Attributes and Data Types

Prof Farhi Marir
Director of Big Data Analytics Research Lab,
College of Technological Innovation,
Zayed University

Lecture Content

- Attributes & Data Types,
  - NOIR Attributes,
  - Vectors
  - Arrays and Matrices
  - Data Frames
  - List
  - Descriptive Statistics

Attributes & Data Types

NOIR

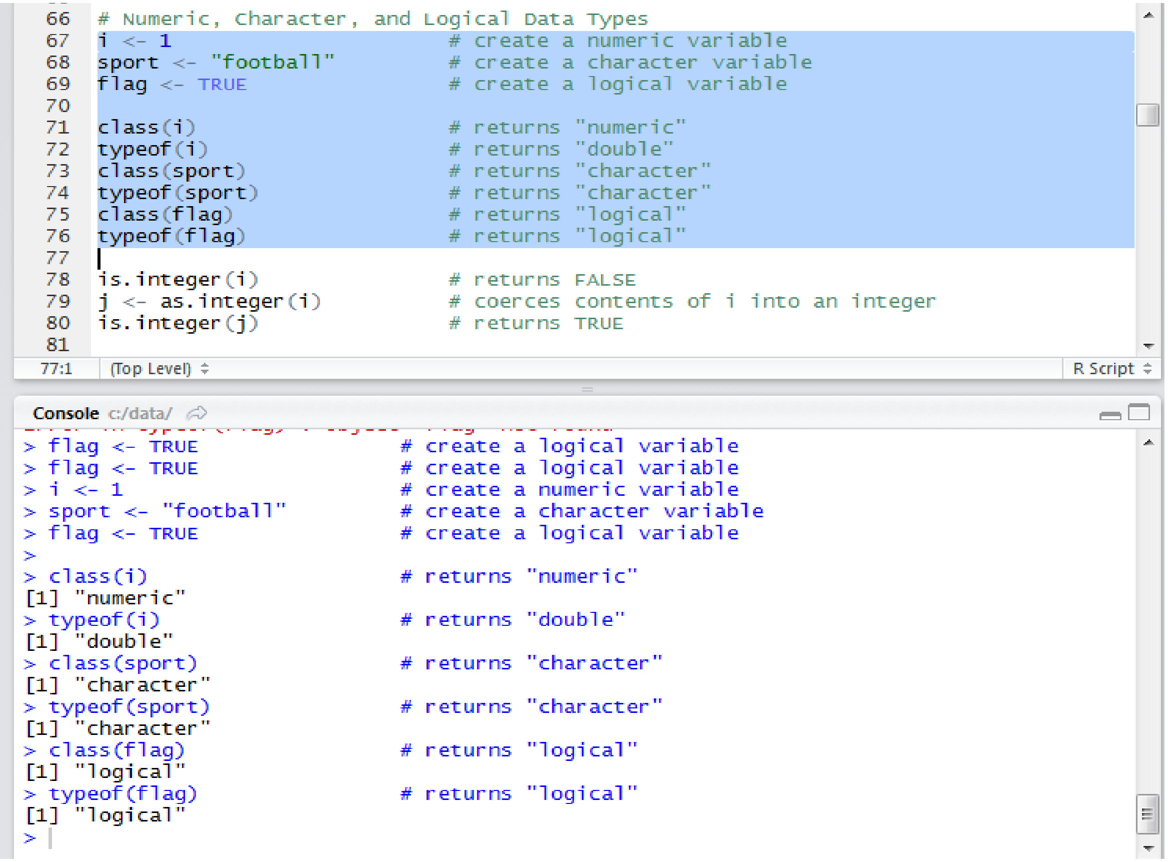
NOIR Attributes Types

- NOIR stands for Nominal, Ordinal, Integer and Ration
- Below are the chrematistic of each of them

	Categorical (Qualitative)		Numeric (Quantitative)	
	Nominal	**Ordinal**	**Interval**	**Ratio**
Definition	The values represent labels that distinguish one from another	Attributes that imply sequence	The difference between two values is meaningful,	Both the difference and the ratio of two values is meaningful
Example	ZIP code, nationality, street names, employee ID numbers, TRUE or FALSE	Quality of Diamond, academic grades, magnitude of earthquakes, etc.	Temperature in Celsius, Fahrenheit, Calendar dates, latitudes	Age, temperature in Kelvin, counts, length, weight
Associated Operations	=, ≠	=, ≠ < , ≤, > , ≥	=, ≠ < , ≤, > , ≥ +, −	=, ≠ < , ≤, > , ≥ +, −, ×, ÷

Numeric, Character, and Logical Data Types

- R supports numeric, character, and Logical (Boolean) data types
- Example of such code



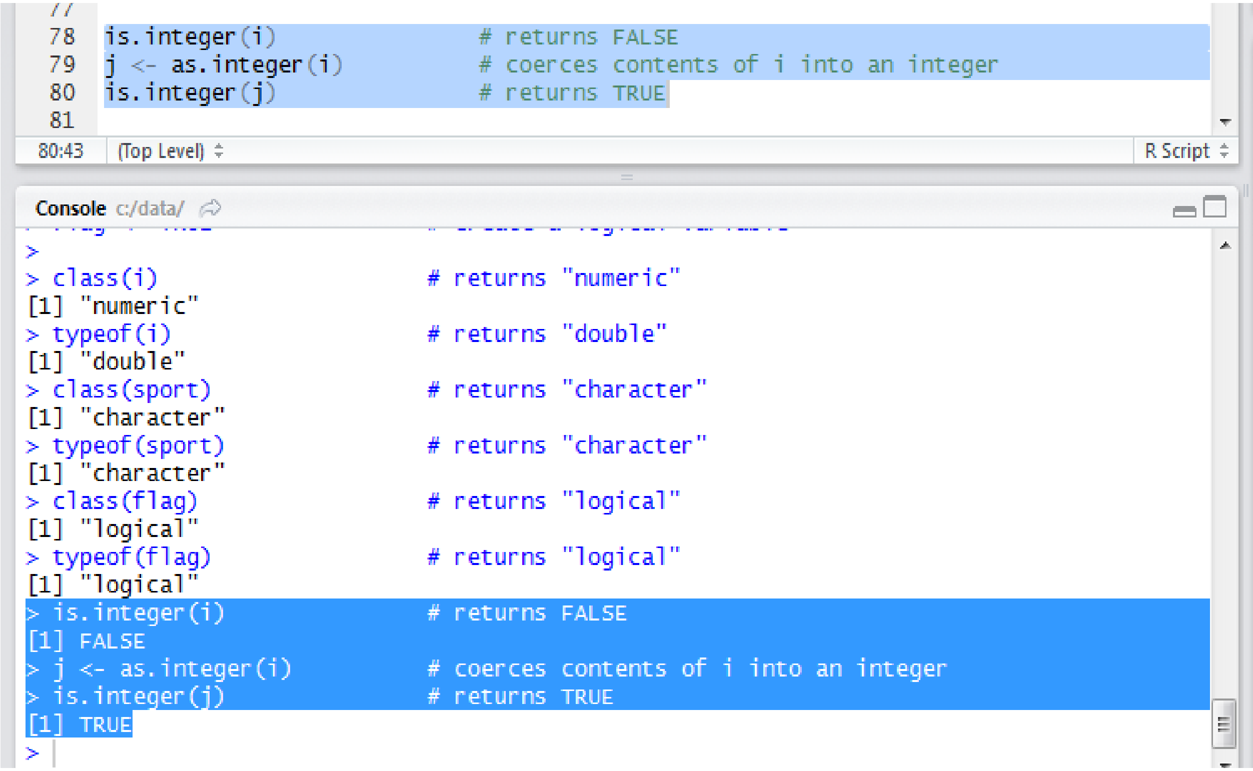
```
# Numeric, Character, and Logical Data Types
i <- 1                          # create a numeric variable
sport <- "football"             # create a character variable
flag <- TRUE                    # create a logical variable

class(i)                        # returns "numeric"
typeof(i)                       # returns "double"
class(sport)                    # returns "character"
typeof(sport)                   # returns "character"
class(flag)                     # returns "logical"
typeof(flag)                    # returns "logical"

is.integer(i)                   # returns FALSE
j <- as.integer(i)              # coerces contents of i into an integer
is.integer(j)                   # returns TRUE
```

```
> flag <- TRUE                # create a logical variable
> flag <- TRUE                # create a logical variable
> i <- 1                      # create a numeric variable
> sport <- "football"         # create a character variable
> flag <- TRUE                # create a logical variable
>
> class(i)                    # returns "numeric"
[1] "numeric"
> typeof(i)                   # returns "double"
[1] "double"
> class(sport)                # returns "character"
[1] "character"
> typeof(sport)               # returns "character"
[1] "character"
> class(flag)                 # returns "logical"
[1] "logical"
> typeof(flag)                # returns "logical"
[1] "logical"
>
```

Coerce a variable into a specific type

is.integer() function can check if a variable is integer or not?

as.integer() can coerce content of a variable into integer (see R code highlights below)

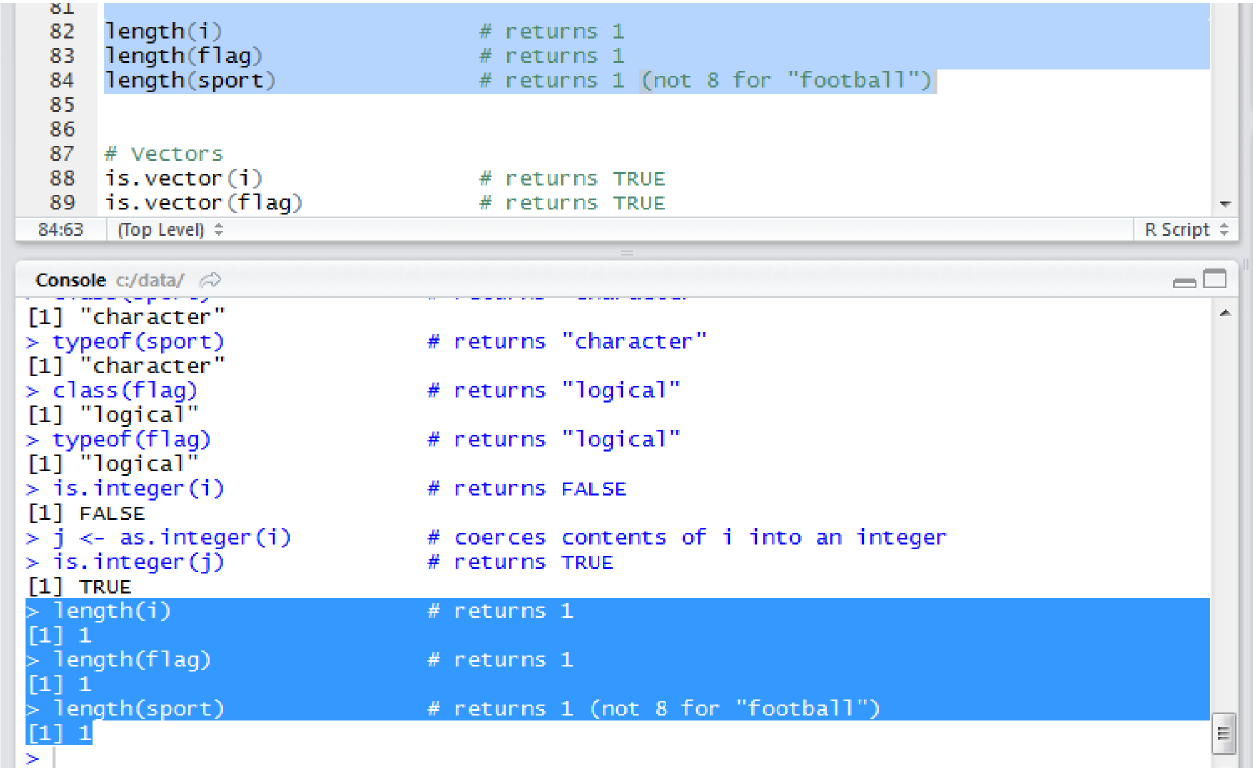


```
is.integer(i)              # returns FALSE
j <- as.integer(i)         # coerces contents of i into an integer
is.integer(j)              # returns TRUE
```

```
> flag <- TRUE                 # create a logical variable
>
> class(i)                     # returns "numeric"
[1] "numeric"
> typeof(i)                    # returns "double"
[1] "double"
> class(sport)                 # returns "character"
[1] "character"
> typeof(sport)                # returns "character"
[1] "character"
> class(flag)                  # returns "logical"
[1] "logical"
> typeof(flag)                 # returns "logical"
[1] "logical"
> is.integer(i)                # returns FALSE
[1] FALSE
> j <- as.integer(i)           # coerces contents of i into an integer
> is.integer(j)                # returns TRUE
[1] TRUE
>
```

length() function

length() function reveals the length of a variable. In the example below even Sport (“Football”) variable has a length of 1 because it is an element of a vector



```
length(i)                # returns 1
length(flag)             # returns 1
length(sport)            # returns 1 (not 8 for "football")


# Vectors
is.vector(i)             # returns TRUE
is.vector(flag)          # returns TRUE
```

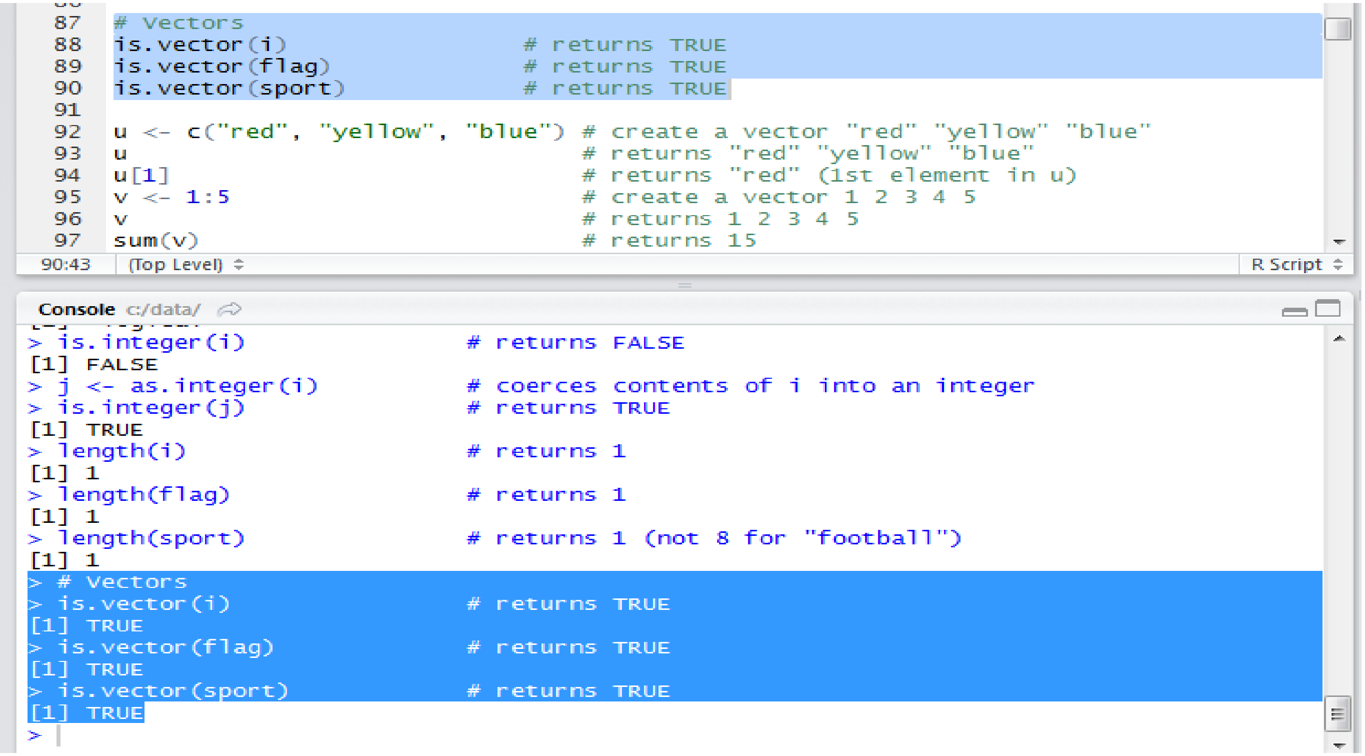
```
[1] "character"
> typeof(sport)         # returns "character"
[1] "character"
> class(flag)           # returns "logical"
[1] "logical"
> typeof(flag)          # returns "logical"
[1] "logical"
> is.integer(i)         # returns FALSE
[1] FALSE
> j <- as.integer(i)    # coerces contents of i into an integer
> is.integer(j)         # returns TRUE
[1] TRUE
> length(i)             # returns 1
[1] 1
> length(flag)          # returns 1
[1] 1
> length(sport)         # returns 1 (not 8 for "football")
[1] 1
>
```

Attributes & Data Types

VECTORS

Vectors

- Vectors are basic building block for data in R
- A vector can only consist of values in the same class e.g. Vector days (Mon, Tue, Wed, Thu, Fri, Sat, Sun)

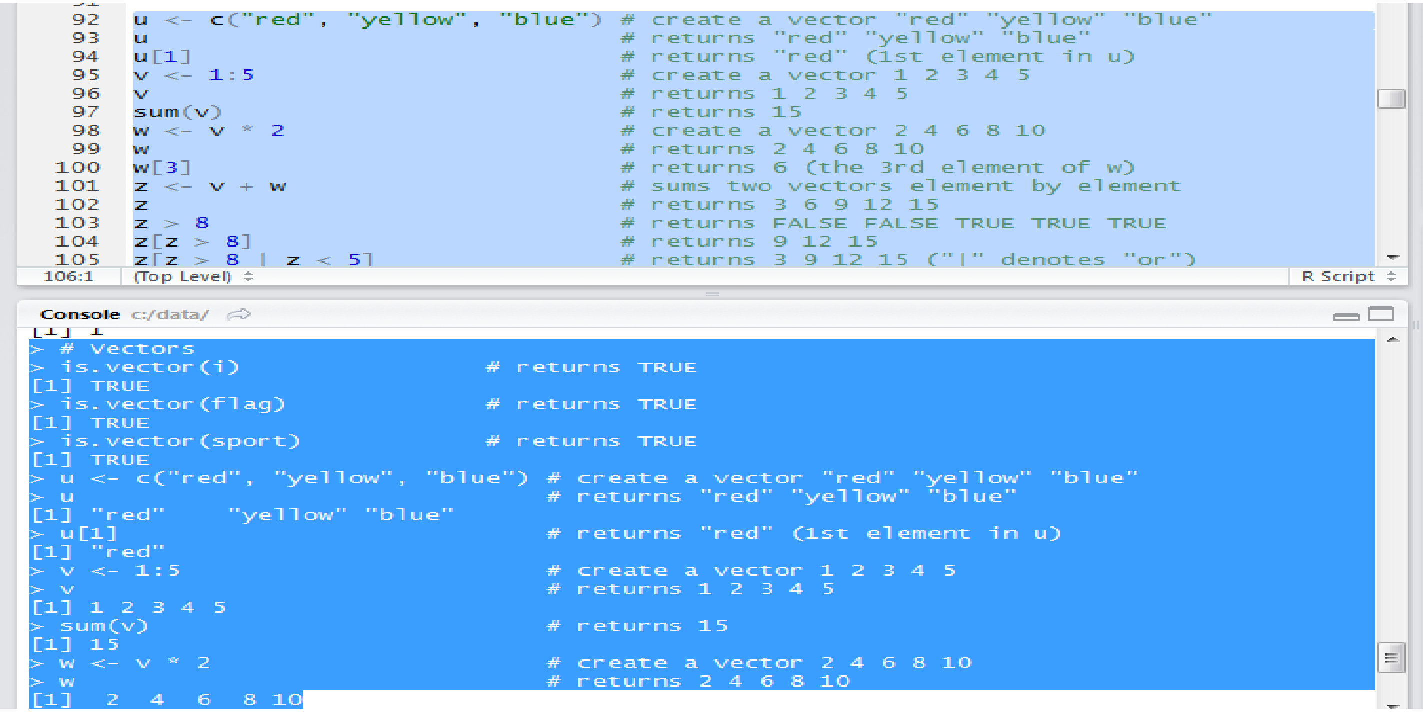


```
# Vectors
is.vector(i)                  # returns TRUE
is.vector(flag)               # returns TRUE
is.vector(sport)              # returns TRUE

u <- c("red", "yellow", "blue") # create a vector "red" "yellow" "blue"
u                               # returns "red" "yellow" "blue"
u[1]                            # returns "red" (1st element in u)
v <- 1:5                        # create a vector 1 2 3 4 5
v                               # returns 1 2 3 4 5
sum(v)                          # returns 15
```

```
> is.integer(i)             # returns FALSE
[1] FALSE
> j <- as.integer(i)        # coerces contents of i into an integer
> is.integer(j)             # returns TRUE
[1] TRUE
> length(i)                 # returns 1
[1] 1
> length(flag)              # returns 1
[1] 1
> length(sport)             # returns 1 (not 8 for "football")
[1] 1
> # Vectors
> is.vector(i)              # returns TRUE
[1] TRUE
> is.vector(flag)           # returns TRUE
[1] TRUE
> is.vector(sport)          # returns TRUE
[1] TRUE
> 
```

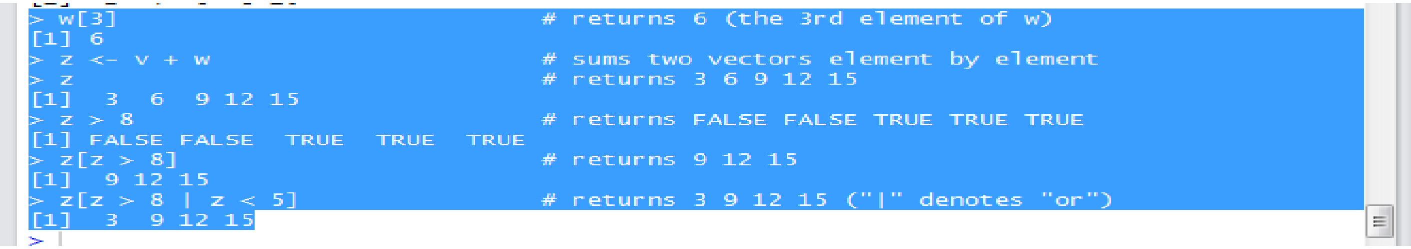
Create a vector and operate on it

Example below shows creation and operations on vectors data type



```
u <- c("red", "yellow", "blue") # create a vector "red" "yellow" "blue"
u                               # returns "red" "yellow" "blue"
u[1]                            # returns "red" (1st element in u)
v <- 1:5                        # create a vector 1 2 3 4 5
v                               # returns 1 2 3 4 5
sum(v)                          # returns 15
w <- v * 2                      # create a vector 2 4 6 8 10
w                               # returns 2 4 6 8 10
w[3]                            # returns 6 (the 3rd element of w)
z <- v + w                      # sums two vectors element by element
z                               # returns 3 6 9 12 15
z > 8                           # returns FALSE FALSE TRUE TRUE TRUE
z[z > 8]                        # returns 9 12 15
z[z > 8 | z < 5]                # returns 3 9 12 15 ("|" denotes "or")
```

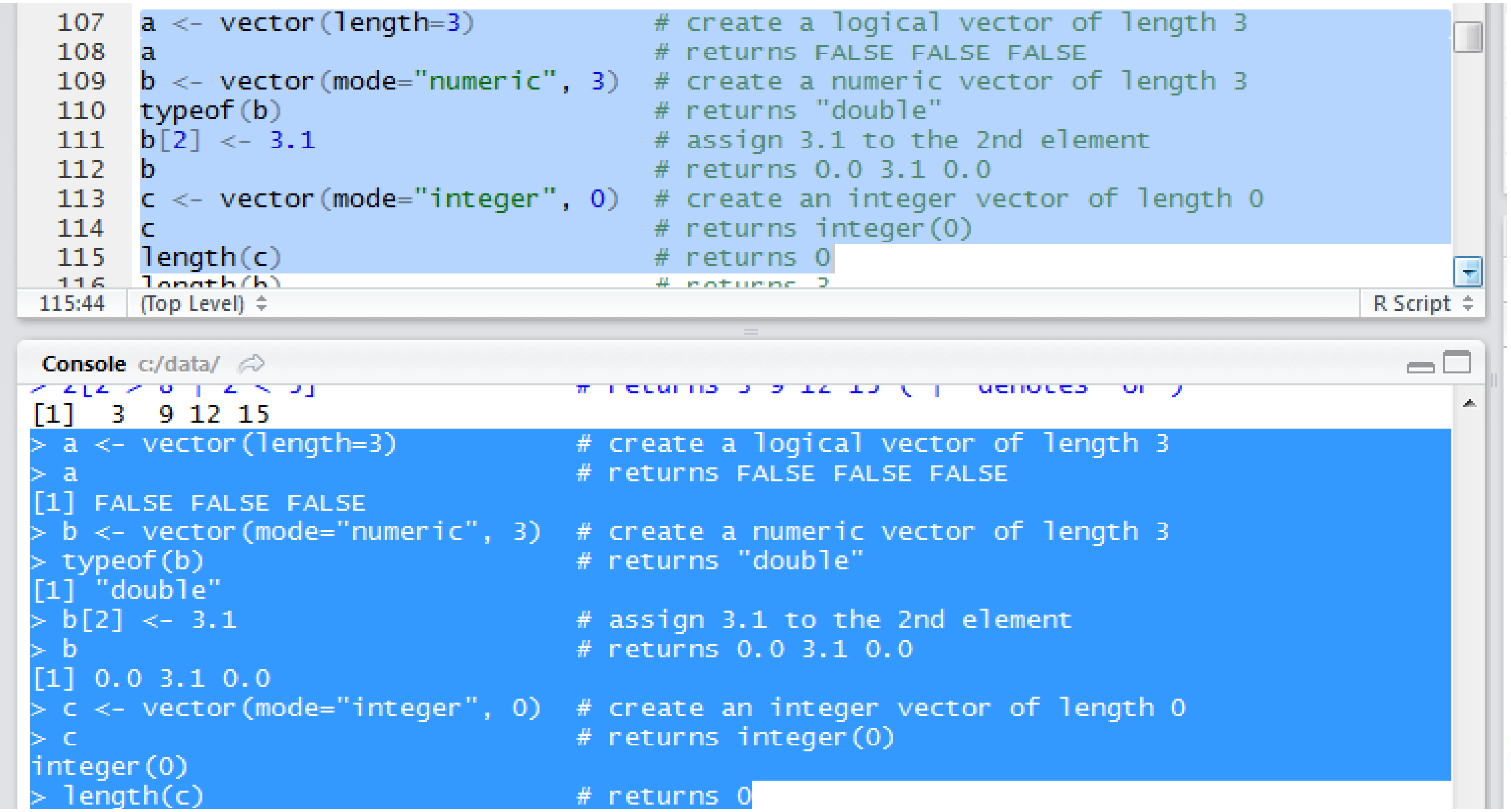
```
> # vectors
> is.vector(i)                  # returns TRUE
[1] TRUE
> is.vector(flag)               # returns TRUE
[1] TRUE
> is.vector(sport)              # returns TRUE
[1] TRUE
> u <- c("red", "yellow", "blue") # create a vector "red" "yellow" "blue"
> u                               # returns "red" "yellow" "blue"
[1] "red"    "yellow" "blue"
> u[1]                            # returns "red" (1st element in u)
[1] "red"
> v <- 1:5                        # create a vector 1 2 3 4 5
> v                               # returns 1 2 3 4 5
[1] 1 2 3 4 5
> sum(v)                          # returns 15
[1] 15
> w <- v * 2                      # create a vector 2 4 6 8 10
> w                               # returns 2 4 6 8 10
[1]  2  4  6  8 10
```



```
> w[3]                            # returns 6 (the 3rd element of w)
[1] 6
> z <- v + w                      # sums two vectors element by element
> z                               # returns 3 6 9 12 15
[1]  3  6  9 12 15
> z > 8                           # returns FALSE FALSE TRUE TRUE TRUE
[1] FALSE FALSE  TRUE  TRUE  TRUE
> z[z > 8]                        # returns 9 12 15
[1]  9 12 15
> z[z > 8 | z < 5]                # returns 3 9 12 15 ("|" denotes "or")
[1]  3  9 12 15
> 
```

Create a Vector with fixed length

- *Vector(length="value")* function create a logical vector in which length could be fixed like in the example below
- You can add new values in the vector
- A vector can be of a different type by using *mode* parameter as shown below



```
a <- vector(length=3)           # create a logical vector of length 3
a                               # returns FALSE FALSE FALSE
b <- vector(mode="numeric", 3)  # create a numeric vector of length 3
typeof(b)                       # returns "double"
b[2] <- 3.1                     # assign 3.1 to the 2nd element
b                               # returns 0.0 3.1 0.0
c <- vector(mode="integer", 0)  # create an integer vector of length 0
c                               # returns integer(0)
length(c)                       # returns 0
length(b)                       # returns 3
```

```
> z[z > 8 | z < 5]              # returns 3 9 12 15 ('|' denotes 'or')
[1]  3  9 12 15
> a <- vector(length=3)           # create a logical vector of length 3
> a                               # returns FALSE FALSE FALSE
[1] FALSE FALSE FALSE
> b <- vector(mode="numeric", 3)  # create a numeric vector of length 3
> typeof(b)                       # returns "double"
[1] "double"
> b[2] <- 3.1                     # assign 3.1 to the 2nd element
> b                               # returns 0.0 3.1 0.0
[1] 0.0 3.1 0.0
> c <- vector(mode="integer", 0)  # create an integer vector of length 0
> c                               # returns integer(0)
integer(0)
> length(c)                       # returns 0
```

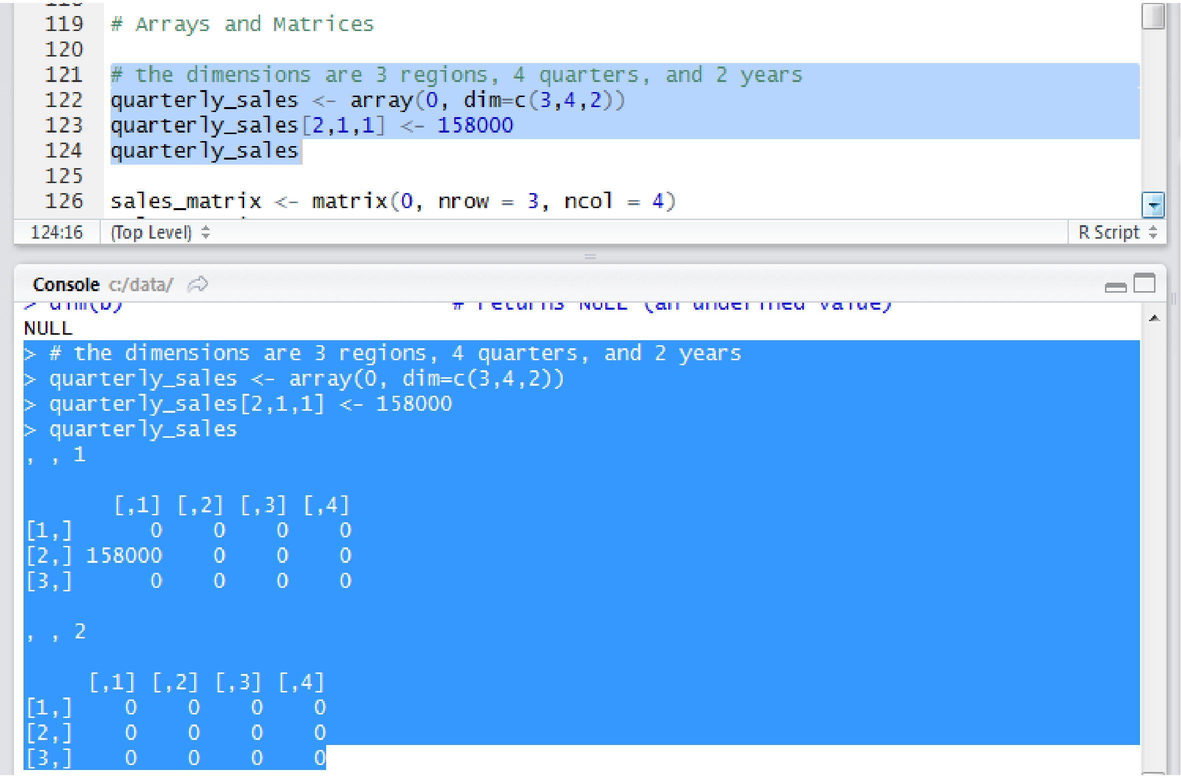
Attributes & Data Types

ARRAYS and MATRICES

Arrays

arrays() function can be used to restructure a vector as an array

Highlighted R code below builds a three dimensional array to hold the quarterly sales for three region over two-year period

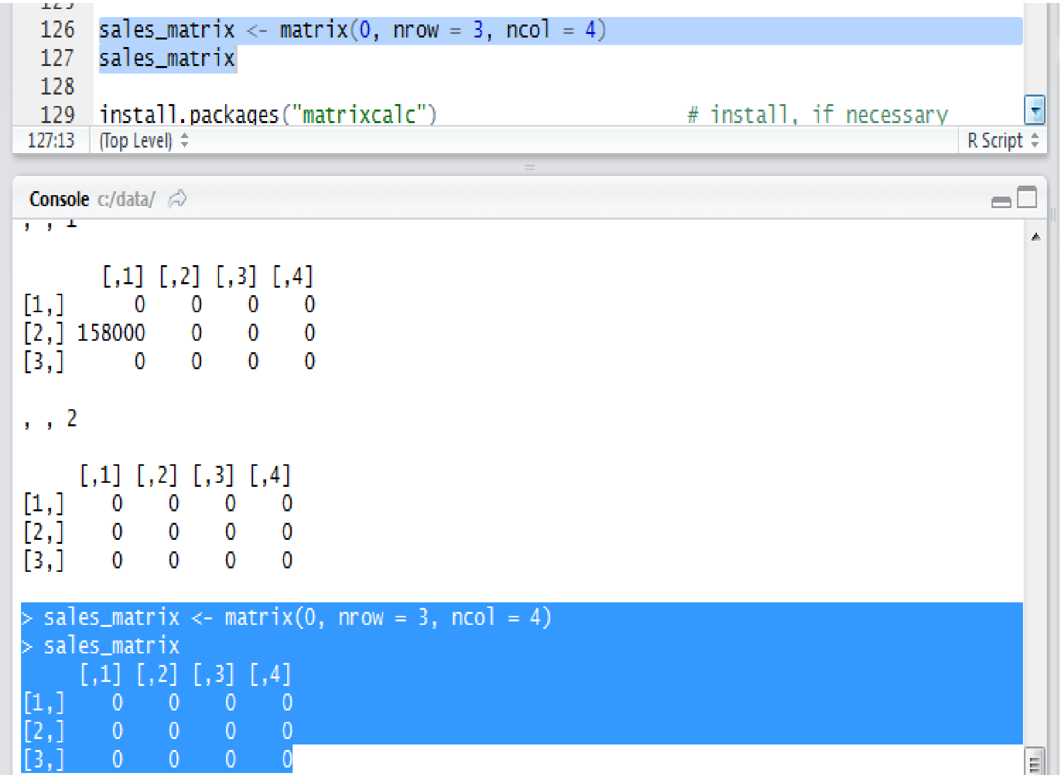


```
# Arrays and Matrices

# the dimensions are 3 regions, 4 quarters, and 2 years
quarterly_sales <- array(0, dim=c(3,4,2))
quarterly_sales[2,1,1] <- 158000
quarterly_sales

sales_matrix <- matrix(0, nrow = 3, ncol = 4)
```

```
> dim(b)                # returns NULL (an undefined value)
NULL
> # the dimensions are 3 regions, 4 quarters, and 2 years
> quarterly_sales <- array(0, dim=c(3,4,2))
> quarterly_sales[2,1,1] <- 158000
> quarterly_sales
, , 1

       [,1] [,2] [,3] [,4]
[1,]      0    0    0    0
[2,] 158000    0    0    0
[3,]      0    0    0    0

, , 2

     [,1] [,2] [,3] [,4]
[1,]    0    0    0    0
[2,]    0    0    0    0
[3,]    0    0    0    0
```

Matrix

- A two dimension array is known as a **matrix**
- The following R code initializes a matrix to hold the quarterly sale for three regions. It uses *matrix()* function with *nrow* & *ncol* as parameters to define number of rows and columns respectively for *Sales_matrix*

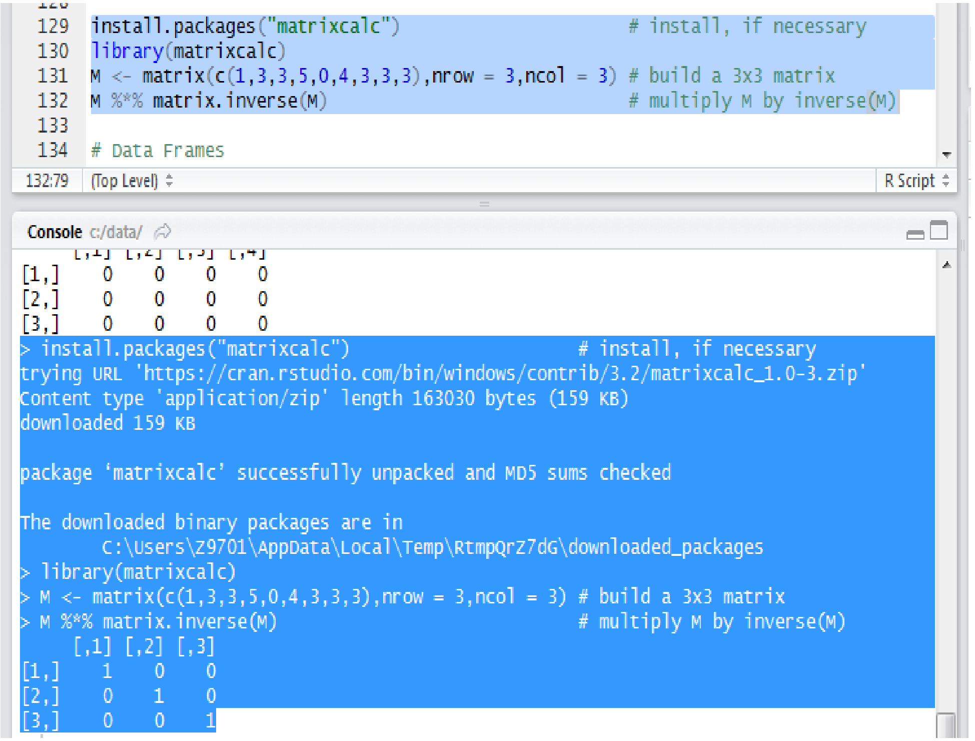


```
sales_matrix <- matrix(0, nrow = 3, ncol = 4)
sales_matrix

install.packages("matrixcalc")                    # install, if necessary
```

```
, , 1

     [,1] [,2] [,3] [,4]
[1,]      0    0    0    0
[2,] 158000    0    0    0
[3,]      0    0    0    0

, , 2

     [,1] [,2] [,3] [,4]
[1,]    0    0    0    0
[2,]    0    0    0    0
[3,]    0    0    0    0

> sales_matrix <- matrix(0, nrow = 3, ncol = 4)
> sales_matrix
     [,1] [,2] [,3] [,4]
[1,]    0    0    0    0
[2,]    0    0    0    0
[3,]    0    0    0    0
```

Operations on Matrix

- R provides the standard matrix operations such as addition, subtraction, and multiplication as well the transpose function t() and the inverse matrix function *matrix.inverse()*
- For this to run you need to include *the matrixcalc* package



```
install.packages("matrixcalc")                          # install, if necessary
library(matrixcalc)
M <- matrix(c(1,3,3,5,0,4,3,3,3),nrow = 3,ncol = 3) # build a 3x3 matrix
M %*% matrix.inverse(M)                                 # multiply M by inverse(M)

# Data Frames
```

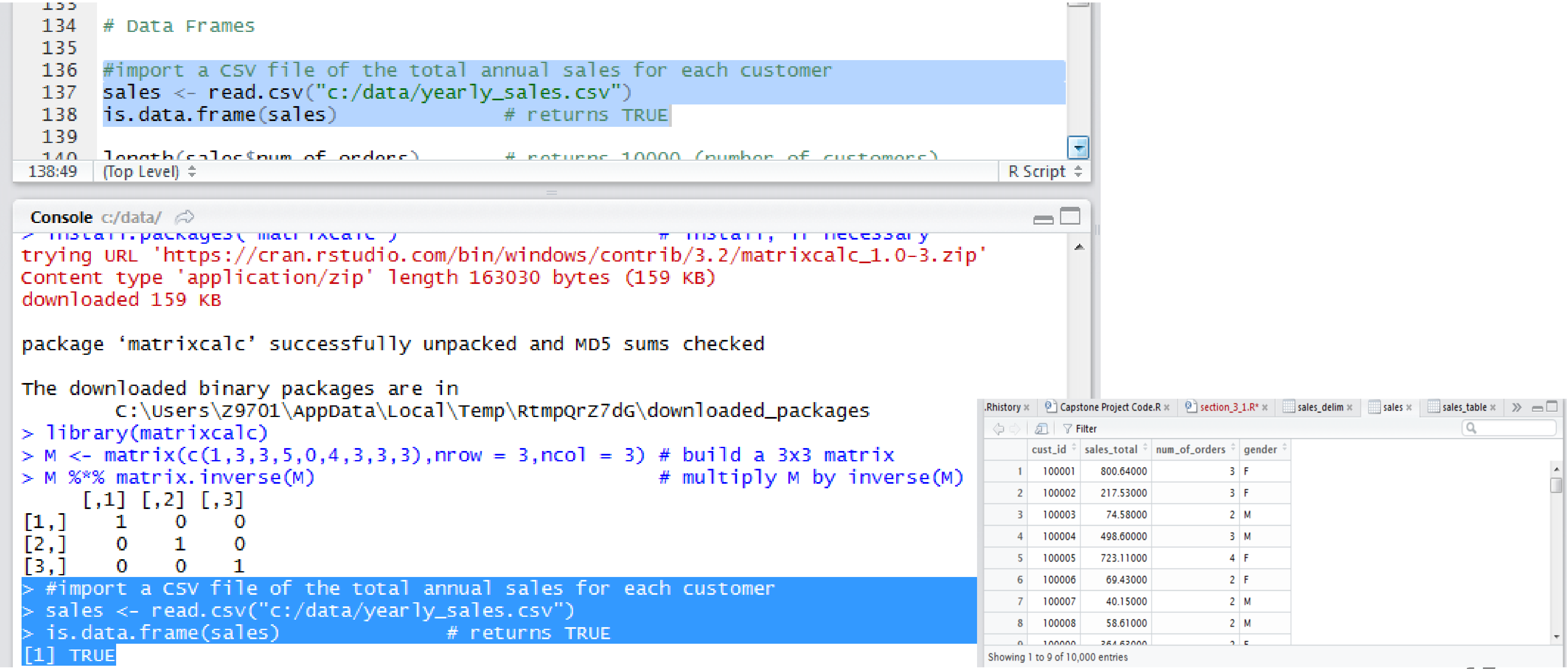
```
     [,1] [,2] [,3] [,4]
[1,]    0    0    0    0
[2,]    0    0    0    0
[3,]    0    0    0    0
> install.packages("matrixcalc")                          # install, if necessary
trying URL 'https://cran.rstudio.com/bin/windows/contrib/3.2/matrixcalc_1.0-3.zip'
Content type 'application/zip' length 163030 bytes (159 KB)
downloaded 159 KB

package ‘matrixcalc’ successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\Z9701\AppData\Local\Temp\RtmpQrZ7dG\downloaded_packages
> library(matrixcalc)
> M <- matrix(c(1,3,3,5,0,4,3,3,3),nrow = 3,ncol = 3) # build a 3x3 matrix
> M %*% matrix.inverse(M)                                 # multiply M by inverse(M)
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1
```

Attributes and Data Types

Data Frames

Data Frames

- Similar to the concepts, data frames provide a structure for storing and accessing several variables of different data types,
- In fact *read.csv()* in the accompanying R example created data frames containing different variable e.g. cust_ID, sales_total, gender, etc..



```
# Data Frames

#import a CSV file of the total annual sales for each customer
sales <- read.csv("c:/data/yearly_sales.csv")
is.data.frame(sales)                # returns TRUE

length(sales$num_of_orders)         # returns 10000 (number of customers)
```

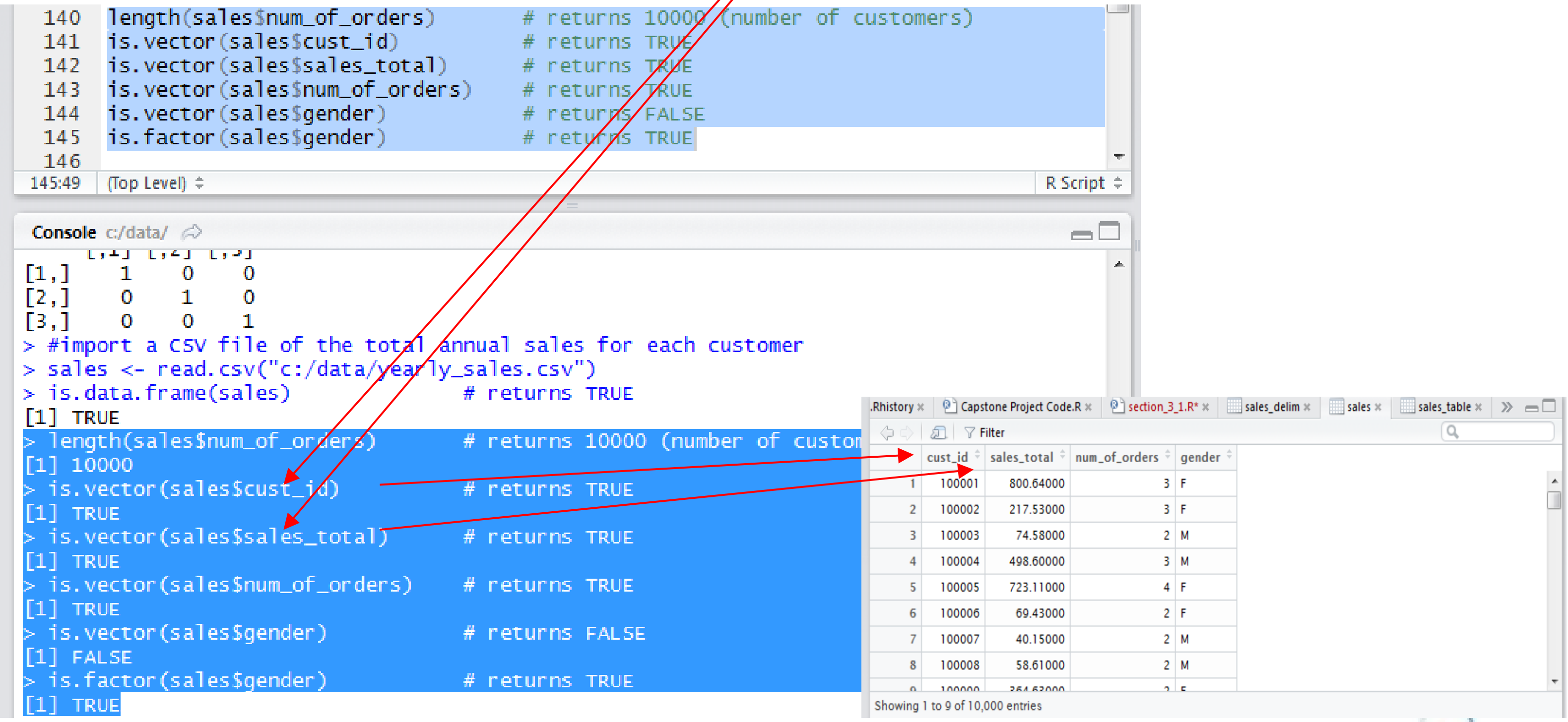
```
> install.packages("matrixcalc")                    # install, if necessary
trying URL 'https://cran.rstudio.com/bin/windows/contrib/3.2/matrixcalc_1.0-3.zip'
Content type 'application/zip' length 163030 bytes (159 KB)
downloaded 159 KB

package ‘matrixcalc’ successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\Z9701\AppData\Local\Temp\RtmpQrZ7dG\downloaded_packages
> library(matrixcalc)
> M <- matrix(c(1,3,3,5,0,4,3,3,3),nrow = 3,ncol = 3) # build a 3x3 matrix
> M %*% matrix.inverse(M)                           # multiply M by inverse(M)
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1
> #import a CSV file of the total annual sales for each customer
> sales <- read.csv("c:/data/yearly_sales.csv")
> is.data.frame(sales)                # returns TRUE
[1] TRUE
```

	cust_id	sales_total	num_of_orders	gender
1	100001	800.64000	3	F
2	100002	217.53000	3	F
3	100003	74.58000	2	M
4	100004	498.60000	3	M
5	100005	723.11000	4	F
6	100006	69.43000	2	F
7	100007	40.15000	2	M
8	100008	58.61000	2	M
9	100009	364.63000	2	F

Showing 1 to 9 of 10,000 entries

Access through $ notation

- The data stored in data frame can be accessed using $ notation,



```
length(sales$num_of_orders)          # returns 10000 (number of customers)
is.vector(sales$cust_id)             # returns TRUE
is.vector(sales$sales_total)         # returns TRUE
is.vector(sales$num_of_orders)       # returns TRUE
is.vector(sales$gender)              # returns FALSE
is.factor(sales$gender)              # returns TRUE
```

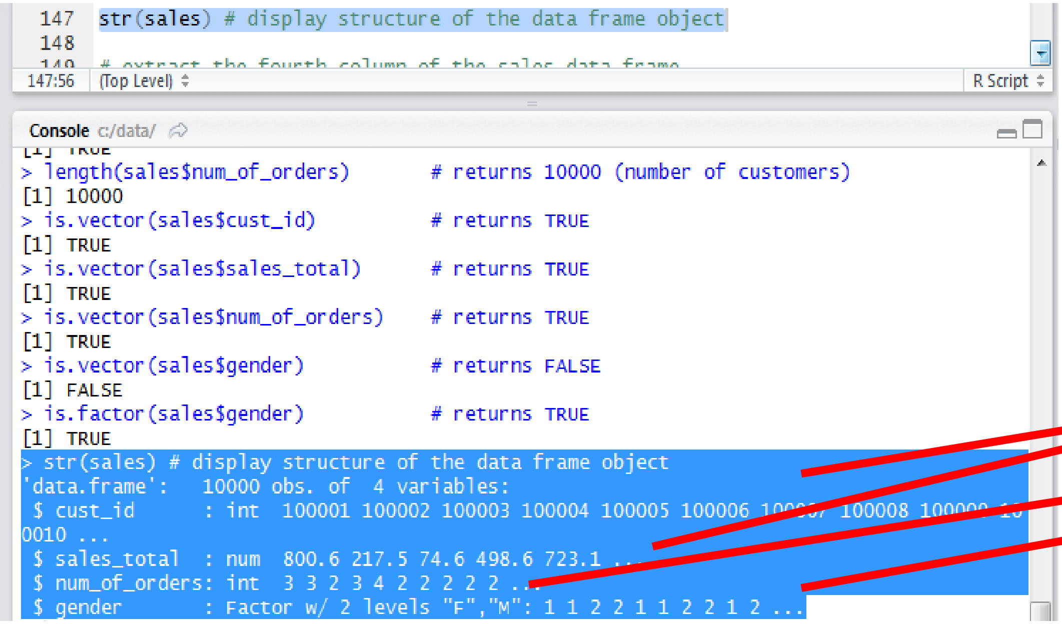
```
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1
> #import a csv file of the total annual sales for each customer
> sales <- read.csv("c:/data/yearly_sales.csv")
> is.data.frame(sales)                 # returns TRUE
[1] TRUE
> length(sales$num_of_orders)          # returns 10000 (number of custon
[1] 10000
> is.vector(sales$cust_id)             # returns TRUE
[1] TRUE
> is.vector(sales$sales_total)         # returns TRUE
[1] TRUE
> is.vector(sales$num_of_orders)       # returns TRUE
[1] TRUE
> is.vector(sales$gender)              # returns FALSE
[1] FALSE
> is.factor(sales$gender)              # returns TRUE
[1] TRUE
```

	cust_id	sales_total	num_of_orders	gender
1	100001	800.64000	3	F
2	100002	217.53000	3	F
3	100003	74.58000	2	M
4	100004	498.60000	3	M
5	100005	723.11000	4	F
6	100006	69.43000	2	F
7	100007	40.15000	2	M
8	100008	58.61000	2	M
9	100009	364.63000	2	F

Showing 1 to 9 of 10,000 entries

str() function

- Because of their flexibility to handle many data types, are the preferred input format for many of the modelling function,
- *str()* function display the structure & values of the data frame e.g. *Sales below*,
- This function identifies integer and numeric (double) data types, the factor variable and levels, as well as first value of each variable



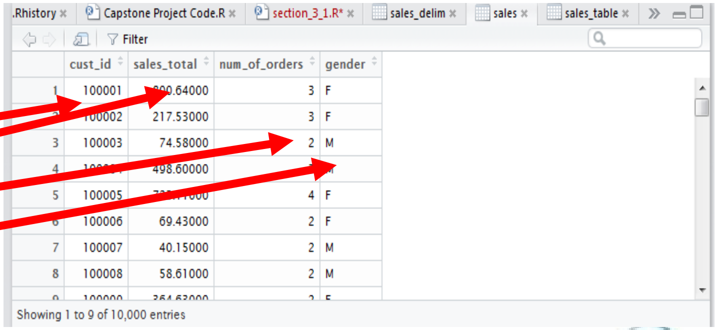
```
str(sales) # display structure of the data frame object

# extract the fourth column of the sales data frame
```

147:56 (Top Level) ÷ R Script ÷

Console c:/data/

```
[1] TRUE
> length(sales$num_of_orders)       # returns 10000 (number of customers)
[1] 10000
> is.vector(sales$cust_id)          # returns TRUE
[1] TRUE
> is.vector(sales$sales_total)      # returns TRUE
[1] TRUE
> is.vector(sales$num_of_orders)    # returns TRUE
[1] TRUE
> is.vector(sales$gender)           # returns FALSE
[1] FALSE
> is.factor(sales$gender)           # returns TRUE
[1] TRUE
> str(sales) # display structure of the data frame object
'data.frame':   10000 obs. of  4 variables:
 $ cust_id      : int  100001 100002 100003 100004 100005 100006 100007 100008 100009 10
0010 ...
 $ sales_total  : num  800.6 217.5 74.6 498.6 723.1 ...
 $ num_of_orders: int  3 3 2 3 4 2 2 2 2 2 ...
 $ gender       : Factor w/ 2 levels "F","M": 1 1 2 2 1 1 2 2 1 2 ...
```

Sales Data Frames



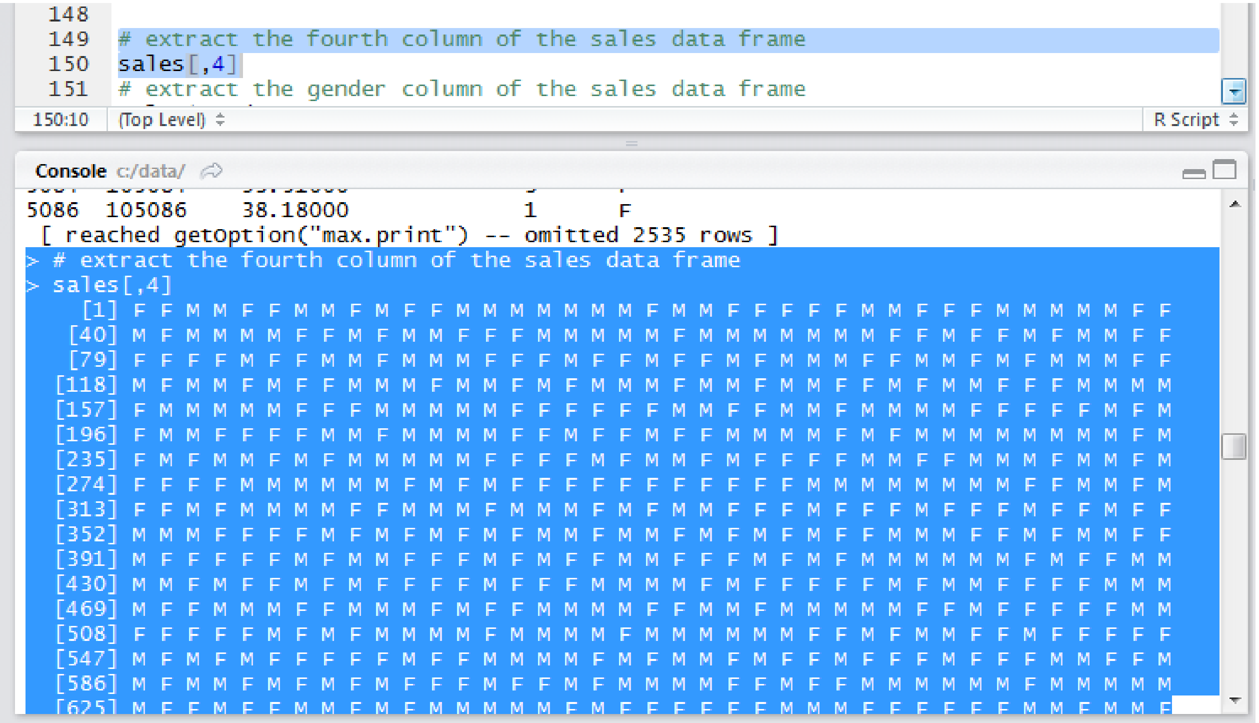
	cust_id	sales_total	num_of_orders	gender
1	100001	800.64000	3	F
2	100002	217.53000	3	F
3	100003	74.58000	2	M
4	100004	498.60000	3	M
5	100005	723.11000	4	F
6	100006	69.43000	2	F
7	100007	40.15000	2	M
8	100008	58.61000	2	M
9	100009	364.63000	2	F

Showing 1 to 9 of 10,000 entries

Retrieve values/data from data frames

- Data frames are lists of variables of the same length,
- Subset of the data frame can be retrieved from ***sub-setting operators***
- R’s sub-setting operators are powerful as they allow complex operations to retrieve subset of the data set

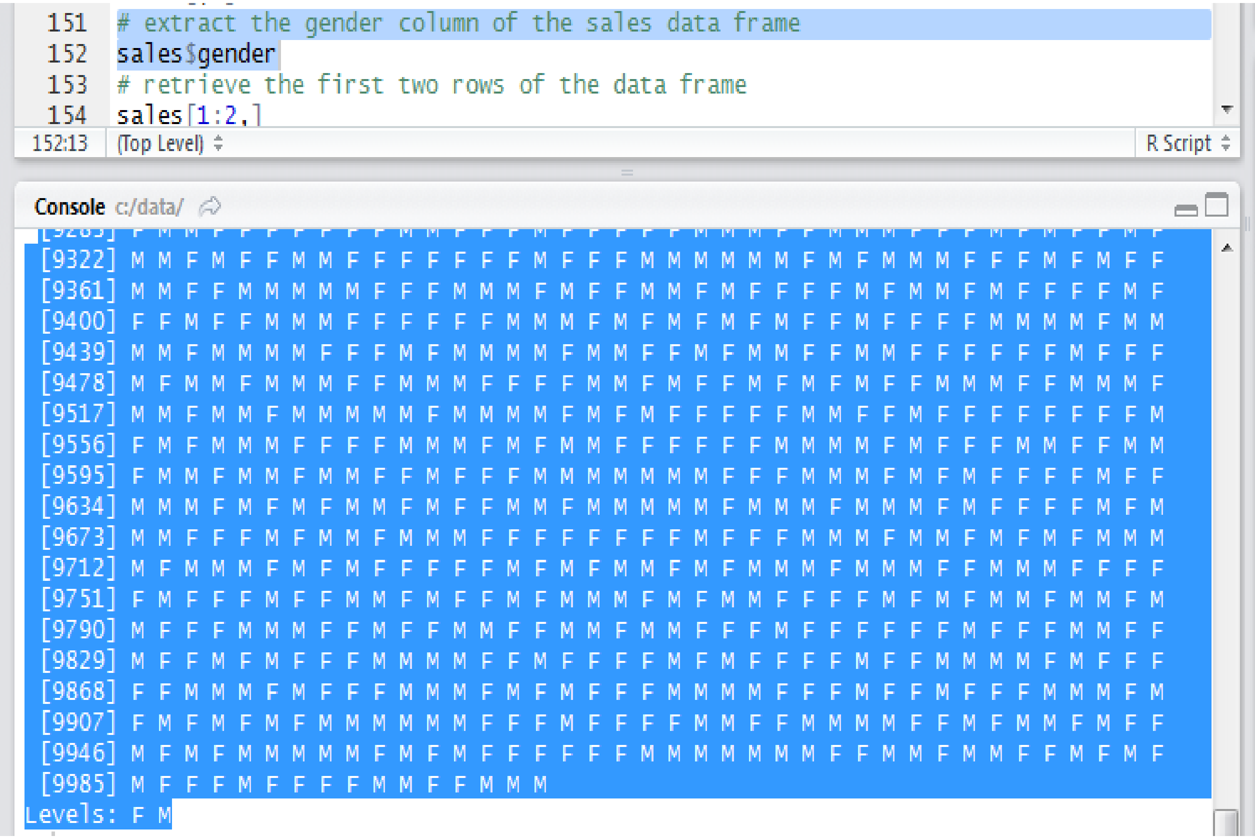
sales[,4]: Extract the fourth Column of Sales



```
# extract the fourth column of the sales data frame
sales[,4]
# extract the gender column of the sales data frame
```

```
5086  105086    38.18000            1       F
 [ reached getOption("max.print") -- omitted 2535 rows ]
> # extract the fourth column of the sales data frame
> sales[,4]
   [1] F F M M F F M M F M F F M M M M M M M F M M F F F F F M M F F F M M M M M F F
  [40] M F M M M M F F M F M M F F F M M M M M F M M M M M M M F F M F F M F M M F F
  [79] F F F F M F F M M F M M M F F F M F F M F F M F M M M F F M M F M F M M M F F
 [118] M F M M F M F F M M M F M M F M F M M M F M M F M M F F M F M M F F F M M M M
 [157] F M M M M M F F F M M M M M F F F F F F M M F F M M F M M M M F F F F F M F M
 [196] F M M F F F F M M F M M M M F F M F F M F F M M M M F M F M M M M M M M M F M
 [235] F M F M M F M F M M M M M F F F F M F M M F M F F F F M M F F M M M F M M F M
 [274] F F F F M M M M M M F M F M F F F F F F F F F F F M M M M M M M M F F M M F M
 [313] F F M F M M M M F F M M M F M M M F M F M M F F F M F F F M F F F M F F M F F
 [352] M M M F F F F M F M F M F M F M M F M M F M F M F M F F M M M F F M F M M F F
 [391] M F F F F F M F M M F F F M F M F F M M F F F M F M F M M M M M M F M F F M M
 [430] M M F M F F M F M F F F F M F F F M M M M F M F F F F F M F M M F F F F M M M
 [469] M F F M M M F F M M M F M F F M M M M F F M M F M M M M M F F M F F F F F M M
 [508] F F F F F M F M F M M M M F M M M M F M M M M M M F F M F M M F F M F F F F F
 [547] M F M F M F F F F F M F F M M M M F M F M M F M F F M F F F M F F F M M F F M
 [586] M F M M F M F M F M F F F M F F M F M M M M F F M F F M M M M M F M M M M M M
 [625] M F F M F F M M F M F M M M M M F M F F F F F F M M M F F F F F F M M F M M F
```

sales$gender retrieves gender column as shown below

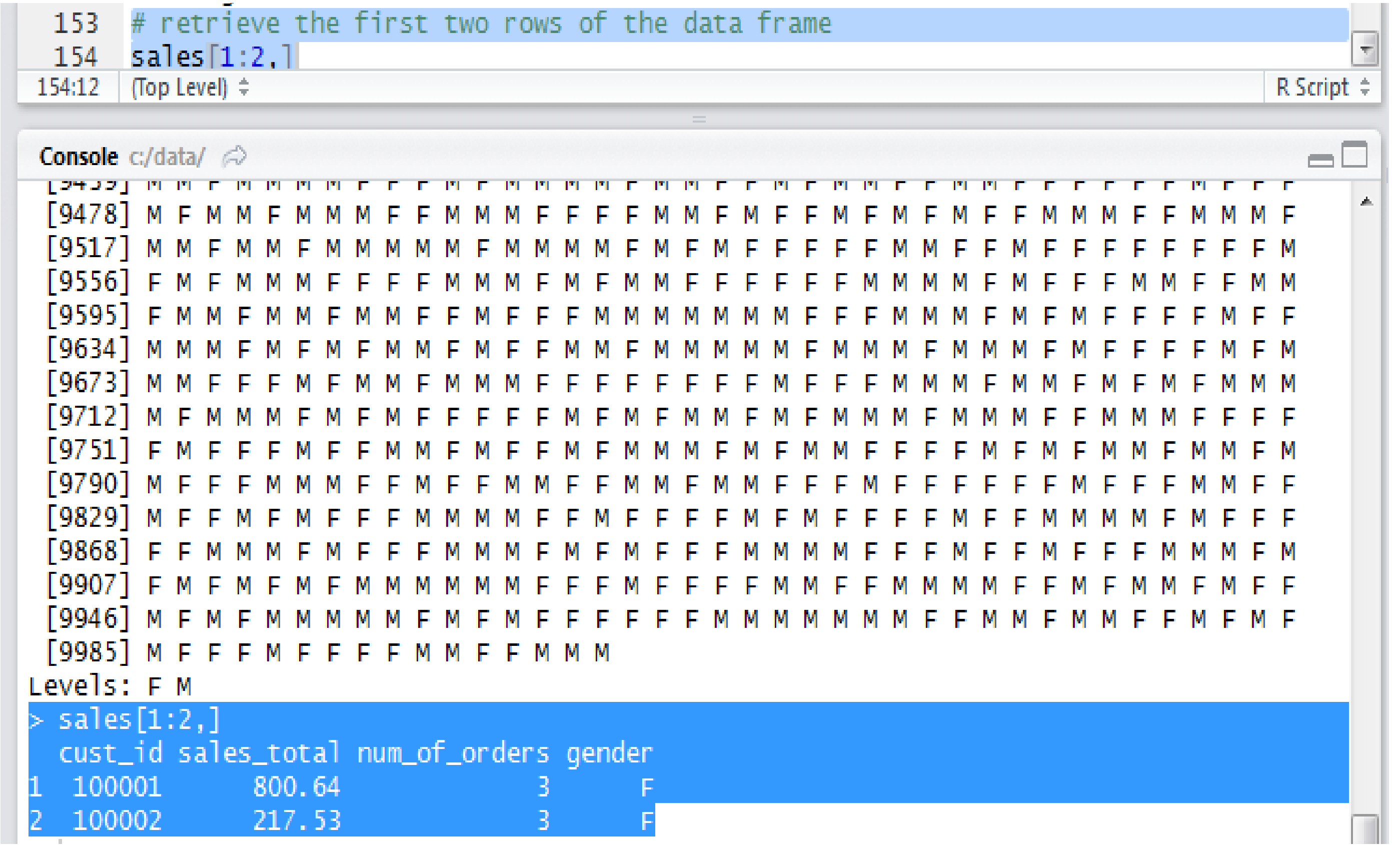


```
# extract the gender column of the sales data frame
sales$gender
# retrieve the first two rows of the data frame
sales[1:2,]
```

```
 [9322] M M F M F F M M F F F F F F F M F F F M M M M M M F M F M M M F F F M F M F F
 [9361] M M F F M M M M M F F F M M M F M F F M M F M F F F F M F M M F M F F F F M F
 [9400] F F M F F M M M F F F F F F M M M F M F M F M F M F F M F F F F M M M M F M M
 [9439] M M F M M M M F F F M F M M M M F M M F F M F M M F F M M F F F F F F M F F F
 [9478] M F M M F M M M F F M M M F F F F M M F M F F M F M F M F F M M M F F M M M F
 [9517] M M F M M F M M M M M F M M M M F M F M F F F F F M M F F M F F F F F F F F M
 [9556] F M F M M M F F F F M M M F M F M M F F F F F F M M M M F M F F F M M F F M M
 [9595] F M M F M M F M M F F M F F F M M M M M M M F F F M M M F M F M F F F F M F F
 [9634] M M M F M F M F M M F M F F M M F M M M M M F M M M F M M M F M F F F F M F M
 [9673] M M F F F M F M M F M M M F F F F F F F F M F F F M M M F M M F M F M F M M M
 [9712] M F M M M F M F M F F F F F M F M F M M F M F M M M F M M M F F M M M F F F F
 [9751] F M F F F M F F M M F M F F M F M M M F M F M M F F F F M F M F M M F M M F M
 [9790] M F F F M M M F F M F F M M F F M M F M M F F F M F F F F F F M F F F M M F F
 [9829] M F F M F M F F F M M M M F F M F F F F M F M F F F F M F F M M M M F M F F F
 [9868] F F M M M F M F F F M M M F M F M F F F M M M M F F F M F F M F F F M M M F M
 [9907] F M F M F M F M M M M M M F F F M F F F F M M F F M M M M F F M F M M F M F F
 [9946] M F M F M M M M M F M F M F F F F F F M M M M M M M F F M M F M M F F M F M F
 [9985] M F F F M F F F F M M F F M M M
Levels: F M
```

Retrieve rows

sales[1:2,] retrieves the first two rows

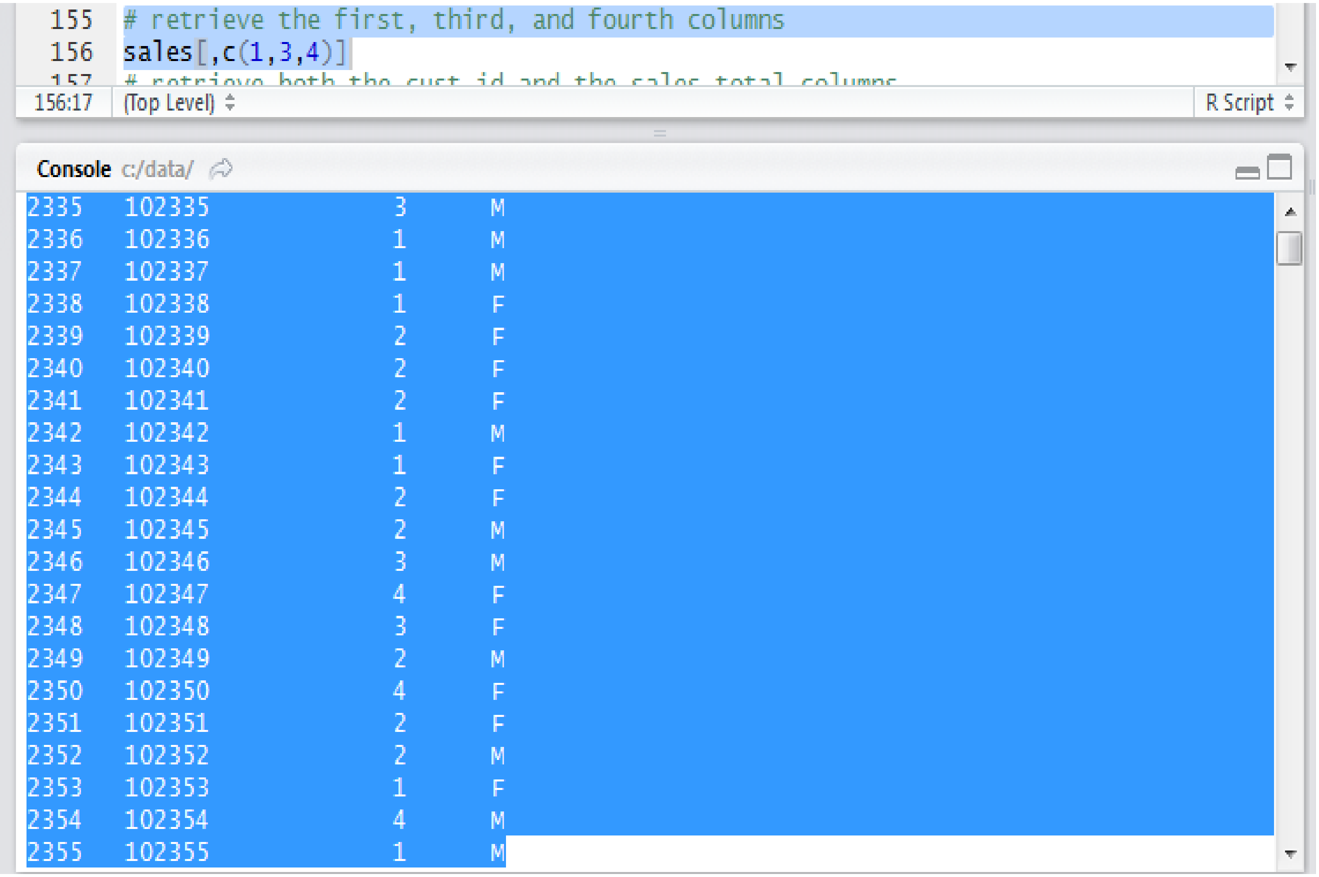


```
# retrieve the first two rows of the data frame
sales[1:2,]
```

```
 [9478] M F M M F M M M F F M M M F F F F M M F M F F M F M F M F F M M M F F M M M F
 [9517] M M F M M F M M M M M F M M M M F M F M F F F F F M M F F M F F F F F F F F M
 [9556] F M F M M M F F F F M M M F M F M M F F F F F F M M M M F M F F F M M F F M M
 [9595] F M M F M M F M M F F M F F F M M M M M M M F F F M M M F M F M F F F F M F F
 [9634] M M M F M F M F M M F M F F M M F M M M M M F M M M F M M M F M F F F F M F M
 [9673] M M F F F M F M M F M M M F F F F F F F F M F F F M M M F M M F M F M F M M M
 [9712] M F M M M F M F M F F F F F M F M F M M F M F M M M F M M M F F M M M F F F F
 [9751] F M F F F M F F M M F M F F M F M M M F M F M M F F F F M F M F M M F M M F M
 [9790] M F F F M M M F F M F F M M F F M M F M M F F F M F F F F F F M F F F M M F F
 [9829] M F F M F M F F F M M M M F F M F F F F M F M F F F F M F F M M M M F M F F F
 [9868] F F M M M F M F F F M M M F M F M F F F M M M M F F F M F F M F F F M M M F M
 [9907] F M F M F M F M M M M M M F F F M F F F F M M F F M M M M F F M F M M F M F F
 [9946] M F M F M M M M M F M F M F F F F F F M M M M M M M F F M M F M M F F M F M F
 [9985] M F F F M F F F F M M F F M M M
Levels: F M
> sales[1:2,]
  cust_id sales_total num_of_orders gender
1  100001      800.64             3      F
2  100002      217.53             3      F
```

Retrieve selected Columns

- *sales [,c(1,2,3)]* retrieves 1, 2, and 3rd columns

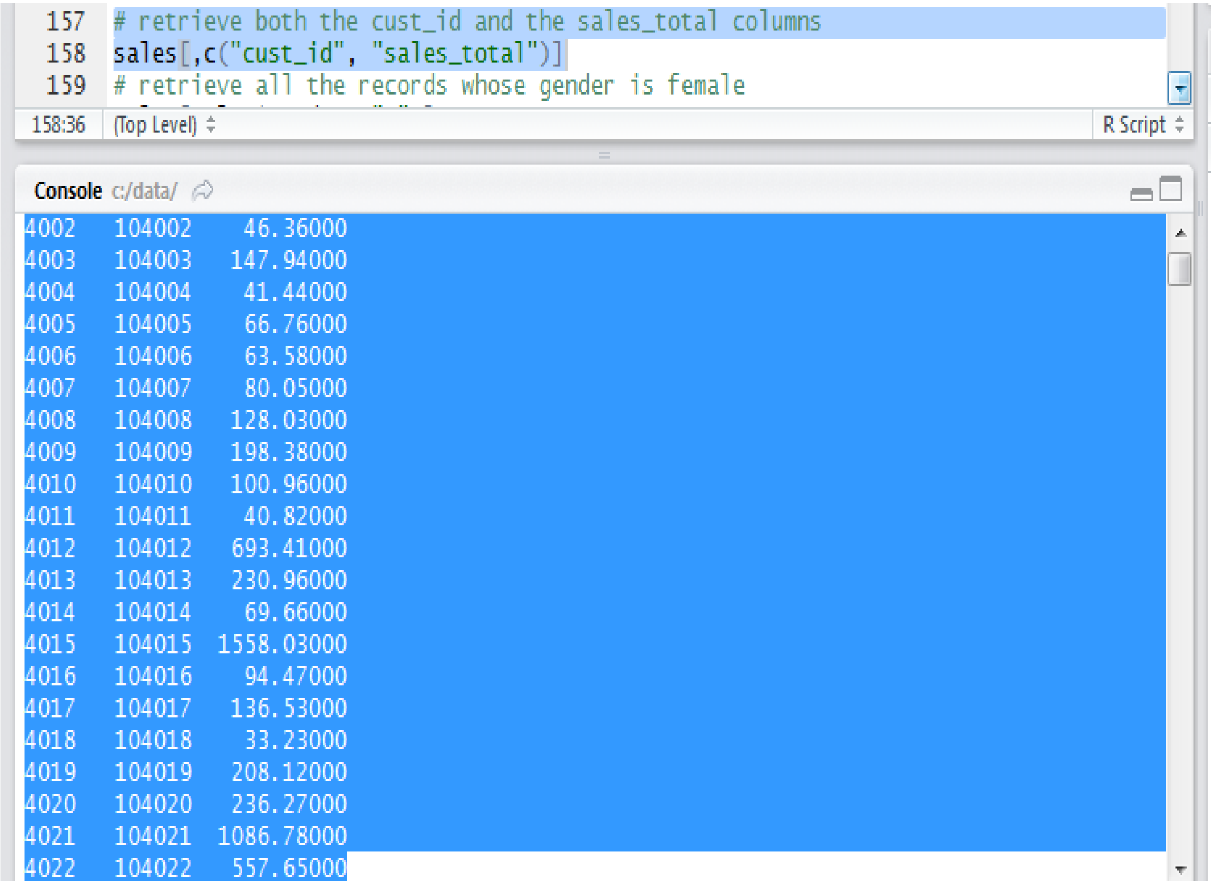


```
# retrieve the first, third, and fourth columns
sales[,c(1,3,4)]
# retrieve both the cust id and the sales total columns
```

```
2335   102335          3      M
2336   102336          1      M
2337   102337          1      M
2338   102338          1      F
2339   102339          2      F
2340   102340          2      F
2341   102341          2      F
2342   102342          1      M
2343   102343          1      F
2344   102344          2      F
2345   102345          2      M
2346   102346          3      M
2347   102347          4      F
2348   102348          3      F
2349   102349          2      M
2350   102350          4      F
2351   102351          2      F
2352   102352          2      M
2353   102353          1      F
2354   102354          4      M
2355   102355          1      M
```

Retrieve named Columns

sales[,c("cust_id", "sales_total")] retrieves cust_id and sales_total columns

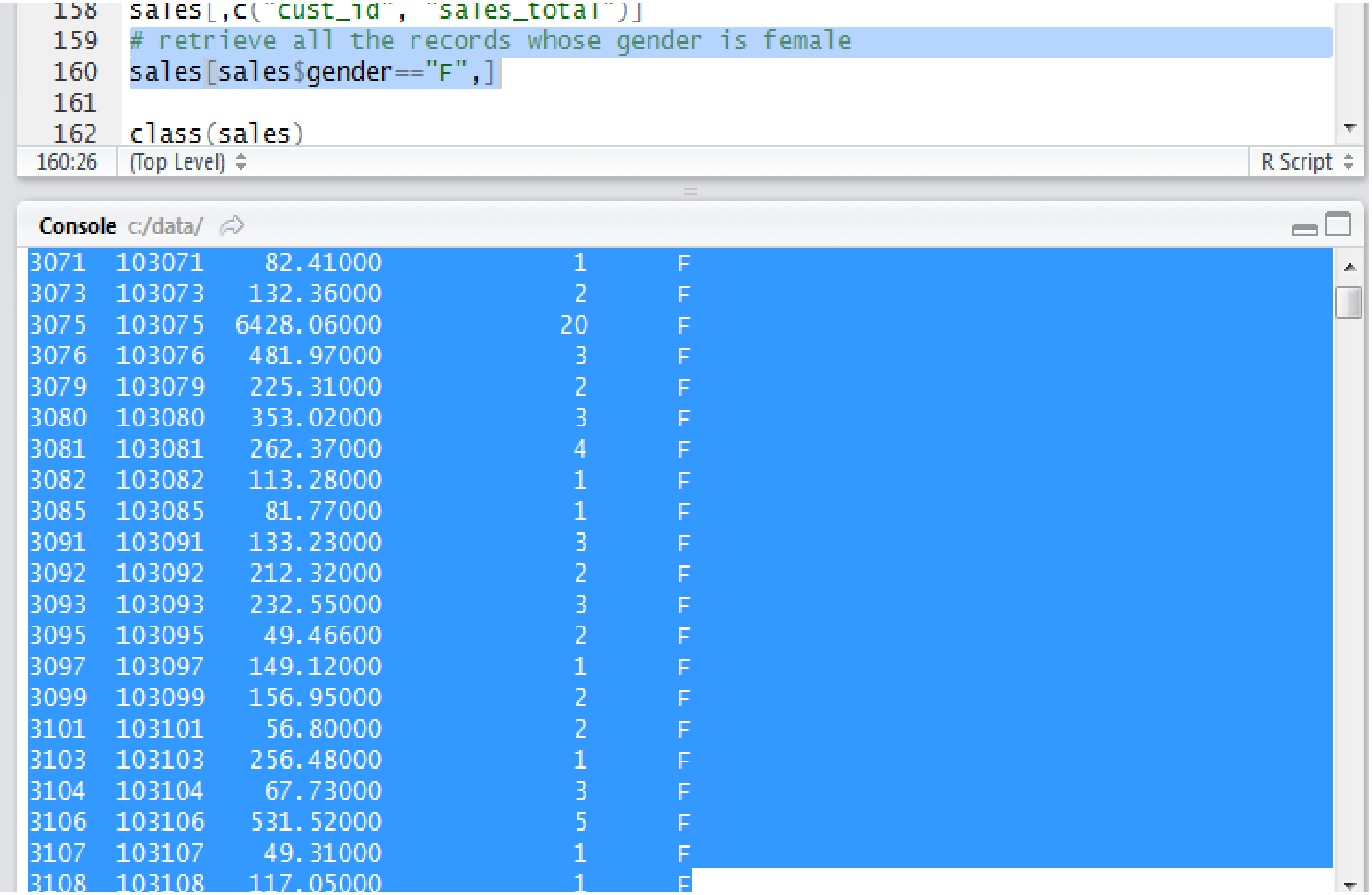


```
# retrieve both the cust_id and the sales_total columns
sales[,c("cust_id", "sales_total")]
# retrieve all the records whose gender is female
```

```
4002   104002    46.36000
4003   104003   147.94000
4004   104004    41.44000
4005   104005    66.76000
4006   104006    63.58000
4007   104007    80.05000
4008   104008   128.03000
4009   104009   198.38000
4010   104010   100.96000
4011   104011    40.82000
4012   104012   693.41000
4013   104013   230.96000
4014   104014    69.66000
4015   104015  1558.03000
4016   104016    94.47000
4017   104017   136.53000
4018   104018    33.23000
4019   104019   208.12000
4020   104020   236.27000
4021   104021  1086.78000
4022   104022   557.65000
```

Retrieve Columns based on a predicate

sales[sales$gender=="F",] retrieves records whose gender is female i.e. “F”



```
sales[,c("cust_id", "sales_total")]
# retrieve all the records whose gender is female
sales[sales$gender=="F",]

class(sales)
```

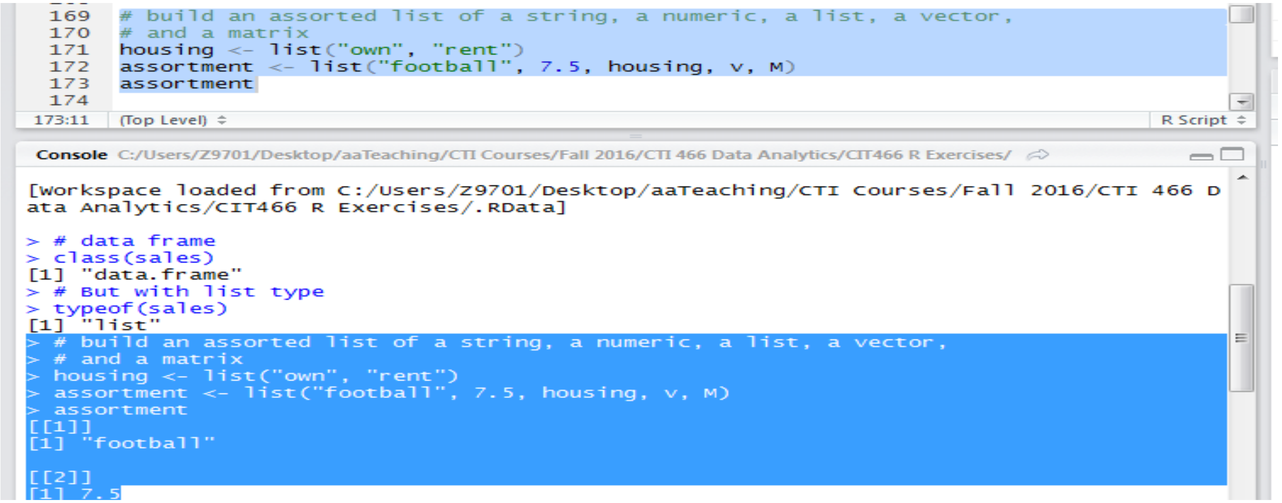
```
3071  103071    82.41000          1       F
3073  103073   132.36000          2       F
3075  103075  6428.06000         20       F
3076  103076   481.97000          3       F
3079  103079   225.31000          2       F
3080  103080   353.02000          3       F
3081  103081   262.37000          4       F
3082  103082   113.28000          1       F
3085  103085    81.77000          1       F
3091  103091   133.23000          3       F
3092  103092   212.32000          2       F
3093  103093   232.55000          3       F
3095  103095    49.46600          2       F
3097  103097   149.12000          1       F
3099  103099   156.95000          2       F
3101  103101    56.80000          2       F
3103  103103   256.48000          1       F
3104  103104    67.73000          3       F
3106  103106   531.52000          5       F
3107  103107    49.31000          1       F
3108  103108   117.05000          1       F
```

Attributes & Data Type here

LISTS

Lists

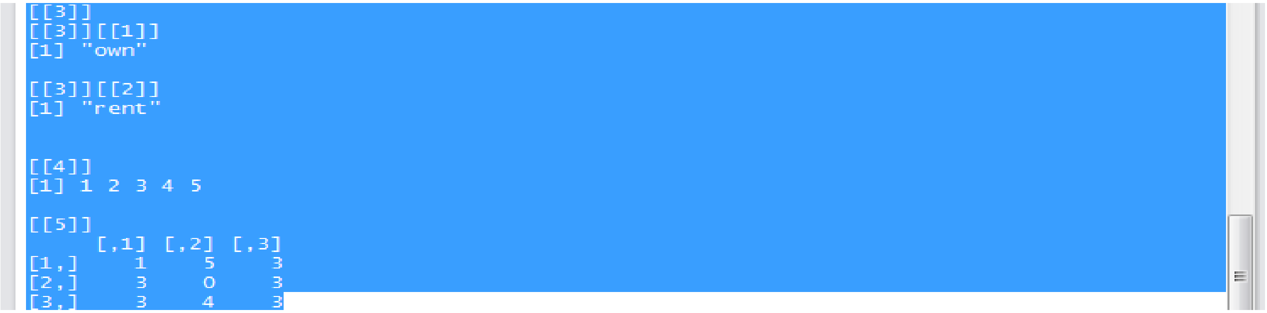
- A list is a collection of objects that can be of various type, including other lists
- *list()* function is used to create lists
- Using the Vector ***v*** and the Matrix *M* created in earlier example, the following R code creates assortments which is a list of different object types



```
# build an assorted list of a string, a numeric, a list, a vector,
# and a matrix
housing <- list("own", "rent")
assortment <- list("football", 7.5, housing, v, M)
assortment
```

173:11
(Top Level)
R Script
Console C:/Users/Z9701/Desktop/aaTeaching/CTI Courses/Fall 2016/CTI 466 Data Analytics/CIT466 R Exercises/

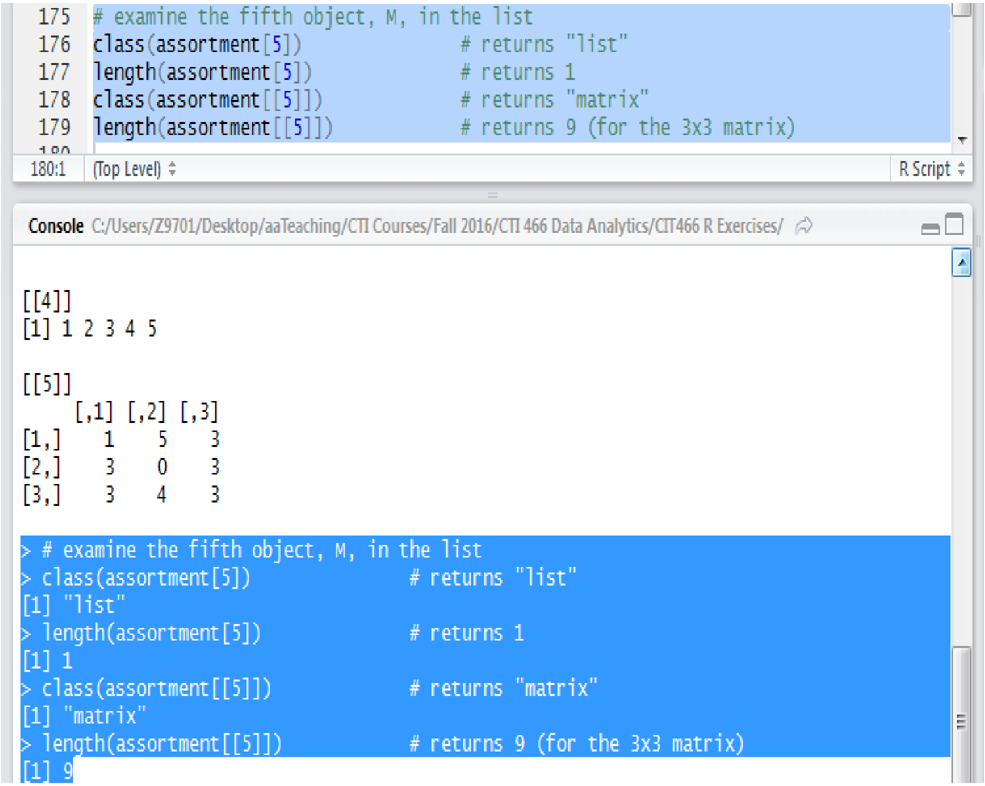
```
[Workspace loaded from C:/Users/Z9701/Desktop/aaTeaching/CTI Courses/Fall 2016/CTI 466 D
ata Analytics/CIT466 R Exercises/.RData]

> # data frame
> class(sales)
[1] "data.frame"
> # But with list type
> typeof(sales)
[1] "list"
> # build an assorted list of a string, a numeric, a list, a vector,
> # and a matrix
> housing <- list("own", "rent")
> assortment <- list("football", 7.5, housing, v, M)
> assortment
[[1]]
[1] "football"

[[2]]
[1] 7.5
```



```
[[3]]
[[3]][[1]]
[1] "own"

[[3]][[2]]
[1] "rent"


[[4]]
[1] 1 2 3 4 5

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

Displaying Content of a List

- Single bracket [ ] in the list() function only accesses an item not its content,
- Double bracket [[ ]] instead is used to access the content
- See highlighted R code and results below where [ ] and [[ ]] are used giving items and content respectively



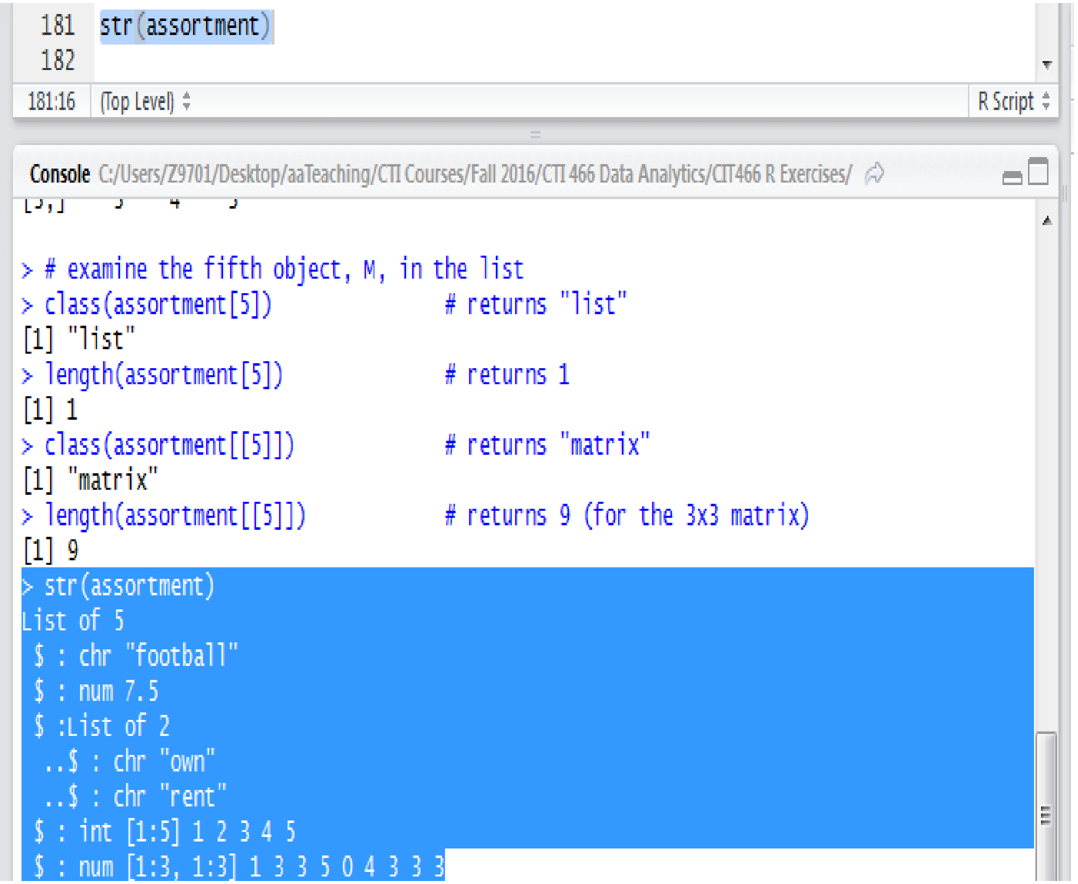
```
# examine the fifth object, M, in the list
class(assortment[5])              # returns "list"
length(assortment[5])             # returns 1
class(assortment[[5]])            # returns "matrix"
length(assortment[[5]])           # returns 9 (for the 3x3 matrix)
```

180:1
(Top Level)
R Script
Console C:/Users/Z9701/Desktop/aaTeaching/CTI Courses/Fall 2016/CTI 466 Data Analytics/CIT466 R Exercises/

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

[[5]]
     [,1] [,2] [,3]
[1,]    1    5    3
[2,]    3    0    3
[3,]    3    4    3

> # examine the fifth object, M, in the list
> class(assortment[5])              # returns "list"
[1] "list"
> length(assortment[5])             # returns 1
[1] 1
> class(assortment[[5]])            # returns "matrix"
[1] "matrix"
> length(assortment[[5]])           # returns 9 (for the 3x3 matrix)
[1] 9
```

Use of str() function in Lists

- *str()* function display details of the structure of the list


181:16
(Top Level)
R Script
Console C:/Users/Z9701/Desktop/aaTeaching/CTI Courses/Fall 2016/CTI 466 Data Analytics/CIT466 R Exercises/

```
str(assortment)
```

```
[3,]    3    4    3

> # examine the fifth object, M, in the list
> class(assortment[5])              # returns "list"
[1] "list"
> length(assortment[5])             # returns 1
[1] 1
> class(assortment[[5]])            # returns "matrix"
[1] "matrix"
> length(assortment[[5]])           # returns 9 (for the 3x3 matrix)
[1] 9
> str(assortment)
List of 5
 $ : chr "football"
 $ : num 7.5
 $ :List of 2
  ..$ : chr "own"
  ..$ : chr "rent"
 $ : int [1:5] 1 2 3 4 5
 $ : num [1:3, 1:3] 1 3 3 5 0 4 3 3 3
```

Descriptive Statistics

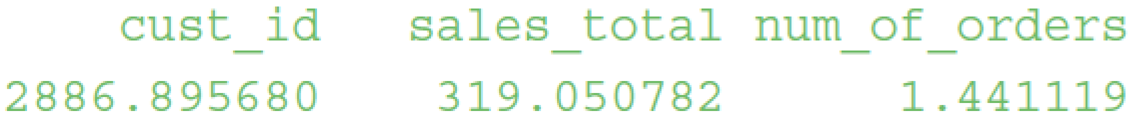
- It has already been shown that the summary() function provides several descriptive statistics, such as the mean and median, about a variable such as the *sales* data frame.
- The following code provides some common R functions that include descriptive statistics. In parentheses, the comments describe the functions.

Descriptive Statistics

```
# to simplify the function calls, assign
x <- sales$sales_total
y <- sales$num_of_orders
cor(x,y) # returns 0.7508015 (correlation)
cov(x,y) # returns 345.2111 (covariance)
IQR(x) # returns 215.21 (interquartile range)
mean(x) # returns 249.4557 (mean)
median(x) # returns 151.65 (median)
range(x) # returns 30.02 7606.09 (min max)
sd(x) # returns 319.0508 (std. dev.)
var(x) # returns 101793.4 (variance)
```

Descriptive Statistics

- The function apply() is useful when the same function is to be applied to several variables in a data frame. For example, the following R code calculates the standard deviation for the first three variables in *sales.*
- apply(sales[,c(1:3)], MARGIN=2, FUN=sd)



```
    cust_id   sales_total num_of_orders
2886.895680    319.050782      1.441119
```