table() function in R Language is used to create a categorical representation of data with variable name and the frequency in the form of a table.

Syntax: table(x)

Parameters:x: Object to be converted

```
# R Program to create

# a tabular representation of data


# Creating a vector

vec = c(2, 4, 3, 1, 2, 3, 2, 1, 4, 2)


# Calling table() Function

table(vec)
```

Output:

```
vec

1 2 3 4

2 4 2 2
```

Example 2:

```
# R Program to create

# a tabular representation of data


# Creating a data frame

df = data.frame(

"Name" = c("abc", "cde", "def"),

"Gender" = c("Male", "Female", "Male")

)
```

```
# Calling table() function

table(df)
```

Output:

```
     Gender

Name  Female Male

  abc      0    1

  cde      1    0

  def      0    1
```

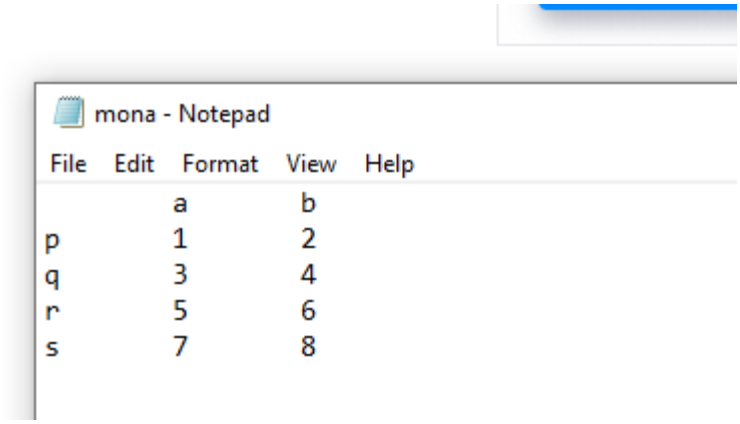
Reading Tabular Data from files in R Programming

he data which is to be read and worked upon is already stored in a file but is present outside the R environment. Hence, importing data into R is a mandatory task in such circumstances. The formats which are supported by R are CSV, JSON, Excel, Text, XML, etc. The majority of times, the data to be read into R is in tabular format. The functions used for reading such data, which is stored in the form of rows and columns, import the data and return data frame in R. Data frame is preferred in R because it is easier to extract data from rows and columns of a data frame for statistical computation tasks than other data structures in R. The most common functions which are used for reading tabular data into R are:- **read.table()**, **read.csv()**, **fromJSON()** and **read.xlxs()** .

Parameters:

- **file:** Specifies the name of the file.
- **header:**The header is a logical flag indicating whether the first line is a header line contains data or not.
- **nrows:** Specifies number of rows in the dataset.
- **skip:** Helps in skipping of lines from the beginning.
- **colClasses:** It is a character vector which indicates class of each column of the data set.
- **sep:** It a string indicating the way the columns are separated that is by commas, spaces, colons, tabs etc.

For small or moderately sized data sets, we can call **read.table()** without any arguments. R will automatically figure out the number of rows, the number of columns, classes of different columns, skip lines that start with #(comment symbol), etc. If we do specify the arguments, it will make the execution faster and efficient but here, since the dataset is small so it would not make much of a difference as it is already fast and efficient.



read.table("mona.txt")

output:

>using read.table to read tabular data from r file

>read.table(“mona.txt”)

	a	b
p	1	2
q	3	4
r	5	6
s	7	8

CSV file is a comma-separated values file

Working with CSV files in R Programming

CSV files are basically the text files wherein the values of each row are separated by a delimiter, as in a comma or a tab. In this article, we will use the following sample CSV file:

sample.csv

```
id, name, department, salary, projects
1,   A,     IT,        60754,   4
2,   B,     Tech,      59640,   2
```

```
3,   C,     Marketing, 69040,   8
4,   D,     Marketing, 65043,   5
5,   E,     Tech,      59943,   2
6,   F,     IT,        65000,   5
7,   G,     HR,        69000,   7
```

Reading a CSV file

The contents of a CSV file can be read as a data frame in R using the read.csv(…) function. The CSV file to be read should be either present in the current working directory or the directory should be set accordingly using the setwd(…) command in R. The CSV file can also be read from a URL using **`read.csv()`** function.

Examples:

```
csv_data <- read.csv(file = 'sample.csv')

print(csv_data)



# print number of columns

print (ncol(csv_data))



# print number of rows

print(nrow(csv_data))
```

```
      id,   name,    department,    salary,      projects
 1      1      A         HR          60754         14
 2      2      B        Tech         59640         3
 3      3      C       Marketing     69040         8
 4      4      D         HR          65043         5
 5      5      E        Tech         59943         2
```

```
 6      6      F          IT          65000          5
 7      7      G          HR          69000          7
 [1] 4
 [1] 7
```

Querying with CSV files

SQL queries can be performed on the CSV content, and the corresponding result can be retrieved using the subset(csv_data,) function in R. Multiple queries can be applied in the function at a time where each query is separated using a logical operator. The result is stored as a data frame in R.

Examples:

```
csv_data <- read.csv(file ='sample.csv')

min_pro <- min(csv_data$projects)

print (min_pro)
```

Output:

2

Aggregator functions (min, max, count etc.) can be applied on the CSV data. Here the **min()** function is applied on projects column using $ symbol. The minimum number of projects which is 2 is returned.

```
csv_data <- read.csv(file ='sample.csv')

new_csv <- subset(csv_data, department == "HR" & projects <10)

print (new_csv)
```

Output:

```
      id,   name,    department,    salary,      projects
4      4      D          HR          65043          5
7      7      G          HR          69000          7
```

The subset of the data that is created is stored as a data frame satisfying the conditions specified as the arguments of the function. The employees D and G are HR and have the number of projects<10. The row numbers are retained in the resultant data frame.

Writing into a CSV file

The contents of the data frame can be written into a CSV file. The CSV file is stored in the current working directory with the name specified in the function write.csv(data frame, output CSV name) in R.

Examples:

```
csv_data <- read.csv(file ='sample.csv')

new_csv <- subset(csv_data, department == "HR" & projects <10)

write.csv(new_csv, "new_sample.csv")

new_data <-read.csv(file ='new_sample.csv')

print(new_data)
```

Output:

```
      X    id,   name,    department,    salary,      projects
1     4      4      D          HR           65043          5
2     7      7      G          HR           69000          7
```

The column X contains the row numbers of the original CSV file. In order to remove it, we can specify an additional argument in the write.csv() function that set row names to FALSE.

```
csv_data <- read.csv(file ='sample.csv')

new_csv <- subset(csv_data, department == "HR" & projects <10)

write.csv(new_csv, "new_sample.csv", row.names = FALSE)

new_data <-read.csv(file ='new_sample.csv')

print(new_data)
```

Output:

```
      id,   name,    department,    salary,      projects
1     4      D          HR          65043         5
2     7      G          HR          69000         7
```

The original row numbers are removed from the new CSV.

Working with Tables in R

Tables are often essential for organzing and summarizing your data, especially with categorical variables. When creating a table in R, it considers your table as a specifc type of object (called “table”) which is very similar to a data frame. Though this may seem strange since datasets are stored as data frames, this means working with tables will be very easy since we have covered data frames in detail over the previous tutorials. In this chapter, we will discuss how to create various types of tables, and how to use various statistical methods to analyze tabular data.

Creating Basic Tables: table() and xtabs()

A contingency table is a tabulation of counts and/or percentages for one or more variables. In R, these tables can be created using **table()** along with some of its variations. To use table(), simply add in the variables you want to tabulate separated by a comma. Note that table() does not have a data= argument like many other functions do (e.g., ggplot2 functions), so you much reference the variable using dataset$variable. Some examples are shown below. By default, missing values are excluded from the counts; if you want a count for these missing values you must specify the argument useNA=“ifany” or useNA=“always”. The below examples show how to use this function.

ghout the chapter, the AOSI dataset will be used.

how to create tables in R Programming Language.

Method 1: Create a table from scratch

We can create a table by using as.table() function, first we create a table using matrix and then assign it to this method to get the...

```
# create matrix with 4 columns and 4 rows
data= matrix(c(1:16), ncol=4, byrow=TRUE)

# specify the column names and row names of matrix
colnames(data) = c('col1','col2','col3','col4')
rownames(data) <- c('row1','row2','row3','row4')

# assign to table
final=as.table(data)

#display

final
```

```
     col1 col2 col3 col4
row1    1    2    3    4
row2    5    6    7    8
row3    9   10   11   12
row4   13   14   15   16
```

How to Use the Table Function in R (With Examples)

The **table()** function in R can be used to quickly create frequency tables.

This tutorial provides examples of how to use this function with the following data frame in R:

```
#create data frame
df <- data.frame(player = c('AJ', 'Bob', 'Chad', 'Dan', 'Eric', 'Frank'),
                 position = c('A', 'B', 'B', 'B', 'B', 'A'),
```

```
                points = c(1, 2, 2, 1, 0, 0))

#view data frame
df

  player position points
1     AJ        A      1
2    Bob        B      2
3   Chad        B      2
4    Dan        B      1
5   Eric        B      0
6  Frank        A      0
```

Example 1: Frequency Table for One Variable

The following code shows how to create a frequency table for the **position** variable in our data frame:

```
#calculate frequency table for position variable
table(df$position)

A B
2 4
```

From the output we can observe:

- 2 players in the data frame have a position of '**A**'
- 4 players in the data frame have a position of '**B**'

Example 2: Frequency Table of Proportions for One Variable

The following code shows how to use **prop.table()** to create a frequency table of proportions for the **position** variable in our data frame:

```
#calculate frequency table of proportions for position variable
prop.table(table(df$position))

        A         B
0.3333333 0.6666667
```

From the output we can observe:

- 33.33% of players in the data frame have a position of '**A**'
- 66.67% of players in the data frame have a position of '**B**'

Note that in a proportion table the sum of the proportions will always be equal to 1.

Example 3: Frequency Table for Two Variables

The following code shows how to create a frequency table for the **position** and **points** variable in our data frame:

```
#calculate frequency table for position and points variable
table(df$position, df$points)

  0 1 2
A 1 1 0
B 1 1 2
```

From the output we can observe:

- 1 player in the data frame has a position of '**A**' and **0** points
- 1 player in the data frame has a position of '**A**' and **1** point
- 0 players in the data frame have a position of '**A**' and **2** points
- 1 player in the data frame has a position of '**B**' and **0** points
- 1 player in the data frame has a position of '**B**' and **1** point
- 2 players in the data frame have a position of '**B**' and **2** points

Example 4: Frequency Table of Proportions for Two Variables

The following code shows how to create a frequency table of proportions for the **position** and **points** variable in our data frame:

```
#calculate frequency table of proportions for position and points variable
prop.table(table(df$position, df$points))
```

```
        0         1         2
A 0.1666667 0.1666667 0.0000000
B 0.1666667 0.1666667 0.3333333
```

From the output we can observe:

- 16.67% of players in the data frame have a position of '**A**' and **0** points
- 16.67% of players in the data frame have a position of '**A**' and **1** point
- 0% of players in the data frame have a position of '**A**' and **2** points
- 16.67% of players in the data frame have a position of '**B**' and **0** points
- 16.67% of players in the data frame have a position of '**B**' and **1** point
- 33.3% of players in the data frame have a position of '**B**' and **2** points

Note that we can also use the **options()** function to specify how many decimals to show in the proportion table:

```
#only display two decimal places
options(digits=2)

#calculate frequency table of proportions for position and points variable
prop.table(table(df$position, df$points))

     0    1    2
A 0.17 0.17 0.00
B 0.17 0.17 0.33
```

How to Create Tables in R

In this R programming tutorial you'll learn how to **create, manipulate, and plot table objects**.

The content of the page is structured as follows:

Example Data

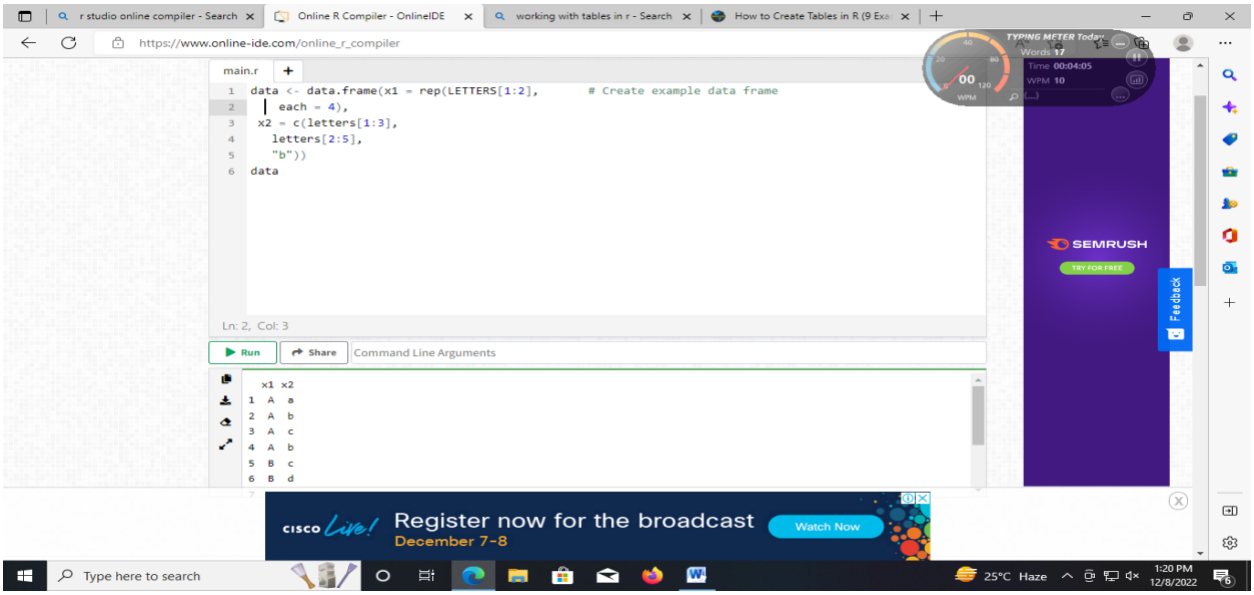
The following data will be used as basement for this R programming language tutorial:

```
data <- data.frame(x1 = rep(LETTERS[1:2],        # Create example data frame
                            each = 4),
                   x2 = c(letters[1:3],
                          letters[2:5],
                          "b"))
data                                             # Print example data frame
```

Table 1

	x1	x2
1	A	a
2	A	b
3	A	c
4	A	b
5	B	c
6	B	d
7	B	e
8	B	b

Table 1 visualizes the output of the RStudio console and shows the structure of our exemplifying data – It is constituted of eight rows and two character columns.



Example 1: Create Frequency Table

This example shows how to make a frequency table in R.

For this task, we can apply the table() function to one of the columns of our example data frame:

```
tab1 <- table(data$x2)                                  # Make frequency table
tab1                                                    # Print frequency table
# a b c d e
# 1 3 2 1 1
```

The previous output shows the frequency counts of each value in the column x2. For instance, the letter a is contained once, and the letter b is contained three times.

Example 2: Create Contingency Table

The following R programming code explains how to make a contingency table, i.e. a table of multiple columns.

The following R code creates a two-way cross tabulation of our example data frame:

```
tab2 <- table(data)                                    # Make contingency table
tab2                                                   # Print contingency table
#    x2
# x1  a b c d e
#   A 1 2 1 0 0
#   B 0 1 1 1 1
```

The previous output shows the frequency distribution among the two columns x1 and x2. For instance, the combination of A and a occurs once, and the combination of B and a appears not at all.

Example 3: Sort Frequency Table

This example explains how to order a table object.

For this example, we use the table object tab1 that we have created in Example 1 as basis.

We sort this table by applying the order function. Within the order function, we set the decreasing argument to be equal to TRUE, to show the values with the most occurrences first.

Have a look at the following R code:

```
tab3 <- tab1[order(tab1, decreasing = TRUE)]     # Order table
tab3                                             # Print ordered table
# b c a d e
# 3 2 1 1 1
```

Example 4: Change Names of Table

In Example 4, I'll demonstrate how to rename the elements of a table.

For this, we can apply the names and paste0 functions as illustrated in the following R code:

```
tab4 <- tab3                                     # Duplicate table
names(tab4) <- paste0("x", 1:length(tab4))       # Change names
tab4                                             # Print renamed table
# x1 x2 x3 x4 x5
#  3  2  1  1  1
```

The previous output contains the same numeric values as the table that we have created in Example 3. However, the labels of those table cells have been changed.

Example 5: Extract Subset of Table

The code below shows how to return only a certain subset of a table object.

To achieve this, we use the table object tab1 that we have constructed in Example1 as basis. We can select a subset of this table object using a logical condition as shown below:

```
tab5 <- tab1[tab1 > 1]                                  # Extract table subset
tab5                                                    # Print table subset
# b c
# 3 2
```

The previously shown table subset consists of all table elements that occur at least two times. All the other table elements have been removed.

Example 6: Create Proportions Table

In Example 6, I'll explain how to create a proportions table (or probabilities).

For this task, we can apply the prop.table command to a table object (i.e. tab1) as illustrated in the following R syntax:

```
tab6 <- prop.table(tab1)                                # Make proportions table
tab6                                                    # Print proportions table
#     a     b     c     d     e
# 0.125 0.375 0.250 0.125 0.125
```

The previous output shows the proportions of each value in our data.

Example 7: Draw Table in Barplot

In Example 7, I'll show how to plot a table object in a barchart.

To do this, we have to apply the barplot function to a table object:

```
barplot(tab1)                                           # Draw table in plot
```

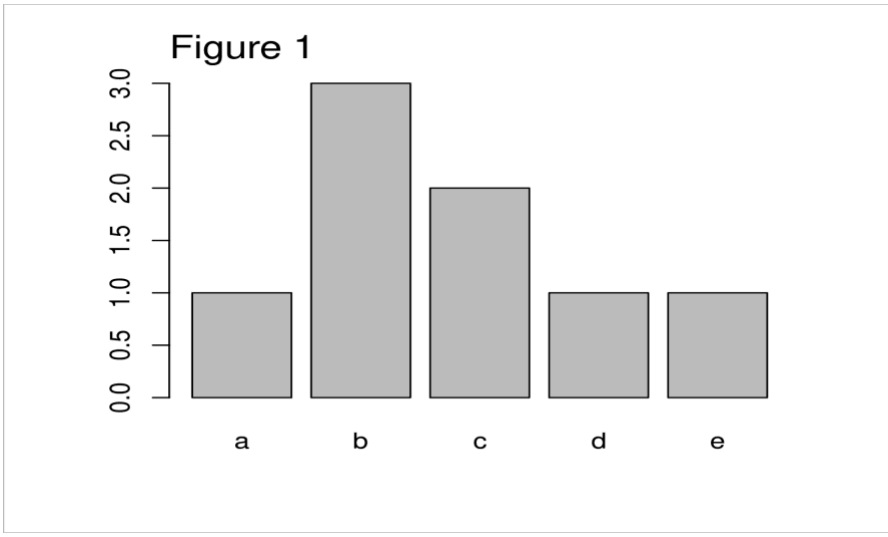


Figure 1 shows the output of the previous R code: A Base R bargraph showing the values in the table we have created in Example 1. The height of the bars corresponds to the occurrences of each value in our data set variable.

Example 8: Convert Matrix to Table

This example explains how to change the data type of a numeric matrix object to the table class.

For this example, we first have to create an exemplifying matrix:

```
mat <- matrix(1:12, ncol = 3)                    # Create example matrix
mat                                               # Print example matrix
```

Table 2

	[,1]	[,2]	[,3]
[1,]	1	5	9
[2,]	2	6	10
[3,]	3	7	11
[4,]	4	8	12

As shown in Table 2, the previous R programming code has created a matrix object with four rows and three columns.

We can now use the as.table function to convert this matrix to the table class:

```
tab7 <- as.table(mat)                                  # Convert matrix to table
tab7                                                   # Print converted table
#    A  B  C
# A  1  5  9
# B  2  6 10
# C  3  7 11
```

Example 9: Check Class of Table Object

This example illustrates how to check whether a data object has the table class.

There are basically two alternatives on how to do this. Either, we can apply the class() function to return the class of a data object…

```
class(tab7)                                            # Return class of table
# [1] "table"
```

…or we can apply the is.table function to return a logical indicator that shows whether our data object has the table class:

```
is.table(tab7)                                    # Test if object is table
# [1] TRUE
```

Both applications return the same result: The data object tab7 that we have created in Example 8 has the table class.

How to subset a data.table object using a range of values in R?

To subset a data.table object using a range of values, we can use single square brackets and choose the range using %between%. For example, if we have a data.table object DT that contains a column x and the values in x ranges from 1 to 10 then we can subset DT for values between 3 to 8 by using the command DT[DT$x %between% c(3,8)].

Example1

Loading data.table package and creating a data.table object −

```
library(data.table)
x1<-rnorm(20)
x2<-rnorm(20,100,3.25)
dt1<-data.table(x1,x2)
dt1
```

Output

```
       x1         x2
1:  0.06546822  102.83348
2:  1.05756760   99.28015
3:  0.09397791   97.36582
4: -0.44478256   96.22510
5: -0.17392089   99.56077
6:  0.32834773   95.85831
7: -0.04563975  104.88298
8:  0.95807930   95.31325
```

```
9:   1.25349243   101.72948
10:  0.47676550   96.76459
11: -1.26790256   98.76222
12: -1.36220203   97.91117
13: -1.31999499   102.81730
14:  1.69391374   96.00380
15: -0.10801512   101.69544
16: -1.13463486   108.11833
17: -0.13885823   102.34798
18: -0.54403332   99.68874
19:  0.55865944   97.33505
20:  0.76511431   96.53975
```

Subsetting rows of dt1 for values 0.5 to 1 of x1 −

```
dt1[dt1$x1 %between% c(0.5,1)]
```

```
      x1        x2
1: 0.9580793  95.31325
2: 0.5586594  97.33505
3: 0.7651143  96.53975
```

```
y1<-rpois(20,5)
y2<-rpois(20,2)
dt2<-data.table(y1,y2)
dt2
```

Output

```
   y1 y2
1:  7  1
2:  3  2
3:  1  5
4:  4  2
5:  5  3
6:  4  1
7:  3  2
8:  5  0
```

9: 5 2
10: 7 4
11: 5 4
12: 6 2
13: 6 4
14: 6 2
15: 5 2
16: 1 1
17: 3 1
18: 2 4
19: 4 0
20: 4 1

Subsetting rows of dt1 for values 2 to 6 of y2 −

dt2[dt2$y2 %between% c(2,6)]

   y1 y2
1: 3 2
2: 1 5
3: 4 2
4: 5 3
5: 3 2
6: 5 2
7: 7 4
8: 5 4
9: 6 2
10: 6 4
11: 6 2
12: 5 2
13: 2 4