Checking & Modifying Data

Contents

Data Snapshot

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

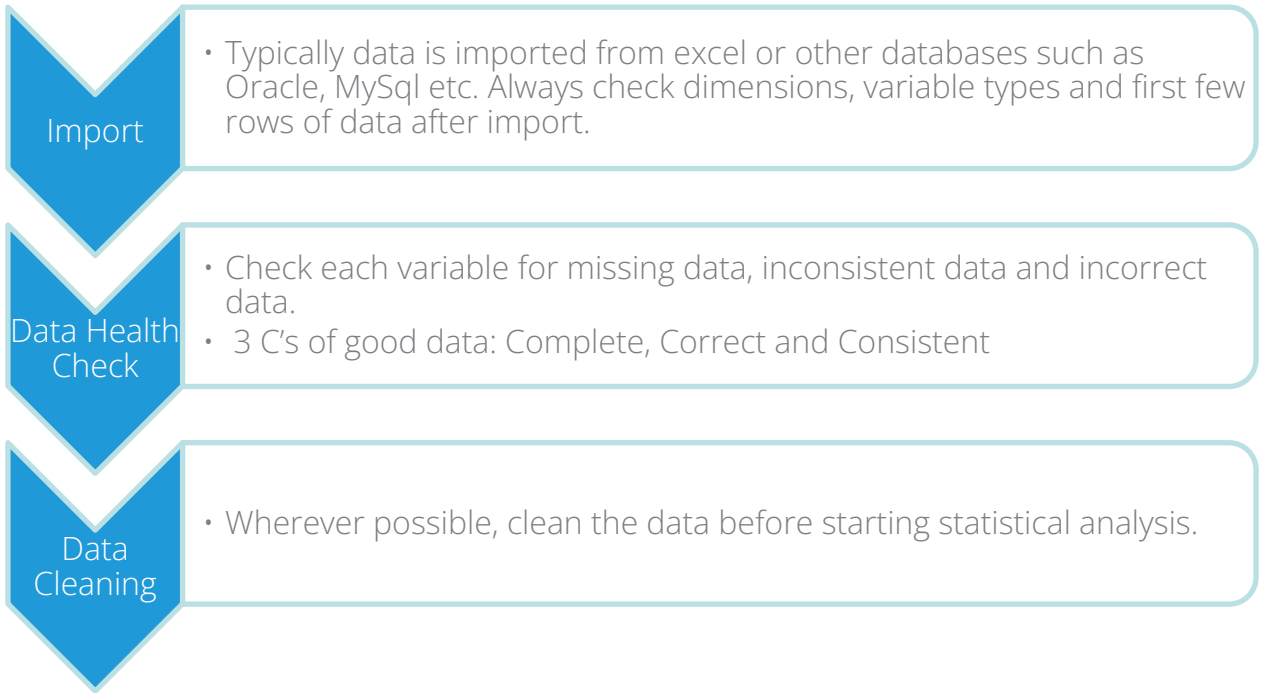
Variables

Observations

First_Name	Last_Name	Grade	Location	ba	ms
Alan	Brown	GR1	DELHI	17990	16070

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

First Look at the Data



Why Data Checking is Important?

After importing the data into R and before analyzing the data, it is very important to understand your data & check that it has maintained the correct format.

It is good practice to check number of rows/columns, variables types, number of missing values and first few rows of the data. .

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

Dimension of Data and Names of the Columns

Suppose we want to know **how many rows and columns are there in our data** and the **names of the columns** it contains, we could ask R like this:

```
# Retrieve the Dimension of your data using dim()
```

```
dim(salary_data)
```

```
 [1] 12  6
```

Data contains 12 rows and 6 columns

- Also if one wants to know no. of rows and columns separately, **`nrow()`** and **`ncol()`** command can be used respectively.

```
# Get the Names of the columns using names()
```

```
names(salary_data)
```

```
[1] "First_Name" "Last_Name"  "Grade"    "Location"
[5] "ba"         "ms"
```

Internal Structure of Data

When R reads data, it treats different variable types in different ways. **`str()`** is the easiest way to inspect how R treats variables in our dataframe. It compactly displays a dataframe's internal structure like this:

```
str(salary_data)
```

Character variables are entered into a dataframe as factors in R

```
# Output
'data.frame':   12 obs. of  6 variables:
 $ First_Name: Factor w/ 12 levels "Aaron","Adela",..: 4 3 10 5 9 11 1 8 12 7 ...
 $ Last_Name : Factor w/ 12 levels "Brown","Chavan",..: 1 12 5 6 8 2 3 7 4 10 ...
 $ Grade     : Factor w/ 2 levels "GR1","GR2": 1 2 1 2 1 2 1 2 1 2 ...
 $ Location  : Factor w/ 2 levels "DELHI","MUMBAI": 1 2 2 1 2 2 2 2 1 1 ...
 $ ba        : int  17990 12390 19250 14780 19235 13390 23280 13500 20660 13760 ...
 $ ms        : int  16070 6630 14960 9300 15200 6700 13490 10760 NA 13220 ...
```

Structure gives the following information:

- Class of the object like in this case 'salary_data' is of 'data.frame' class
- Type of the variable.
- Number of levels of each factor.
- Some values of the first few rows

Check the levels

Our data has 4 factor variables. A factor is a categorical variable that can take only one of a fixed, finite set of possibilities. Those possible categories are the levels. We can check the levels using **`levels()`** function

```
levels(salary_data$Grade)
```

```
[1] "GR1" "GR2"
```

```
# Assigning individual levels :
```

```
levels(salary_data$Grade)[1]<-“G1”
levels(salary_data$Grade)
```

```
[1] "G1" "GR2"
```

```
# Assigning levels as a group :
```

```
levels(salary_data$Grade)<-c(“G1”, “G2”)
levels(salary_data$Grade)
```

```
[1] "G1" "G2"
```

Check the Size of an Object

Suppose we want to know how much memory space is used to store **salary_data** object, we can use **object.size()** function to get an estimate in bytes.

```
object.size(salary_data)
```

```
4584 bytes
```

```
# Get the size of all the objects present in the current environment
```

```
for(i in ls()){
  message(i); print(object.size(get(i)))
}
```

```
a3
72 bytes
access_secret
136 bytes
access_token
.
.
```

This **for** loop prints all the objects in the current environment and their respective memory size in bytes

- **ls()** gives the vector of names of objects in the specified environment. With no argument, it shows what data sets and functions a user has defined.
- **message()** coerces the object/s to character (which is pasted together with no separator).
- **get()** returns the value of a named object.

Number of Missing Observations

Our data might contain some missing values or observations. In R missing data are usually recorded as NA. We can check the number of missing observations like this:

```
nmiss<-sum(is.na(salary_data$ms))
nmiss
```

```
[1] 1
```

- **$** operator after a data frame object helps in selecting a column from the data frame.
- **is.na()** returns a logical vector giving information about missing values.
- **sum()** displays the sum of missing observations.

First ‘n’ Rows of Data

Now if we want to have an idea about how our data looks like without displaying the entire data set, which could have millions of rows and thousands of columns then we can use **`head()`** to obtain first n observations.

```
head(salary_data)

# Output
```

	First_Name	Last_Name	Grade	Location	ba	ms
1	Alan	Brown	GR1	DELHI	17990	16070
2	Agatha	Williams	GR2	MUMBAI	12390	6630
3	Rajesh	Kolte	GR1	MUMBAI	19250	14960
4	Ameet	Mishra	GR2	DELHI	14780	9300
5	Neha	Rao	GR1	MUMBAI	19235	15200
6	Sagar	Chavan	GR2	MUMBAI	13390	6700

By default, **`head()`** displays the first 6 rows

First n Rows of Data

The no. of rows to be displayed can be customised to n

```
head(salary_data, n=2)
```

```
# Output
```

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

Last ‘n’ Rows of data

Now we will see the last n rows of our data using **`tail()`**. By default, it displays last 6 rows.

```
tail(salary_data)
```

```
# Output
```

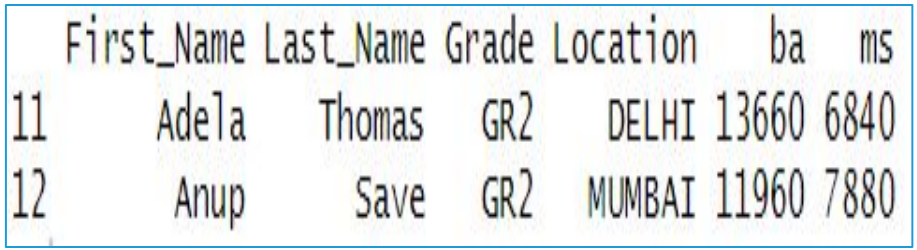
	First_Name	Last_Name	Grade	Location	ba	ms
7	Aaron	Jones	GR1	MUMBAI	23280	13490
8	John	Patil	GR2	MUMBAI	13500	10760
9	Sneha	Joshi	GR1	DELHI	20660	NA
10	Gaurav	Singh	GR2	DELHI	13760	13220
11	Adela	Thomas	GR2	DELHI	13660	6840
12	Anup	Save	GR2	MUMBAI	11960	7880

Last n Rows of Data

The no. of rows to be displayed can be customised to n

```
tail(salary_data, n=2)
```

```
# Output
```



	First_Name	Last_Name	Grade	Location	ba	ms
11	Adela	Thomas	GR2	DELHI	13660	6840
12	Anup	Save	GR2	MUMBAI	11960	7880

Summarizing Data

We can also inspect our data using **`summary()`**. This function gives summary of objects including datasets, variables, linear models, etc.

```
# Variables are summarized based on their type
summary(salary_data)
```

```
First_Name   Last_Name    Grade     Location
 Aaron  :1    Brown  :1   GR1:5   DELHI :5
 Adela  :1    Chavan :1   GR2:7   MUMBAI:7
 Agatha :1    Jones  :1
 Alan   :1    Joshi  :1
 Ameet  :1    Kolte  :1
 Anup   :1    Mishra :1
 (Other):6    (Other):6
       ba               ms
 Min.   :11960   Min.   : 6630
 1st Qu.:13472   1st Qu.: 7360
 Median :14270   Median :10760
 Mean   :16155   Mean   :11005
 3rd Qu.:19239   3rd Qu.:14225
 Max.   :23280   Max.   :16070
                 NA's   :1
```

When **summary()** is applied to a dataframe, it is essentially applied to each column, and it summarizes all the columns.

- For a continuous variable, it gives summary in the form of : min, 1st quantile, median, mean, 3rd quantile, max and count of NA's (if any).
- For a factor(categorical) variable, it gives the frequency of each level or category like in this case: Grade has 2 levels-GR1 and GR2 occurring 5 and 7 times respectively.

Quick Recap

In this session, we learnt how to check data features in R and why we should do it.

Here is the quick recap of functions used for checking data features:

Check the dimensionality and variable names or column names	• **`dim()`** returns the count of rows and columns • **`names()`** returns variable names or column names.
Check the compact internal structure, levels of categorical variable and size of an object	• **`str()`** returns many useful pieces of information like class of the object, data type of each column. • **`levels()`** returns the value of the levels of the object • **`object.size()`** returns the size of an object in bytes.
Check the missing values (if any), first and fast n rows	• **`is. na()`** returns a logical vector telling whether the data has missing values or not. • **`head()`** returns the first n rows of data. • **`tail()`** returns the last n rows of data.
Summaries data	• **`summary()`** summarizes the data based on the type of variable it contains.

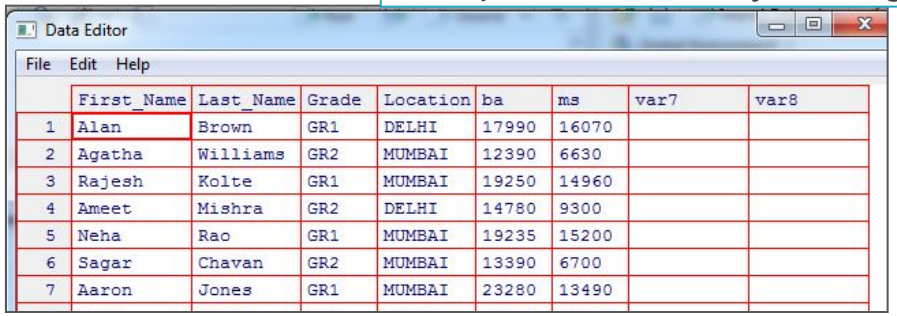
Change Variable Names and Content – fix()

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

In case we want to change the name of some variable or column and its values. We can use the following command which lets us make changes interactively.

```
fix(salary_data)
```

fix() function will open the data in a separate, editable Data Editor like the screenshot below. You can go to the desired column name or cell and make the changes manually. Complete the task by closing the dialogue box File >Close.



	First_Name	Last_Name	Grade	Location	ba	ms	var7	var8
1	Alan	Brown	GR1	DELHI	17990	16070		
2	Agatha	Williams	GR2	MUMBAI	12390	6630		
3	Rajesh	Kolte	GR1	MUMBAI	19250	14960		
4	Ameet	Mishra	GR2	DELHI	14780	9300		
5	Neha	Rao	GR1	MUMBAI	19235	15200		
6	Sagar	Chavan	GR2	MUMBAI	13390	6700		
7	Aaron	Jones	GR1	MUMBAI	23280	13490		

* Alternative function for editing data interactively is **edit()**, similar to **fix()**. The difference is: **edit()** lets you edit an object and returns the new version and **fix()** lets you edit an object and modifies the original.

Change Variable Names – names()

```
#Renaming column names with R’s built-in function names() :
```

```
names(salary_data)[names(salary_data)==“ba”] <- “basic allowance”
names(salary_data)

[1] "First_Name" "Last_Name"  "Grade"  "Location"  "basic allowance"
[6] "ms"
```

Note that this modifies **salary_data** directly; i.e. you don’t have to save the result back into **salary_data**.

Remove Columns from a Data Frame

```
# To remove column Last_Name from salary_data.
```

```
salary_data$Last_Name<-NULL
head(salary_data)
```

To remove a column, set it to **NULL**

```
  First_Name  Grade Location    ba    ms newvariable
1       Alan    GR1    DELHI 17990 16070      899.50
2     Agatha    GR2   MUMBAI 12390  6630      619.50
3     Rajesh    GR1   MUMBAI 19250 14960      962.50
4      Ameet    GR2    DELHI 14780  9300      739.00
5       Neha    GR1   MUMBAI 19235 15200      961.75
6      Sagar    GR2   MUMBAI 13390  6700      669.50
```

You can remove columns using integer indexing also like this:

```
salary_data[6]<-NULL
```

* **Index is the position of column. It starts from 1.**
In this case index number of First_Name is 1, Grade is 2 and so on and so forth.

Remove Rows from a Data Frame

We can remove unwanted rows from our data by using their index nos.

Suppose we want to remove rows 2, 3 and 4 from salary_data then we will write the following command:

```
salary_data[-(2:4),]
```

```
   First_Name Last_Name Grade Location    ba    ms
1        Alan     Brown   GR1    DELHI 17990 16070
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
11      Adela    Thomas   GR2    DELHI 13660  6840
12       Anup      Save   GR2   MUMBAI 11960  7880
```

Remove Rows from a Data Frame

```
# Remove only rows which has Location as 'MUMBAI'

salary_data[!(salary_data$Location=="MUMBAI"),]

   First_Name Last_Name Grade Location    ba    ms
1        Alan     Brown   GR1    DELHI 17990 16070
4       Ameet    Mishra   GR2    DELHI 14780  9300
9       Sneha     Joshi   GR1    DELHI 20660    NA
10     Gaurav     Singh   GR2    DELHI 13760 13220
11      Adela    Thomas   GR2    DELHI 13660  6840
```

Negation (**!**) symbol used to remove the row which satisfies the condition.

```
# You can also specify multiple conditions like this:

salary_data[!(salary_data$Location=="MUMBAI" & salary_data$Grade=="GR2"),]
```

Derive a New Variable

```
#Add a new variable serial_no to salary_data .
```

```
salary_data$serial_no <- c(1:nrow(salary_data))
head(salary_data,n=3)
```

Specify new variable name after $ operator and assign values to it

```
First_Name Last_Name Grade Location basic allowance    ms serial_no
1       Alan     Brown   GR1    DELHI           17990 16070         1
2     Agatha  Williams   GR2   MUMBAI           12390  6630         2
3     Rajesh     Kolte   GR1   MUMBAI           19250 14960         3
```

Variables are always added horizontally in a dataframe. One can use operators like * for multiplying, + for addition, - for subtraction, and / for division to create new variables.

Recode a Categorical Variable – ifelse()

One data manipulation task that you need to do in pretty much any data analysis is recode data. It's almost never the case that the data are set up exactly the way you need them for your analysis.

Let's recode **Location** 'MUMBAI' as 1 and 'DELHI' as 2

```
salary_data$Location<-ifelse(salary_data$Location=="MUMBAI", 1, 2)
head(salary_data)
```

```
  First_Name Last_Name Grade Location    ba    ms
1       Alan     Brown   GR1        2 17990 16070
2     Agatha  Williams   GR2        1 12390  6630
3     Rajesh     Kolte   GR1        1 19250 14960
4      Ameet    Mishra   GR2        2 14780  9300
5       Neha       Rao   GR1        1 19235 15200
6      Sagar    Chavan   GR2        1 13390  6700
```

Location column is recoded using **ifelse** statement.

* Make a note of this syntax. It's great for recoding within R programs

Recode a Continuous Variable into Categorical Variable – ifelse()

Let's categorise the employees on the basis of their basic allowance(ba) in three categories, namely, Low, Medium and High

```
salary_data$category<-ifelse(salary_data$ba <14000, "Low",
ifelse(salary_data$ba <19000, "Medium", "High"))
head(salary_data)
```

```
  First_Name Last_Name Grade Location    ba    ms category
1       Alan     Brown   GR1        2 17990 16070   Medium
2     Agatha  Williams   GR2        1 12390  6630      Low
3     Rajesh     Kolte   GR1        1 19250 14960     High
4      Ameet    Mishra   GR2        2 14780  9300   Medium
5       Neha       Rao   GR1        1 19235 15200     High
6      Sagar    Chavan   GR2        1 13390  6700      Low
```

Nested **ifelse** statement is used

```
class(salary_data$category)
```

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

Note that column **category** is of character type. You will have to convert it to a factor using **as.factor().**

Get an Edge!

- Another way of recoding continuous variable to categorical variable is through cut() function. Only difference is you don't need to convert its output into factor as it by default produces output of factor type.
- Previous example using **cut()**:

```
salary_data$category<-cut(salary_data$ba,breaks=c(0,14000,19000,Inf),labels=c("Low","Medium","High"))
```

- Note that 14000 is assigned to "Low", 19000 to "Medium" and 0 to <NA>
- Cut points giving the number of intervals into which ba is to be cut are defined using **breaks=**
- Labels for the levels of the resulting category. are defined using **levels=**

Quick Recap

In this session, we learnt how to modify data in different ways. Here is the quick recap:

Change variable names	• **`edit()`** lets you edit an object and returns the new version • **`fix()`** lets you edit an object and modifies the original**.** • **`names()`** is a built-in function which lets you rename the column and modifies the original object
Change the content of data	• **`edit()`** and **`fix()`** lets you change the content of data
Remove a column	• Specify column name after **`$`** operator and assign **`NULL`** to it. • Specify the index in **`[]`** brackets and assign **`NULL`** to it
Add a new column	• Specify new variable name after **`$`** operator and assign values to it
Recoding	• Categorical Variable using **`ifelse`** statement • Continuous variable to categorical variable using nested **`ifelse`** statement