csv format
library(readr)
df_1<-read.csv("data/iris.csv")   # we call the iris data in csv format.
head(df_1)                        # we call the first few rows of data set.
```

```
##   X Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1 1          5.1         3.5          1.4         0.2  setosa
## 2 2          4.9         3.0          1.4         0.2  setosa
## 3 3          4.7         3.2          1.3         0.2  setosa
## 4 4          4.6         3.1          1.5         0.2  setosa
## 5 5          5.0         3.6          1.4         0.2  setosa
## 6 6          5.4         3.9          1.7         0.4  setosa
```

now we see how to save the builtin data set in xlsx , csv and tsv format.

Data is the collection of row facts and figures ,when the data is arrange in specific format become information.In dataset we see the observations of same characteristics are displayed in a single colonmns and we have data of form the one type of characteristics are arrange in more than one colomns become a dataset of multiple colomns.

Data set must be save in many format but i highlight only three types that are xlsx (excel format),csv (coma seperated values) and tsv (tab seperated values) format.

```r
# first we call the built in dataset in R.
data()

# now we see the air quality dataset that is built in dataset in R.
df_2<-airquality
head(df_2)
```

```
##   Ozone Solar.R Wind Temp Month Day
## 1    41     190  7.4   67     5   1
## 2    36     118  8.0   72     5   2
## 3    12     149 12.6   74     5   3
## 4    18     313 11.5   62     5   4
## 5    NA      NA 14.3   56     5   5
## 6    28      NA 14.9   66     5   6
```

```r
# now save the dataset in excel format
library(writexl)
write_xlsx(df_2,"Data/ariquality.xlsx")
```

```r
# now save in csv format
write.csv(df_2,"Data/airquality.csv")
```

```r
# now save the file in tsv format
library(readr)
write_tsv(df_2,"Data/airquality.tsv")
```

Missing Values in R

Missing values in R are represented by `NA`, which stands for "Not Available." These values can occur due to various reasons such as data collection issues, data entry errors, or incomplete records. Handling missing values is crucial for accurate data analysis and modeling.

Identifying Missing Values

To identify missing values in a data frame, you can use the `is.na()` function. This function returns a logical matrix indicating which elements are missing (`TRUE`) and which are not (`FALSE`).

```r
# Create a sample data frame with missing values
df <- data.frame(A = c(1, 2, NA, 4), B = c("a", NA, "c", "d"), C = c(TRUE, FALSE, TRUE, NA))

# Find missing values in the data frame
is.na(df)
```

```
##          A     B     C
## [1,] FALSE FALSE FALSE
## [2,] FALSE  TRUE FALSE
## [3,]  TRUE FALSE FALSE
## [4,] FALSE FALSE  TRUE
```

Counting Missing Values

You can count the total number of missing values in a data frame, in each column, or in a vector using the `sum()` function in combination with `is.na()`.

```
# Count the total number of missing values in the data frame
sum(is.na(df))
```

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

Count Missing Values in Each Column:

```
# Count missing values in each column using sapply()
sapply(df, function(x) sum(is.na(x)))
```

```
## A B C
## 1 1 1
```

```
# Or using colSums()
colSums(is.na(df))
```

```
## A B C
## 1 1 1
```

Count Missing Values in a Vector:

```
# Create a sample vector with missing values
vec <- c(1, NA, 3, NA, 5)

# Count missing values in the vector
sum(is.na(vec))
```

```
## [1] 2
```

Identifying Rows with Missing Values

To identify rows in a data frame that contain missing values, you can use the `complete.cases()` function. This function returns a logical vector indicating which rows have complete data (`TRUE`) and which rows have missing values (`FALSE`).

```
# Identify rows with missing values
complete.cases(df)
```

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

Handling Missing Values

Once you've identified the missing values, you can handle them in various ways, such as removing rows with missing values, replacing them with a specific value, or imputing them based on other data.

1. Remove Rows with Missing Values:

```
# Remove rows with any missing values
df_clean <- df[complete.cases(df), ]
```

2. Replace Missing Values with a Specific Value:

```
# Replace NA values with 0
df[is.na(df)] <- 0
df
```

```
##   A B C
## 1 1 a 1
## 2 2 0 0
## 3 0 c 1
## 4 4 d 0
```

Impute Missing Values:

You can use packages like `mice` or `missForest` for more advanced imputation methods.To replace missing values using the `tidyverse` package in R, you can use the `replace_na()` function from the `tidyr` package. This function allows you to replace `NA` values with specified values for each column in a data frame.

Overview of `replace_na()`

The `replace_na()` function is used to replace `NA` values in a data frame or vector with specified values. If the input is a data frame, you can provide a named list of values to replace `NA` values in each column. If the input is a vector, you can provide a single value to replace all `NA` values in the vector.

Syntax

replace_na(data, replace)

1. `data`: A data frame or vector containing `NA` values.
2. `replace`: A named list of values for each column (if `data` is a data frame) or a single value (if `data` is a vector) to replace `NA` values.

Replacing `NA` Values in a Data Frame

```r
# Load the tidyverse package
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v purrr     1.0.2
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.1     v tibble    3.2.1
## v lubridate 1.9.4     v tidyr     1.3.1
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```r
# Create a sample data frame with missing values
df <- tibble(
  A = c(1, 2, NA, 4),
  B = c("a", NA, "c", "d"),
  C = c(TRUE, FALSE, TRUE, NA)
)

# Replace NA values in the data frame
df_clean <- df %>%
  replace_na(list(
    A = 0,          # Replace NA in column A with 0
    B = "unknown",  # Replace NA in column B with "unknown"
    C = FALSE       # Replace NA in column C with FALSE
  ))

# Print the cleaned data frame
print(df_clean)
```

```
## # A tibble: 4 x 3
##       A B       C
##   <dbl> <chr>   <lgl>
## 1     1 a       TRUE
## 2     2 unknown FALSE
## 3     0 c       TRUE
## 4     4 d       FALSE
```

Replacing NA Values in a Vector

```r
# Create a sample vector with missing values
vec <- c(1, NA, 3, NA, 5)

# Replace NA values in the vector with 0
vec_clean <- replace_na(vec, 0)

# Print the cleaned vector
print(vec_clean)
```

```
## [1] 1 0 3 0 5
```

Additional Functions for Handling Missing Values

1. `dplyr::na_if()`: Replaces specified values with `NA`.
2. `dplyr::coalesce()`: Replaces `NA` values with values from other vectors.

Example Using `dplyr::coalesce()`

```
# Create two vectors
vec1 <- c(1, NA, 3, NA, 5)
vec2 <- c(NA, 2, NA, 4, NA)

# Replace NA values in vec1 with corresponding values from vec2
vec_combined <- dplyr::coalesce(vec1, vec2)
# Print the combined vector
print(vec_combined)
```

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

Using Piping Operator

Using the piping operator(%>%) in R,we can chain together multiple operations in a clear and readable manner.This is particularly useful when working with the `tidyverse` package.Here's how you can replace missing values using the piping operator.

Replacing `NA` Values in a Data Frame

Let's start with a sample data frame that contains missing values:

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

# Create a sample data frame with missing values
df <- tibble(
  A = c(1, 2, NA, 4),
  B = c("a", NA, "c", "d"),
  C = c(TRUE, FALSE, TRUE, NA)
)

# Replace NA values in the data frame using the piping operator
df_clean <- df %>%
  replace_na(list(
    A = 0,          # Replace NA in column A with 0
    B = "unknown",  # Replace NA in column B with "unknown"
    C = FALSE       # Replace NA in column C with FALSE
  ))

# Print the cleaned data frame
print(df_clean)
```

```
## # A tibble: 4 x 3
##       A B       C
##   <dbl> <chr>   <lgl>
## 1     1 a       TRUE
## 2     2 unknown FALSE
## 3     0 c       TRUE
## 4     4 d       FALSE
```

In this example: 1. We load the `tidyverse` package. 2. We create a sample data frame `df` with missing values. 3. Using the piping operator `%>%`, we chain the `replace_na()` function to replace `NA` values in each column with specified values.

Replacing `NA` Values in a Vector

You can also use the piping operator to replace `NA` values in a vector:

```
# Create a sample vector with missing values
vec <- c(1, NA, 3, NA, 5)

# Replace NA values in the vector with 0 using the piping operator
vec_clean <- vec %>%
  replace_na(0)

# Print the cleaned vector
print(vec_clean)
```

```
## [1] 1 0 3 0 5
```

In this example: 1. We create a sample vector `vec` with missing values. 2. Using the piping operator `%>%`, we chain the `replace_na()` function to replace `NA` values with `0`.

Additional Example Using `dplyr::na_if()` and `dplyr::coalesce()`

You can also use `dplyr::na_if()` to replace specific values with `NA` and then use `dplyr::coalesce()` to replace `NA` values with desired replacements:

```
# Create a sample vector with missing values
vec1 <- c(1, NA, 3, NA, 5)

# Use na_if() to replace 2 with NA (for demonstration purposes)
vec_combined_1 <- vec1 %>%
  dplyr::na_if(2)

# Replace NA values in vec_combined_1 with 0 using coalesce()
vec_combined_1 <- vec_combined_1 %>%
  dplyr::coalesce(0)

# Print the cleaned vector
print(vec_combined_1)
```

```
## [1] 1 0 3 0 5
```

Sorting,Selecting,Filtering and Mutating the Data on builtin data in R

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

1. Selecting Columns

The `select()` function is used to choose specific columns from a data frame.

```
# Select specific columns from the mtcars dataset
selected_data <- mtcars %>%
  select(mpg, cyl, hp)

# Print the selected data
print(head(selected_data))
```

```
##                    mpg cyl  hp
## Mazda RX4         21.0   6 110
## Mazda RX4 Wag     21.0   6 110
## Datsun 710        22.8   4  93
## Hornet 4 Drive    21.4   6 110
## Hornet Sportabout 18.7   8 175
## Valiant           18.1   6 105
```

2. Filtering Rows

The `filter()` function is used to filter rows based on specific conditions.

```
# Filter rows where mpg is greater than 20
filtered_data <- mtcars[,1:8] %>%
  filter(mpg > 20)

# Print the filtered data
print(head(filtered_data))
```

```
##                mpg cyl  disp  hp drat    wt  qsec vs
## Mazda RX4      21.0   6 160.0 110 3.90 2.620 16.46  0
## Mazda RX4 Wag  21.0   6 160.0 110 3.90 2.875 17.02  0
## Datsun 710     22.8   4 108.0  93 3.85 2.320 18.61  1
## Hornet 4 Drive 21.4   6 258.0 110 3.08 3.215 19.44  1
## Merc 240D      24.4   4 146.7  62 3.69 3.190 20.00  1
## Merc 230       22.8   4 140.8  95 3.92 3.150 22.90  1
```

3. Mutating (Adding or Modifying Columns)

The `mutate()` function is used to add new columns or modify existing ones.

```r
# Add a new column 'hp_per_cyl' which is horsepower per cylinder
mutated_data <- mtcars %>%
  mutate(hp_per_cyl = hp / cyl)

# Print the mutated data
mutated_data_1<-mutated_data%>%
  select(cyl,hp,hp_per_cyl)
head(mutated_data_1)
```

```
##                   cyl  hp hp_per_cyl
## Mazda RX4           6 110   18.33333
## Mazda RX4 Wag       6 110   18.33333
## Datsun 710          4  93   23.25000
## Hornet 4 Drive      6 110   18.33333
## Hornet Sportabout   8 175   21.87500
## Valiant             6 105   17.50000
```

4. Sorting (Arranging Rows)

The `arrange()` function is used to sort rows based on one or more columns.

```r
# Sort the data by mpg in ascending order
sorted_data <- mtcars[,1:7] %>%
  arrange(mpg)

# Print the sorted data
print(head(sorted_data))
```

```
##                      mpg cyl disp  hp drat    wt  qsec
## Cadillac Fleetwood  10.4   8  472 205 2.93 5.250 17.98
## Lincoln Continental 10.4   8  460 215 3.00 5.424 17.82
## Camaro Z28          13.3   8  350 245 3.73 3.840 15.41
## Duster 360          14.3   8  360 245 3.21 3.570 15.84
## Chrysler Imperial   14.7   8  440 230 3.23 5.345 17.42
## Maserati Bora       15.0   8  301 335 3.54 3.570 14.60
```

```r
# Sort the data by mpg in descending order
sorted_data_desc <- mtcars[,1:7] %>%
  arrange(desc(mpg))

# Print the sorted data in descending order
print(head(sorted_data_desc))
```

```
##                 mpg cyl  disp  hp drat    wt  qsec
## Toyota Corolla 33.9   4  71.1  65 4.22 1.835 19.90
## Fiat 128       32.4   4  78.7  66 4.08 2.200 19.47
## Honda Civic    30.4   4  75.7  52 4.93 1.615 18.52
## Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.90
## Fiat X1-9      27.3   4  79.0  66 4.08 1.935 18.90
## Porsche 914-2  26.0   4 120.3  91 4.43 2.140 16.70
```