

Exploring Dataframes

Aug 09, 2023.

The mtcars dataset is a readily available set in R, originally sourced from the 1974 Motor Trend US magazine. It includes data related to fuel consumption and 10 other factors pertaining to car design and performance, recorded for 32 vehicles from the 1973-74 model years.

To load the mtcars dataset in R, use this command:

```
data(mtcars)
```

Reviewing a dataframe

View(): This function opens the dataset in a spreadsheet-style data viewer.

```
View(mtcars)
```

head(): This function prints the first six rows of the dataframe.

```
head(mtcars)
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160	110	3.90	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108	93	3.85	2.320	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360	175	3.15	3.440	17.02	0	0	3	2
Valiant	18.1	6	225	105	2.76	3.460	20.22	1	0	3	1

tail(): This function prints the last six rows of the dataframe.

```
tail(mtcars)
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Porsche 914-2	26.0	4	120.3	91	4.43	2.140	16.7	0	1	5	2
Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.9	1	1	5	2
Ford Pantera L	15.8	8	351.0	264	4.22	3.170	14.5	0	1	5	4
Ferrari Dino	19.7	6	145.0	175	3.62	2.770	15.5	0	1	5	6
Maserati Bora	15.0	8	301.0	335	3.54	3.570	14.6	0	1	5	8
Volvo 142E	21.4	4	121.0	109	4.11	2.780	18.6	1	1	4	2

`dim()`: This function retrieves the dimensions of a dataframe, i.e., the number of rows and columns.

`nrow()`: This function retrieves the number of rows in the dataframe.

`ncol()`: This function retrieves the number of columns in the dataframe.

```
dim(mtcars)
```

```
[1] 32 11
```

```
nrow(mtcars)
```

```
[1] 32
```

```
ncol(mtcars)
```

```
[1] 11
```

`names()`: This function retrieves the column names of a dataframe.

`colnames()`: This function also retrieves the column names of a dataframe.

```
names(mtcars)
```

```
[1] "mpg" "cyl" "disp" "hp" "drat" "wt" "qsec" "vs" "am" "gear"
[11] "carb"
```

```
colnames(mtcars)
```

```
[1] "mpg" "cyl" "disp" "hp" "drat" "wt" "qsec" "vs" "am" "gear"
[11] "carb"
```

Accessing data within a dataframe

\$: In R, the dollar sign \$ is a unique operator that lets us retrieve specific columns from a dataframe or elements from a list.

For instance, consider the dataframe mtcars. If we wish to fetch the data from the mpg (miles per gallon) column, we would use mtcars\$mpg. This action will yield a vector containing the data from the mpg column.

```
# Extract the mpg column in mtcars dataframe as a vector
mpg_vector <- mtcars$mpg

# Print the mpg vector
print(mpg_vector)
```

```
[1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4 17.3 15.2 10.4
[16] 10.4 14.7 32.4 30.4 33.9 21.5 15.5 15.2 13.3 19.2 27.3 26.0 30.4 15.8 19.7
[31] 15.0 21.4
```

This operator offers a simple and readable shortcut for accessing data.

[: The usage of \$ is limited since it doesn't support character substitution for dynamic column access inside functions. In such cases, we resort to using double square brackets [[or single square brackets [.

As an example, if we have a character string stored in a variable var as var <- "mpg", using mtcars\$var will not return the mpg column. But if we use mtcars[[var]] or mtcars[, var], we will correctly get the mpg column.

```
# Let's say we have a variable var
var <- "mpg"

# Now we can access the mpg column in mtcars dataframe using [[
mpg_data1 <- mtcars[[var]]
print(mpg_data1)
```

```
[1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4 17.3 15.2 10.4
[16] 10.4 14.7 32.4 30.4 33.9 21.5 15.5 15.2 13.3 19.2 27.3 26.0 30.4 15.8 19.7
[31] 15.0 21.4
```

```
# Alternatively, we can use [
mpg_data2 <- mtcars[, var]
print(mpg_data2)
```

```
[1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4 17.3 15.2 10.4
[16] 10.4 14.7 32.4 30.4 33.9 21.5 15.5 15.2 13.3 19.2 27.3 26.0 30.4 15.8 19.7
[31] 15.0 21.4
```

Data Structures

str(): This function displays the internal structure of an R object.

```
str(mtcars)
```

```
'data.frame': 32 obs. of 11 variables:
 $ mpg : num 21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
 $ cyl : num 6 6 4 6 8 6 8 4 4 6 ...
 $ disp: num 160 160 108 258 360 ...
 $ hp : num 110 110 93 110 175 105 245 62 95 123 ...
 $ drat: num 3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
 $ wt : num 2.62 2.88 2.32 3.21 3.44 ...
 $ qsec: num 16.5 17 18.6 19.4 17 ...
 $ vs : num 0 0 1 1 0 1 0 1 1 1 ...
 $ am : num 1 1 1 0 0 0 0 0 0 0 ...
 $ gear: num 4 4 4 3 3 3 3 4 4 4 ...
 $ carb: num 4 4 1 1 2 1 4 2 2 4 ...
```

class(): This function is used to determine the class or data type of an object. It returns a character vector specifying the class or classes of the object.

```
x <- c(1, 2, 3) # Create a numeric vector
class(x)        # Output: "numeric"
```

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

```
y <- "Hello, My name is Sameer Mathur!" # Create a character vector
class(y)                                # Output: "character"
```

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

`class(x)` returns “numeric” because `x` is a numeric vector. Similarly, `class(y)` returns “character” because `y` is a character vector.

```
z <- data.frame(a = 1:5, b = letters[1:5]) # Create a data frame
class(z) # Output: "data.frame"
```

```
[1] "data.frame"
```

`class(z)` returns “data.frame” because `z` is a data frame.

```
sapply(mtcars, class)
```

```
      mpg      cyl      disp      hp      drat      wt      qsec      vs
"numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric"
      am      gear      carb
"numeric" "numeric" "numeric"
```

Factors

In R, factors are a specific data type used for representing categorical variables or data with discrete levels or categories. They are employed to store data that has a limited number of distinct values, such as “male” or “female,” “red,” “green,” or “blue,” or “low,” “medium,” or “high.”

Factors in R consist of both values and levels. The values represent the actual data, while the levels correspond to the distinct categories or levels within the factor. Factors are particularly useful for statistical analysis as they facilitate the representation and analysis of categorical data efficiently.

To change the data type of the `am`, `cyl`, `vs`, and `gear` variables in the `mtcars` dataset to factors, you can utilize the `factor()` function. Here’s an example demonstrating how to achieve this:

```
# Convert variables to factors
mtcars$am <- factor(mtcars$am)
mtcars$cyl <- factor(mtcars$cyl)
mtcars$vs <- factor(mtcars$vs)
mtcars$gear <- factor(mtcars$gear)
```

The code above applies the `factor()` function to each variable, thereby converting them to factors. By assigning the result back to the respective variables, we effectively change their data type to factors. This conversion retains the original values while establishing levels based on the distinct values present in each variable.

After executing this code, the `am`, `cyl`, `vs`, and `gear` variables in the `mtcars` dataset will be of the factor data type. And we can verify this by re-running the `str()` function

```
str(mtcars)
```

```
'data.frame':  32 obs. of  11 variables:
 $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
 $ cyl : Factor w/ 3 levels "4","6","8": 2 2 1 2 3 2 3 1 1 2 ...
 $ disp: num  160 160 108 258 360 ...
 $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
 $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
 $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
 $ qsec: num  16.5 17 18.6 19.4 17 ...
 $ vs  : Factor w/ 2 levels "0","1": 1 1 2 2 1 2 1 2 2 2 ...
 $ am  : Factor w/ 2 levels "0","1": 2 2 2 1 1 1 1 1 1 1 ...
 $ gear: Factor w/ 3 levels "3","4","5": 2 2 2 1 1 1 1 2 2 2 ...
 $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

When the `cyl` variable in the `mtcars` dataset is converted to a factor, the `levels()` function can be used to extract the distinct levels or categories of that factor. By executing `levels(mtcars$cyl)`, you will receive an output that reveals the levels present in the `cyl` variable.

For example, if the `cyl` variable has been transformed into a factor with levels “4”, “6”, and “8”, the result of `levels(mtcars$cyl)` will be a character vector displaying these three levels:

```
levels(mtcars$cyl)
```

```
[1] "4" "6" "8"
```

It is important to note that the order of the levels in the output corresponds to their appearance in the original data.

Utilizing the `levels()` function on factor variables in R allows you to examine the particular categories or levels present within a factor, aiding in understanding the data’s composition and facilitating operations that target specific levels if necessary.

To change the base level of a factor variable in R, you can use the `relevel()` function. This function allows you to reassign a new base level by rearranging the order of the levels in the factor variable.

Here's an example of how you can change the base level of a factor variable:

```
# Assuming 'cyl' is a factor variable with levels "4", "6", and "8"
mtcars$cyl <- relevel(mtcars$cyl, ref = "6")
```

In the code above, we apply the `relevel()` function to the `cyl` variable, specifying `ref = "6"` to set "6" as the new base level.

After executing this code, the levels of the `mtcars$cyl` factor variable will be reordered, with "6" becoming the new base level. The order of the levels will be "6", "4", and "8" instead of the original order.

Changing the base level can be particularly useful when conducting statistical modeling or interpreting the effects of categorical variables in regression models. By selecting a specific level as the base, we can compare the effects of the other levels relative to the chosen base level, facilitating more meaningful analysis and interpretation.

For convenience, we will change the base level back to "4".

```
# Assuming 'cyl' is a factor variable with levels "4", "6", and "8"
mtcars$cyl <- relevel(mtcars$cyl, ref = "4")
```

`droplevels()`: This function is helpful for removing unused factor levels. It removes levels from a factor variable that do not appear in the data, reducing unnecessary levels and ensuring that the factor only includes relevant levels.

```
# Assuming 'cyl' is a factor variable with levels "4", "6", and "8"
# Check the levels of 'cyl' before removing unused levels
levels(mtcars$cyl)
```

```
[1] "4" "6" "8"
```

```
# Remove unused levels from 'cyl'
mtcars$cyl <- droplevels(mtcars$cyl)

# Check the levels of 'cyl' after removing unused levels
levels(mtcars$cyl)
```

```
[1] "4" "6" "8"
```

We apply `droplevels()` to `mtcars$cyl` to remove any unused levels from the factor variable. This function removes factor levels that are not present in the data. In this case all three levels were present in the data and therefore nothing was removed.

`cut()`: The `cut()` function allows you to convert a continuous variable into a factor variable by dividing it into intervals or bins. This is useful when you want to group numeric data into categories or levels.

```
# Create a new factor variable 'mpg_category' by cutting 'mpg' into intervals
mtcars$mpg_category <- cut(mtcars$mpg,
                           breaks = c(0, 20, 30, Inf),
                           labels = c("Low", "Medium", "High"))

# Summarize the resulting 'mpg_category' variable
summary(mtcars$mpg_category)
```

Low	Medium	High
18	10	4

In the provided code, a new factor variable called `mpg_category` is generated based on the `mpg` (miles per gallon) variable from the `mtcars` dataset. This is achieved using the `cut()` function, which segments the `mpg` values into distinct intervals and assigns appropriate factor labels.

The `cut()` function takes several arguments:

`mtcars$mpg` represents the variable to be divided.

`breaks` specifies the cutoff points for interval creation. Here, we define three intervals: values up to 20, values between 20 and 30 (inclusive), and values greater than 30. The `breaks` argument is defined as `c(0, 20, 30, Inf)` to indicate these intervals.

`labels` assigns labels to the resulting factor levels. In this instance, the labels “Low”, “Medium”, and “High” are provided to correspond with the respective intervals.

Having demonstrated how to create the new column `mpg_category`, we will now drop this column from the dataframe.

```
# drop the column `mpg_category`
mtcars$mpg_category = NULL
```


Logical operations

Here are some logical operations functions in R.

`subset()`: This function returns a subset of a data frame according to condition(s).

```
# Find cars that have cyl = 4 and mpg < 28
subset(mtcars, cyl == 4 & mpg < 22)
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Toyota Corona	21.5	4	120.1	97	3.70	2.465	20.01	1	0	3	1
Volvo 142E	21.4	4	121.0	109	4.11	2.780	18.60	1	1	4	2

```
# Find cars that have wt > 5 or mpg < 15
subset(mtcars, wt > 5 | mpg < 15)
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Duster 360	14.3	8	360	245	3.21	3.570	15.84	0	0	3	4
Cadillac Fleetwood	10.4	8	472	205	2.93	5.250	17.98	0	0	3	4
Lincoln Continental	10.4	8	460	215	3.00	5.424	17.82	0	0	3	4
Chrysler Imperial	14.7	8	440	230	3.23	5.345	17.42	0	0	3	4
Camaro Z28	13.3	8	350	245	3.73	3.840	15.41	0	0	3	4

`which()`: This function returns the indexes of a vector's members that satisfy a condition.

```
# Find the indices of rows where mpg > 20
indices <- which(mtcars$mpg > 20)
indices
```

```
[1] 1 2 3 4 8 9 18 19 20 21 26 27 28 32
```

`ifelse()`: This function applies a logical condition to a vector and returns a new vector with values depending on whether the condition is TRUE or FALSE.

```
# Create a new column "high_mpg" based on mpg > 20
mtcars$high_mpg <- ifelse(mtcars$mpg > 20, "Yes", "No")
```

Dropping a column: We can drop a column by setting it to NULL.

```
# Drop the column "high_mpg"
mtcars$high_mpg <- NULL
```

all(): If every element in a vector satisfies a logical criterion, this function returns TRUE; otherwise, it returns FALSE.

```
# Check if all values in mpg column are greater than 20
all(mtcars$mpg > 20)
```

```
[1] FALSE
```

any(): If at least one element in a vector satisfies a logical criterion, this function returns TRUE; otherwise, it returns FALSE.

```
# Check if any of the values in the mpg column are greater than 20
any(mtcars$mpg > 20)
```

```
[1] TRUE
```

Subsetting based on a condition:

The logical expression `[]` and square bracket notation can be used to subset the mtcars dataset according to one or more conditions.

```
# Subset mtcars based on mpg > 20
mtcars_subset <- mtcars[mtcars$mpg > 20, ]
mtcars_subset
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
Fiat 128	32.4	4	78.7	66	4.08	2.200	19.47	1	1	4	1
Honda Civic	30.4	4	75.7	52	4.93	1.615	18.52	1	1	4	2
Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1
Toyota Corona	21.5	4	120.1	97	3.70	2.465	20.01	1	0	3	1

Fiat X1-9	27.3	4	79.0	66	4.08	1.935	18.90	1	1	4	1
Porsche 914-2	26.0	4	120.3	91	4.43	2.140	16.70	0	1	5	2
Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.90	1	1	5	2
Volvo 142E	21.4	4	121.0	109	4.11	2.780	18.60	1	1	4	2

sort(): This function arranges a vector in an increasing or decreasing sequence.

```
sort(mtcars$mpg) # increasing order
```

```
[1] 10.4 10.4 13.3 14.3 14.7 15.0 15.2 15.2 15.5 15.8 16.4 17.3 17.8 18.1 18.7
[16] 19.2 19.2 19.7 21.0 21.0 21.4 21.4 21.5 22.8 22.8 24.4 26.0 27.3 30.4 30.4
[31] 32.4 33.9
```

```
sort(mtcars$mpg, decreasing = TRUE) # decreasing order
```

```
[1] 33.9 32.4 30.4 30.4 27.3 26.0 24.4 22.8 22.8 21.5 21.4 21.4 21.0 21.0 19.7
[16] 19.2 19.2 18.7 18.1 17.8 17.3 16.4 15.8 15.5 15.2 15.2 15.0 14.7 14.3 13.3
[31] 10.4 10.4
```

order(): This function provides an arrangement which sorts its initial argument into ascending or descending order.

```
mtcars[order(mtcars$mpg), ] # ascending order
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250	17.98	0	0	3	4
Lincoln Continental	10.4	8	460.0	215	3.00	5.424	17.82	0	0	3	4
Camaro Z28	13.3	8	350.0	245	3.73	3.840	15.41	0	0	3	4
Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
Chrysler Imperial	14.7	8	440.0	230	3.23	5.345	17.42	0	0	3	4
Maserati Bora	15.0	8	301.0	335	3.54	3.570	14.60	0	1	5	8
Merc 450SLC	15.2	8	275.8	180	3.07	3.780	18.00	0	0	3	3
AMC Javelin	15.2	8	304.0	150	3.15	3.435	17.30	0	0	3	2
Dodge Challenger	15.5	8	318.0	150	2.76	3.520	16.87	0	0	3	2
Ford Pantera L	15.8	8	351.0	264	4.22	3.170	14.50	0	1	5	4
Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.40	0	0	3	3
Merc 450SL	17.3	8	275.8	180	3.07	3.730	17.60	0	0	3	3
Merc 280C	17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4

Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4
Pontiac Firebird	19.2	8	400.0	175	3.08	3.845	17.05	0	0	3	2
Ferrari Dino	19.7	6	145.0	175	3.62	2.770	15.50	0	1	5	6
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
Volvo 142E	21.4	4	121.0	109	4.11	2.780	18.60	1	1	4	2
Toyota Corona	21.5	4	120.1	97	3.70	2.465	20.01	1	0	3	1
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
Porsche 914-2	26.0	4	120.3	91	4.43	2.140	16.70	0	1	5	2
Fiat X1-9	27.3	4	79.0	66	4.08	1.935	18.90	1	1	4	1
Honda Civic	30.4	4	75.7	52	4.93	1.615	18.52	1	1	4	2
Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.90	1	1	5	2
Fiat 128	32.4	4	78.7	66	4.08	2.200	19.47	1	1	4	1
Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1

```
mtcars[order(-mtcars$mpg), ] # descending order
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1
Fiat 128	32.4	4	78.7	66	4.08	2.200	19.47	1	1	4	1
Honda Civic	30.4	4	75.7	52	4.93	1.615	18.52	1	1	4	2
Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.90	1	1	5	2
Fiat X1-9	27.3	4	79.0	66	4.08	1.935	18.90	1	1	4	1
Porsche 914-2	26.0	4	120.3	91	4.43	2.140	16.70	0	1	5	2
Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
Toyota Corona	21.5	4	120.1	97	3.70	2.465	20.01	1	0	3	1
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
Volvo 142E	21.4	4	121.0	109	4.11	2.780	18.60	1	1	4	2
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
Ferrari Dino	19.7	6	145.0	175	3.62	2.770	15.50	0	1	5	6
Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4
Pontiac Firebird	19.2	8	400.0	175	3.08	3.845	17.05	0	0	3	2
Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2

Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
Merc 280C	17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4
Merc 450SL	17.3	8	275.8	180	3.07	3.730	17.60	0	0	3	3
Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.40	0	0	3	3
Ford Pantera L	15.8	8	351.0	264	4.22	3.170	14.50	0	1	5	4
Dodge Challenger	15.5	8	318.0	150	2.76	3.520	16.87	0	0	3	2
Merc 450SLC	15.2	8	275.8	180	3.07	3.780	18.00	0	0	3	3
AMC Javelin	15.2	8	304.0	150	3.15	3.435	17.30	0	0	3	2
Maserati Bora	15.0	8	301.0	335	3.54	3.570	14.60	0	1	5	8
Chrysler Imperial	14.7	8	440.0	230	3.23	5.345	17.42	0	0	3	4
Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
Camaro Z28	13.3	8	350.0	245	3.73	3.840	15.41	0	0	3	4
Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250	17.98	0	0	3	4
Lincoln Continental	10.4	8	460.0	215	3.00	5.424	17.82	0	0	3	4

Statistical functions

`mean()`: This function computes the arithmetic mean.

```
mean(mtcars$mpg)
```

```
[1] 20.09062
```

`median()`: This function computes the median.

```
median(mtcars$mpg)
```

```
[1] 19.2
```

`sd()`: This function computes the standard deviation.

```
sd(mtcars$mpg)
```

```
[1] 6.026948
```

`var()`: This function computes the variance.

```
var(mtcars$mpg)
```

```
[1] 36.3241
```

cor(): This function computes the correlation between variables.

```
cor(mtcars$mpg, mtcars$wt)
```

```
[1] -0.8676594
```

unique(): This function extracts the unique elements of a vector.

```
unique(mtcars$mpg)
```

```
[1] 21.0 22.8 21.4 18.7 18.1 14.3 24.4 19.2 17.8 16.4 17.3 15.2 10.4 14.7 32.4  
[16] 30.4 33.9 21.5 15.5 13.3 27.3 26.0 15.8 19.7 15.0
```

Summarizing a dataframe

summary(): This function is a convenient tool to generate basic descriptive statistics for your dataset. It provides a succinct snapshot of the distribution characteristics of your data.

```
summary(mtcars$mpg)
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
10.40	15.43	19.20	20.09	22.80	33.90

When applied to a vector or a specific column in a dataframe, it generates the following:

Min: This represents the smallest recorded value in the mpg column.

1st Qu: This indicates the first quartile or the 25th percentile of the mpg column. It implies that 25% of all mpg values fall below this threshold.

Median: This value signifies the median or the middle value of the mpg column, also known as the 50th percentile. Half of the mpg values are less than this value.

Mean: This denotes the average value of the mpg column.

3rd Qu: This represents the third quartile or the 75th percentile of the mpg column. It shows that 75% of all mpg values are less than this value.

Max: This indicates the highest value observed in the mpg column.

When we use `summary(mtcars$mpg)`, it returns these six statistics for the mpg (miles per gallon) column in the mtcars dataset.

When used with an entire dataframe, it applies to each column individually and provides a quick overview of the data.

```
summary(mtcars$cyl)
```

```
 4  6  8
11  7 14
```

The output of `summary(mtcars$cyl)` displays the frequency distribution of the levels within the cyl factor variable. It shows the count or frequency of each level, which in this case are “4”, “6”, and “8”. The summary will provide a concise overview of the distribution of these levels within the dataset.

```
summary(mtcars)
```

mpg	cyl	disp	hp	drat
Min. :10.40	4:11	Min. : 71.1	Min. : 52.0	Min. :2.760
1st Qu.:15.43	6: 7	1st Qu.:120.8	1st Qu.: 96.5	1st Qu.:3.080
Median :19.20	8:14	Median :196.3	Median :123.0	Median :3.695
Mean :20.09		Mean :230.7	Mean :146.7	Mean :3.597
3rd Qu.:22.80		3rd Qu.:326.0	3rd Qu.:180.0	3rd Qu.:3.920
Max. :33.90		Max. :472.0	Max. :335.0	Max. :4.930

wt	qsec	vs	am	gear	carb
Min. :1.513	Min. :14.50	0:18	0:19	3:15	Min. :1.000
1st Qu.:2.581	1st Qu.:16.89	1:14	1:13	4:12	1st Qu.:2.000
Median :3.325	Median :17.71			5: 5	Median :2.000
Mean :3.217	Mean :17.85				Mean :2.812
3rd Qu.:3.610	3rd Qu.:18.90				3rd Qu.:4.000
Max. :5.424	Max. :22.90				Max. :8.000

Creating new functions in R

We illustrate how to create a custom function in R that computes the mean of any given numeric column in the mtcars dataframe:

```
# Function creation
compute_average <- function(df, column) {
  # Compute the average of the specified column
  average_val <- mean(df[[column]], na.rm = TRUE)

  # Return the computed average
  return(average_val)
}

# Utilize the created function
average_mpg <- compute_average(mtcars, "mpg")
print(average_mpg)
```

```
[1] 20.09062
```

```
average_hp <- compute_average(mtcars, "hp")
print(average_hp)
```

```
[1] 146.6875
```

In the above code, `compute_average` is a custom function which takes two arguments: a dataframe (`df`) and a column name (as a string) `column`. The function computes the mean of the specified column in the provided dataframe, with `na.rm = TRUE` ensuring that NA values (if any) are removed before the mean calculation.

After defining the function, we utilize it to calculate the average values of the “mpg” and “hp” columns in the `mtcars` dataframe. These computed averages are then printed.

This demonstrates a simple way to create a custom function in R.

Function to calculate average mileage for cars with a specific number of cylinders:

```
avg_mileage_by_cyl <- function(data, cyl) {
  mean(data$mpg[data$cyl == cyl])
}
```



```
# Usage
```

```
# Returns the average mileage of cars with 4 cylinders  
avg_mileage_by_cyl(mtcars, 4)
```

```
[1] 26.66364
```

```
# Returns the average mileage of cars with 6 cylinders  
avg_mileage_by_cyl(mtcars, 6)
```

```
[1] 19.74286
```

Summary of Chapter 5 – Exploring Dataframes

Chapter 5 guides the reader through a comprehensive understanding of data manipulation, logical operations, statistical functions, and custom function creation in R. The chapter highlights a number of significant elements integral to any data analysis task in R.

Firstly, the chapter introduces an essential toolbox for data manipulation in R, focusing on ‘dplyr’. It provides a step-by-step tutorial on using key ‘dplyr’ verbs such as `select()`, `filter()`, `arrange()`, `mutate()`, and `summarise()`. The lesson is further enriched with a discussion on the grouping data with the ‘`group_by()`’ function and the application of the pipe operator ‘`%>%`’, which provides a more readable and organized approach to data manipulation.

Secondly, the chapter explains logical operations in R, demonstrating how they can be employed in subsetting and data extraction tasks. It covers the workings of various functions including `subset()`, `which()`, `ifelse()`, `all()`, and `any()`. The chapter elaborates on data subsetting using square brackets and logical expressions, and introduces the `sort()` and `order()` functions, essential for arranging data in a particular sequence.

Thirdly, the chapter transitions into an examination of key statistical functions, showcasing the usage of `mean()`, `median()`, `sd()`, `var()`, and `cor()`. An interesting aspect is the inclusion of the `unique()` function, which allows extraction of distinct elements from a vector.

Fourthly, the chapter discusses the utility of the `summary()` function, providing basic descriptive statistics. This function furnishes a snapshot of dataset characteristics by generating the minimum, 1st quartile, median, mean, 3rd quartile, and maximum values for a specified dataset or column.

Lastly, the chapter unveils how to create and utilize custom functions in R. It provides an in-depth illustration of creating a custom function to calculate the mean of a given numeric

column in a dataframe and the average mileage for cars with a specific number of cylinders. These examples highlight the extensibility of R and how custom functions can enhance its capabilities.

In summary, Chapter 5 serves as a comprehensive guide to effectively managing, manipulating, and analyzing data in R. Through the demonstration of custom functions, it underscores how R's functionalities can be extended according to the specific needs of a task, thus, strengthening the flexibility and power of R programming.