R Cheat Sheet

Nicholas Piccaro

4/18/23

Contents

What is R

R is an open-source programming language and environment for statistical computing and graphics. Widely used in data analysis, statistical modeling, and visualization, R offers a rich ecosystem of packages and has a large community of users and developers. Known for its flexibility, extensibility, and optimized syntax for data analysis, R is popular in academia, industry, and research for various applications such as data science, bioinformatics, finance, and social sciences.

R is a programming langage for statistics that uses packages to create an easy and powerful framework

Different Data Types in R

Data types are the foundation of most coding languages. ##Base R Data Types:

```
# Different data types in R
x <- 5  # Numeric
y <- "hello"  # Character
z <- TRUE  # Logical
a <- c(1, 2, 3)  # Vector
b <- list(1, "two", TRUE)  # List
c <- data.frame(x = 1:3, y = c("A", "B", "C"))  # Data frame

# Print the data types
cat("x| Numeric:", class(x), "\n")
```

```
## x| Numeric: numeric
```

```
cat("y| Character:", class(y), "\n")
```

```
## y| Character: character
```

```
cat("z| Logical:", class(z), "\n")
```

```
## z| Logical: logical
```

```
cat("a| Vector:", class(a), "\n")
```

```
## a| Vector: numeric
```

```
cat("b| List:", class(b), "\n")
```

```
## b| List: list
```

```
cat("c| Data frame:", class(c), "\n")
```

```
## c| Data frame: data.frame
```

Numeric Type Operations

What can we do with Numeric Data Types ### Basic Operations

```
var1<-11 #A comment is a # in a code block this code wont run you can use it for notes
var2<-16
#above we assigned two variables var1 and var2 both of these are type numeric
```

addition

```r
addV1V2 <- var1 + var2 # here we assign a new variable addV1V2 to the addition of var1 and var2
cat("var1+var2 = ", addV1V2, "\n") #we need to add ,"\n" so it prints on a newline
```

```
## var1+var2 =  27
```

subtraction

```r
subV1V2 <- var1 - var2 # here we assign a new variable subV1V2 to the subtration of var2 from var1
cat("var1-var2 = ", subV1V2, "\n")
```

```
## var1-var2 =  -5
```

multiplication

```r
mulV1V2 <- var1 * var2 # here we assign a new variable mulV1V2 to the product of var2 and var1
cat("var1*var2 = ", mulV1V2, "\n")
```

```
## var1*var2 =  176
```

division

```r
divV1V2 <- var1 / var2 # here we assign a new variable divV1V2 to the division of var1 by var2
cat("var1/var2 = ", divV1V2, "\n")
```

```
## var1/var2 =  0.6875
```

Exponentiation

```r
base <- 2
exponent <- 3
result <- base^exponent
print(result) # Output: 8
```

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

Modulus

```r
dividend <- 10
divisor <- 3
remainder <- dividend %% divisor
print(remainder) # Output: 1
```

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

Comparison

```
a <- 5
b <- 10
less_than <- a < b
print(less_than) # Output: TRUE
```

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

```
greater_than_equal_to <- a >= b
print(greater_than_equal_to) # Output: FALSE
```

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

```
equality <- a == b
print(equality) # Output: FALSE
```

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

Mathematical Functions

```
x <- 9
square_root <- sqrt(x)
print(square_root) # Output: 3
```

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

```
absolute_value <- abs(-5)
print(absolute_value) # Output: 5
```

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

```
sine_value <- sin(pi/2)
print(sine_value) # Output: 1
```

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

```
natural_logarithm <- log(10)
print(natural_logarithm) # Output: 2.302585
```

```
## [1] 2.302585
```

Character Type Operations

Concatenation

```r
# Concatenation using paste()
first_name <- "John"
last_name <- "Doe"
full_name <- paste(first_name, last_name)
print(full_name) # Output: "John Doe"
```

```
## [1] "John Doe"
```

```r
# Concatenation using c()
greeting <- "Hello"
audience <- "world"
message <- c(greeting, audience)
print(message) # Output: "Hello" "world"
```

```
## [1] "Hello" "world"
```

Substring Extraction

```r
# Substring Extraction using substr()
text <- "Hello world"
substring <- substr(text, 1, 5)
print(substring) # Output: "Hello"
```

```
## [1] "Hello"
```

```r
# Substring Extraction using []
substring <- text[7:11]
print(substring) # Output: "world"
```

```
## [1] NA NA NA NA NA
```

String Manipulation

```r
# String Manipulation
text <- "   R programming is fun!   "
length <- nchar(text)
print(length) # Output: 27
```

```
## [1] 27
```

```r
upper_text <- toupper(text)
print(upper_text) # Output: "   R PROGRAMMING IS FUN!   "
```

```
## [1] "   R PROGRAMMING IS FUN!   "
```

```r
lower_text <- tolower(text)
print(lower_text) # Output: "   r programming is fun!   "
```

```
## [1] "   r programming is fun!   "
```

```r
trimmed_text <- trimws(text)
print(trimmed_text) # Output: "R programming is fun!"
```

```
## [1] "R programming is fun!"
```

Comparison Operators

```r
# Comparison
text1 <- "apple"
text2 <- "banana"
less_than <- text1 < text2
print(less_than) # Output: TRUE
```

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

```r
equality <- text1 == text2
print(equality) # Output: FALSE
```

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

Regular Expressions

```r
# Regular Expressions
text <- "The quick brown fox jumps over the lazy dog"
contains_fox <- grepl("fox", text)
print(contains_fox) # Output: TRUE
```

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

```r
replaced_text <- gsub("fox", "cat", text)
print(replaced_text) # Output: "The quick brown cat jumps over the lazy dog"
```

```
## [1] "The quick brown cat jumps over the lazy dog"
```

Boolean or Logical Type Operations

Logical Operators

```r
# Logical Operators
x <- TRUE
y <- FALSE

# AND
result1 <- x & y
print(result1) # Output: FALSE
```

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

```r
# OR
result2 <- x | y
print(result2) # Output: TRUE
```

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

```r
# NOT
result3 <- !x
print(result3) # Output: FALSE
```

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

```r
# Exclusive OR
result4 <- xor(x, y)
print(result4) # Output: TRUE
```

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

Comparison Operators

```r
# Comparison Operators
a <- TRUE
b <- FALSE

equality <- a == b
print(equality) # Output: FALSE
```

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

```r
inequality <- a != b
print(inequality) # Output: TRUE
```

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

Conditional Statments

```r
# Conditional Statements
age <- 25
is_adult <- age >= 18

if (is_adult) {
  print("You are an adult.")
} else {
  print("You are not an adult.")
}
```

```
## [1] "You are an adult."
```

```r
# Using ifelse()
result <- ifelse(is_adult, "You are an adult.", "You are not an adult.")
print(result) # Output: "You are an adult."
```

```
## [1] "You are an adult."
```

Logical Functions

```r
# Logical Functions
x <- TRUE
y <- FALSE

is_logical <- is.logical(x)
print(is_logical) # Output: TRUE
```

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

```r
is_true <- isTRUE(x)
print(is_true) # Output: TRUE
```

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

```r
any_true <- any(x, y)
print(any_true) # Output: TRUE
```

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

```r
all_true <- all(x, y)
print(all_true) # Output: FALSE
```

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

Vector Operations

Vector Creation

```r
# Creating a numeric vector
numeric_vector <- c(1.2, 2.3, 3.4, 4.5)
cat("numeric vector: ", numeric_vector ,"\n")
```

```
## numeric vector:  1.2 2.3 3.4 4.5
```

```r
# Creating an integer vector
integer_vector <- c(1L, 2L, 3L, 4L)
cat("integer vector: ", integer_vector ,"\n")
```

```
## integer vector:  1 2 3 4
```

```r
# Creating a character vector
character_vector <- c("a", "b", "c", "d")
cat("character vector: ", character_vector ,"\n")
```

```
## character vector:  a b c d
```

Generate Vector Data

```r
# Create a vector of even numbers from 2 to 20
numbers2 <- seq(from = 2, to = 20, by = 2)
print(numbers2)
```

```
##  [1]  2  4  6  8 10 12 14 16 18 20
```

```r
# Create a vector of random numbers between 0 and 1
numbers3 <- runif(n = 10)
print(numbers3)
```

```
##  [1] 0.80640512 0.68207079 0.57277017 0.86398585 0.90297717 0.76692067
##  [7] 0.96672134 0.24865691 0.15125953 0.08398508
```

Vector Indexing

```r
# Accessing the first element of a vector
first_element <- numeric_vector[1]
cat("first element: ", first_element ,"\n")
```

```
## first element:  1.2
```

```r
# Accessing a range of elements
range_of_elements <- integer_vector[2:4]
cat("range_of_elements: ", range_of_elements,"\n")
```

```
## range_of_elements:  2 3 4
```

```r
# Accessing multiple elements using a vector of indices
multiple_elements <- character_vector[c(2, 4)]
cat("multiple_elements: ", multiple_elements ,"\n")
```

```
## multiple_elements:  b d
```

Vector Arithmetic

```r
# Vector addition
sum_vector <- numeric_vector + integer_vector
cat("sum_vector: ", sum_vector ,"\n")
```

```
## sum_vector:  2.2 4.3 6.4 8.5
```

```r
# Vector subtraction
diff_vector <- numeric_vector - integer_vector
cat("diff_vector: ", diff_vector,"\n")
```

```
## diff_vector:  0.2 0.3 0.4 0.5
```

```r
# Vector multiplication
prod_vector <- numeric_vector * integer_vector
cat("prod vector: ", prod_vector,"\n")
```

```
## prod vector:  1.2 4.6 10.2 18
```

```r
# Vector division
quot_vector <- numeric_vector / integer_vector
cat("quot_vector: ", quot_vector,"\n")
```

```
## quot_vector:  1.2 1.15 1.133333 1.125
```

Vector Functions

```r
# Calculating the length of a vector
length_vector <- length(numeric_vector)
cat("length of vector: ", length_vector ,"\n")
```

```
## length of vector:  4
```

```r
# Finding the maximum value in a vector
max_value <- max(numeric_vector)
cat("max value of vector: ", max_value,"\n")
```

```
## max value of vector:  4.5
```

```r
# Calculating the sum of all elements in a vector
sum_of_elements <- sum(numeric_vector)
cat("fsum of elements: ", sum_of_elements,"\n")
```

```
## fsum of elements:  11.4
```

```r
# Applying a function to each element of a vector
sqrt_vector <- sqrt(numeric_vector)
cat("sqrt of vector: ", sqrt_vector ,"\n")
```

```
## sqrt of vector:  1.095445 1.516575 1.843909 2.12132
```

Vector Comparison

```r
# Comparing two vectors element-wise
comparison_vector <- numeric_vector > integer_vector
cat("comparison vector: ", comparison_vector ,"\n")
```

```
## comparison vector:  TRUE TRUE TRUE TRUE
```

```r
# Checking if all elements of a vector are equal
all_equal <- all(numeric_vector == integer_vector)
cat("all equal: ", all_equal,"\n")
```

```
## all equal:  FALSE
```

```r
# Checking if any element of a vector satisfies a condition
any_satisfy <- any(numeric_vector > 3)
cat("satisfy condition: ", any_satisfy ,"\n")
```

```
## satisfy condition:  TRUE
```

List Operations

List Creation

```r
# Creating a list with named elements
my_list <- list(a = 1, b = "hello", c = c(1.2, 2.3, 3.4))

# Creating a list with unnamed elements
my_list <- list(1, "hello", c(1.2, 2.3, 3.4))
```

List Indexing

```r
# Accessing a list element using single brackets
list_element <- my_list["a"]

# Accessing a list element using double brackets
list_element <- my_list[["a"]]
```

List Manipulation

```r
# Adding an element to a list
my_list$c <- 3.14

# Removing an element from a list
my_list$d <- NULL

# Modifying an element in a list
my_list$a <- 10
```

List Functions

```r
# Checking the length of a list
list_length <- length(my_list)

# Checking the names of list elements
list_names <- names(my_list)
```

List Unlisting

```r
# Converting a list to a vector
my_vector <- unlist(my_list)
```

List iteration

```r
# Iterating over list elements with a for loop
for (i in 1:length(my_list)) {
  print(my_list[[i]])
}
```

```
## [1] 1
## [1] "hello"
## [1] 1.2 2.3 3.4
## [1] 3.14
## [1] 10
```

```r
# Iterating over list elements with lapply()
lapply(my_list, function(x) print(x))
```

```
## [1] 1
## [1] "hello"
## [1] 1.2 2.3 3.4
## [1] 3.14
## [1] 10
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] "hello"
##
## [[3]]
## [1] 1.2 2.3 3.4
##
## $c
## [1] 3.14
##
## $a
## [1] 10
```

Data Frame Operations

Creating a Data Frame

```r
# Creating a data frame
df <- data.frame(
  Name = c("John", "Jane", "Alice", "Bob"),
  Age = c(25, 30, 22, 35),
  Height = c(6.1, 5.5, 5.9, 5.7),
  Married = c(FALSE, TRUE, FALSE, TRUE),
  Weight = c(180, 150, 140, 160),
  stringsAsFactors = FALSE
)

# Printing the entire data frame
print(df)
```

```
##    Name Age Height Married Weight
## 1  John  25    6.1   FALSE    180
## 2  Jane  30    5.5    TRUE    150
## 3 Alice  22    5.9   FALSE    140
## 4   Bob  35    5.7    TRUE    160
```

Accessing Column from DataFrame

```r
# Access the age column of the data frame
df_age <- df$Age
print(df_age)
```

```
## [1] 25 30 22 35
```

Getting Frequency of Categorical Variables in DataFrame

```r
# Use the table() function to get the frequency of each true and false
marry_freq <- table(df$Married)

# Print the frequency table
print(marry_freq)
```

```
##
## FALSE  TRUE
##     2     2
```

Subsetting DataFrame

```r
# Subset data frame to get rows where Age is greater than 25
df_subset <- df[df$Age > 25, ]
print(df_subset)
```

```
##   Name Age Height Married Weight
## 2 Jane  30    5.5    TRUE    150
## 4  Bob  35    5.7    TRUE    160
```

Adding New Columns

```r
# Add a new column "BMI" calculated from Height and Age
df$BMI <- df$Weight / (df$Height^2)
print(df)
```

```
##    Name Age Height Married Weight      BMI
## 1  John  25    6.1   FALSE    180 4.837409
## 2  Jane  30    5.5    TRUE    150 4.958678
## 3 Alice  22    5.9   FALSE    140 4.021833
## 4   Bob  35    5.7    TRUE    160 4.924592
```

Changing Data Types

```r
# Convert Age column to a factor
df$Age <- as.factor(df$Age)
print(df)
```

```
##    Name Age Height Married Weight      BMI
## 1  John  25    6.1   FALSE    180 4.837409
## 2  Jane  30    5.5    TRUE    150 4.958678
## 3 Alice  22    5.9   FALSE    140 4.021833
## 4   Bob  35    5.7    TRUE    160 4.924592
```

Sorting Data

```
# Sort data frame by Age in ascending order
df <- df[order(df$Age), ]
print(df)
```

```
##    Name Age Height Married Weight      BMI
## 3 Alice  22    5.9   FALSE    140 4.021833
## 1  John  25    6.1   FALSE    180 4.837409
## 2  Jane  30    5.5    TRUE    150 4.958678
## 4   Bob  35    5.7    TRUE    160 4.924592
```

Aggregating Data

```
# Calculate mean Height and Age by Married status
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
df_summary <- df %>%
  group_by(Married) %>%
  summarize(mean_Height = mean(Height), mean_Age = mean(Age))
```

```
## Warning: There were 2 warnings in `summarize()`.
## The first warning was:
## i In argument: `mean_Age = mean(Age)`.
## i In group 1: `Married = FALSE`.
## Caused by warning in `mean.default()`:
## ! argument is not numeric or logical: returning NA
## i Run `dplyr::last_dplyr_warnings()` to see the 1 remaining warning.
```

```
print(df_summary)
```

```
## # A tibble: 2 x 3
##   Married mean_Height mean_Age
##   <lgl>         <dbl>    <dbl>
## 1 FALSE           6         NA
## 2 TRUE            5.6       NA
```

Matrices

Creating a Matrix

```
# Creating a matrix
mat <- matrix(c(1, 2, 3, 4, 5, 6), nrow = 2, ncol = 3)
print(mat)
```

```
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

Matrix Arithmetic

```
# Matrix addition
mat1 <- matrix(c(1, 2, 3, 4, 5, 6), nrow = 2, ncol = 3)
mat2 <- matrix(c(6, 5, 4, 3, 2, 1), nrow = 2, ncol = 3)
mat_sum <- mat1 + mat2
print(mat_sum)
```

```
##      [,1] [,2] [,3]
## [1,]    7    7    7
## [2,]    7    7    7
```

```
# Matrix multiplication
mat_prod <- mat1 %*% t(mat2)  # Note: %*% is used for matrix multiplication, t() is used for transposit
print(mat_prod)
```

```
##      [,1] [,2]
## [1,]   28   19
## [2,]   40   28
```

Matrix Indexing

```
# Accessing elements of a matrix
element <- mat[1, 2]  # Access element in the first row and second column
print(element)
```

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

```
# Accessing rows and columns of a matrix
row <- mat[1, ]  # Access the first row
col <- mat[, 2]  # Access the second column
print(row)
```

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

```
print(col)
```

```
## [1] 3 4
```

Matrix Manipulation

```
# Combining matrices
mat1 <- matrix(c(1, 2, 3, 4, 5, 6), nrow = 2, ncol = 3)
mat2 <- matrix(c(6, 5, 4, 3, 2, 1), nrow = 2, ncol = 3)
mat_combined <- cbind(mat1, mat2)  # Combine matrices by column
print(mat_combined)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    1    3    5    6    4    2
## [2,]    2    4    6    5    3    1
```

```
# Extracting diagonal elements
diag_elements <- diag(mat1)  # Extract diagonal elements of a matrix
print(diag_elements)
```

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

Reading in CSV type data

Using the Camera Data set

```
# Read a CSV file with additional parameters
cameraData <- read.csv("camera_dataset.csv", header = TRUE, sep = ",", encoding = "UTF-8", na.strings =

head(cameraData)
```

```
##                   Model Release.date Max.resolution Low.resolution
## 1       Agfa ePhoto 1280         1997           1024            640
## 2       Agfa ePhoto 1680         1998           1280            640
## 3       Agfa ePhoto CL18         2000            640              0
## 4       Agfa ePhoto CL30         1999           1152            640
## 5 Agfa ePhoto CL30 Clik!         1999           1152            640
```

```
## 6       Agfa ePhoto CL45         2001           1600            640
##   Effective.pixels Zoom.wide..W. Zoom.tele..T. Normal.focus.range
## 1                0            38           114                 70
## 2                1            38           114                 50
## 3                0            45            45                  0
## 4                0            35            35                  0
## 5                0            43            43                 50
## 6                1            51            51                 50
##   Macro.focus.range Storage.included Weight..inc..batteries. Dimensions Price
## 1                40                4                     420         95   179
## 2                 0                4                     420        158   179
## 3                 0                2                       0          0   179
## 4                 0                4                       0          0   269
## 5                 0               40                     300        128  1299
## 6                20                8                     270        119   179
```

Data Exploration

Structure of Data

We see that this data has 13 variables: Model, release date, max resolution, low resolution, effective pixels, zoom wide, zoom tele, normal focus, macro focus, storage included, weight, dimensions, and price

```
str(cameraData)
```

```
## 'data.frame':    1038 obs. of  13 variables:
##  $ Model                  : chr  "Agfa ePhoto 1280" "Agfa ePhoto 1680" "Agfa ePhoto CL18" "Agfa ePho
##  $ Release.date           : int  1997 1998 2000 1999 1999 2001 1999 1997 1996 2001 ...
##  $ Max.resolution         : num  1024 1280 640 1152 1152 ...
##  $ Low.resolution         : num  640 640 0 640 640 ...
##  $ Effective.pixels       : num  0 1 0 0 0 1 1 0 0 1 ...
##  $ Zoom.wide..W.          : num  38 38 45 35 43 51 34 42 50 35 ...
##  $ Zoom.tele..T.          : num  114 114 45 35 43 51 102 42 50 105 ...
##  $ Normal.focus.range     : num  70 50 0 0 50 50 0 70 40 76 ...
##  $ Macro.focus.range      : num  40 0 0 0 0 20 0 3 10 16 ...
##  $ Storage.included       : num  4 4 2 4 40 8 8 2 1 8 ...
##  $ Weight..inc..batteries.: num  420 420 0 0 300 270 0 320 460 375 ...
##  $ Dimensions             : num  95 158 0 0 128 119 0 93 160 110 ...
##  $ Price                  : num  179 179 179 269 1299 ...
```

Summarize Data

This displays the quartile data of all the columns of the Data frame

```
summary(cameraData)
```

```
##     Model            Release.date  Max.resolution Low.resolution
##  Length:1038        Min.   :1994   Min.   :   0   Min.   :   0
##  Class :character   1st Qu.:2002   1st Qu.:2048   1st Qu.:1120
##  Mode  :character   Median :2004   Median :2560   Median :2048
```

```
##                    Mean   :2004   Mean   :2475   Mean   :1774
##                    3rd Qu.:2006   3rd Qu.:3072   3rd Qu.:2560
##                    Max.   :2007   Max.   :5616   Max.   :4992
##
##  Effective.pixels Zoom.wide..W.   Zoom.tele..T.   Normal.focus.range
##  Min.   : 0.000   Min.   : 0.00   Min.   :  0.0   Min.   :  0.00
##  1st Qu.: 3.000   1st Qu.:35.00   1st Qu.: 96.0   1st Qu.: 30.00
##  Median : 4.000   Median :36.00   Median :108.0   Median : 50.00
##  Mean   : 4.596   Mean   :32.96   Mean   :121.5   Mean   : 44.15
##  3rd Qu.: 7.000   3rd Qu.:38.00   3rd Qu.:117.0   3rd Qu.: 60.00
##  Max.   :21.000   Max.   :52.00   Max.   :518.0   Max.   :120.00
##
##  Macro.focus.range Storage.included Weight..inc..batteries.   Dimensions
##  Min.   : 0.000    Min.   :  0.00   Min.   :   0.0          Min.   :  0.0
##  1st Qu.: 3.000    1st Qu.:  8.00   1st Qu.: 180.0          1st Qu.: 92.0
##  Median : 6.000    Median : 16.00   Median : 226.0          Median :101.0
##  Mean   : 7.788    Mean   : 17.45   Mean   : 319.3          Mean   :105.4
##  3rd Qu.:10.000    3rd Qu.: 20.00   3rd Qu.: 350.0          3rd Qu.:115.0
##  Max.   :85.000    Max.   :450.00   Max.   :1860.0          Max.   :240.0
##  NA's   :1         NA's   :2        NA's   :2               NA's   :2
##      Price
##  Min.   :  14.0
##  1st Qu.: 149.0
##  Median : 199.0
##  Mean   : 457.4
##  3rd Qu.: 399.0
##  Max.   :7999.0
##
```

Check first few rows of Data

```
# Check the first few rows of the dataset
head(cameraData)
```

```
##                    Model Release.date Max.resolution Low.resolution
## 1       Agfa ePhoto 1280         1997           1024            640
## 2       Agfa ePhoto 1680         1998           1280            640
## 3       Agfa ePhoto CL18         2000            640              0
## 4       Agfa ePhoto CL30         1999           1152            640
## 5 Agfa ePhoto CL30 Clik!         1999           1152            640
## 6       Agfa ePhoto CL45         2001           1600            640
##   Effective.pixels Zoom.wide..W. Zoom.tele..T. Normal.focus.range
## 1                0            38           114                 70
## 2                1            38           114                 50
## 3                0            45            45                  0
## 4                0            35            35                  0
## 5                0            43            43                 50
## 6                1            51            51                 50
##   Macro.focus.range Storage.included Weight..inc..batteries. Dimensions Price
## 1                40                4                    420         95   179
## 2                 0                4                    420        158   179
## 3                 0                2                      0          0   179
```

```
## 4                 0                 4                     0          0   269
## 5                 0                40                   300        128  1299
## 6                20                 8                   270        119   179
```

Check missing values

we see some of the fields have missing values, its ultimate up to us how we want to handle missing data. We could remove the data, we could just leave it, we could set it to the mean. Generally if the data is big enough removing it is sufficient.

```
sapply(cameraData, function(x) sum(is.na(x)))
```

```
##                   Model            Release.date          Max.resolution
##                       0                       0                       0
##          Low.resolution       Effective.pixels           Zoom.wide..W.
##                       0                       0                       0
##           Zoom.tele..T.      Normal.focus.range       Macro.focus.range
##                       0                       0                       1
##        Storage.included Weight..inc..batteries.              Dimensions
##                       2                       2                       2
##                   Price
##                       0
```

Remove missing values from dataframe

There are several ways to use this

```
camera_clean <- na.omit(cameraData)
```

Renaming Data Frame Column Names

```
# Assign new column names
colnames(camera_clean) <- c('Model', 'ReleaseDate', 'MaxResolution', 'LowResolution', 'EffectivePixels'

head(camera_clean)
```

```
##                    Model ReleaseDate MaxResolution LowResolution
## 1       Agfa ePhoto 1280        1997          1024           640
## 2       Agfa ePhoto 1680        1998          1280           640
## 3       Agfa ePhoto CL18        2000           640             0
## 4       Agfa ePhoto CL30        1999          1152           640
## 5 Agfa ePhoto CL30 Clik!        1999          1152           640
## 6       Agfa ePhoto CL45        2001          1600           640
##   EffectivePixels ZoomWide ZoomTele NormalFocusRange MacroFocusRange Storage
## 1               0       38      114               70              40       4
## 2               1       38      114               50               0       4
## 3               0       45       45                0               0       2
## 4               0       35       35                0               0       4
## 5               0       43       43               50               0      40
```

```
## 6              1       51       51               50             20       8
##   Weight Dimensions Price
## 1    420         95   179
## 2    420        158   179
## 3      0          0   179
## 4      0          0   269
## 5    300        128  1299
## 6    270        119   179
```

Adding New Column to DataFrame

```
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v forcats   1.0.0     v readr     2.1.4
## v ggplot2   3.4.2     v stringr   1.5.0
## v lubridate 1.9.2     v tibble    3.2.1
## v purrr     1.0.1     v tidyr     1.3.0
## -- 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
```

```
# Extract the first word from the 'strings' column
camera_clean <- camera_clean %>% mutate(Brand = str_extract(Model, "\\b\\w+"))

# View the dataframe with the new 'first_word' column
head(camera_clean)
```

```
##                   Model ReleaseDate MaxResolution LowResolution
## 1       Agfa ePhoto 1280        1997          1024           640
## 2       Agfa ePhoto 1680        1998          1280           640
## 3       Agfa ePhoto CL18        2000           640             0
## 4       Agfa ePhoto CL30        1999          1152           640
## 5 Agfa ePhoto CL30 Clik!        1999          1152           640
## 6       Agfa ePhoto CL45        2001          1600           640
##   EffectivePixels ZoomWide ZoomTele NormalFocusRange MacroFocusRange Storage
## 1               0       38      114               70              40       4
## 2               1       38      114               50               0       4
## 3               0       45       45                0               0       2
## 4               0       35       35                0               0       4
## 5               0       43       43               50               0      40
## 6               1       51       51               50              20       8
##   Weight Dimensions Price Brand
## 1    420         95   179  Agfa
## 2    420        158   179  Agfa
## 3      0          0   179  Agfa
## 4      0          0   269  Agfa
## 5    300        128  1299  Agfa
## 6    270        119   179  Agfa
```

Useful Packages

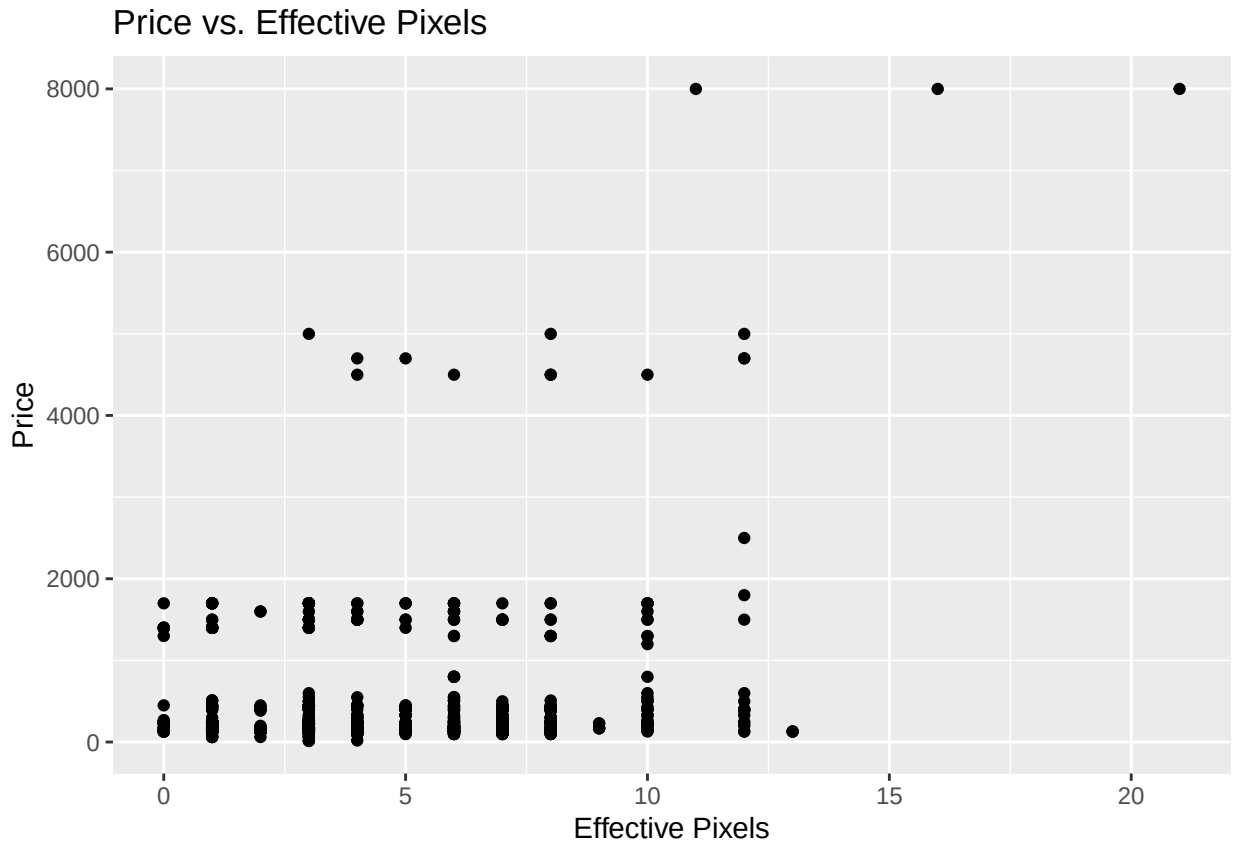
GGPLOT2 for plotting

Importing GGPLOT2

```
library(ggplot2)
```

Scatter Plot

```
ggplot(camera_clean, aes(x = EffectivePixels, y = Price)) +
  geom_point() +
  labs(x = "Effective Pixels", y = "Price") +
  ggtitle("Price vs. Effective Pixels")
```

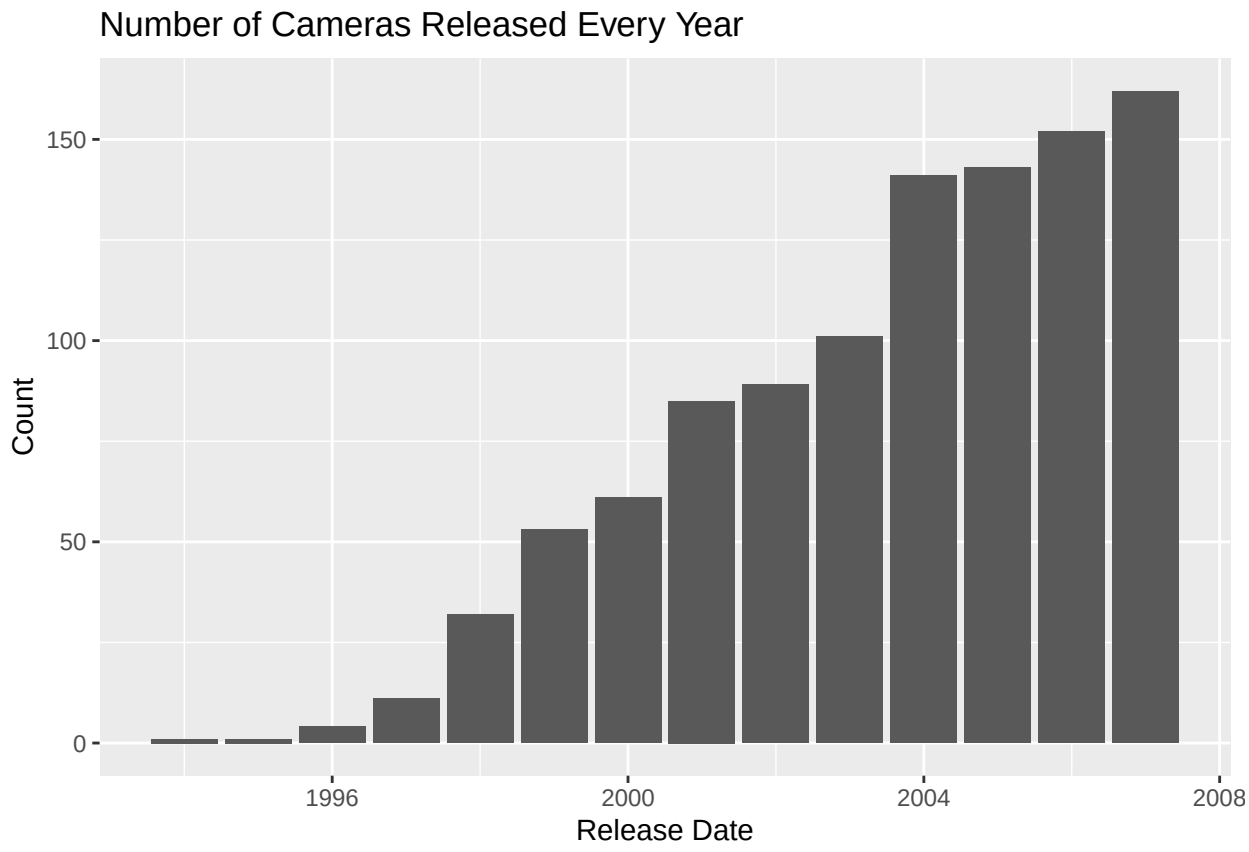


Bar Chart

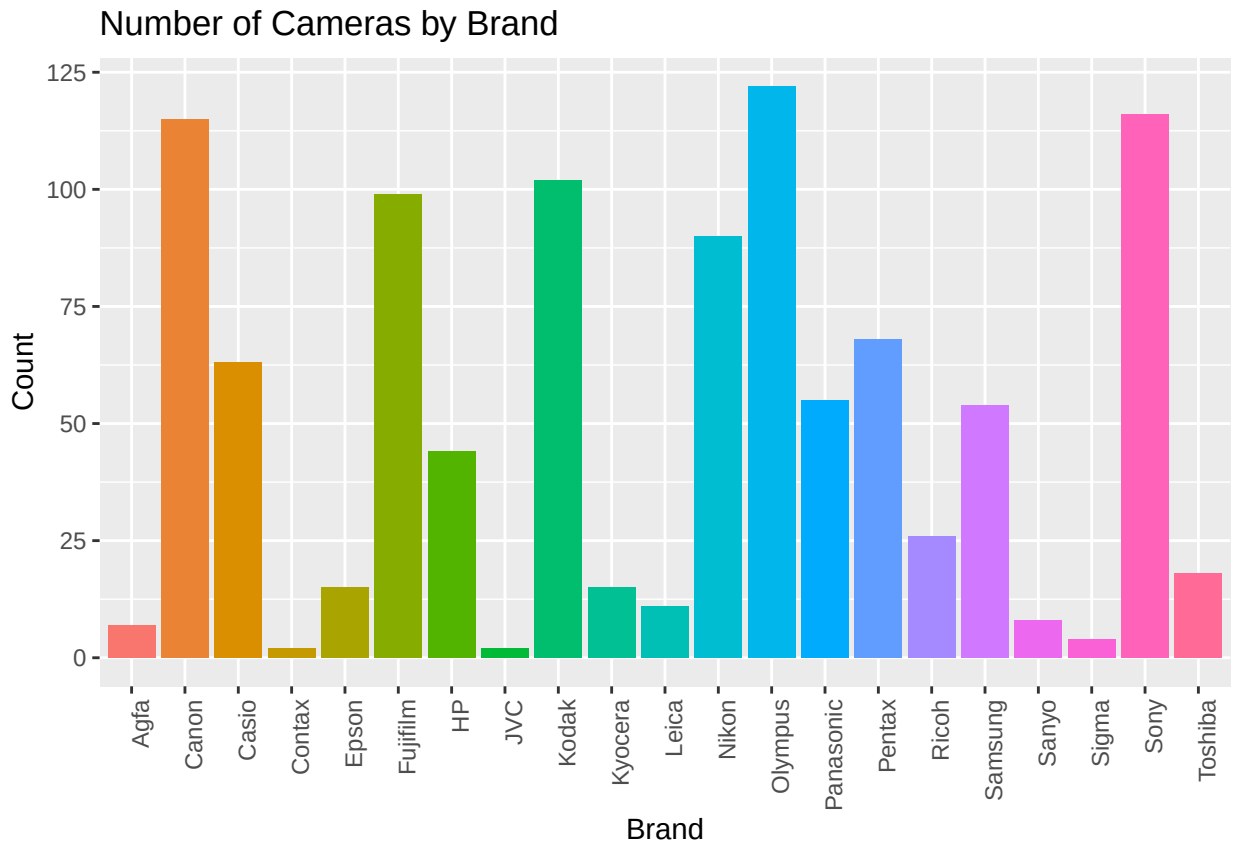
```
# Bar chart of number of cameras by brand
ggplot(camera_clean, aes(x = ReleaseDate, fill = ReleaseDate)) +
  geom_bar() +
```

```
  labs(x = "Release Date", y = "Count") +
  ggtitle("Number of Cameras Released Every Year") +
  theme(legend.position = "none")
```

```
## Warning: The following aesthetics were dropped during statistical transformation: fill
## i This can happen when ggplot fails to infer the correct grouping structure in
##   the data.
## i Did you forget to specify a 'group' aesthetic or to convert a numerical
##   variable into a factor?
```

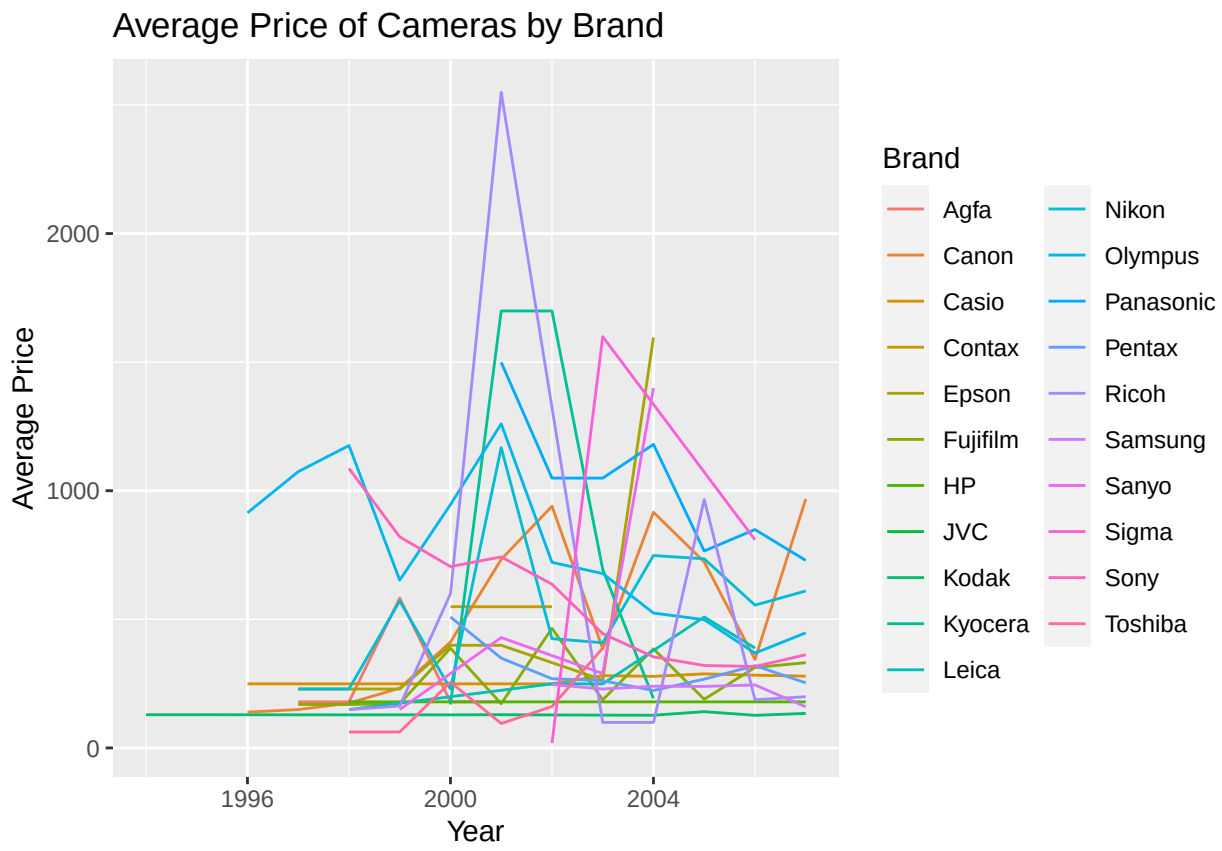



```
# Bar chart of number of cameras by brand
ggplot(camera_clean, aes(x = Brand, fill = Brand)) +
  geom_bar() +
  labs(x = "Brand", y = "Count") +
  ggtitle("Number of Cameras by Brand") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1), legend.position = "none")
```



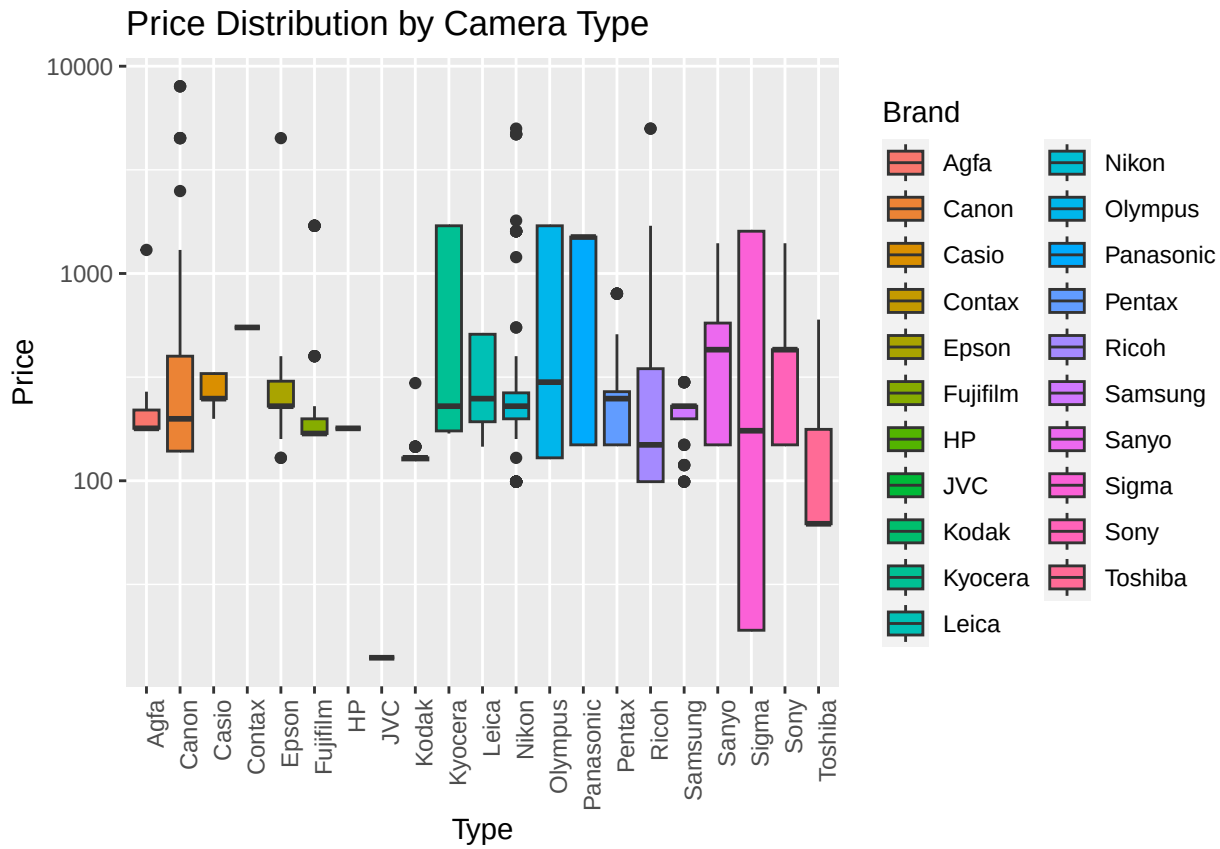
Line Plot

```
ggplot(camera_clean, aes(x = ReleaseDate, y = Price, color = Brand)) +
  geom_line(stat = "summary", fun = "mean") +
  labs(x = "Year", y = "Average Price") +
  ggtitle("Average Price of Cameras by Brand")
```



BoxPlot

```
# Box plot of price by Brand note this has a log applied to the y axis to change the appearance
ggplot(camera_clean, aes(x = Brand, y = Price, fill = Brand)) +
  geom_boxplot() +
  labs(x = "Type", y = "Price") +
  ggtitle("Price Distribution by Camera Type")+
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  scale_y_continuous(trans = "log10")
```



Line of Best Fit through scatter plot

```
# Scatter plot with line of best fit
ggplot(camera_clean, aes(x = MaxResolution, y = Price)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE, color = "red") +  # Add line of best fit
  labs(x = "Max Resolution", y = "Price") +
  ggtitle("Scatter Plot of Price vs. Max Resolution with Line of Best Fit")+
  scale_y_continuous(trans = "log10")
```

```
## `geom_smooth()` using formula = 'y ~ x'
```

DPLYR for data manipulation

Import DPLYR

```
library(dplyr)
```

Filter Data

```
# Filter the data frame to include only rows where the Brand is "Canon"
canon_cameras <- camera_clean %>%
  filter(Brand == "Canon") %>%
  select(Model, Price)

# Display the filtered data frame
head(canon_cameras) #canon_cameras is a new dataframe with only canon cameras and the price
```

```
##                  Model Price
## 1  Canon PowerShot 350   149
## 2  Canon PowerShot 600   139
## 3  Canon PowerShot A10   139
## 4 Canon PowerShot A100   139
```

```
## 5  Canon PowerShot A20   139
## 6 Canon PowerShot A200   139
```

Grouping by specific column names and summarize

```
# Group the data frame by Brand and calculate the average Price for each Brand
brand_avg_price <- camera_clean %>%
  group_by(Brand) %>%
  summarize(AveragePrice = mean(Price, na.rm = TRUE))

# Display the summarized data frame
head(brand_avg_price) # brand_avg_price is a new data frame with the brands and the average price
```

```
## # A tibble: 6 x 2
##   Brand    AveragePrice
##   <chr>           <dbl>
## 1 Agfa             352.
## 2 Canon            682.
## 3 Casio            270.
## 4 Contax           549
## 5 Epson            530.
## 6 Fujifilm         297.
```

Sort and Arrange Data Frame

```
# Sort the data frame by Price in descending order
sorted_cameras <- camera_clean %>%
  arrange(desc(Price))

# Display the sorted data frame
head(sorted_cameras)
```

```
##                   Model ReleaseDate MaxResolution LowResolution
## 1          Canon EOS-1Ds        2002          4064          2032
## 2  Canon EOS-1Ds Mark II        2004          4992          3600
## 3 Canon EOS-1Ds Mark III        2007          5616          4992
## 4               Nikon D3        2007          4256          3184
## 5       Ricoh GR Digital        2005          3264          2592
## 6         Ricoh RDC-i500        2001          2048          1024
##   EffectivePixels ZoomWide ZoomTele NormalFocusRange MacroFocusRange Storage
## 1              11        0        0                0               0       0
## 2              16        0        0                0               0       0
## 3              21        0        0                0               0       0
## 4              12        0        0                0               0       0
## 5               8       28       28               30               2      26
## 6               3       35      105               24               1       8
##   Weight Dimensions Price Brand
## 1   1585        156  7999 Canon
## 2   1565        156  7999 Canon
## 3   1385        150  7999 Canon
```

```
## 4   1300         160  4999 Nikon
## 5    200         107  4999 Ricoh
## 6    320         142  4999 Ricoh
```

TIDYR for cleaning or tidying data

###Load TIDYR

```
library(tidyr)
```

Handle missing values

```
missing_values <- cameraData %>%
  is.na() %>%
  colSums()

# Display the columns with missing values and their corresponding counts
missing_values
```

```
##                   Model            Release.date          Max.resolution
##                       0                       0                       0
##          Low.resolution        Effective.pixels           Zoom.wide..W.
##                       0                       0                       0
##           Zoom.tele..T.      Normal.focus.range       Macro.focus.range
##                       0                       0                       1
##        Storage.included Weight..inc..batteries.              Dimensions
##                       2                       2                       2
##                   Price
##                       0
```

```
# Remove rows with missing values in the Price column
camera_df <- cameraData %>%
  drop_na(Storage.included)

# Fill in missing values in the Megapixels column with the mean Megapixels value
camera_df <- camera_df %>%
  fill(Weight..inc..batteries. , .direction = "down")

# Display the updated data frame
head(camera_df)
```

```
##                    Model Release.date Max.resolution Low.resolution
## 1       Agfa ePhoto 1280         1997           1024            640
## 2       Agfa ePhoto 1680         1998           1280            640
## 3       Agfa ePhoto CL18         2000            640              0
## 4       Agfa ePhoto CL30         1999           1152            640
## 5 Agfa ePhoto CL30 Clik!         1999           1152            640
## 6       Agfa ePhoto CL45         2001           1600            640
##   Effective.pixels Zoom.wide..W. Zoom.tele..T. Normal.focus.range
## 1                0            38           114                 70
```

```
## 2                1            38           114                50
## 3                0            45            45                 0
## 4                0            35            35                 0
## 5                0            43            43                50
## 6                1            51            51                50
##   Macro.focus.range Storage.included Weight..inc..batteries. Dimensions Price
## 1                40                4                    420         95   179
## 2                 0                4                    420        158   179
## 3                 0                2                      0          0   179
## 4                 0                4                      0          0   269
## 5                 0               40                    300        128  1299
## 6                20                8                    270        119   179
```

Reshape Data

```
# Convert the data frame from wide to long format
camera_df_long <- camera_df %>%
  pivot_longer(cols = c(Effective.pixels, Price),
               names_to = "Variable",
               values_to = "Value")

# Display the reshaped data frame
head(camera_df_long)
```

```
## # A tibble: 6 x 13
##   Model   Release.date Max.resolution Low.resolution Zoom.wide..W. Zoom.tele..T.
##   <chr>          <int>          <dbl>          <dbl>         <dbl>         <dbl>
## 1 Agfa e~         1997           1024            640            38           114
## 2 Agfa e~         1997           1024            640            38           114
## 3 Agfa e~         1998           1280            640            38           114
## 4 Agfa e~         1998           1280            640            38           114
## 5 Agfa e~         2000            640              0            45            45
## 6 Agfa e~         2000            640              0            45            45
## # i 7 more variables: Normal.focus.range <dbl>, Macro.focus.range <dbl>,
## #   Storage.included <dbl>, Weight..inc..batteries. <dbl>, Dimensions <dbl>,
## #   Variable <chr>, Value <dbl>
```

Clean Data

```
# Replace "0" values in the Weight column with NA
camera_df <- cameraData
camera_df <- camera_df %>%
  mutate(Weight..inc..batteries. = ifelse(Weight..inc..batteries. == 0, NA, Weight..inc..batteries.))

# Replace all spaces in the Model column with underscores
camera_df <- camera_df %>%
  mutate(Model = gsub(" ", "_", Model))

# Display the cleaned data frame
head(camera_df)
```

```
##                    Model Release.date Max.resolution Low.resolution
## 1       Agfa_ePhoto_1280         1997           1024            640
## 2       Agfa_ePhoto_1680         1998           1280            640
## 3       Agfa_ePhoto_CL18         2000            640              0
## 4       Agfa_ePhoto_CL30         1999           1152            640
## 5 Agfa_ePhoto_CL30_Clik!         1999           1152            640
## 6       Agfa_ePhoto_CL45         2001           1600            640
##   Effective.pixels Zoom.wide..W. Zoom.tele..T. Normal.focus.range
## 1                0            38           114                70
## 2                1            38           114                50
## 3                0            45            45                 0
## 4                0            35            35                 0
## 5                0            43            43                50
## 6                1            51            51                50
##   Macro.focus.range Storage.included Weight..inc..batteries. Dimensions Price
## 1                40                4                     420         95   179
## 2                 0                4                     420        158   179
## 3                 0                2                      NA          0   179
## 4                 0                4                      NA          0   269
## 5                 0               40                     300        128  1299
## 6                20                8                     270        119   179
```

Convert Data from Long to Wide Format

```
# Convert the data frame from long to wide format
camera_df_wide <- camera_df_long %>%
  pivot_wider(names_from = "Variable",
              values_from = "Value")

# Display the spread data frame
camera_df_wide
```

```
## # A tibble: 1,036 x 13
##    Model  Release.date Max.resolution Low.resolution Zoom.wide..W. Zoom.tele..T.
##    <chr>         <int>          <dbl>          <dbl>         <dbl>         <dbl>
##  1 Agfa ~         1997           1024            640            38           114
##  2 Agfa ~         1998           1280            640            38           114
##  3 Agfa ~         2000            640              0            45            45
##  4 Agfa ~         1999           1152            640            35            35
##  5 Agfa ~         1999           1152            640            43            43
##  6 Agfa ~         2001           1600            640            51            51
##  7 Agfa ~         1999           1280            640            34           102
##  8 Canon~         1997            640              0            42            42
##  9 Canon~         1996            832            640            50            50
## 10 Canon~         2001           1280           1024            35           105
## # i 1,026 more rows
## # i 7 more variables: Normal.focus.range <dbl>, Macro.focus.range <dbl>,
## #   Storage.included <dbl>, Weight..inc..batteries. <dbl>, Dimensions <dbl>,
## #   Effective.pixels <dbl>, Price <dbl>
```

Simple Tests

Frequency of Variables

```r
# Create a vector of categorical videos
videos <- c("Funny", "Music", "Sports", "News", "Funny", "Music", "Sports", "Music")

# Use the table() function to get the frequency of each category
video_freq <- table(videos)

# Print the frequency table
print(video_freq)
```

```
## videos
##  Funny  Music   News Sports
##      2      3      1      2
```

Mean

```r
numbers <- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
mean(numbers)
```

```
## [1] 5.5
```

Median

```r
median(numbers)
```

```
## [1] 5.5
```

Standard Deviation

```r
sd(numbers)
```

```
## [1] 3.02765
```

Statistical Tests

One Sample T-Test

```r
# One-sample t-test to compare mean Price to a hypothesized value of $500
t.test(camera_clean$Price, mu = 500)
```

```
##
##  One Sample t-test
##
## data:  camera_clean$Price
## t = -1.7795, df = 1035, p-value = 0.07545
## alternative hypothesis: true mean is not equal to 500
## 95 percent confidence interval:
##  411.5225 504.3211
## sample estimates:
## mean of x
##  457.9218
```

Paired Samples T-Test

```
# Paired samples t-test to compare mean Low Resolution to Max Resolution
t.test(camera_clean$LowResolution, camera_clean$MaxResolution, paired = TRUE)
```

```
##
##  Paired t-test
##
## data:  camera_clean$LowResolution and camera_clean$MaxResolution
## t = -49.884, df = 1035, p-value < 2.2e-16
## alternative hypothesis: true mean difference is not equal to 0
## 95 percent confidence interval:
##  -725.6913 -670.7604
## sample estimates:
## mean difference
##       -698.2259
```

Two or Independent Sample T-Test

Two-Sample T-Test: The t.test() function is used to compare the means of two groups. The result of the t-test provides a p-value, which is used to determine if there is statistically significant evidence to reject the null hypothesis that the means of the two groups are equal. A low p-value (typically less than 0.05) suggests that there is evidence to support the conclusion that the means of the two groups are different.

```
# Two-sample t-test to compare Price between Brand A and Brand B
t.test(Price ~ Brand, data = camera_clean, subset = Brand %in% c("Canon", "Sigma"))
```

```
##
##  Welch Two Sample t-test
##
## data:  Price by Brand
## t = -0.26648, df = 3.5656, p-value = 0.8046
## alternative hypothesis: true difference in means between group Canon and group Sigma is not equal to
## 95 percent confidence interval:
##  -1515.241  1261.415
## sample estimates:
## mean in group Canon mean in group Sigma
##             682.087             809.000
```

Chi Square Test

Chi-Square Test: The chisq.test() function is used to test the independence between two categorical variables. The result of the chi-square test provides a p-value, which is used to determine if there is statistically significant evidence to reject the null hypothesis of independence between the two variables. A low p-value (typically less than 0.05) suggests that there is evidence to support the conclusion that the two categorical variables are associated or dependent.

```
# Chi-square test for independence between Resolution and Brand
chisq.test(camera_clean$MaxResolution, camera_clean$Brand)
```

```
## Warning in chisq.test(camera_clean$MaxResolution, camera_clean$Brand):
## Chi-squared approximation may be incorrect
```

```
##
##  Pearson's Chi-squared test
##
## data:  camera_clean$MaxResolution and camera_clean$Brand
## X-squared = 4386.2, df = 1940, p-value < 2.2e-16
```

Correlation Test

Correlation Test: The cor.test() function is used to test the association between two continuous variables. The result of the correlation test provides a correlation coefficient (typically Pearson's correlation coefficient), which indicates the strength and direction of the association between the two variables, as well as a p-value to determine if the association is statistically significant. A correlation coefficient close to 1 or -1 indicates a strong association, while a coefficient close to 0 indicates a weak or no association. A low p-value (typically less than 0.05) suggests that there is evidence to support the conclusion that the two continuous variables are correlated.

```
# Correlation test between Price and Max Resolution
cor.test(camera_clean$Price, camera_clean$MaxResolution)
```

```
##
##  Pearson's product-moment correlation
##
## data:  camera_clean$Price and camera_clean$MaxResolution
## t = 6.0263, df = 1034, p-value = 2.331e-09
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.1246939 0.2423876
## sample estimates:
##       cor
## 0.1842009
```

One Way ANOVA

One-Way ANOVA: The aov() function is used to compare means across multiple groups. The result of the one-way ANOVA provides an F-statistic and a p-value, which are used to determine if there is statistically significant evidence to reject the null hypothesis that the means of all groups are equal. A high F-statistic and a low p-value (typically less than 0.05) suggest that there is evidence to support the conclusion that there are statistically significant differences among the means of the groups.

```r
# One-way ANOVA to compare Price across different Brand categories
model <- aov(Price ~ Brand, data = camera_clean)
summary(model)
```

```
##               Df    Sum Sq Mean Sq F value   Pr(>F)
## Brand         20  54710963 2735548   5.096 2.89e-12 ***
## Residuals   1015 544819062  536768
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Regression Tests

Simple Linear Regression

The output includes coefficients, standard errors, t-values, p-values, and goodness of fit statistics, which provide information about the estimated coefficients, significance of the coefficients, goodness of fit of the models, and statistical significance of the relationships between the independent and dependent variables.

```r
# Simple Linear Regression
# Fit a simple linear regression model with Price as the dependent variable and MaxResolution as the in
simple_reg_model <- lm(Price ~ MaxResolution, data = camera_clean)

# Print the summary of the simple linear regression model
summary(simple_reg_model)
```

```
##
## Call:
## lm(formula = Price ~ MaxResolution, data = camera_clean)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -709.8 -339.3 -204.9  -50.9 7247.4
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)    1.35555   79.25075   0.017    0.986
## MaxResolution  0.18461    0.03063   6.026 2.33e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 748.4 on 1034 degrees of freedom
## Multiple R-squared:  0.03393,    Adjusted R-squared:  0.033
## F-statistic: 36.32 on 1 and 1034 DF,  p-value: 2.331e-09
```

Mutliple Linear Regression

```r
multiple_reg_model <- lm(Price ~ MaxResolution + Brand + ZoomWide, data = camera_clean)

# Print the summary of the multiple linear regression model
summary(multiple_reg_model)
```

```
##
## Call:
## lm(formula = Price ~ MaxResolution + Brand + ZoomWide, data = camera_clean)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1208.8  -286.4   -52.2    96.7  6251.9
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)    1759.57656  265.10003   6.637 5.19e-11 ***
## MaxResolution     0.01378    0.02972   0.464   0.6429
## BrandCanon      -68.52121  253.76711  -0.270   0.7872
## BrandCasio     -216.81826  258.46205  -0.839   0.4017
## BrandContax    -635.06685  518.71007  -1.224   0.2211
## BrandEpson      -67.27334  294.91552  -0.228   0.8196
## BrandFujifilm  -304.70499  254.51148  -1.197   0.2315
## BrandHP        -315.10430  263.55372  -1.196   0.2321
## BrandJVC       -475.18961  516.23814  -0.920   0.3575
## BrandKodak     -583.08793  253.66463  -2.299   0.0217 *
## BrandKyocera    369.50443  295.78718   1.249   0.2119
## BrandLeica     -559.09943  314.75095  -1.776   0.0760 .
## BrandNikon     -138.21528  255.84822  -0.540   0.5892
## BrandOlympus     24.95318  252.51901   0.099   0.9213
## BrandPanasonic  243.45824  261.63327   0.931   0.3523
## BrandPentax    -402.03340  258.67589  -1.554   0.1204
## BrandRicoh      -84.07026  276.40776  -0.304   0.7611
## BrandSamsung   -405.64226  262.42492  -1.546   0.1225
## BrandSanyo       49.41713  333.38758   0.148   0.8822
## BrandSigma     -738.76149  409.04184  -1.806   0.0712 .
## BrandSony       -10.61992  252.32560  -0.042   0.9664
## BrandToshiba   -312.22412  287.10723  -1.087   0.2771
## ZoomWide        -35.09182    2.17827 -16.110  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 643.2 on 1013 degrees of freedom
## Multiple R-squared:  0.3011, Adjusted R-squared:  0.2859
## F-statistic: 19.84 on 22 and 1013 DF,  p-value: < 2.2e-16
```

Poison Regression

```
# Poisson Regression
# Fit a Poisson regression model with Price as the dependent variable and EffectivePixels, ZoomWide, an
poisson_model <- glm(Price ~ EffectivePixels + ZoomWide + Brand, data = camera_clean, family = poisson)

# Print the summary of the Poisson regression model
summary(poisson_model)
```

```
##
## Call:
## glm(formula = Price ~ EffectivePixels + ZoomWide + Brand, family = poisson,
```

```
##     data = camera_clean)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -49.680  -12.625   -2.837    3.766  118.722
##
## Coefficients:
##                   Estimate Std. Error  z value Pr(>|z|)
## (Intercept)      7.7120341  0.0206556  373.363  < 2e-16 ***
## EffectivePixels  0.0071897  0.0005301   13.562  < 2e-16 ***
## ZoomWide        -0.0464402  0.0001130 -411.027  < 2e-16 ***
## BrandCanon      -0.1030771  0.0207031   -4.979 6.40e-07 ***
## BrandCasio      -0.4235171  0.0216493  -19.563  < 2e-16 ***
## BrandContax     -0.9312534  0.0364547  -25.545  < 2e-16 ***
## BrandEpson      -0.0433757  0.0231122   -1.877   0.0606 .
## BrandFujifilm   -0.5896988  0.0210640  -27.996  < 2e-16 ***
## BrandHP         -0.8406437  0.0231526  -36.309  < 2e-16 ***
## BrandJVC        -3.3762572  0.1900576  -17.764  < 2e-16 ***
## BrandKodak      -1.6840082  0.0220615  -76.333  < 2e-16 ***
## BrandKyocera     0.7313173  0.0219727   33.283  < 2e-16 ***
## BrandLeica      -1.0033060  0.0265037  -37.855  < 2e-16 ***
## BrandNikon      -0.2503339  0.0208098  -12.030  < 2e-16 ***
## BrandOlympus     0.1223684  0.0206047    5.939 2.87e-09 ***
## BrandPanasonic   0.4737498  0.0208067   22.769  < 2e-16 ***
## BrandPentax     -0.8877182  0.0215771  -41.142  < 2e-16 ***
## BrandRicoh       0.0982245  0.0217819    4.509 6.50e-06 ***
## BrandSamsung    -0.9735603  0.0224063  -43.450  < 2e-16 ***
## BrandSanyo       0.2880881  0.0250459   11.502  < 2e-16 ***
## BrandSigma      -0.8398484  0.0269947  -31.112  < 2e-16 ***
## BrandSony        0.1191837  0.0206637    5.768 8.03e-09 ***
## BrandToshiba    -0.9863560  0.0279213  -35.326  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 633555  on 1035  degrees of freedom
## Residual deviance: 318375  on 1013  degrees of freedom
## AIC: 326115
##
## Number of Fisher Scoring iterations: 6
```

Regression Example with Prediction

Split Data into Training and Testing

We use the caret package for splitting into training and testing its easy

```
camera_clean_example = subset(camera_clean, select = -c(Model)) #we remove the Model attribute here
# Load the caret package
library(caret)
```

```
## Loading required package: lattice

##
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':
##
##     lift
```

```
# Set the seed for reproducibility
set.seed(123)

# Split the data into training and testing sets
train_indices <- createDataPartition(y = camera_clean_example$Price, p = 0.9, list = FALSE)
trainCamera <- camera_clean_example[train_indices, ]
testCamera <- camera_clean_example[-train_indices, ]
```

Training the Model

Lets see if we can predict the brand of the Price based on all the of the camera except model

```
# Simple Linear Regression Using the training data
test_model <- lm(Price ~ Brand + ReleaseDate + MacroFocusRange , data = trainCamera)

# Print the summary of the simple linear regression model
summary(test_model)
```

```
##
## Call:
## lm(formula = Price ~ Brand + ReleaseDate + MacroFocusRange, data = trainCamera)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1033.3  -363.6  -106.1    26.8  7318.4
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)     40442.418  20096.089   2.012  0.04447 *
## BrandCanon        370.772    320.480   1.157  0.24761
## BrandCasio         -5.165    328.020  -0.016  0.98744
## BrandContax       197.298    617.400   0.320  0.74937
## BrandEpson        193.548    365.287   0.530  0.59634
## BrandFujifilm      15.784    321.552   0.049  0.96086
## BrandHP           -72.019    336.924  -0.214  0.83079
## BrandJVC         -394.963    816.839  -0.484  0.62884
## BrandKodak       -134.497    321.501  -0.418  0.67580
## BrandKyocera      694.309    371.074   1.871  0.06165 .
## BrandLeica         17.018    392.552   0.043  0.96543
## BrandNikon        307.108    323.448   0.949  0.34263
## BrandOlympus      368.676    320.118   1.152  0.24975
## BrandPanasonic    575.784    331.618   1.736  0.08285 .
## BrandPentax       -15.626    328.088  -0.048  0.96202
```

```
## BrandRicoh        266.896    346.610   0.770  0.44149
## BrandSamsung      -67.683    332.925  -0.203  0.83895
## BrandSanyo        271.895    421.263   0.645  0.51881
## BrandSigma        722.533    537.936   1.343  0.17955
## BrandSony         169.429    319.992   0.529  0.59660
## BrandToshiba     -182.604    359.287  -0.508  0.61141
## ReleaseDate       -19.996     10.048  -1.990  0.04688 *
## MacroFocusRange   -10.411      3.275  -3.179  0.00153 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 755.8 on 912 degrees of freedom
## Multiple R-squared:  0.09676,    Adjusted R-squared:  0.07497
## F-statistic: 4.441 on 22 and 912 DF,  p-value: 7.613e-11
```

Predicting Price

```
# Perform prediction using the linear regression model on the test data
predicted_prices <- predict(test_model, newdata = testCamera)
# Store predicted prices in a named numeric vector
predicted_prices <- as.vector(predicted_prices)
sum(predicted_prices)
```

```
## [1] 42524.89
```

Evaluating Accuracy

```
# Calculate the accuracy of the predictions
actual_prices <- testCamera$Price
actual_prices <- as.vector(actual_prices)
accuracy <- mean(abs(predicted_prices - actual_prices)/actual_prices)

# Print the accuracy
cat("Accuracy:", accuracy, "\n")
```

```
## Accuracy: 1.593525
```

In short this model is terrible