CSE1006
Foundations of Data Analytics

Adla Padma
Assistant Professor
School of Computer Science & Engineering
VIT-AP University

Module - 3

Data manipulation

- Data sorting
- Find and remove duplicates record
- Cleaning data
- Recording data
- Merging data

Data Sorting

- R provides a different way to sort the data either in ascending or descending order.
- Data-analysts, and Data scientists use order(), sort() and packages like dplyr to sort data depending upon the structure of the obtained data.
- The function **Order()** can sort Vector, Matrix and also a Dataframe can be sorted in ascending and descending order.

Syntax of order():

```
order(x, [decreasing = TRUE or FALSE], [na.last = TRUE or FLASE],
[method = c("auto", "shell", "quick", "radix")])
```

- The first argument **x** is the vector or data frame to be sorted.
- The second argument **decreasing** is a logical value that determines whether the sorting should be in decreasing order (**TRUE**) or increasing order (**FALSE**).
- The third argument **na.last** is also a logical value that determines whether missing values should be placed at the end of the sorted vector (**TRUE**) or at the beginning (**FALSE**).
- The fourth argument **method** is an optional argument that specifies the sorting algorithm to be used.

- The available options are "auto" (default), "shell", "quick", and "radix".
- The "auto" option automatically selects the best algorithm based on the size and type of the input data.
- Overall, this code snippet is used to sort a vector or data frame in R using the order() function with various options for sorting order, handling of missing values, and sorting algorithm.

Example:

```
y = c(4,12,6,7,2,9,5)
order(y)
```

- In this case, the output would be: 5 1 7 3 4 6 2, which indicates that the smallest value is at index 5, the next smallest is at index 1, and so on.
- Note that the original vector "y" is not modified by this function.

Example:

```
y = c(4,12,6,7,2,9,5)
y[order(y)]
```

- The "order" function returns the indices of the sorted values, so when we use these indices to subset "y" with square brackets, we get the sorted values of "y".
- Therefore, the output of this code will be: 2 4 5 6 7 9 12.

Example:

```
x <- c(8,2,4,1,-4,NA,46,8,9,5,3)
order(x,na.last = TRUE)
x[order(x)]
```

- The "na.last = TRUE" argument specifies that missing values (NA) should be placed at the end of the sorted vector.
- So, the output of this code would be:

```
 [1]  5  4  2 11  3 10  1  8  9  7  6
 [1] -4  1  2  3  4  5  8  8  9 46 NA
```

Example:

```
x <- c(8,2,4,1,-4,NA,46,8,9,5,3)
order(x,na.last=FALSE)
```

- The "na.last = FALSE" argument specifies that missing values (NA) should be placed at the end of the sorted vector.
- Therefore, the output of this code will be:

```
[1]  6  5  4  2 11  3 10  1  8  9  7
[1] NA -4  1  2  3  4  5  8  8  9 46
```

Example:

```
x <- c(8,2,4,1,-4,NA,46,8,9,5,3)
order(x,decreasing=TRUE,na.last=TRUE)
```

- The **decreasing=TRUE** argument specifies that the indices should be sorted in descending order instead of ascending order.
- The **na.last=TRUE** argument specifies that missing values should be placed at the end of the sorted vector.
- Overall, this code sorts the input vector **x** in descending order and returns the indices that would sort **x** in that order.

```
[1]  7  9  1  8  3 10  2 11  4  5  6
[1] 46  9  8  8  4  5  3  2  1 -4 NA
```

- In R DataFrame is a two-dimensional tabular data structure that consists of rows and columns.
- Sorting a DataFrame allows us to reorder the rows based on the values in one or more columns.
- This can be useful for various purposes, such as organizing data for analysis or presentation.
- Methods to sort a dataframe:
  - order() function (increasing and decreasing order)
  - arrange() function from dplyr package
  - setorder() function from data.table package

Using order() function:

This function is used to sort the dataframe based on the particular column in the dataframe.

Syntax: order(dataframe$column_name,decreasing = TRUE))

where

dataframe is the input dataframe

Column name is the column in the dataframe such that dataframe is sorted based on this column

Decreasing parameter specifies the type of sorting order - If it is TRUE dataframe is sorted in descending order. Otherwise, in increasing order.

return type: Index positions of the elements.

Example 1: R program to create a dataframe with 2 columns and order based on particular columns in decreasing order. Displayed the Sorted dataframe based on subjects in decreasing order, displayed the Sorted dataframe based on roll no in decreasing order.

```
# create dataframe with roll no and subjects columns
        data = data.frame(
                rollno = c(1, 5, 4, 2, 3),
                subjects = c("java", "python", "php", "sql", "c"))
        print(data)
        print("sort the data in decreasing order based on subjects
")
        print(data[order(data$subjects, decreasing = TRUE), ]   )
        print("sort the data in decreasing order based on rollno ")
        print(data[order(data$rollno, decreasing = TRUE), ]   )
```

```
print(data)
print("sort the data in decreasing order based on subjects ")
print(data[order(data$subjects, decreasing = TRUE), ]   )
print("sort the data in decreasing order based on rollno ")
print(data[order(data$rollno, decreasing = TRUE), ]   )
```

```
  rollno subjects
1      1     java
2      5   python
3      4      php
4      2      sql
5      3        c
[1] "sort the data in decreasing order based on subjects "
  rollno subjects
4      2      sql
2      5   python
3      4      php
1      1     java
5      3        c
[1] "sort the data in decreasing order based on rollno "
  rollno subjects
2      5   python
3      4      php
5      3        c
4      2      sql
1      1     java
```

Example 2: R program to create a dataframe with 3 columns named roll no, names, and subjects with a vector, displayed the Sorted dataframe based on subjects in increasing order, displayed the Sorted dataframe based on roll no in increasing order, displayed the Sorted dataframe based on names in increasing order

```
# create dataframe with roll no, names and subjects columns
data=data.frame(rollno = c(1, 5, 4, 2, 3),
                names = c("sravan", "bobby","pinkey", "rohith","ganesh"),
                subjects = c("java", "python","php", "sql", "c"))

print(data)

print("sort the data in increasing order based on subjects")
print(data[order(data$subjects, decreasing = FALSE), ]   )


print("sort the data in increasing order based on rollno")
print(data[order(data$rollno, decreasing = FALSE), ]   )

print("sort the data in increasing order based on names")
print(data[order(data$names,decreasing = FALSE), ]   )
```

Sorting Matrix Elements

```
> m1<-matrix(sample(1:100,9),ncol=3)
> m1
     [,1] [,2] [,3]
[1,]  100   64   69
[2,]    6   14   89
[3,]   70   66   83
> order(m1)
[1] 2 5 4 6 7 3 9 8 1
> m1[order(m1)]
[1]   6  14  64  66  69  70  83  89 100
> m1[order(m1[,2])
+ ]
[1]   6 100  70
> 
```

Find and remove duplicates record

- Storage administrators are struggling to handle spiraling volumes of documents, audio, video, images and large email attachments
- Adding storage is not always the best solution
- Many companies are turning to data reduction technologies such as data deduplication.

Data Duplicacy:

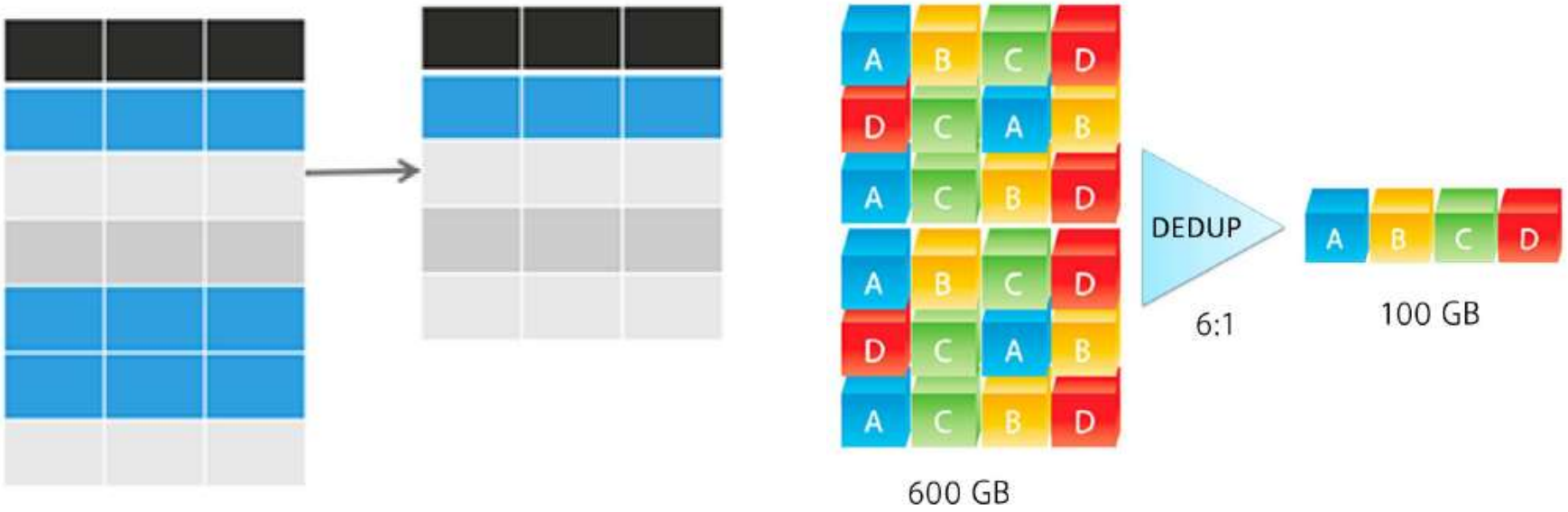
- Entries that have been added by a system user multiple times
- for example, re registering because you have forgotten your details, It is one of the problem which causes inconsistency in databases.

Data Redundancy:

- Same of data is stored at multiple locations or tables.
- Data redundancy is costly to address as it requires
  - additional storage, synchronization between databases.
  - design work to align the information represented by different presentation of the same data.

- A dataset can have duplicate values and to keep it redundancy-free and accurate, duplicate rows need to be identified and removed.
- In this module, we will see how to identify and remove duplicate data in R. First we will check if duplicate data is present in our data, if yes then, we will remove it.

Removing Duplicate Data in R



Identifying Duplicate Data in vector:

We can use duplicated() function to find out how many duplicates value are present in a vector.

Syntax : duplicated(vector_name)

The R function duplicated() returns a logical vector where TRUE specifies which elements of a vector or data frame are duplicates.

Example:

```
# Create a sample vector with duplicate elements
vector1<- c(1, 2, 3, 4, 4, 5)
# Identify duplicate elements
duplicated(vector1)  #FALSE   FALSE   FALSE   FALSE   TRUE   FALSE
# count of duplicated data
sum(duplicated(vector1))   #1
```

Removing Duplicate Data in vector:

We can remove duplicate data from vectors by using unique() functions so it will give only unique values.

Syntax : unique(vector_name)

The R function duplicated() returns a logical vector where TRUE specifies which elements of a vector or data frame are duplicates.

Example:

```
# Create a sample vector with duplicate elements
vector1<- c(1, 2, 3, 4, 4, 5)
# Remove duplicate elements
unique(vector1)      #1      2       3       4       5
```

Identifying Duplicate Data in a data frame:

For identification, we will use the duplicated() function.

Syntax : duplicated(dataframe)

Approach:

Create data frame

Pass it to duplicated() function

This function returns the rows which are duplicated in form of boolean values

Apply the sum() function to get the number of duplicates.

Removing Duplicate Data in a data frame:

we use unique() and distinct() functions.

Approach:

Create data frame

Select rows which are unique

Retrieve those rows

Display result

Identifying Duplicate Data in a data frame:

```
# Creating a sample data frame of students and their marks in respective subjects.
studen_result=data.frame(name=c("Ram","Geeta","John","Paul","Cassie","Geeta","Paul"),
                         maths=c(7,8,8,9,10,8,9),
                         science=c(5,7,6,8,9,7,8),
                         history=c(7,7,7,7,7,7,7))

# Printing data
student_result
duplicated(student_result)
sum(duplicated(student_result))
```

```
    name maths science history
1    Ram     7       5       7
2  Geeta     8       7       7
3   John     8       6       7
4   Paul     9       8       7
5 Cassie    10       9       7
6  Geeta     8       7       7
7   Paul     9       8       7
[1] FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE
[1] 2
```

Identifying Duplicate Data in a data frame:

```
# create dataframe
data=data.frame(names=c("manoj","bobby","sravan",
"deepu","manoj","bobby") ,
                id=c(1,2,3,1,1,2),
                subject=c("java","python","php","html","java","python"))
data
# remove duplicate rows in subject column
print(data[!duplicated(data$subject), ])
# remove  duplicate rows in names column
print(data[!duplicated(data$names), ])
# remove  duplicate rows in  id column
print(data[!duplicated(data$id), ])
```

```
   names id subject
1  manoj  1    java
2  bobby  2  python
3 sravan  3     php
4  deepu  1    html
5  manoj  1    java
6  bobby  2  python
```

```
   names id subject
1  manoj  1    java
2  bobby  2  python
3 sravan  3     php
4  deepu  1    html
```

```
   names id subject
1  manoj  1    java
2  bobby  2  python
3 sravan  3     php
```

Removing Duplicate Data in a data frame:

Method 1: Using unique()

We use unique() to get rows having unique values in our data.

Syntax: unique(dataframe)

Example:

```
# Creating a sample data frame of students and their marks in respective subjects.
student=data.frame(name=c("Ram","Geeta","John","Paul“,"Cassie","Geeta", "Paul"),
                                        maths=c(7,8,8,9,10,8,9),
                                        science=c(5,7,6,8,9,7,8),
                                        history=c(7,7,7,7,7,7,7))

# Printing data
student
# Printing data without duplicates using unique
unique(student)
```

```
    name maths science history
1    Ram     7       5       7
2  Geeta     8       7       7
3   John     8       6       7
4   Paul     9       8       7
5 Cassie    10       9       7
6  Geeta     8       7       7
7   Paul     9       8       7
    name maths science history
1    Ram     7       5       7
2  Geeta     8       7       7
3   John     8       6       7
4   Paul     9       8       7
5 Cassie    10       9       7
```

Method 2: Using distinct()

- This method is available in dplyr package which is used to get the unique rows from the dataframe.
- We can remove rows from the entire which are duplicates and also we can remove duplicate rows in a particular column.

Syntax: distinct(dataframe,keepall)

Where dataframe -> data in use and keepall -> decides which variables to keep

Example:

```
library(tidyverse) #load library
# Creating a sample data frame of students and their marks in respective subjects.
student=data.frame(name=c("Ram","Geeta","John","Paul“,"Cassie","Geeta", "Paul"),
                                        maths=c(7,8,8,9,10,8,9),
                                        science=c(5,7,6,8,9,7,8),
                                        history=c(7,7,7,7,7,7,7))
 student  # Printing data
dplyr::distinct(student) # Printing data without duplicates using distinct
```

```
    name maths science history
1    Ram     7       5       7
2  Geeta     8       7       7
3   John     8       6       7
4   Paul     9       8       7
5 Cassie    10       9       7
6  Geeta     8       7       7
7   Paul     9       8       7
    name maths science history
1    Ram     7       5       7
2  Geeta     8       7       7
3   John     8       6       7
4   Paul     9       8       7
5 Cassie    10       9       7
```

```
# load the package
library(dplyr)
# create dataframe
data=data.frame(names=c("manoj","bobby","sravan","deepu","ma
noj","bobby") ,
                id=c(1,2,3,4,1,2),

subjects=c("java","python","php","html","java","python")
# remove all duplicate rows
print(dplyr::distinct(data))
# remove  duplicate rows in subjects column
print(dplyr::distinct(data,subjects))
# remove  duplicate rows in namescolumn
print(dplyr::distinct(data,names))
```

# remove all duplicate rows
print(dplyr::distinct(data))

```
    names id subjects
1   manoj  1     java
2   bobby  2   python
3  sravan  3      php
4   deepu  4     html
```

# remove duplicate rows in subjects column
print(dplyr::distinct(data,subjects))

```
   subjects
1      java
2    python
3       php
4      html
```

# remove duplicate rows in namescolumn
print(dplyr::distinct(data,names))

```
     names
1    manoj
2    bobby
3   sravan
4    deepu
```

Example: # Creating a sample data frame of students and their marks in respective subjects.

```
student=data.frame(name=c("Ram","Geeta","John","Paul",
"Cassie","Geeta","Paul"),
                   maths=c(7,8,8,9,10,8,9),
                   science=c(5,7,6,8,9,7,8),
                   history=c(7,7,7,7,7,7,7))

student_result  # Printing data
 dplyr::distinct(student_result,maths,.keep_all = TRUE)
```

```
     name maths science history
1     Ram     7       5       7
2   Geeta     8       7       7
3    John     8       6       7
4    Paul     9       8       7
5  Cassie    10       9       7
6   Geeta     8       7       7
7    Paul     9       8       7
     name maths science history
1     Ram     7       5       7
2   Geeta     8       7       7
3    Paul     9       8       7
4  Cassie    10       9       7
```

Duplicates in Matrix

```
> m2<-matrix(c(rep(1,6),7:9),3)
> m2
     [,1] [,2] [,3]
[1,]    1    1    7
[2,]    1    1    8
[3,]    1    1    9
> duplicated(m2)
[1] FALSE FALSE FALSE
> duplicated(m2[,2])
[1] FALSE  TRUE  TRUE
> unique(m2)
     [,1] [,2] [,3]
[1,]    1    1    7
[2,]    1    1    8
[3,]    1    1    9
```

Cleaning data

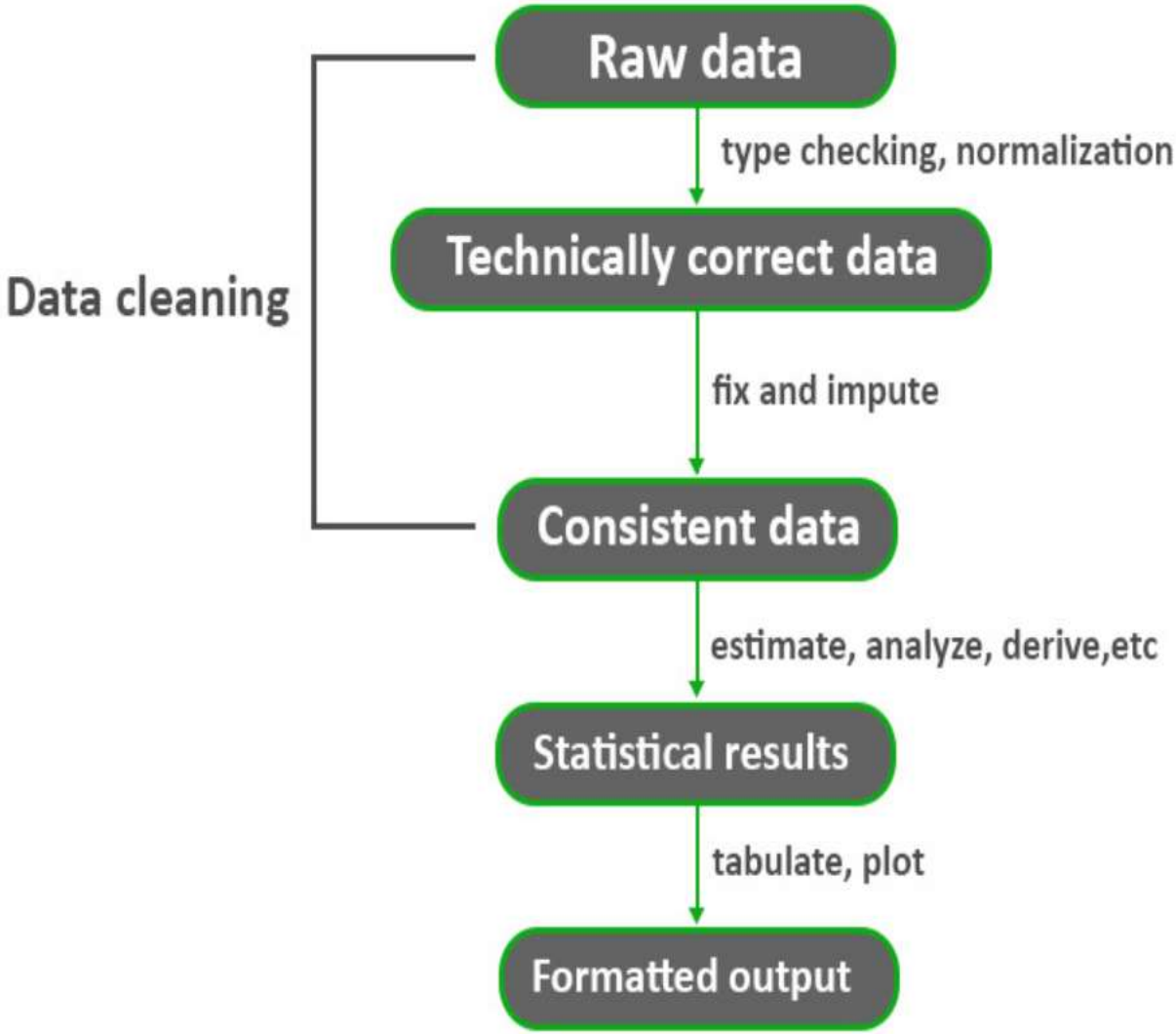
- Data Cleaning in R is the process to transform raw data into consistent data that can be easily analysed.
- It is aimed at filtering the content of statistical statements based on the data as well as their reliability.
- Moreover, it influences the statistical statements based on the data and improves your data quality and overall productivity.

Purpose of Data Cleaning:

The following are the various purposes of data cleaning in R

Eliminate Errors
Eliminate Redundancy
Increase Data Reliability
Delivery Accuracy
Ensure Consistency
Assure Completeness
Standardize your approach

Overview of a typical data analysis chain:



Example1:

For better understanding let us discuss with one example

1) Creation of Example Data (Data Frame)
2) Modify Column Names
3) Format Missing Values
4) Remove Empty Rows & Columns
5) Remove Rows with Missing Values
6) Remove Duplicates
7) Modify Classes of Columns
8) Detect & Remove Outliers
9) Remove Spaces in Character Strings

1) Creation of Example Data:

```
data <- data.frame(x1 = c(1:4, 99999, 1, NA, 1, 1, NA),   # Create example data frame
                   x1 = c(1:5, 1, "NA", 1, 1, "NA"),
                   x1 = c(letters[c(1:3)], "x  x",  "x", "   y    y y", "x", "a", "a", NA),
                   x4 = "",
                   x5 = NA)
data                  # Printing data frame
```

	x1	x1.1	x1.2	x4	x5
1	1	1	a		NA
2	2	2	b		NA
3	3	3	c		NA
4	4	4	x x		NA
5	99999	5	x		NA
6	1	1	y y y		NA
7	NA	NA	x		NA
8	1	1	a		NA
9	1	1	a		NA
10	NA	NA	NA		NA

2) Modify Column Names:

```
data <- data.frame(x1 = c(1:4, 99999, 1, NA, 1, 1, NA),   # Create example data frame
                   x1 = c(1:5, 1, "NA", 1, 1, "NA"),
                   x1 = c(letters[c(1:3)], "x  x",  "x",  "   y    y y", "x", "a", "a",
NA),
                   x4 = "",
                   x5 = NA)
colnames(data)
ncol(data)
```

- The colnames() function returns or sets the names
- of the columns in a data frame.
- ncol() function in R Language is used to return the number of columns of the object.

```
      x1 x1.1         x1.2 x4 x5
1      1    1            a    NA
2      2    2            b    NA
3      3    3            c    NA
4      4    4          x  x    NA
5  99999    5            x    NA
6      1    1    y     y y    NA
7     NA   NA            x    NA
8      1    1            a    NA
9      1    1            a    NA
10    NA   NA         <NA>    NA
[1] "x1"   "x1.1" "x1.2" "x4"   "x5"
[1] 5
```

2) Modify Column Names:

Let's assume that we want to change these column names to a consecutive range with the prefix "col". Then, we can apply the **colnames**, **paste0**, and **ncol** functions as shown below.

```
# Modify all column names
colnames(data) <- paste0("col", 1:ncol(data))
data          # Print updated data frame
```

	col1	col2	col3	col4	col5
1	1	1	a		NA
2	2	2	b		NA
3	3	3	c		NA
4	4	4	x x		NA
5	99999	5	x		NA
6	1	1	y y y		NA
7	NA	NA	x		NA
8	1	1	a		NA
9	1	1	a		NA
10	NA	NA	NA		NA

3) Format Missing Values:

- In the R programming language, missing values are usually represented by NA. For that reason, it is useful to convert all missing values to this NA format.
- In our specific example data frame, we have the problem that some missing values are represented by blank character strings.
- If we want to assign NA values to those blank cells, we can use the following syntax

```
data[data == ""] <- NA
data[data == "NA"] <- NA
data
```

	col1	col2	col3	col4	col5
1	1	1	a	NA	NA
2	2	2	b	NA	NA
3	3	3	c	NA	NA
4	4	4	x x	NA	NA
5	99999	5	x	NA	NA
6	1	1	y y y	NA	NA
7	NA	NA	x	NA	NA
8	1	1	a	NA	NA
9	1	1	a	NA	NA
10	NA	NA	NA	NA	NA

4) Remove Empty Rows & Columns:

The syntax below demonstrates how to use the rowSums, is.na, and ncol functions to remove only-NA rows.

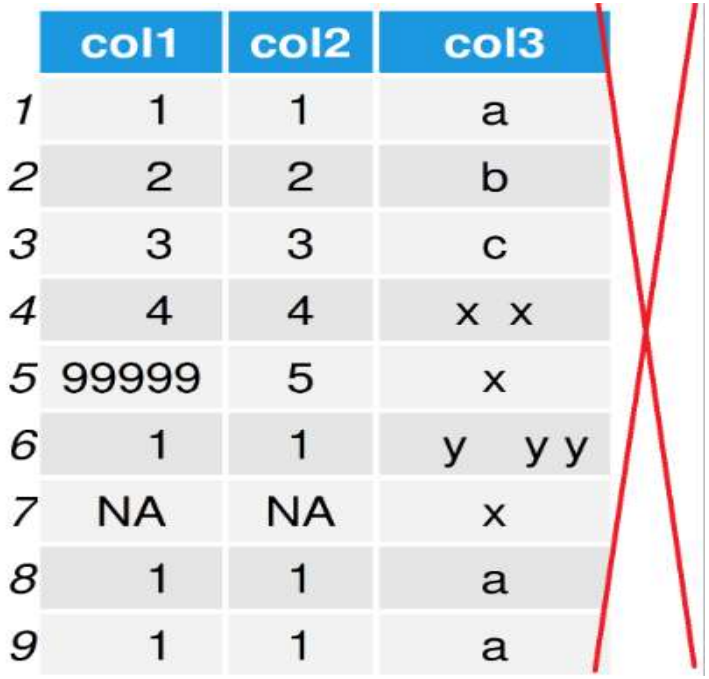
```
# Drop empty rows

data <- data[rowSums(is.na(data)) != ncol(data), ]
data                    # Print updated data frame
```

	col1	col2	col3	col4	col5
1	1	1	a	NA	NA
2	2	2	b	NA	NA
3	3	3	c	NA	NA
4	4	4	x x	NA	NA
5	99999	5	x	NA	NA
6	1	1	y y y	NA	NA
7	NA	NA	x	NA	NA
8	1	1	a	NA	NA
9	1	1	a	NA	NA

4) Remove Empty Rows & Columns:

Similar to that, we can also exclude columns that contain only NA values.

```
data <- data[ , colSums(is.na(data)) != nrow(data), ]
# Drop empty cols
data                    # Print updated data frame
```



	col1	col2	col3
1	1	1	a
2	2	2	b
3	3	3	c
4	4	4	x x
5	99999	5	x
6	1	1	y y y
7	NA	NA	x
8	1	1	a
9	1	1	a

5) Remove Rows with Missing Values:

However, in case you have decided to remove all rows with one or more NA values, you may use the **na.omit** function as shown below.

```
data <- na.omit(data)   # Drop rows with missing vals
data                    # Print updated data frame
```

	col1	col2	col3
1	1	1	a
2	2	2	b
3	3	3	c
4	4	4	x x
5	99999	5	x
6	1	1	y y y
8	1	1	a
9	1	1	a

6) Remove Duplicates:

we can apply the unique function to our data frame to remove duplicates.

```
data <- unique(data)   # delete duplicate rows
data                   # Print updated data frame
```

	col1	col2	col3
1	1	1	a
2	2	2	b
3	3	3	c
4	4	4	x x
5	99999	5	x
6	1	1	y y y

7) Modify Classes of Columns:

- The class of the columns of a data frame is another critical topic when it comes to data cleaning.
- This example explains how to format each column to the most appropriate data type automatically.
- Let's first check the current classes of our data frame columns.

```
sapply(data, class)                  # Print classes of all columns
#      col1          col2          col3
#  "numeric" "character" "character“
```

- We can now use the type.convert function to change the column classes whenever it is appropriate

```
data <- type.convert(data, as.is = TRUE)
sapply(data, class)                  # Print classes of all columns
#       col1       col2       col3
#  “integer" “integer" "character“
```

8) Detect & Remove Outliers:

- One method to detect outliers is provided by the boxplot.stats function. The following R code demonstrates how to test for outliers in our data frame column col1

```
# Identify outliers in column
data$col1[data$col1 %in% boxplot.stats(data$col1)$out]
[1] 99999 #This value is obviously much higher than the other values in this column.
```

- Let's assume that we have confirmed theoretically that the observation containing this outlier should be removed. Then, we can apply the R code below

```
# Remove rows with outliers
data <- data[! data$col1 %in% boxplot.stats(data$col1)$out, ]
data
```

	col1	col2	col3
1	1	1	a
2	2	2	b
3	3	3	c
4	4	4	x x
6	1	1	y y y

9) Remove Spaces in Character Strings:

- The manipulation of character strings is another important aspect of the data cleaning process.
- This example demonstrates how to avoid blank spaces in the character strings of a certain variable.
- For this task, we can use the **gsub** function as demonstrated below:

```
# Delete white space in character strings
data$col3 <- gsub(" ", "", data$col3)
data
```

	col1	col2	col3
1	1	1	a
2	2	2	b
3	3	3	c
4	4	4	xx
6	1	1	yyy

```r
# Create example data frame with unique column names
data <- data.frame(
  x1 = c(1:4, 99999, 1, NA, 1, 1, NA),
  x2 = c(1:5, 1, "NA", 1, 1, "NA"),
  x3 = c(letters[c(1:3)], "x  x", "x", "   y    y y", "x", "a", "a", NA),
  x4 = "",  # Empty column
  x5 = NA   # Column with only NA
)

print("Original Data:")
print(data)

# Rename columns dynamically
colnames(data) <- paste0("col", 1:ncol(data))

print("After Renaming Columns:")
print(data)

# Convert empty strings and "NA" strings to NA
data[data == ""] <- NA
data[data == "NA"] <- NA

print("After Converting Empty Strings to NA:")
print(data)

# Remove rows where all values are NA
data <- data[rowSums(is.na(data)) != ncol(data), ]

print("After Removing Fully NA Rows:")
print(data)

# Remove columns where all values are NA
data <- data[, colSums(is.na(data)) != nrow(data)]

print("After Removing Fully NA Columns:")
print(data)
```

```r
# Drop rows with any missing values
data <- na.omit(data)

print("After Dropping Rows with Missing Values:")
print(data)

# Remove duplicate rows
data <- unique(data)

print("After Removing Duplicate Rows:")
print(data)

# Convert data types automatically
data <- type.convert(data, as.is = TRUE)

print("After Converting Data Types:")
print(sapply(data, class))

# Detect and remove outliers in col1 using boxplot statistics
outliers <- data$col1[data$col1 %in% boxplot.stats(data$col1)$out]
print("Outliers in col1:")
print(outliers)

data <- data[!data$col1 %in% boxplot.stats(data$col1)$out, ]

print("After Removing Outliers in col1:")
print(data)

# Remove spaces from col3
data$col3 <- gsub(" ", "", data$col3)

print("After Removing Spaces in col3:")
print(data)
```

Example 2:

Create a data frame patients with columns PatientID, Name, Age, Gender, Disease, AdmissionDate, TreatmentCost, inserting at least 10 rows with missing values (NA), duplicate entries, and extra spaces in text fields.

- Rename columns to P_ID, Full_Name, Age_Years, Gender_Type, Diagnosis, Admit_Date, Cost_Treatment.
- Replace empty strings and "NA" stored as text with actual NA values.
- Remove rows where all values are NA and drop columns that contain only NA values.
- Drop rows with at least one missing value and remove duplicate rows.
- Convert PatientID to integer, AdmissionDate to Date format, and TreatmentCost to numeric.
- Identify and remove outliers in Age_Years and Cost_Treatment using boxplot statistics.
- Remove spaces from Full_Name and Diagnosis columns.

Example 3:

Create a data frame shipments with columns ShipmentID, Sender, Receiver, Weight, ShipmentDate, DeliveryDate, Status, Cost, inserting at least 10 rows with missing values (NA), duplicate records, and extra spaces in text fields.

- Rename columns to S_ID, Sender_Name, Receiver_Name, Weight_Kg, Ship_Date, Del_Date, Delivery_Status, Shipping_Cost.
- Replace empty strings and "NA" stored as text with actual NA values.
- Remove rows where all values are NA and drop columns that contain only NA values.
- Drop rows with at least one missing value and remove duplicate rows.
- Convert ShipmentID to integer, ShipmentDate and DeliveryDate to Date format, and Cost to numeric.
- Identify and remove outliers in Weight_Kg and Shipping_Cost using boxplot statistics.
- Remove spaces from Sender_Name, Receiver_Name, and Delivery_Status columns.

Data Recoding

- Recoding allows you to create new variables and to replace existing values of a variables based on a criterion.

Example: Let us consider a Data frame

```
df <- data.frame(player = c('P1', 'P2', 'P3', 'P4'),
                 points = c(124, 229, 313, 415),
                 result = c('Win', 'Loss', 'Win', 'Loss'))
df
```

Output:

```
  player points result
1     P1    124    Win
2     P2    229   Loss
3     P3    313    Win
4     P4    415   Loss
```

- To recode, The easiest way is to use revalue() or mapvalues() from the plyr package. These are defined in **plyr** package.

Example:

```
df <- data.frame(player = c('P1', 'P2', 'P3', 'P4'),
                 points = c(124, 229, 313, 415),
                 result = c('Win', 'Loss', 'Win', 'Loss'))
df
df$scode <- plyr::revalue(df$result, c("Win"="1", "Loss"="2"))
```

	player	points	result
1	P1	124	Win
2	P2	229	Loss
3	P3	313	Win
4	P4	415	Loss

#Creaing new Variable

```
df
```

	player	points	result	scode
1	P1	124	Win	1
2	P2	229	Loss	2
3	P3	313	Win	1
4	P4	415	Loss	2

```
df$result <- plyr::mapvalues(df$result, from = c("Win","Loss"),
to = c("1", "0"))
df
```

#Modify the existing variable

	player	points	result	scode
1	P1	124	1	1
2	P2	229	0	2
3	P3	313	1	1
4	P4	415	0	2

It is also possible to recode using ifelse.

Example:

```
df <- data.frame(player = c('P1', 'P2', 'P3', 'P4'),
                 points = c(124, 229, 313, 415),
                 result = c('Win', 'Loss', 'Win', 'Loss'))
df
```

	player	points	result
1	P1	124	Win
2	P2	229	Loss
3	P3	313	Win
4	P4	415	Loss

```
df$scode <- ifelse(df$result=="Win",1,2)
df                          #Creaing new Variable
```

	player	points	result	scode
1	P1	124	Win	1
2	P2	229	Loss	2
3	P3	313	Win	1
4	P4	415	Loss	2

```
df$result <- ifelse(df$result=="Win",1,0)
df                                  #Modify the existing variable
```

	player	points	result	scode
1	P1	124	1	1
2	P2	229	0	2
3	P3	313	1	1
4	P4	415	0	2

Data Merging

- Merging data is a common task in data analysis, especially when working with large datasets.
- The merge function in R is a powerful tool that allows you to combine two or more datasets based on shared variables.
- In R there are various ways to merge data frames, using the 'merge()' function from base R, using the 'dplyr' package, and the 'data.table' package.

Using 'merge()' from base R:

- The merge() function in base R helps us to combine two or more data frames based on common columns.
- It performs various types of joins such as inner join, left join, right join, and full join.

Syntax: merged_df <- merge(x,y,by = "common_column",..)

- x' and 'y' are the data frames that you want to merge.
- 'by' specifies the common columns on which the merge will be performed.
- Additional arguments like 'all.x',all.y' and 'all' control the type of join that is to be performed.

Example:

Consider two data frames ‘df1’ and ‘df2’

```
df1 <- data.frame(ID = c(1, 2, 3, 4),
                  Name = c("A", "B", "C", "D"),
                  Age = c(25, 30, 35, 40))
```

	ID	Name	Age
1	1	A	25
2	2	B	30
3	3	C	35
4	4	D	40

```
df2 <- data.frame(ID = c(2, 3, 4, 5),
                  Occupation = c("Engineer", "Teacher", "Doctor", "Lawyer"),
                  Salary = c(5000, 4000, 6000, 7000))
```

	ID	Occupation	Salary
1	2	Engineer	5000
2	3	Teacher	4000
3	4	Doctor	6000
4	5	Lawyer	7000

1. Inner join (default behavior):

```
inner_join <- merge(df1, df2, by = "ID")
print(inner_join)
```

*The resulting ‘inner_join’ dataframe will only include the common rows where **‘ID’** is present in **both ‘df1’ and ‘df2’.***

	ID	Name	Age	Occupation	Salary
1	2	B	30	Engineer	5000
2	3	C	35	Teacher	4000
3	4	D	40	Doctor	6000

Example:

Df1->

	ID	Name	Age
1	1	A	25
2	2	B	30
3	3	C	35
4	4	D	40

df2->

	ID	Occupation	Salary
1	2	Engineer	5000
2	3	Teacher	4000
3	4	Doctor	6000
4	5	Lawyer	7000

2. Left join('all.x=TRUE'):

```
left_join <- merge(df1, df2, by = "ID", all.x=TRUE)
print(left_join)
```

The resulting 'left_join' data frame will include all rows from 'df1' and the matching rows from 'df2'. Non-matching rows from 'df2' will have an 'NA' value

	ID	Name	Age	Occupation	Salary
1	1	A	25	<NA>	NA
2	2	B	30	Engineer	5000
3	3	C	35	Teacher	4000
4	4	D	40	Doctor	6000

Example:

Df1->

	ID	Name	Age
1	1	A	25
2	2	B	30
3	3	C	35
4	4	D	40

df2->

	ID	Occupation	Salary
1	2	Engineer	5000
2	3	Teacher	4000
3	4	Doctor	6000
4	5	Lawyer	7000

3. Right join('all.y=TRUE'):

```
join <- merge(df1, df2, by = "ID", all.y=TRUE)
print(join)
```

The resulting 'right_join' data frame will include all rows from 'df2' and the matching rows from 'df1'. Non-matching rows from 'df1' will have 'NA' values.

	ID	Name	Age	Occupation	Salary
1	2	B	30	Engineer	5000
2	3	C	35	Teacher	4000
3	4	D	40	Doctor	6000
4	5	<NA>	NA	Lawyer	7000

Example:

Df1->

	ID	Name	Age
1	1	A	25
2	2	B	30
3	3	C	35
4	4	D	40

df2->

	ID	Occupation	Salary
1	2	Engineer	5000
2	3	Teacher	4000
3	4	Doctor	6000
4	5	Lawyer	7000

4. Full outer join('all =TRUE')

```
join <- merge(df1, df2, by = "ID", all=TRUE)
print(join)
```

The resulting 'full_join' data frame will include all rows from both 'df1' and 'df2'. Non-matching values will have 'NA' values.

	ID	Name	Age	Occupation	Salary
1	1	A	25	<NA>	NA
2	2	B	30	Engineer	5000
3	3	C	35	Teacher	4000
4	4	D	40	Doctor	6000
5	5	<NA>	NA	Lawyer	7000

5. Cross join(by=NULL)

```
join <- merge(df1, df2, by = NULL)
print(join)
```

A Cross Join also known as cartesian join results in every row of one data frame is being joined to every other row of another data frame.

Df1->

	ID	Name	Age
1	1	A	25
2	2	B	30
3	3	C	35
4	4	D	40

df2->

	ID	Occupation	Salary
1	2	Engineer	5000
2	3	Teacher	4000
3	4	Doctor	6000
4	5	Lawyer	7000

	ID.x	Name	Age	ID.y	Occupation	Salary
1	1	A	25	2	Engineer	5000
2	2	B	30	2	Engineer	5000
3	3	C	35	2	Engineer	5000
4	4	D	40	2	Engineer	5000
5	1	A	25	3	Teacher	4000
6	2	B	30	3	Teacher	4000
7	3	C	35	3	Teacher	4000
8	4	D	40	3	Teacher	4000
9	1	A	25	4	Doctor	6000
10	2	B	30	4	Doctor	6000
11	3	C	35	4	Doctor	6000
12	4	D	40	4	Doctor	6000
13	1	A	25	5	Lawyer	7000
14	2	B	30	5	Lawyer	7000
15	3	C	35	5	Lawyer	7000
16	4	D	40	5	Lawyer	7000

Using ‘dplyr’ Package:

The primary function for merging in ‘dplyr’ is ‘join()’, which supports various types of joins.

Syntax :

```
merged_df<- join(x,y,by="common_column",type="type_of_join")
```

‘x’ and ‘y’ are the data frames to be merged.

‘by’ specifies the common columns on which the merge is to be performed

‘type_of_join’ can be ‘inner’, ‘left’,’ right’ or ‘full’ to specify the type of join.

Example:

Consider two data frames ‘df1’ and ‘df2’

```
df1 <- data.frame(ID = c(1, 2, 3, 4),
                  Name = c("A", "B", "C", "D"),
                  Age = c(25, 30, 35, 40))

df2 <- data.frame(ID = c(2, 3, 4, 5),
                  Occupation = c("Engineer", "Teacher", "Doctor", "Lawyer"),
                  Salary = c(5000, 4000, 6000, 7000))
```

```
  ID Name Age
1  1    A  20
2  2    B  30
3  3    C  40
4  4    D  50
  ID Occupation Salary
1  2   Engineer   2000
2  3    Teacher   4000
3  4     Doctor   6000
4  5     Lawyer   7000
```

Inner join:

```
inner_join <- dplyr::inner_join(df1, df2, by = "ID")
print(inner_join)
```

Output:

```
  ID Name Age Occupation Salary
1  2    B  30   Engineer   2000
2  3    C  40    Teacher   4000
3  4    D  50     Doctor   6000
```

Left join:

```
left_join <- dplyr::left_join(df1, df2, by = "ID")
print(left_join)
```

Output:

```
  ID Name Age Occupation Salary
1  1    A  20       <NA>     NA
2  2    B  30   Engineer   2000
3  3    C  40    Teacher   4000
4  4    D  50     Doctor   6000
```

Right join:

```
right_join <- dplyr::right_join(df1, df2, by = "ID")
print(right_join)
```

Output:

```
 ID Name Age Occupation Salary
1  2    B  30   Engineer   2000
2  3    C  40    Teacher   4000
3  4    D  50     Doctor   6000
4  5 <NA>  NA     Lawyer   7000
```

Full outer join:

```
left_join <- dplyr::full_join(df1, df2, by = "ID")
print(full_join)
```

Output:

```
  ID Name Age Occupation Salary
1  1    A  20       <NA>     NA
2  2    B  30   Engineer   2000
3  3    C  40    Teacher   4000
4  4    D  50     Doctor   6000
5  5 <NA>  NA     Lawyer   7000
```

Example 1:

Create two data frames: patients with columns PatientID, Name, Age, Gender, Disease, AdmissionDate and treatments, in second dataframe columns PatientID, TreatmentType, Doctor, Cost.

- Perform an inner join to find patients who received treatment. Perform a left join to get all patients, including those who haven't received treatment.
- Perform a right join to get all treatments, including those for which patient details are missing.
- Perform a full join to combine all records from both tables. Filter the result to find patients whose treatment cost exceeds $5000. Group by Disease and calculate the total treatment cost per disease.

Example 2:

Create two data frames: Logistics with columns OrderID, CustomerID, ProductID, Quantity, OrderDate and customers, 2nd dataframe with columns CustomerID, Name, Location.

- Perform an inner join to find all orders with customer details. Perform a left join to get all orders, including those placed by unknown customers.
- Perform a right join to get all customers, including those who haven't placed any orders.
- Perform a full join to merge all records. Find the top 3 locations with the highest number of orders. Group by ProductID and calculate the total quantity ordered per product.