```
getwd()

setwd("C:\\Users\\Swapnil bandekar\\Downloads\\Swapnil\\Data Analytics\\My
Work\\R\\Datasets")


save.image("Data Manipulation in R_2.RData")

### Factor ( 6th Data Structure)

V1 <- c("Male","Female","Female","Female","Male","Unknown")

V1

class(V1)

V1_fac <- as.factor(V1)

V1_fac

class(V1_fac)

# Double iverted commas are missing in case of Factor as compared to Vector (for
character values)
# result of factor is not a character ( just appears as a character)

## factor is a way using which we can assign a representative number against a
level
# we have 3 different levels here ( Male , Female , Unknown)
# each level is assigned a number

as.integer(V1_fac)


# We can see the number is assigned to particular variable

# Number allocation is done alphabatically

## We can assign our own numbering as well


V2 <- c("Agree","Strongly Agree","Partially Agree","Disagree")

as.integer(as.factor(V2))



## How to enforce my own labelling to factor levels ( Assignment)
```

```
V2_new <- factor(V2, levels =c("Agree","Strongly Agree","Partially
Agree","Disagree"), labels = c(1,2,3,4) )

V2_new


## Why do we need factors??

# 1. Lot of algorithms don't understand the characters so we need to provide
numeric representative. Hence, factor is important.
# 2. It occupies less space in memory. So, computation is faster and less storage
space is required.

# Whenever reading a table if we parse "stringasFactors=TRUE" then all the
character cols will be read as factors
# Have to mention while reading a table ( in the syntax )

V3 <- c("abc","1","male")

V3_fac <- as.factor(V3)

V3_fac

as.integer(V3_fac)

# Numbers get the first priority

V4 <- c("abc","1","male","Male","Abc","ABc","ABC")

V4_fac <- as.factor(V4)

V4_fac

as.integer(V4_fac)

# Lowercase characters get the highest priority than the Uppercase characters


## How to combine the dataframes

Retail <- read.csv("22 Sep - retail_sales.csv")

View(Retail)

Retail1 <- Retail[5:10,]
Retail2 <- Retail[14:20,]

dim(Retail1)
dim(Retail2)
```

```
# Combining dataframes is known as "Row binding"

Retail12 <- rbind(Retail1,Retail2)
dim(Retail12)
View(Retail12)


# All the dataframes should have same no and names of cols while using rbind


Retail3 <- Retail[5:10,-1]
Retail4 <- Retail[14:20,]

dim(Retail3)
dim(Retail4)

Retail34 <- rbind(Retail3,Retail4)

# if the no of cols of dataframes are not matching then it will give an error


Retail5 <- Retail[5:10,]
Retail6 <- Retail[14:20,]

dim(Retail5)
dim(Retail6)

names(Retail5)[which(names(Retail5)=="City")] <- "Region"

View(Retail5)
View(Retail6)


Retail56 <- rbind(Retail5,Retail6)

# if the names of cols of dataframes are not matching then it will also give an
error


Retail12$Index <- 1:nrow(Retail12)

View(Retail12)

# for indexing the dataframe


## Column binding of dataframe

Retail7 <- Retail[5:10,c(1,2,5)]
```

```
Retail8 <- Retail[5:10,c(6,9)]

dim(Retail7)
dim(Retail8)

Retail78 <- cbind(Retail7,Retail8)
dim(Retail78)
View(Retail78)

# All the dataframes of cbind should have same number and names of rows

### I have 30 files of month and I want to combine all the files and also want
particular date against each file ( Similar to extrating filename) ((Assignment))


## combining dataframes having unequal no of rows and columns
# We can use gtools package

library(gtools)


dim(Retail3)
dim(Retail4)

View(Retail3)
View(Retail4)

Retail34 <- smartbind(Retail3,Retail4)

View(Retail34)

Retail56 <- smartbind(Retail5,Retail6)
View(Retail56)

# Smartbind can work even if col names are different


## Dataframe Merging

Revenue <- read.csv("Revenue.csv")
Cost <- read.csv("Cost.csv")

View(Revenue)
View(Cost)

Total <- merge(Revenue,Cost,by =c("Item_Category","Month"))
View(Total)

# Merging the column with the reference of Item_Category and Month
```

```
Total1 <- merge(Revenue,Cost,by =c("Item_Category"))
View(Total1)

# cross product or cartesian

Total2 <- merge(Revenue,Cost,c("Item_Category","Month"))
View(Total2)

# "by=" is optional argument


A <- data.frame(letter=LETTERS[8:12],a=1:5)
B <- data.frame(letter=LETTERS[sample(10)],b=runif(10))

View(A)
View(B)

C <- merge(A,B)

View(C)

# Inner Join
# finding out common values in dataframes

D <- merge(A,B,all.x = TRUE)

View(D)

# Left Join
# all.x = TRUE : I want all the records of left table

E <- merge(A,B,all.y = TRUE)

View(E)

# Right Join
# all.y = TRUE : I want all the records of right table

F1 <- merge(A,B,all.x = FALSE)

View(F1)

# all.x=FALSE and all.y=FALSE don't have practical use while merging

# "F" is a keyword in R


G <- merge(A,B,all.x = TRUE,all.y = TRUE)
```

```
View(G)

# Full Outer Join

# We can't join 3 tables with 1 piece of code as it will give the result which will
be difficult to interpret

# It is advisable to join 2 tables first and then join the 3rd table with o/p of
first two

# If we have more than 1 common values


H <- merge(A,B,by="letter")

View(H)


# In Table A , col name is "Employee_ID"
# In Table B , col name is "Emp_ID"

I <- merge(A,B,by.x = "Employee_ID",by.y = "Emp_ID")


## How to create a dataframe which contains only numeric/character cols?? (
Assignment)



## Apply family of a function


View(Retail)

Retail_New <- Retail[,5:9]

View(Retail_New)


## What is mean of each row/col ( run a function )

# Syntax : apply(dataframe name , row/col(1or2) , function )

# row/col : have to specify whether function to be applied on row/col

# 1 stands for row and 2 stands for col

apply(Retail_New,1,mean) # gives mean of each row

apply(Retail_New,2,mean) # gives mean of each col
```

```
# functions such as "min , max , median , sd , var " can also be used
# can create custom functions as well
# size of data does not matter

Retail_New$Mean_of_each_row <- apply(Retail_New,1,mean)

View(Retail_New)



# Apply family have wide range of functions

# apply , tapply , lapply , mapply , vapply , idply , ddply

## r-bloggers.com/r-tutorial-on-the-apply-family-of-functions/

tapply(Retail$Revenue, list(Retail$Item_Category), mean)

# mean of revenue col per item category

## blank data can be treated as well so that it won't affect the result. We have to
pass additional argument "na.rm=TRUE"

tapply(Retail$Revenue,list(Retail$Item_Category,Retail$Month), max)

View(tapply(Retail$Revenue,list(Retail$Item_Category,Retail$Month), max))

class(tapply(Retail$Revenue,list(Retail$Item_Category,Retail$Month), max))


## How to find out number of records for each category

table(Retail$Item_Category)

table(Retail$Item_Category,Retail$Month) # Cross Tabulation

class(table(Retail$Item_Category,Retail$Month))

table(Retail$Item_Category,Retail$Month,Retail$Supplier) # Cross Tabulation for
more than 1 category

class(table(Retail$Item_Category,Retail$Month,Retail$Supplier))



library(openxlsx)

write.xlsx(Retail,"Retail.xlsx")
```

```
# Extracting results in Excel file (.xlsx)


## Transpose the data ( Reshaping)

Retail_T <- t(Retail)

View(Retail_T)

# "t" function is used to Transpose the data

## Long form to wide form conversion ?? (Assignment)

## wide form to long form conversion ?? (Assignment)

dat <- data.frame(
  name = rep(c("firstName", "secondName"), each=4),
  numbers = rep(1:4, 2),
  value = rnorm(8)
)

dat


wide <- reshape(dat, idvar = "name", timevar = "numbers", direction = "wide")
wide

long <- reshape(wide, direction = "long" )
long



library(dplyr)

mydata <- mtcars

View(mtcars)
View(mydata)

## I want to remove the duplicate rows from a dataframe

mtcars_1 <- distinct(mtcars)

View(mtcars_1)

dim(mtcars)
dim(mtcars_1)

# "distinct" command is used to remove the duplicate rows
```

```
## I want to rename cols

Mydata1 <- rename(mydata,displacment=disp,cylinder=cyl)

head(Mydata1)

## Subset of dataframe

Mydata2 <- filter(mydata,cyl==6)

View(Mydata2)

mydata%>%
  filter(cyl==6)

# Pipe Operator(%>%)

## Mutate Function
# It is used to create a new col in the dataset

Mydata3 <- mutate(mydata,mpg_cyl_ratio=mpg/cyl)

View(Mydata3)


mydata%>%
  mutate(mpg_cyl_ratio=mpg/cyl)


Mydata4 <- mutate_all(mtcars,funs("percent"=./100))

# This is wrong statement

Mydata4 <- mutate_all(mtcars[,-1],funs("percent"=./100))

View(Mydata4)

# This is right statement
# -1 : exclude 1st col as it is having character values

## check detailed documentation of dplyr package



#### Missing Values ( Imp Topic)

# Representated as "NA" in R ( NaN in Python)


View(airquality) # missing values in the dataset
```

```
mean(airquality$Temp)

mean(airquality$Ozone) # gives "NA" as it is having missing value

## To treat cols with missing value

mean(airquality$Ozone,na.rm = TRUE)

mean(na.omit(airquality$Ozone))

# We can parse "na.rm=TRUE" statement or can use "na.omit" command

## How to spot missing values in the data

is.na(airquality$Ozone) # not the best solution

## Tell me the records where I have missing value

which(is.na(airquality$Ozone))

# gives the indexes of missing values

## Tell me the count of data which is missing

length(which(is.na(airquality$Ozone)))

##  What % of data is missing in ozone col

length(which(is.na(airquality$Ozone)))/nrow(airquality)*100 # 24% of data missing

## How to find out count of missing values in each cols

colSums(is.na(airquality))

colSums(is.na(airquality))/nrow(airquality)*100

## Try the same with the help of "sapply" function


### How to do Treatment of missing values

# 1. Impute or replace missing values
# 2. Ignore or drop the missing value
# 3. Drop the col having the missing values
```

```
## 3. Drop the col having the missing values

# Criteria to consider

# If it is not a business imp col
# having 80% of missing values

## Two considerations for missing value treatment

# 1. For model building , I should have sizeable amount of data
# 2. Missing value treatment is an expensive exercise and doesn't give me the exact
results

# Ex1 : 5 million records and 20% of the data is missing (Ignore)
# Ex2 : 5k records and 20% of data is missing (Impute)

# Default mindset is to ignore the missing values but If I have shortage of data
then I have to do Imputation on original data


## I want to create the new dataframe without any missing values

airquality_without_missing_rec <- na.omit(airquality)

nrow(airquality)

nrow(airquality_without_missing_rec)

dim(airquality)
dim(airquality_without_missing_rec)

colSums(is.na(airquality_without_missing_rec)) # no missing records

## How to impute
## I want to impute the missing values with average of that col

airquality$Ozone[5]


airquality$Ozone[5] <- mean(airquality$Ozone,na.rm = TRUE)

airquality$Ozone[5]

airquality$Ozone[which(is.na(airquality$Ozone))] <-
mean(airquality$Ozone,na.rm=TRUE)

View(airquality)

colSums(is.na(airquality))
```

```
## MICE package ( cool ways to treat missing values)

## na.roughfix ( randomforest package)

## UCI Machine Learning Repository ( Open Source Databases)




## Transposing data using reshape2() package

library(reshape2)

wide1 <- data.frame(persons = c('ram','shyam','govind'), age = c(28,30,32), weight
= c(60,65,70))
wide1



# converting the data into the long form from wide form
# will use melt command

long1 <- melt(wide1, id.vars = 'persons', value.name = 'Value1')



# converting the data into wide form from long form
# will use dcast command


dcast(long1, persons~variable, value.var = 'Value1')




### SQL Queries within R using sqldf()

install.packages('sqldf')

library(sqldf)

# Using Select statement : Selecting the data

ret1 <- sqldf('select Month, Cost from Retail' )
head(ret1)
```

```
# Using where statement : to filter the data

ret2 <- sqldf('select Month, Cost from Retail where Cost>10000')
ret2

ret3 <- sqldf('select * from Retail where Cost >10000')
ret3

# * is used to select all the columns


# Using order by statement :

ret4 <- sqldf('select Month,Cost from Retail order by Cost')
ret4


ret5 <- sqldf('select * from Retail order by Cost')
ret5

ret5_1 <- sqldf('select * from Retail order by Cost desc')
ret5_1


# order by will sort the data according to given column
# ordering will ascending(asc) by default, for descending sort we can use 'desc'




##### Case Study



hd <- read.csv('case_study_heart_disease_data_set.csv')

str(hd)

hd$Sex <- as.factor(hd$Sex)
hd$cp <- as.factor(hd$cp)
hd$fbs <- as.factor(hd$fbs)
hd$restecg <- as.factor(hd$restecg)
hd$exang <- as.factor(hd$exang)
hd$slope <- as.factor(hd$slope)
hd$DV <- as.factor(hd$DV)

str(hd)


# Subsetting the data
```

```
dat1 <- hd[hd$thal==3,]
head(dat1)
dim(dat1)


dat2 <- hd[hd$thal==3 & hd$Sex==1, c('Sex','thal')]
head(dat2)
dim(dat2)



# sorting the data


dat3 <- hd[order(-hd$thalach),]
head(dat3)



# Tabulations


table(hd$thal)


table(cp = hd$cp, sex = hd$Sex, dv = hd$DV)

table(hd[,c('cp', 'Sex', 'DV')])



# Finding Aggregates


aggregate(hd$trestbps~hd$Sex, data = hd, mean)

aggregate(hd$trestbps~hd$Sex+hd$exang, data = hd, mean)

aggregate(hd$trestbps , by=list(hd$Sex,hd$exang), mean)


# Groupby using sqldf() package


sqldf('select thal, cp, count(DV) from hd group by thal,cp')

sqldf('select thal,cp,avg(trestbps) from hd group by thal,cp')
```

```
hd %>% group_by(thal,cp) %>% summarise(mean(trestbps))

# filter , groupby and summarise data using dplyr() package

hd %>% filter(thal==3 & cp ==4) %>% group_by(DV) %>% summarise(n())
```