Creating Subsets & Sorting Data

Contents

Contents

Data Snapshot

basic_salary data consists salary of each employee with it's Location & Grade.

Variables

Observations

First_Name	Last_Name	Grade	Location	ba	ms
Alan	Brown	GR1	DELHI	17990	16070
Agatha	Williams	GR2	MUMBAI	12390	6630

Columns	Description	Type	Measurement	Possible values
First_Name	First Name	character	-	-
Last_Name	Last Name	character	-	-
Grade	Grade	character	GR1, GR2	2
Location	Location	character	DELHI, MUMBAI	2
ba	Basic Allowance	numeric	Rs.	positive values
ms	Management Supplements	numeric	Rs.	positive values

* Here we continue to use previous data for our further analysis.

* Here we continue to use previous data for our further analysis.

Need for Creating Subsets

- Sometimes we want to view filtered snippet, or to extract just the data we are interested in from a data frame, for obtaining this we will be using some of R's built-in subsetting functions.
- In terms of R, Subsetting is extracting the needed rows (observations) and columns (variables) from a dataset.
- This tutorial aims to teach the ways in which it is possible to subset data set in R.

Row Subsetting

- We can use the **[ ]** bracket notation to access the indices of rows and columns like this: [R, C]. Here, the first index is for **Rows** and the second is for **Columns**.

```
# Import basic_salary data
salary_data <- read.csv("basic_salary.csv", header=TRUE)
# Display rows from 5th to 10th & save it as new object data
salary1 <- salary_data [5:10, ]
salary1
# Output
```

Colon notation(:) used since rows have consecutive positions.

```
   First_Name Last_Name Grade Location    ba    ms
5        Neha       Rao   GR1   MUMBAI 19235 15200
6       Sagar    Chavan   GR2   MUMBAI 13390  6700
7       Aaron     Jones   GR1   MUMBAI 23280 13490
8        John     Patil   GR2   MUMBAI 13500 10760
9       Sneha     Joshi   GR1    DELHI 20660    NA
10     Gaurav     Singh   GR2    DELHI 13760 13220
```

* When we only want to subset rows we use the first index and leave the second index blank. Leaving an index blank indicates that you want to keep all the elements in that dimension. Use names() to know which column corresponds to which no. in the index.

Row Subsetting

```
# Display only selected rows & save it as new object data
```

```
salary2 <- salary_data[c(1,3,5), ]
salary2
```

```
# Output
```

	First_Name	Last_Name	Grade	Location	ba	ms
1	Alan	Brown	GR1	DELHI	17990	16070
3	Rajesh	Kolte	GR1	MUMBAI	19250	14960
5	Neha	Rao	GR1	MUMBAI	19235	15200

c() can be used to give a list of row indices if the rows are not in sequential order

Column Subsetting

- As we know, we can subset variables using **[ ]** bracket notation but now we **use the second index and leave the first index blank**. This indicates that we want all the rows for specific columns.

```
# Display columns 1 to 4 & save it as new object data
```

```
salary3<-salary_data[ ,1:4]
head(salary3)
```

```
# Output
```

```
  First_Name Last_Name Grade Location
1       Alan     Brown   GR1    DELHI
2     Agatha  Williams   GR2   MUMBAI
3     Rajesh     Kolte   GR1   MUMBAI
4      Ameet    Mishra   GR2    DELHI
5       Neha       Rao   GR1   MUMBAI
6      Sagar    Chavan   GR2   MUMBAI
```

Row-Column Subsetting

```
# Display rows 1,5,8 and columns 1 and 2  & save it as new object
data
salary4<-salary_data[c(1,5,8),c(1,2)]
salary4
```

```
# Output
```

```
  First_Name Last_Name
1       Alan     Brown
5       Neha       Rao
8       John     Patil
```

We can also subset the columns by name as in select for subset() like this:

```
salary6<-salary_data[c(1,5,8),c("First_Name", "Last_Name")]
```

Subsetting Observations

- The **subset()** function with conditions made from logical operators involving the columns of the data frame will let you subset the data frame by observations.
- Our data is saved as an object named **salary_data** which is of class **data.frame.**

```
# Create a subset with all details of employees of MUMBAI with ba
more than 15000
```

```
salary5<-subset(salary_data,Location=="MUMBAI" & ba>15000)
salary5
```

```
# Output
```

	First_Name	Last_Name	Grade	Location	ba	ms
3	Rajesh	Kolte	GR1	MUMBAI	19250	14960
5	Neha	Rao	GR1	MUMBAI	19235	15200
7	Aaron	Jones	GR1	MUMBAI	23280	13490

- **subset()** will return a data frame and so this newly created object can be used as an argument in **subset()**
- **Note :** There is no limit on how many

hieve

* R assumes variable names are within the data frame being subset, so there is no need to tell R where to look for **location** and **ba** in the above example.

Subsetting Observations

- We can also specify which columns we would like **subset()** to return by adding a **select** argument

```
# Create a subset of only Location, First name and ba of salary5
data, created in the first example
```

```
salary6<-subset(salary5,select=c(Location,First_Name,ba))
salary6
```

```
# Output
```

	Location	First_Name	ba
3	MUMBAI	Rajesh	19250
5	MUMBAI	Neha	19235
7	MUMBAI	Aaron	23280

- **select=** takes columns to be selected from a data frame.
- The order of columns in the output totally depends on how you place the column names in **select** argument.

Subsetting Both Observations and Variables

- We can subset observations and variables by simply combining the previous two methods of subsetting.

```
# Select First_Name, Grade and Location of employees of GR1 with ba
more than 15000
```

```
salary7<-subset(salary_data, Grade=="GR1" & ba>15000,
select= c(First_Name, Grade, Location))

salary7
```

```
# Output
```

	First_Name	Grade	Location
1	Alan	GR1	DELHI
3	Rajesh	GR1	MUMBAI
5	Neha	GR1	MUMBAI
7	Aaron	GR1	MUMBAI
9	Sneha	GR1	DELHI

Subsetting Using Not Operator

Suppose we want all the details of employees not having GR1 and not from MUMBAI, we will write the following command.

```
salary8<-subset(salary_data,!(Grade=="GR1") & !(Location=="MUMBAI"))
salary8
```

```
# Output
```

	First_Name	Last_Name	Grade	Location	ba	ms
4	Ameet	Mishra	GR2	DELHI	14780	9300
10	Gaurav	Singh	GR2	DELHI	13760	13220
11	Adela	Thomas	GR2	DELHI	13660	6840

Not Equal To (!) operator is used to give condition.

Quick Recap

In this session, we learnt many ways to subset data using **`subset()`** and row/column index. Here is the quick recap of how we can create subsets :

Using index	• **Row Subsetting:** By specifying the row indices using **`[]`** notation • **Column Subsetting:** By specifying the column indices using [] notation or column names using **`c()`**. • **Row-Column Subsetting:** By combining the above two methods.
Using **`subset()`**	• **Subsetting observations:** By giving conditions on columns using this function. • **Subsetting variables:** By specifying columns in **`select`** argument. • **Subsetting both observations and variables:** By simply combining above two methods. • **Subsetting using Not Operator :** By giving conditions on those columns which we do not want using this function.

Introduction to Sorting

Sorting data is one of the common activity in preparing data for analysis.

Sorting is storage of data in sorted order, it can be in ascending or descending order.

We will be exploring all the ways in which sorting can be done.

```
# Import and attach basic_salary data
salary_data <- read.csv("basic_salary.csv",header=TRUE)
attach(salary_data)
```

attach() attaches the database to the R search path, so the variables in the database can be accessed by simply giving their names

Ascending Data

```
# Sort salary_data by ba in Ascending order
```

```
ba_sorted_1<-salary_data[order(ba),]
ba_sorted_1
```

```
# Output
```

	First_Name	Last_Name	Grade	Location	ba	ms
12	Anup	Save	GR2	MUMBAI	11960	7880
2	Agatha	Williams	GR2	MUMBAI	12390	6630
6	Sagar	Chavan	GR2	MUMBAI	13390	6700
8	John	Patil	GR2	MUMBAI	13500	10760
11	Adela	Thomas	GR2	DELHI	13660	6840
10	Gaurav	Singh	GR2	DELHI	13760	13220
4	Ameet	Mishra	GR2	DELHI	14780	9300
1	Alan	Brown	GR1	DELHI	17990	16070
5	Neha	Rao	GR1	MUMBAI	19235	15200
3	Rajesh	Kolte	GR1	MUMBAI	19250	14960
9	Sneha	Joshi	GR1	DELHI	20660	NA
7	Aaron	Jones	GR1	MUMBAI	23280	13490

order() is used to sort a vector, matrix or data frame. By default, it sorts in ascending order

Descending Order

```
# Sort salary_data by ba in Descending order
```

```
ba_sorted_2<-salary_data[order(-ba),]
ba_sorted_2
```

```
# Output
```

	First_Name	Last_Name	Grade	Location	ba	ms
7	Aaron	Jones	GR1	MUMBAI	23280	13490
9	Sneha	Joshi	GR1	DELHI	20660	NA
3	Rajesh	Kolte	GR1	MUMBAI	19250	14960
5	Neha	Rao	GR1	MUMBAI	19235	15200
1	Alan	Brown	GR1	DELHI	17990	16070
4	Ameet	Mishra	GR2	DELHI	14780	9300
10	Gaurav	Singh	GR2	DELHI	13760	13220
11	Adela	Thomas	GR2	DELHI	13660	6840
8	John	Patil	GR2	MUMBAI	13500	10760
6	Sagar	Chavan	GR2	MUMBAI	13390	6700
2	Agatha	Williams	GR2	MUMBAI	12390	6630
12	Anup	Save	GR2	MUMBAI	11960	7880

The '-' sign before a numeric column reverses the default order.

Alternatively , you can also use decreasing=TRUE

```
ba_sorted_2<-salary_data[order(ba, decreasing = TRUE),]
```

Sorting by Factor Variable

Sort data by column with characters / factors

```
# Sort salary_data by Grade
```

```
gr_sorted<-salary_data[order(Grade),]
gr_sorted
```

```
# Output
```

	First_Name	Last_Name	Grade	Location	ba	ms
1	Alan	Brown	GR1	DELHI	17990	16070
3	Rajesh	Kolte	GR1	MUMBAI	19250	14960
5	Neha	Rao	GR1	MUMBAI	19235	15200
7	Aaron	Jones	GR1	MUMBAI	23280	13490
9	Sneha	Joshi	GR1	DELHI	20660	NA
2	Agatha	Williams	GR2	MUMBAI	12390	6630
4	Ameet	Mishra	GR2	DELHI	14780	9300
6	Sagar	Chavan	GR2	MUMBAI	13390	6700
8	John	Patil	GR2	MUMBAI	13500	10760
10	Gaurav	Singh	GR2	DELHI	13760	13220
11	Adela	Thomas	GR2	DELHI	13660	6840
12	Anup	Save	GR2	MUMBAI	11960	7880

Note that by default **order()** sorts in ascending order

Sorting by Factor Variable

Sort data by column with characters / factors in Descending order

```
# Sort salary_data by Grade in Descending order
```

```
gr_sorted<-salary_data[order(Grade,decreasing=TRUE),]
gr_sorted
```

```
# Output
```

	First_Name	Last_Name	Grade	Location	ba	ms
2	Agatha	Williams	GR2	MUMBAI	12390	6630
4	Ameet	Mishra	GR2	DELHI	14780	9300
6	Sagar	Chavan	GR2	MUMBAI	13390	6700
8	John	Patil	GR2	MUMBAI	13500	10760
10	Gaurav	Singh	GR2	DELHI	13760	13220
11	Adela	Thomas	GR2	DELHI	13660	6840
12	Anup	Save	GR2	MUMBAI	11960	7880
1	Alan	Brown	GR1	DELHI	17990	16070
3	Rajesh	Kolte	GR1	MUMBAI	19250	14960
5	Neha	Rao	GR1	MUMBAI	19235	15200
7	Aaron	Jones	GR1	MUMBAI	23280	13490
9	Sneha	Joshi	GR1	DELHI	20660	NA

In case of **factor type variables**, if the ordering is to be made descending, then the logical argument of **decreasing= TRUE** needs to be included.

Sorting Data by Multiple Variables

- Sort data by giving multiple columns; one column with characters / factors and one with numerals

```
# Sort salary_data by Grade and ba
```

```
grba_sorted<-salary_data[order(Grade,ba),]
grba_sorted
```

```
# Output
```

	First_Name	Last_Name	Grade	Location	ba	ms
1	Alan	Brown	GR1	DELHI	17990	16070
5	Neha	Rao	GR1	MUMBAI	19235	15200
3	Rajesh	Kolte	GR1	MUMBAI	19250	14960
9	Sneha	Joshi	GR1	DELHI	20660	NA
7	Aaron	Jones	GR1	MUMBAI	23280	13490
12	Anup	Save	GR2	MUMBAI	11960	7880
2	Agatha	Williams	GR2	MUMBAI	12390	6630
6	Sagar	Chavan	GR2	MUMBAI	13390	6700
8	John	Patil	GR2	MUMBAI	13500	10760
11	Adela	Thomas	GR2	DELHI	13660	6840
10	Gaurav	Singh	GR2	DELHI	13760	13220
4	Ameet	Mishra	GR2	DELHI	14780	9300

Here, data is first sorted in increasing order of **Grade** then **ba.**

Multiple Variables & Multiple Ordering Levels

- Sort data by giving multiple columns; one column with characters / factors and one with numerals and multiple ordering levels

```
# Sort salary_data by Grade in Descending order and then by ms in
# Ascending order
```

```
grba_sorted<-salary_data[order(Grade,decreasing=TRUE,ms),]
grba_sorted
```

```
# Output
```

	First_Name	Last_Name	Grade	Location	ba	ms
10	Gaurav	Singh	GR2	DELHI	13760	13220
8	John	Patil	GR2	MUMBAI	13500	10760
4	Ameet	Mishra	GR2	DELHI	14780	9300
12	Anup	Save	GR2	MUMBAI	11960	7880
11	Adela	Thomas	GR2	DELHI	13660	6840
6	Sagar	Chavan	GR2	MUMBAI	13390	6700
2	Agatha	Williams	GR2	MUMBAI	12390	6630
1	Alan	Brown	GR1	DELHI	17990	16070
5	Neha	Rao	GR1	MUMBAI	19235	15200
3	Rajesh	Kolte	GR1	MUMBAI	19250	14960
7	Aaron	Jones	GR1	MUMBAI	23280	13490
9	Sneha	Joshi	GR1	DELHI	20660	NA

- Here, data is sorted by **Grade** and **ms** in decreasing order.
- By default missing values in data are put last.
- You can put it first by adding an argument **na.last=FALSE** in **order().**

Quick Recap

In this session, we learnt sorting data using **`order()`** in various ways. Here is a quick recap

Ascending/ Descending order	• **`order()`** by default sorts in ascending order. • For descending order: specify **`decreasing=TRUE`** for factor type variable and put'-' sign before numeric type variable
Multiple Columns	• **`order()`** allows us to sort by multiple columns of different type
Multiple columns and multiple ordering levels	• **`order()`** provides flexibility to order by multiple columns with different ordering levels