→ imbalance vectors :- 2 vectors of imbalance length can be added.
→ the shorter one repeats itself and adds it to the larger one until the larger one is completed.
2nd R

→ [a ← val] - vector with one element

- c (values) = vector (which is 1-D) - grp of elements
- only one type of data can be their in a vector.
  If numeric and character type are in vector then vector will convert the numeric one to character and the Data Type of whole vector would be character.

- In R, indexing starts from 1 [NULL - changes the org] [after = 2] - 2nd index
  → comma lagana jaroori. → (container)
- List → Collection of data of Different DT - Just like Python - list (values).
  It is a 2-D Data structure.
  ★ [• c(L1, L2) - Merge 2 list]
  ★ [append (L1, L2)]

→ we can provide alias to the values of a list

var ← list (name = "Gaurav", age = 18) | var ['name'] after = index
(name = alias)

☆ [container] → Stores element as container | can also use [names()]

To remove element from list → Assign it as NULL
lis [2] ← NULL

$ - Access

- We can have list and it can have a vector as one of its member → NESTING.

Providing alias in vector → [names()]

Vec ← c(1, 2, 3, 4)
→ Providing alias to elements of vector.

- names (vec) ← c ('one', 'two', 'three', 'four')
  . vec ['two'] is equal as calling vec [2]

• list [index] ← new value
↳ manipulating list Elements.

8

unlist() → converting list to a vector

[concept of Recycling] ↳

Sort(vec) – Sort(vec, decreasing = TRUE).

length(vec)

- lowest/min and Highest/max value of a vector

→ range(vec)

- Generating sequence of Numbers.

→ seq(1, 100, by = 10)

↳ skiping steps

- mean – sd – var (variance)

↳ std. deviation (√var)

mean(vec)
sd(vec)
var(vec)

"ele" %in% list.

[in = Python
%in% ≠ R]

%in% = R

- To remove a element from vector.

INDEX

→ Just call negative of the Index we want to Remove.

→ if I have to remove vec[1] element, then Just call vec[-1] [It will remove that element vec[1]].

- Just call negative of the Index we want to Remove.

* aliasing names to col and Rows

param → dimnames = list(rname, cname);
rname = c("","")
cname = c("","")

True → Repeat data Row wise

[Data, nrow, ncol, byrow]

Matrix → 2D [vector, nrow = , ncol =] parameters.
Data for matrix

mat → matrix(c(1,2,3,4), nrow=3, ncol=3, byrow=TRUE).

Array → Multi-Dimensional Data structure.

→ [Data, dim] [dim as vector] [dim ← c(a,b,c)]
30 or more.

arr → array(c(1,2,3,4), dim ← c(3,3,4))

• Indexing is same as in python → |os[r, c]| |os[r1:r2, c1:c2]|

Factors → Categorize the Data and Store it as levels.
- They can store Both Strings and integers
- Useful in columns which have a limited number of unique values.

→ To convert use | factor(Data structure) |

- Categorize + Store as levels → identifies unique values and store it as levels.

example → data ← c("East", "west", "East", "North", "North", "East", "west", "west", "East", "North").

data-fac ← factor(data)
→ cat the Data - stores unique values as levels.

- data-fac → East , North, West

- Summary(data-fac) → East , North, West
  4 3 4 → Categorize the Data*

For Dataframes [To create dataframe → data.frame ( )]

1) create features using data structures (vec, list etc.)
2) create Dataframe → data.frame (feat1, feat2)
   ⇕ o
   data.frame (feat1, feat2, stringAsFactors = TRUE)

Imp

- Generally / usually / in prev version when we create a col / feature of text or categorical col in dataframe → factors of it are automatically created BUT

⇔ Now we have to specify it by using stringAsFactors.

Refering Columns → data-frame ['col name'] | data-frame $ col name

---

f - strings in R → library (glue)

glue ("Addition of {a} and {b} is {a+b}")

Functions in R.

→ function_name ← function ( ) — parameters

(Keyword)

→ add ← function ( ) — No parameters

eg. add ← function()
{
a = 5
b = 10
print(glue("Addition of {a} and {b} is {a+b}"))
}

★ We can create global variables INSIDE A Function in R.

var <<- value

add_param ← function(a, b = 10)
{
global_var <<- 10 → global var declaration inside an func^n.
return(a+b)
}

Type Conversion → as.desired_Output_Data_Type( )

c ← 9.5
d ← 60
] numeric - default Data Type for numbers.

c ← as.integer(c) | d ← as.integer(d)

To check Data Type → class()

Imp

→ Integers can be converted into characters (65 → "65").
→ Reverse is not True (if character is a alphabet).

Characters can be converted to integers only if that character is a number.
(Just like python).

Try-catch [exception / Error Handling].

→ Syntax → try catch ( )

- try catch (operation, error = function(e) print()). — For errors
- try catch (operation, warning = function(e) { print(). — for warnings

ch = 'a' | ch + var - error
var = 5 | as.integer(ch) - warning |.

READING FILES IN R.

1) csv files → read.csv()

2) Excel files → 1) install package
2) load that package (library()).
3) use it

→ install.packages('readxl')
→ library(readxl)
→ Syntax → [read_excel()].

3] Text files → readlines() [var ← readlines()]
↳ reads line by line

len(var) → Number of lines in txt files
nchar(var) → Number of characters in each line.

Reading char by char in txt file → scan()

var1 ← scan("pathoffile", "").

WRITING and saving into files.

1] Txt

Data ← matrix(c(1,2,3,4), nrow=3, ncol=4, byrow=TRUE).

write(Data, file='filepath', ncol=5, sep=' ')
↳ sep Blw values
↳ irrespective of Data - in how many columns data should be saved in txt file.

2] into csv

write.csv (df, file = "path", row.names = FALSE)

dataframe

↳ as in df we already have col names.

3] in excel

1) install package - xlsx.
2) load - library()
3) use - write.xlsx.

→ install.packages('xlsx')
→ library(xlsx)
→ write.xlsx(df, file = 'path', sheetname = 'Sheet1', col.names = TRUE, row.names = FALSE).

Basic String Operations in R.

#1 → Number of characters in a string (= or —)

→ nchar(var)

#2 → Conver to 'Lower' case and 'Upper case'

→ tolower(var)
toupper(var)

#3 To replace characters in a String

chartr (' ', '.', 'string')
[tr → to replace] — in which we have to replace

- what to replace — space
- with what — Dot

every space in string will be replaced by Dot.

x — x — x — x

#4 String Split

→ 1st → strsplit + 2nd unlist
└ To get individual words.

- 1st → strsplit ('Death and Freedom', ' ')
[by / from where]

[wherever we have a space - split the content]

- 2nd → and then unlist those words using unlist

eg →
```
catch ← strsplit ("Death and Freedom", ' ')
words ← unlist (catch)
```

x — x — x — x

#5 Sub-String (start = , end =)

→ SubStr ("Death and freedom", Start = 1, end = 7)

[Indexing in R starts from 1].

#6 - trim.ws → Removes white space from front and end of the string.
└ white space

Dates in R

YYYY - MM - DD
(Year - Month - Date)

1) UNIX (from 1970-01-01) To Date format

var ← as.POSIXct(col, origin = '1970-01-01')

result ← as.Date(var)

└ Date class object [YYYY-MM-DD]

ex ← as.Date(col, '%Y/%m/%d')

└ last / This argument tells to expect the input in this slash format. (and Y, m, d).

→ %Y → year (4-Digit) Representation

→ %y → 2-Digit year Representation

→ %M → Month Representation

→ %B → Full Month name

→ %b → Three Character abbr month name

%A = full weekday Name.

%a = 3 Char abbr weekday Name.

String to Date → as.Date()

as.Date("1994/06/07", "%Y/%m/%d")

→ We can use comparison and arithmetic operators B/W Dates. (subtraction /, <, >=, <=).

useful date functions.

→ Sys.Date() → current date as R date object.

→ date() → current DATE and TIME

getting what we want from DATE

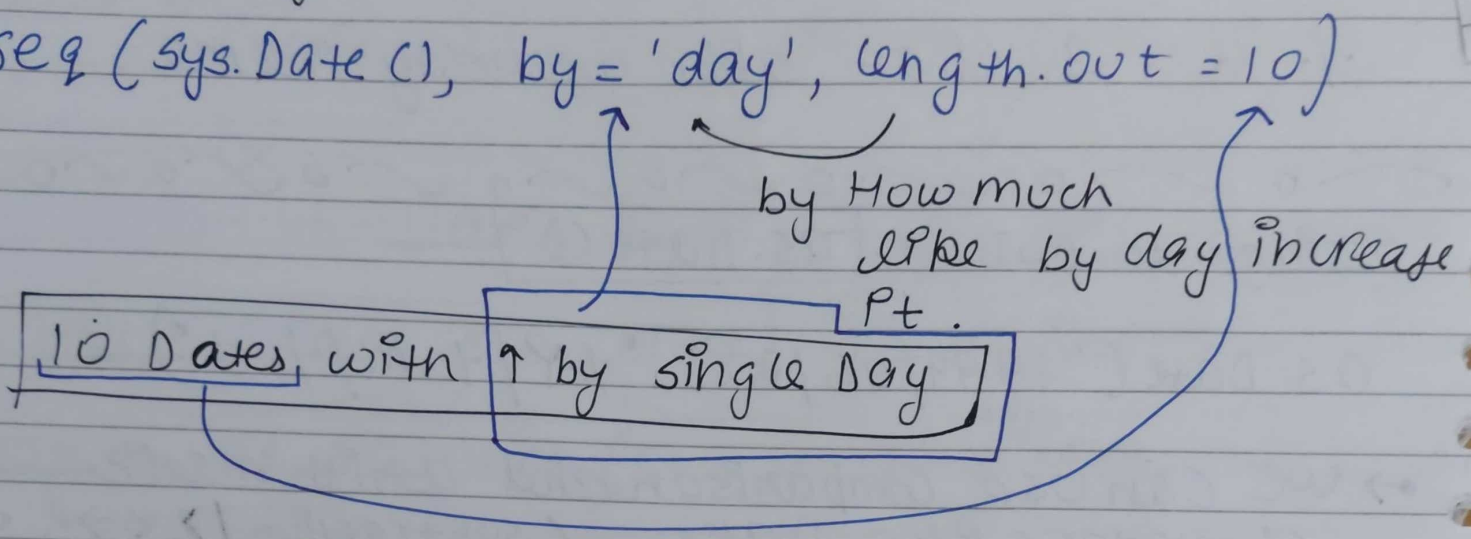
weekdays(sys.Date())

months(sys.Date())

quarters(sys.Date())

Sequence of Dates

```
seq(Sys.Date(), by = 'day', length.out = 10)
```



Regular expressions in R

↳ match pattern and strings in text.

. [period] = matches with any character

+ → matches the Preceding Pattern Element one or more time

\\. → ensures period matches with period and not any character.

```
# grep('@+\\.', c(values), value = TRUE)
```
↳ gives values that matches RE

→ To get result (extract the RE just not values).

```
result ← regexpr("@+\\.", c(values))
regmatches(c(values), result).
```

↳ gives out the extracted RE

%/% %>% %/%

More matrices

- Adding more rows and col^n's → rbind / cbind.
  - matrix combine
  - cbind

Removing Rows and cols — rbind(matrix, c(values))

mymatrix[-RowIndex, -colIndex] (removes whole Row and col).

X X X

Dplyr

%/% %>% %/% → pipeline → x %/% %>% function(y) = function(x, y)

select() → mutate() → COLUMN FUNC

filter() → distinct() → arrange() → ROW FUNC

group_by() → summarise() → count() → Group func

(group_by and summarise go together)

select() → Select Columns by Name or Helper func

- df %>% select(col1, col2, col3, col4, newcol_name = col5)
- df %>% select(coln : colm) — includes n and m
- df %>% select(-col1, -col2) — will not show col1 and 2 but every other

→ one_of(c("", "")) → starts_with("")

- select(
  - → Contains("") → Select columns whose name contains spec string
  - → ends_with("")
  - everything() ← every column
  - matches(".t.") ← name matches with REGULAR EXP

mutate() → Compute or Append one or more new columns.

- grepl(" ") → matches pattern and text within
  └ gives True and/or False

mutate (newcolname = condⁿ on which colⁿ name is divided)
└ mutate (is_collab = grepl("Featuring", artist)) %>% ↓
  └ colⁿ name select()

---

filter() → extract Rows that meet logical criteria

select() %>% filter (col-name and condⁿ) [1] for mul col
(| → or, & → and)

- arrange → sorting or viewing data order wise
  └ arrange (data) | arrange (desc (data))

- distinct() → To remove duplicate data (rows)
  └ distinct (row-name)

Group_by (group_by) → Summarise [max, min, mean, var, std]

Count() → count number of rows with unique value

Analysis of Songs By Taylor Swift + Her features

```
df %>%
  select(song, artist, rank, weekly_rank = "weeks.on.board")
  filter(grepl("Featuring Taylor Swift", artist) | grepl("Taylor Swift", artist))
  group_by(song) %>%
  summarise(highest_week_rank = max(weekly_rank)) %>%
  arrange(desc(highest_week_rank))
```

More Advanced

* Data cleaning → lubridate and stringr (date and Time)
* Reshaping Data → TIDYR [Data Tidying] ← cleaning/null values
* Joining/merging Datasets →

lubridate

Parse Date-Time (convert strings, char, no, to Date-Times)

ymd(col) | Date: y-m-d (day) | Time: h-m-s | y-m-d | ymd-hms()

To extract:
- date(x), year(x), day(x), wday(x) ← day of week, qday(x) → quater of Day
- hour(x), minute(x), seconds(x)
- week(x), quater(x)

Round Date-time

floor_date(x, 'month'/'second'/'year')

↳ round_date | ceiling_date

STRINGR

Detect Matches → **str_detect(string, pattern)**

→ **str_which(string, pattern)** → Find Indexes of strings that contain a patternmatch.

→ str_count(string, pattern) → Coun the no. of matches in a string.

Subset Strings

→ str_sub(string, start, end) → Extract substring from a character vector

→ str_extract(string, pattern) → Returns the first pattern match found in each string, as a vector. → str_extract_all to return every pattern

→ str_match(string, pattern) → Returns the first pattern match found in each string, as a matrix with a col for each () group in pattern.

↳ returns the Result

- **Manage Lengths**
  - str_length(string)
  - str_trunc(string, width, side = c("right", "left", "both"), pad = " ") → Truncate width of strings, replacing content with ellipsis
  - str_trim(string, side = c("right", "left", "both")) → Trim whitespaces from the start and/or end.

- **Mutate Strings**
  - str_to_lower(string) → str_to_upper(string) → str_to_title(string) → str_replace(string, replacement)

× × ×

Tidyr → Data Tidyness

→ mutate(Primary_artist = ifelse(str_detect(artrist, 'featuring'), str_match(artist, "(.*) \\s featuring") ← [,2], .* (one and more characters) artist) .

\\s (space)

[,2] → First it will run finding the whole match string .* \\s featuring and then will return and find-return the required (.*). so [,2]

→ Tidyr

Sapply (data, function(x) is.na(x)) → data.1.>.1 Sapply (function(x) is.na(x))

↳ To check null values in each columns.

is.na() → To check null values

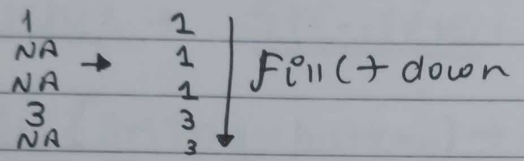
sum(is.na()) → Total null values

Tidyr

- Handle Missing values

1) drop_na( ) → Drop rows containing NA's in column

2) Fill (data.., direction = "down") → fill data in NA's columns (using the next or previous down value)

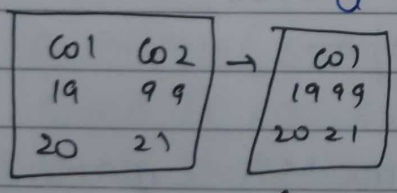


3) replace_na (data, replace) → replace with ~~max~~, mean, median or mode.

- Functions on cells (rows).

1) unite → unite (data, col, ... sep = " ", remove = TRUE, na.rm = FALSE)

↳ Collapse cells across several columns into a single column



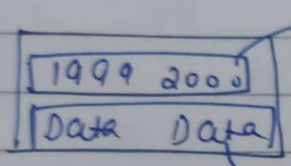
2) seperate → seperate (data, col, into, sep=, remove TRUE)

↳ seperate each cell in column into several col

↳ opposite of unite

Reshape Data

- pivot longer → pivot_longer (data, cols = a:b, names_to = , (col new name)
  values_to = , values_drop_na = FALSE)
  - Collapses several columns into rows, hence lengthing the Data



pivot_longer (data, cols = 1:2, names_to = year, values_to = cases)

- pivot wider → Inverse of pivot_longer
  - widen data by expanding rows into columns

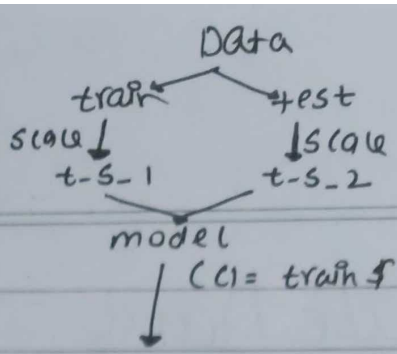
  → pivot_wider (data, names_from = "", value_from = "") (values to impute)
  (col to make wider)

  → pivot_wider (data, names_from "Type", values_from = "count()").

Merging data frames — Inner Join, Left Join, Right Join — Full outer Join

* Inner_join (Dataset 2, by = c("colname from D1" = "col from D2")

KNN



- library(class)
  library('caTools') → for spliting data

data ← iris

Data split (sample.split → obj → use in subset())

```
set.seed(123)
split ← sample.split(data, SplitRatio = 0.8) //obj
train_data ← subset(data, split=='TRUE')
test_data ← subset(data, split=='FALSE')
```

Feature scaling (scale)

```
train_data_scale
train_data ← scale(train_data[, 1:4])
test_data ← scale(test_data[,1:4])
test_data_scale
```

Model

```
model ⇒ knn(train = train_data_scale,
            test = test_data_scale,
            cl = train_data$species,
            K = 5)

conf_matrix = table(test_data$species, model)
conf_matrix

Acc = 1 - mean(model != test_data$species)
Acc
```

na.omit()
↳ removes rows with null values

K-means

```
library(factoextra)
Library(cluster)

df ← USArrests
df

df ← na.omit(df)  // Removes Rows with null values
df ← scale(df)    // scalling
df

# Elbow curve to find Optimal Value of K (fviz_nbclust)

fviz_nbclust(df, Kmeans, method = "gap_start")  // gap-statistic
fviz_nbclust(df, Kmeans, method = "wss")  // sum of square

# Model with K=4
set.seed(123)
model ← Kmeans(df, centers=4, nstart=25)
model

# Graph
fviz_cluster(model, data=df)
```

LR

[Library(tidyverse)]

```
Model ← lm(y~x, data=df)
summary(model) — R-score and all other summary
coef(model) → Intercept value (1) and x (2)   (y = mx+b)

ggplot(data=df, aes(x=x, y=y)) +
    geom_point() +
    geom_abline(intercept = (coef(model))[1], slope = ((coef(model))[2]).
```

OOPs in R — everything is a object

- Modular pieces of code → build blocks for large systems.
- Classes - objects - Abstraction - Encapsulation - Inheritence - polymor.

S3 → overload any func^n [No Inheri or encapsulation
No safety

S4 → Important characteristic of OOPs. [Build after S4
Add safty

Nowadays we use R6

Classes: User-defined data Types that serve Blueprint for creating Objects - attributes - methods

Object → Instances of Individual classes

Use Blueprint to create seperate instance with different values.

- constructor ⇒ initialize = function ()
- public = list()
- private = list()
- Inherit = Parent-class name

Inside the constructor → super $ initialize ($1, $2, $3) // calling Base class constructor

- Personal Info = Private → access them By using private $ to access them
- Personal Info = Public → access them By using [redacted] self $

```
Dog <- R6 Class (
    classname = " ",

    private = list (
      name = NULL,
      age = NULL,
      hair_color = NULL), →

    public = list (
    initialize = function (name = NA,
    age = NA, hair_color = NA) {
      private $ name = name
      private $ age = age
      private $ hair_color = hair_
                             color
    } ,
```

```
bark = function ()
{
    cat (private $name, ' says fuck off')
}
)
)
```

// Obj creation

```
Dog1 ← Dog$new ('Milo', 4, 'white').
Dog1$Bark()
Dog1$name → NULL as it is Private.
```

In Inheritence create a Parent class → A
In its child B add → Inherit = A.
In B's public constructor call parent constructor (super$initializer)
will initialize the properties ↵
Inherited from Parent.

Graph Points

```
ggplot(data, aes(x,y)) +
    geom_type (aes(color, size)) +
    theme_options() +
    xlab() +
    Ylab(),+
    title()
```

- geom_area()
- geom_line()
- geom_point() ↳ Scatter
- geom_abline()
- geom_text()
- geom_histogram()
- geom_bar()
- geom_jitter()
- geom_col()
- geom_boxplot()
- geom_violin()

Normal

- Barchart

```
barplot(data, xlab, ylab, main, col=, border=)
                          └ title        └ Bar color
```

- Historgram

```
hist(data, xlab, ylab, main, col=, border,
     breaks = )
                └ Data Distribution over graph
```

- Boxplot

```
Boxplot(y~x, data, xlab, ylab)
```

- Scatterplot()

```
plot( Speed ~ dist, data, xlab, ylab, main, pch=,
      cex = , col= )
      └ size of points        └ design of points (•, o, Δ, □)
                                └ Points size
```

palette() → use for colors or colors() [has more colors]

```
var <- grep('dark', colors(), value = TRUE)
   └ will gives all colors related to dark.
col = var  in graph
```

X X X