Data frames :-

A Data frame is like a matrix with a two-dimensional rows & columns structure. It differes from a matrix in that each coloumn may have a different mode.

A data frame is a list with component of that list being equal length vectors. R does allows the component to be other type of objects including other data frames. In this all components of data frame are vectors.

Creating data frames :-

```
Eg:- > kids ← c("x", "y")
     > ages ← c(11, 12)
     > d ← data.frame(kids, ages, stringAsFactor = FALSE)
     > d
```

```
o/p:  kids  age
       x     11
       y     12
```

The first two arguments in the call to data.frame() is clear The third argument string asfactor = FALSE requires. because if the named argument sting asfactor is not specified then by deau: deafult string asfactor will be true. this means that if we create a data frame from a character vector kids will convert that vector to a factor. that's why w-here we will said string as factors to FALSE.

Accessing data frames:

We can Access it such as via component index values or component names

```
Eg:- > d[[1]]
     o/p: "x" "y"   [1] x y
     > d$kids
     o/p: "x" "y"   [1] x y
     > d[,1]
     o/p: "x" "y"   [1] x y
     > str(d)
data.frame 2 obs. of 2 variables
 $ kids: string "x" "y"
 $ age : number 11  12
```

read.table

"Exam1"	"Exam2"	"Que"
2.0	3.3	4.0
3.3	2.0	3.7
4.0	4.0	4.0
2.3	0.0	3.3
2.3	1.0	3.3
3.3	3.7	4.0

> examplequiz ← read.table ("Exam", header = TRUE)

> head (examplequiz)

including headings like exam

O/p:

"Exam1"	"Exam2"
2.0	3.3
3.3	2.0
4.0	4.0
2.3	0.0
2.3	1.0
3.3	3.7

Other matrix like operations:

Various matrix operations also apply two data frames we can do filtering to extract various sub data frames

Extracting sub data frames:

A data frame can be viewed in rows and columns terms we can extract sub data frame by rows, cols.

> examsquiz [2:5,] all cols

O/p:

3.3	2.0	3.7
4.0	4.0	4.0
2.3	0.0	3.3
2.3	1.0	3.3

> Exams quiz [2:5, 2] (col 2)

O/p:

3.3	2.0
4.0	4.0
2.3	0.0
2.3	1.0

> class (examsquiz [2.5, 2]) O/p: numeric

> examsquiz [2:5, 2, drop = FALSE] print in matrix formate

if not it takes vector formate.

1	2.0
2	4.0
3	0.0
4	1.0

> class (examsquiz [2:5, 2, drop = FALSE]) O/p: data.frame

More on Treatment of NA values:-

Suppose the 2nd exam score for the first student had been missing 2.0 NA 4.0
In any subsequent statistical analysis R would do its best with the missing data na.rm=TRUE explicitly telling R to ignore NA values.

Eg: 2.0 NA 4.0

```
> x <- c(2, NA, 4)
> mean(x)
[1] NA
> mean(x, na.rm=TRUE)
[1] 3
> examsquiz[examsquiz$Exam.1 >= 3.8,]
```

o/p: 4.0

searched in matrix form using rows, cols

```
> subset(examsquiz, Exam.1 >= 3.8]
```

o/p: 4.0

function is different

The columns names are taken in the content of the given data frame the data frame of any observation that has atleast one NA value a function for this complete.cases()

```
> d4
```

kids	states
Jack	T
<NA>	AP
Jhon	MP
Suman	<NA>

If we take empty row then it will print NULL

```
> complete.cases(d4)
[1] TRUE FALSE TRUE FALSE
```

Using the rbind() and cbind()
In this using rbind() to add a row

```
> d4
```

kids	ages
Jack	10
Jhon	12

```
> rbind(d4, list("Lavanya", 19))
```

It will print exam1, exam2, Qwiz,

o/p: 1.3
-1.3
0
-2.3
-1.3
0.4

```
> Eq <- cbind(examsquiz, examsquiz$Exam.2 - examsquiz$Exam.1)
> Eq
```

> examsquiz $ examsdiff ← (examsquiz $ exam.2 - examsquiz $exam.1)

> head ("examsquiz")

o/p:	"Examsquiz"	"Exam.1"	Exam.2	"Quiz"
	1.3	2.0	3.3	4.0
	-1.3	3.3	2.0	3.7
	0	4.0	4.0	4.0
	-2.3	2.3	0.0	3.3
	-1.3	2.3	4.0	3.3
	0.4	2.3	3.7	4.0

27/04/2022

> d

kids	ages
Jhon	12
Jack	10

> d $one ← 1

o/p:	kids	ages	one
	Jhon	12	1
	Jack	10	1

Applying the apply() function:

> apply (examsquiz, 1, max) up to 6 it gives vector formate

o/p: 4.0 4.0 3.7 4.0 3.3 3.3 4.0 more than 6 it gives matrix format

~~4.0~~
~~3.7~~
~~4.0~~
~~3.3~~
~~3.3~~
~~4.0~~

Merging data frames:

In R two data frames can be similar Combine using the merge()

merge (x,y) this merge data frames. x&y it assumes that two data frames have one or more columns with names in ~~column~~ common.

> d1

kids	states
Jack	CA
John	MA
Suman	MA
Lavanya	HI

> d2

ages	kids
10	John
7	Smith
12	Jack

> d ← merge (d1, d2)

> d

o/p:

kids	states	ages
Jhon	MA	10
Jack	CA	12

it omits only similar Output

Checks variables

The two data frames have the variables kids in common & form founds the rows in which this variable had the same value of kids in both data frames.

The merge() has named arguments by.x and by.y which handles cases in which variables have similar information but different names in two data frames

```
> d3
  ages  pals
1   12  Jack
2   10  John
3    7  Stef

> merge (d1, d3, by.x = "kids", by.y = "pals")
```

o/p:

	kids	states	ages
	Jhon	MA	10
	Jack	CA	12

Even though our variables was called kids in one data frame and pals in another data frame it was mean to store the same information and does the merge made sence

Duplicate matches will appear in the full in the result. Possiblibly in undesirable ways

```
> d2a ← rbind (d2, list ("15", "Jhon"))
> d2a
```

o/p:

	kids	states	ages
	Jhon		10
	Jack		12
	Stef		7
	Jhon		15

```
> merge (d1, d2a)
```

o/p:

	kids	state	ages
	Jack	CA	12
	Jhon	MA	10
	Jhon	MA	15

There are two Jhons in d2a there is a Jhon in d2 who lives leaves in MA and another Jhon with unknowing residency in the previous merge (d1, d2a) there was only one Jhon in both data frame. but here in the call merge (d1, d2a). it may have been the case that only one of the John was a MA it is clear from these eg that we must use matching variable with great care

28/04/2022

Applying functions to data frames

The lapply and sapply functions with data frames using lapply and sapply on data frames with a specified function R-D the f() will be called on each of the frame column with the return value place in a list

```
>d
kids ages
Jack  12
John  10
>d1<-lapply(d,sort)
>d1
kids ages
John  10
Jack  12
```

```
as.data frame(d1)
John  10
Jack  12
Prints in data frame
format (matrix)
```

$kids o/p: Jack John

$ages o/p: 10 12

Characteristics of data frames:

- The column names should be non-empty.
- The row names should be unique.
- The data frame can hold the data which can be numeric, character or of factor type.
- Each column should contain the number of positive data items.

Factors and tables:

- Factors forms the bases for R's powerfull operations including many of those performed on tabular data.
- Factors comes from notation of nominal, chatagrical, variables in statistics.

These values are non-numerical in nature corresponding to catagerious such as republican, demorgan, unapplicatricated all thu they may be coded using numbers.

Factors and levels

29/04/2022

an R factor might be viewed simply as a vector with a bit more information added. the extra information consist of a record of the distint values in that vector called levels

eg: `>x<-c(5,12,13,12)`

`>xf<-factor(x)`

`>xf` o/p: 5 12 13 12

[1]

Levels 5 12 13

> str(xf) → o/p: "5" "12" "13" 1 2 3 2

> unclass (xf) → 1 2 3 2

> attr("levels"). o/p: "5" "12" "13"

> length (x) o/p: 4

> length (xf) o/p: 3

> level (xf) o/p: 3 "5" "12" "13"

The core of xf here is not (5, 12, 13, 12) but rather 1 2 3 2. the letter means that our dat consist first of a level 1 value and level 2 and level 3 values and finally another level 2 value. So, the data has been recorded by level. the levels themselves are recorded to the length of the factor is still defined in terms of the length of the data rather than being a count of the no. of the levels.

> x ← c(5, 12, 13, 12)

> xff ← factor (x, levels = c(5, 12, 13, 2))

> xff

o/p: 5 12 13 2

5 12 13 02

> xff [2] ← 88

> xff o/p: 5 88 13 2

→ x ← c(1, 2, 3, 4, 1) xff ← factor(x)

> length (x) factor (levels(x)).

o/p: 5

factor (length (x)) o/p: 5

level (x) o/p: 4

factor (x) o/p: [1] 1 2 3 4

levels : 1 2 3 4

Print (nlevels (factor (x))) = o/p: 4

> array (x) o/p: 1 2 3 4 1

> length (array (x)) o/p: 5

> xff ← factor (x, levels = c(1, 2, 3, 4, 1))

> xff o/p: [1] 1 2 3 4 1

levels 1 2 3 4

```
> x1 <- c(5,12,13,12)
> xff <- factor(x1, levels = c(5,12,13,88))
> xff
o/p: 5 12 13 12
  Levels 5 12 13 88
> xff[2] <- 88
> xff
o/p: 5 88 12 13
  levels 5 12 13 88
```

xff didnot contain the value 88 but in defining it we allowed for that feture possibility. we did add the value by the same token. If we can add in an illigal level it displays like

```
> xff[2] <- 88
  waring message:
  In [<-, factor(x, step = 1:2, value = 28):
  Invalid factor level, NAs Generated
```

Common functions used with factors.

Here we have know about the member of the family of apply() i.e tapply as well as two other functions commonly used with the factors split(), by()

Tapply():

```
> ages <- c(25,26,55,37,21,42)
> affils <- c("R","D","D","R","V","D")
> tapply(ages, affils, mean)
```

R = (25+37)/2 = 31 D = (26+55+42)/3 = 41 V = 21

o/p: 31 41 21

We have a vector x6 of ages of voters and a factor of showing some non numeric values such as party affiliations in x. we might wish to find the mean ages in x with in each of the party group.

To print a table :-

```
> d <- data.frame(list(gender = c("M","M","F","M",
                                  "F","F"),
  ages <- c(47,59,21,32,33,24), income = c(55K, 88K, 32K,
                                  52K, 34K, 24K))
```

```
>d
  gender  ages  income
1      M    47    55k
2      M    59    88k
3      F    21    32k
4      M    32    52k
5      F    33    34k
6      F    24    24k
```

> d $ over 25 ← ifelse (d$age > 25, 1, 0)

```
>d
  gender  ages  income  over25
       M    47    55k       1
       M    59    88k       1
       f    21    32k       0
       M    32    52k       1
       F    33    34k       1
       f    24    24k       0
```

> tapply (d$income, list (d$gender, $over25), mean)

```
     0        1
F  2800.00  34000.00
M    NA     65000.00
```

30/04/2022

Split()

Split is a vector which splits the vector into group and then applies a specified function on each group.

Split function stops at a that first stage stage just forming the groups

S(x,f) here x and f playing roles similar to those in the call tapply(x,f,g) i.e x being a vector or data frame and f being a factor or a list of factors. The action is to split x into groups which are returned in a list

eg:- >d

```
  gender  ages  income  Over25
       M    47    55k       1
       M    59    88K       1
       F    21    32K       0
       M    32    52K       1
       F    33    34K       1
       F    24    24K       0
```

```
> split(d$income, list(d$gender, d$over25))
```

o/p: F 0
38K 24K
M 0
NA •
F 1
34K
M 1
55K 88K 52K

```
> g ← c("M", "M", "F", "F")
> split(1:4, g)
$F
 3
$M
 1 2 4
```

The results shows the female cases are in records 3 the Male cases are in records 1, 2, 4

there w
text() /txt()

Here we create the txt file the file contains the word = world 5 times and climate was their 10 times Then the world & Climate will have the 2 levels of txt. It has 2 levels of th text
here we call the split() then determine where these and the other word apper in text.

```
> findwords ← function(tf)
{
 txt ← scan(tf, "")
  words ← split(1:length(txt), txt)
  return(words)
}
```

o/p:

world					
w	6	c	10	c	
o	6	$	:	l	
r	6	g	10	i	
l	16	m	10	m	
d	6	a	10	a	
		t	10	t	
		e	10		

by()

02/05/2022

by() function works like tapply() function but it is applied to objects rather than vectors.

> abc ← read.table ("example.csv", header = TRUE)

> by (aba, aba$gender, function (m) lm(m[,2] ~ m[,3]))

aba$gender : F

Coefficient

	F	M
ages	26	46.
Incomes	30000	65000

$(\beta_0 + \beta_1)$ Exams

[26 + 30000] 6 = 180156

[46 + 65000] 6 = 390276

By function very similar to tapply() with the first argument specifing our data. the 2nd argument is grouping factor and the 3rd the function to be applied to each group

working with tables

Exploring R tables :-

Example :- f1 ← list (c(5, 12, 13, 12, 13, 5, 13)

c("a", "bc", "a", "a", "bc", "a", "a"))

> tapply (f1, length)

	a	bc
5	2	NA
12	1	1
13	2	1

Here tapply() function breaks into sub vectors and then applies the length() function to each sub vector.

this sub vector lengths are the count of the occurances of each of the 3x2 = 6 combinations of the two factors.

for instance 5 occurs twice with "a" and not at all with "bc" hence the entry is 2 and NA in the first row of the output.

In statistics this is called as contingency table

- In the above example there is one problem that is NA value it really should be zero. meaning that in no cases with the did the factor have level 5 and 2 the 2nd have level bc
- But the table function creates contingency tables correctly,

> table (f1)

o/p :-

	a	bc
5	2	1
12	1	1
13	2	1

ct.csv

"Vote for x"	"Voted for x Last time"
"Yes"	"Yes"
"Yes"	"NO"
"NO"	"NO"
"Not sure"	"Yes"
"NO"	"NO"

In the usual statical analysis each row in these file represents one subject under study in this case, we have Asked 5 people the following 2 Questions.

05/05/2022

1. Do you plan to vote for candidate x,
2. Did you vote for x in the last election

```
en:> ct <-read table ("ct.csv", header=T)
>ct
```

Vote for x	Voted for x last time
"Yes"	"Yes"
"Yes"	"No"
"No"	"No"
"Not sure"	"Yes"
"No"	"No"

```
> cttab <-table(ct)
> cttab
```

Vote for x	voted for x last time	
	NO	Yes
NO	2	NA
Yes	1	1
Not sure	0	1

The 2 in the upper left corner of the table shows that we had, for example, two people who said No to the first and second question, The 1 in the middle right indicates that 1 person not sure to frost question and Yes to the and Question we can get 1D counts, which are counts on a single factor >table (c(5,12,13,12,8,5))

5	12	13	8
2	2	1	1

An example of 3D table involving voters genders. race (white, black, Asian and others) and political views (liberal or conservative)

ex:- > vt dataframe

gender	race	pol
M	W	L
M	W	L
F	A	C
M	O	L
F	B	L
F	B	C

>vt ← table (v)

>vt

```
, , pol = L

      race
gender A B W O
     F 0 0 2 0
     M 0 1 0 0

, , pol = C

      race
gender A B W O
     F 0 0 2 0
     M 1 0 0 0
```

R programing points out a three dimensional table as a series of two dimensional table. In these case it generates a table of gender and race for conservations and then a corresponding table for liberals.

For example the second 2D table says that there were two white male liberals.

Matrix or array:-

Matrix or Array operations can be on dataframe, they can applied on table too. for ex:- we can access the table cell counts using matrix notation Apply these to our voting example from the previous

Ex:- > class (cttab) o/p: "Table"

> cttab [1,1] o/p: 2

> cttab [1,] o/p: 2 0

In the second command, even though the first command had shown that cttab had class has class (cttab) we treated it as a matrix and printed as [1,1] element Continuing the third command printed the first column of these matrix.

> cttab /s

Vote for x	voted for x : NO	last time Yes
Yes	0.2	0.2
NO	0.4	0
Notsure	0	0.2

we can multiple the matrix by a scalar

ex:- apply (cttab, 1, sum)

Yes	Notsure	NO
2	1	2

> add margins (cttabs)

vote for x	voted for x	last time	
NO	NO	Yes	Sum
Yes	2	0	2
Not-Sure	1	1	2
Sum	0	1	1
	3	2	5

The marginal value of a variable are those obtained when this variable is held constant while others are summed. In the voting example the marginal value of vote for x variable are 2+0=2, 0+1=1 & 1+1=2. Here we can obtained thes via the matrix apply() function But R's applys a function add margins () from these to find marginal totals. It shows in the above example.

> dimnames (cttab)

vote for x

"No" "Yes" "Notsure"

voted for x last time

"No" "Yes"

Extracting a sub table :-

continuing with voting example

vote for x	voted for x	last time
No	NO	Yes
Yes	2	0
Notsure	1	1
	0	1

Suppose, we wish to present these data at a meeting Concentrating on those despone who know they will vote for x in the current elections. i.e we wish to eliminate

the not sure entries and present a sub table.

> subtable (cttab, vote for x = c("No", "yes", voted for x last time = c("No", "yes"))

o/p: vote for x voted for x last time

Yes	yes	No
No	1/2	0

The function subtable performs subtable extraction & has two arguments.

1. tbl → the table of interest of class "table"
2. subnames - A list specifying the desired subtable extraction, each component of list is named after some dimensions of tbl, and value of that component is a vector of the names of the desired levels.

From the above example, the argument cttab will be a 2D table with dimension names vote for x & voted for x last time within these 2D the level names are No, Notsure, Yes in the 1st Dimension and No, Yes in the second dimension

we wish to exclude the not sure cases so our Actual argument corresponding to the formal argument subnames

```
Ex:- >subtable ← function (tbl, subnames)
       {
         tblarray ← unclass (tbl)
         dcarg) ← list (tblarray)
         ndims ← length (subnames)
         for (i in 1:ndims) {
       dcargs [[i+1]] ← subnames [[i]]
       }
       sub array ← do.call ("[", dargs)
       dims ← lapply (subnames, length)
     Subtbl ← array (sub array, dims, dimnames
                        = subnames)
       Class (Subtbl) ← "table"
       return (Subtbl)
     }
```

The function table we found that at its core on object of class table consist of an array whose elements are the cell counts so the strategy is to extract the desired subarray then add names to the dimensions of the subarray and then class status to the result.

For the code here, then the first task is to form the sub array corresponding to the users desired sub table, and this constitutes most of the code. In line three we first extract the full cell counts array, sorting it in the tbl array

Finding the largest cells in a table:

Tabdom(): Tabdom() function reports the dominant frequencies in a table.

Syntax: tabdom(tbl, k)

This reports the cells in the table tbl that have the k largest frequencies

```
> d ← c(5,12,13,14,3,28,12,12,9,5,5,13,5,4,12)
> dtab ← table(d) → o/p:-
```

3	4	5	9	12	13	28
1	2	4	1	4	2	1

```
> tabdom(dtab, 3)
```

o/p:

d	freq
3	5	4
5	12	4
2	4	2

o/p:-

5	12	13	4	3	28	9
4	4	2	2	1	1	1

largest frequency

```
> as.dataframe(dtab)
```

```
> tabdom(dtab, 2)
```

	Vote for n	Voted for X 1 time	frequency
1	NO	NO	2
2	Not sure	NO	0
3	Yes	No	1
4	NO	Yes	0
5	Not sure	Yes	1
6	Yes	Yes	1

o/p:- Vote for X

NO
Yes

Voted for X 1 time

NO
NO

freq

2
1

The function tells us that the values 5 & 12 were the most frequent in d with 4 instances each. and the next most frequent value was 4 with 2 instances. Consider our table ctlab in the example we use, tabdom (above example)

So the combination "No, No" was most frequent with 2 instance which the 2nd most frequent being. yes-No which is 1 instance

This not the original data frame ct from which the table ctlab was constructed. It is simply a different presentation of the table itself. They adds 1 row for each combination of the tad with a frequency column. Added to show the no. of instances of each combination

```
tabdom ← function (tbl, k)
{
tbldf ← as.data frame (tbl)
freqord ← order (tbldf $freq, decreasing = TRUE)
dom ← tbl df [freqord, ] [1:k, ]
return (dom)
}
```

09/05/2022

- The first line finds the cells in table tbl with the k highest frequencies.
- second line indicates the enters into the function
- 3rd line indicates the create a data frame representation of tbl, adding a frequency column
- 4th line determine the proper positions of the frequencies in a sorted order
- 5th line is a rearranging the data frame in that order and that takes the 1st k rows.
- 6th line returns the result stored in dom
- 7th line close the working functions.

Other factors and table related functions:-

R includes a no. of other functions that are handy for working with tables and factors. Here we discuss two of them - 1. is aggregate()

2. is cut()

aggregate() :-

The aggregate() function calls tapply once for each variable in a group

- aggregate() is used to get the summary statistics of the data by group. The statistics includes sum, mean, MAX, MIN and etc.,

Syntax :- aggregate (data-frame $ aggregate_col, list-(data-frame $ group_col), FUN)

Where data-frame → is the input data frame

aggregate_col → is the column to be aggregated in the data frame

group_col → is the column to be grouped with FUN

FUN- Represents sum or mean or MAX or MIN etc.

Eg:- data ← data.frame (subjects = c("R", "DAA", "R", "SE", "DAA", "SE"

id = c(1,2,3,4,5,6)

names = c("A", "B", "C", "D", "E", "F")

marks = c(89, 89, 76, 89, 90, 67))

Print (data)

o/p:-

	subject	id	names	marks
1	R	1	A	89
2	DAA	2	B	89
3	R	3	C	76
4	SE	4	D	89
5	DAA	5	E	90
6	SE	6	F	67

> Print (aggregate (data $ marks, list (data $ subjects), FUN = sum))

SE
- 89+67 = 156

DAA
- 89+90 = 179

R
- 89+76 = 155

cut():-

The cut() function enables us to divide the numeric vector into a range of certain intervals in a customised fassion. with cut() the values get divided into a n, intervals from the n data values depending upon the breaking interval criteria.

Syntax:- Cut (x, b, labels = FALSE)

Where x - is a data i.e the numeric vector which has to be transformed

b - indicates bins are defined to be the semi open intervals. # b[1], b[2], b[3] (indexes)

labels - they are logical labels attached to the result. if labels = FALSE integer code will come to use.

```
> z ← c(0.8, 0.2, 0.5, 0.4, 0.4, 0.2, 0.0, 0.8)
> seq (from = 0.0, to = 1.0, by = 0.1)
```

o/p 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0

```
> binmarks ← seq (from = 0.0, to = 1.0, by = 0.1)
> cut (z, binmarks, labels = false)
```

o/p: [1] 9 3 6 5 5 3 1 9

The above example says that z[1] is 0.8 fell into bin 9 which was 0, 0, 0.1. z[2] is 0.2 fell into bin 3 and so on.