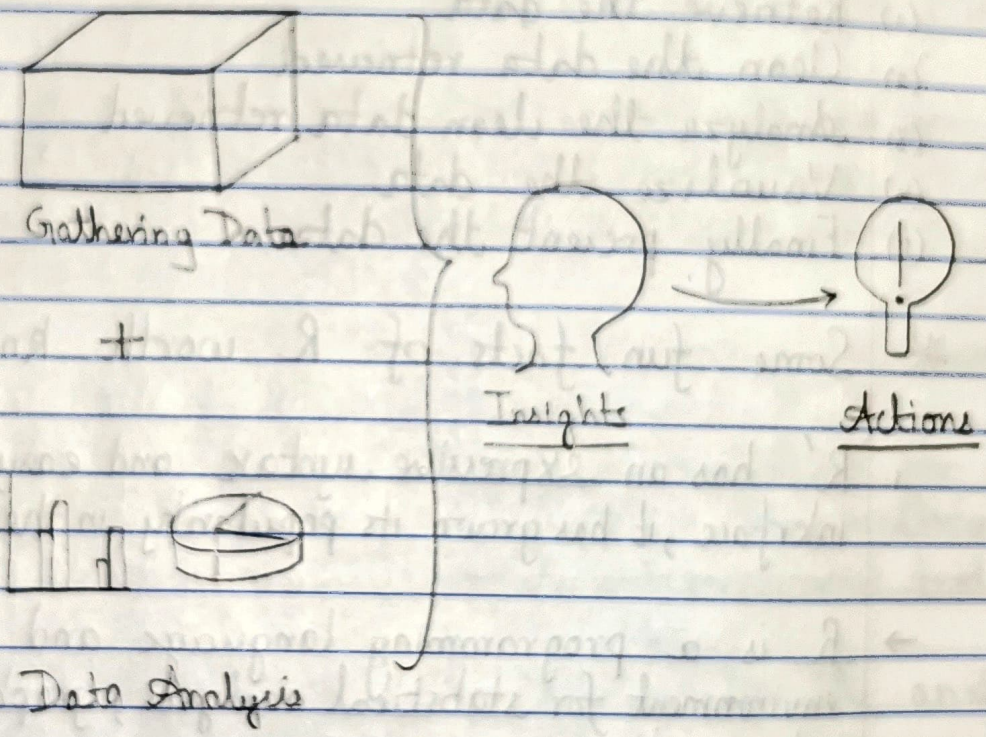
By vhh2105

Community Contribution - Understanding EDAV (for real)

~~Understanding Exploratory Data Analysis (Using R)~~

Introduction

It is said it's a good practice to understand the data primarily and then try to gather as many insights from it.



So exploratory data analysis can be said to be the critical process of performing initial investigations on data so as to discover patterns, to spot anomalies, to test hypothesis and to check the assumptions with the help of summary statistics and graphical representations.

* R programming for EDAV

- R is a programming language and environment commonly used in statistical computing, data analytics and scientific research

- One of the most popular languages used by the statisticians, data analysts, researchers and marketers to

(1) Retrieve the data
(2) Clean the data retrieved
(3) Analyze the clean data retrieved
(4) Visualize the data
(5) Finally present the data

* Some fun facts of R worth knowing :-

1. 'R' has an expressive syntax and easy-to-use interface, it has grown its popularity in the recent years

→ R is a programming language and software environment for statistical analysis, graphics, representation and reporting.
→ R was created by Ross Ihaka and Robert Gentleman at the University of Aukland, New Zealand and is currently developed by the R Development Core Team.
→ The core of R is an interpreted computer language which allows branching and looping as well as modular programming using functions.
→ R allows integration with procedures written in the C, C++, .Net, Python or FORTRAN languages for efficiency.

2. R is freely available under the GNU-General Public License and pre-compiled binary versions are provided for various operating systems like Linux, Windows and Mac.

3. R is a free software which is distributed under a GNU-style copy left and an official part of the GNU project called GNU S.

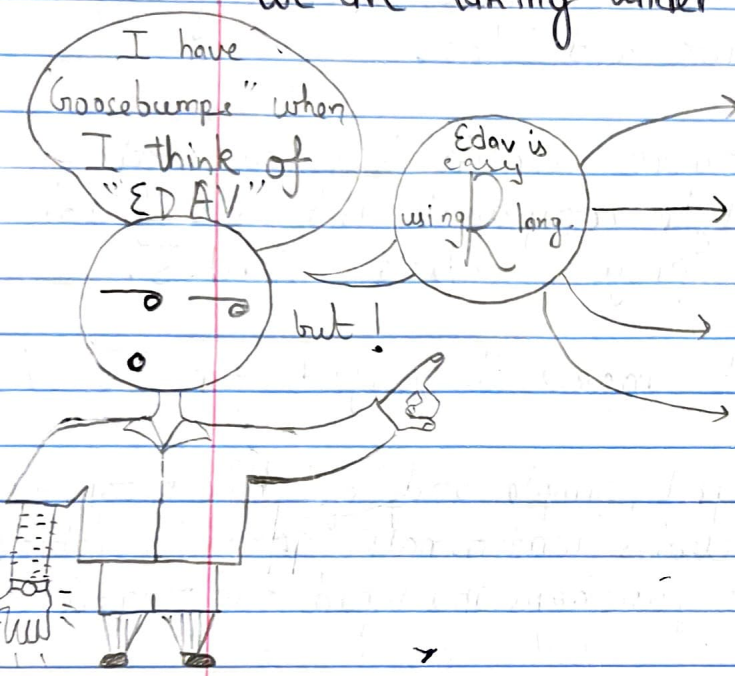
* Features of R that makes it perfect for EDAV

1. R is a well developed, simple and effective programming language and includes conditionals, loops (iterations), user defined recursive functions and input and output facilities.
2. R has an effective data handling measure and storage facility.
3. R also gives us a suite of operators for calculations on arrays, lists, vectors and matrices.
4. R provides a large, coherent and integrated tools collected for data analysis.
5. R gives graphical facilities too for the analysis of data and display either directly on the computer or printing at the papers.

Is R programming an easy language to learn?

It is quite difficult to answer. Many researchers are learning R as their first language to solve their needs in data analysis.
That's the power of R programming language, it's simple enough to LEARN as you 'GO'. All you

need is data and clear intention to draw an observable conclusion based on analysis on the data we are taking under consideration.



In fact, R is built on top of the language S programming that was originally intended as a programming language that would help the students learn programming while playing around the data.

"But", programmers that have a background from a Python, PHP or JAVA background might find R to be quirky and a little bit confusing at FIRST!

The syntax that R uses is a bit different from other common programming languages. This is quite amusing too, while R have all the capabilities of a programming language, but we will not find ourself writing a lot of if conditions

or loops while writing code in R language. It's probably due to the other programming constructs like vectors, lists, frames, data tables and matrices, that allows us to perform data transformations in a bulk.

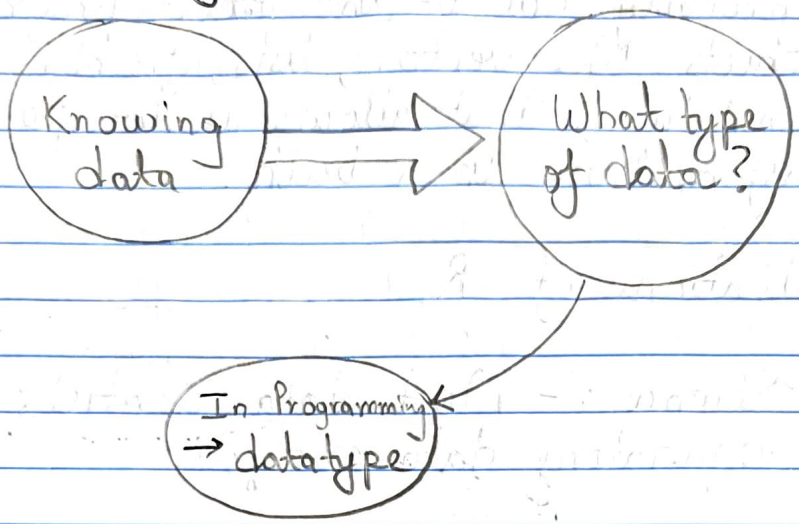
Applications of R

1. Data Science :- Most data scientists uses R to do exploratory data analysis.

2. Statistical Computing :- Did we know R was built by statisticians for statisticians. It has more than 9100 programming based R packages with every statistical tool one can imagine.

3. Machine learning :- R capability is more extensible when used for predictive analytics and machine learning.

Understanding basic data types in R



- To make best of R, you'll need a strong understanding of the basic data types and data structures and more importantly how to operate on them.

It is very important to understand because these are the things we will manipulate on a day to day basis in R. Most common source of frustration among beginners

- Everything is object in "R".

R has "5" basic atomic classes:-

1. logical (e.g True, False)
2. integer (e.g. 2L, as.integer(3))
3. numeric (real or decimal)(e.g 2, 2.0, pi)
4. complex (e.g 4+0.1i, 3+2i)
5. character (e.g "a", "vipul")

Now some handy functions)

```
typeof()      # What is it?
class()       # (Sorry) What is it, again?
length()      # How long is it? What about 2-d
              # objects?
attributes()  # Does it have any metadata?
```

Now what about the data structures?

The way of storing data is structural format for easy retrieval, and understanding.

→ Vector
→ list
→ matrix
→ data frame * Important for us to know
→ factors * We will avoid these, but they have their own uses
→ tables

A Vectors

- A vector is most common
- Basic data structure in R
- Pretty much like workhorse of R

Vectors are of 2 types
→ atomic vectors
→ lists

Creating an empty vector with vector().

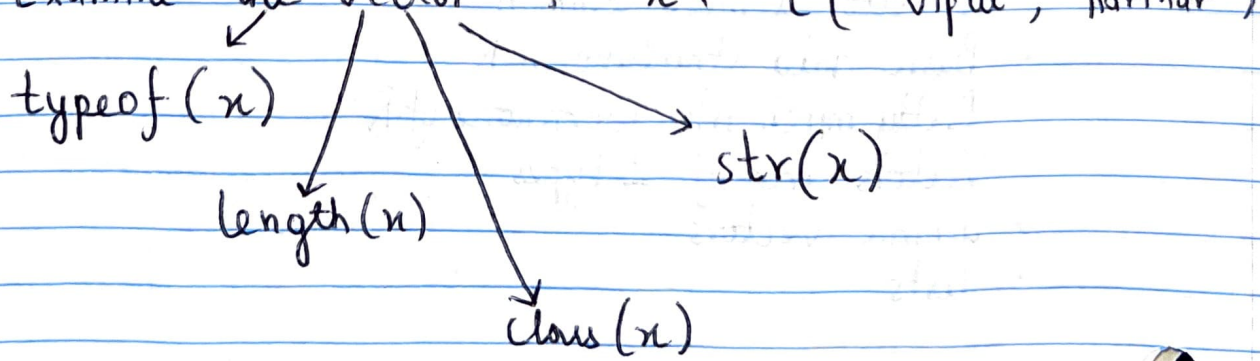
- The general pattern is vector (class of object, length)
- We can also create vectors by concatenating them in c() function.

e.g
```
x <- vector()    # Creating empty vector()
# with predefined length
x <- vector(length = 10)
# with a length and type
vector("character", length = 10)
vector("numeric", length = 10)
vector("integer", length = 10)
vector("logical", length = 10)
```

Using c() function

```
x <- c("a", "b", "c") : character vector
y <- c(1,2,3)  it's a numeric vector, treated as numerical objects as double precision
x1 <- c(1L,2L,3L) : to explicitly specify integers, use a 'L'
y <- c(TRUE, FALSE) : logical vectors
```

Examine the vector : x <- c("vipul", "harihar")



* You can also create vectors as sequence of numbers

```
series <- 1:10
seq(10)
seq(1, 10, by = 0.1)
```

Other objects

Inf is Infinity. You might find positive or negative infinity both.

```
e.g  1/0
#  [1] Inf
1/ Inf
# [1] 0
```

NaN means Not a number, it is in simple words the undefined value.

```
0/0
NaN
```

Each object has an attribute and these attributes can be a part of an Robject. These include:

- names
- dimnames
- length
- class
- attributes (contains metadata)

* For a vector, length(vector_name) is just the total number of elements.

Vectors may have only one data type

R will create a resulting vector that is the least common denominator. The coercion will move towards the one that's easiest to coerce to.

The coercion rules goes

logical → integer → numeric → complex → character

We can also coerce vectors explicitly using the as.<class_name>

```
as.numeric()
as.character()
```

Note :-

① When you coerce an existing numeric vector with as.numeric, it does nothing
② Sometimes coercions, especially nonsensical ones won't work.
③ Sometimes there is implicit conversion

```
1 < "2"
# TRUE
"1" > 2
# FALSE
1 < "a"
# TRUE
```

B Matrix

- Matrices are a special vector in R
- They are not a separate class of object but simply a vector but now with dimensions added on it.
- Matrices have rows and columns

```
m <- matrix(nrow = 2, ncol = 2)
m
dim(m)
Same as attributes(m)
```

Matrices are constructed columnwise

```
m <- matrix (1:6, nrow = 2, ncol = 3)
```

Other ways to construct a matrix

```
m <- 1:10
dim(m) <- c(2,5)
```

↳ This takes a vector and transform into a matrix with 2 rows and 5 columns.

ANOTHER WAY to bind columns or rows using cbind() and rbind()

```
x <- 1:3
y <- 10:12
cbindy <- 10:12   (wrong) it is a variable
cbind(x, y)
# or
rbind(x, y)
```

List

- In R list act as containers
- Unlike atomic vectors, its contents are not restricted to a single mode and can encompass any data type
- Lists are sometimes called recursive vectors, because a list can contain other lists.
- This makes them very different from atomic vectors.

Lists is a special vector. Each element can be a different class

```
x <- list(1, "a", TRUE, 1+4i)
x <- 1:10
x <- as.list(x)
length(x)
```

What is the class of x[1]? how about x[[1]]?

```
xlist <- list(a = "Vipul", b = 1:10, data = head(iris))
```

* List can be contained as many listed list inside one another list.

```
temp <- list(list(list(list())))
temp
is.recursive(temp)
```

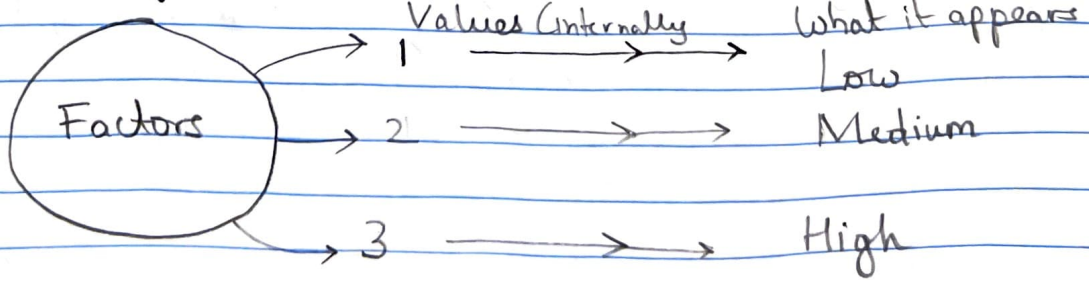
→ Lists are extremely useful inside functions.
→ You can staple together lots of different kind of results into a single object that a function can return.
→ It doesn't print out like a vector. Prints a new line
→ Elements are indexed by double brackets. Single brackets will still return another list.

Factors - VERY IMPORTANT

- Factors are special vectors that represent categorical data
- Factors can be ordered / unordered and are important when for modelling functions such as lm() and glm() and also in plot methods
- Factors can only contain pre-defined values.
- Factors can be thought as integers that have labels on them.
- While factors look (and often behave) like character vectors, they are actually integers underneath and we need to be very careful when treating them like strings.
- Some string methods will coerce factors to string, while others will throw an error

Sometimes factors can be left unordered. e.g male or female.
Other times you might want factors to be ordered (or ranked). Example low, medium, high.
Underlying it's represented by numbers 1, 2, 3.

They are better than using simple integer labels because factors are what are called self describing male and female is more descriptive than 1s and 2s. It's more helpful when there is no additional metadata



e.g Which is male? 1 or 2? We will not be able to tell that by just integer value.
Factors have this information built in

Factors can be created with factor(). Input is a character vector

```
x <- factor(c("yes", "no", "yes", "no", "maybe"))
x
```

table(x) ⟶ will return a frequency table

unclass(x) ⟶ strips out the class information

In modelling functions, important to know what baseline level is. This is the first factor but by default the ordering is determined by alphabetical order of words entered. You can change this by specifying the levels.

```
x ← factor(c("yes", "no", "yes"),
           levels = ("yes", no"))
```

Data Frame

A data frame is a very important data type in R. It's pretty much the de-facto data structure for most tabular data and what we use for statistics data frames can have additional attributes such as rownames() This can be useful for annotation of data, like subject_id or sample_id. But mostly they are not used

e.g rownames() used for annotating data, subject names. other times they are not useful.

- Data frames usually created by read.csv and read.table
- Can convert to matrix with data.matrix()
- Coercion will force and not always what you expect
- Can you also be using data.frame() function

With and data frame, you can do nrow(df) and ncol(df) rownames are usually 1...n.

Combining data frames

e.g

```
df <- data.frame(id = letters[1:10], x = [1:10],
                 y = rnorm(10))
```

```
> df
  id  x      y
1  a  1 -1.012
2  b  2  0.086
3  c  3  0.072
4  d  4  0.016
```

5	e	5	1.123
6	f	6	0.871
7	g	7	0.941
8	h	8	0.732
9	i	9	-0.514
10	j	10	0.291

cbind(df, data.frame(z=4))

When you combine column wise, only row numbers need to match. If you are adding a vector, it will get repeated.

Useful functions:

head() - see first 5 rows

tail() - see last 5 rows

dim() - see dimensions

nrow() - number of rows

ncol() - number of columns

str() - structure of each column

names() - will list column names for a data.frame (or any object really)

A data frame is a special type of list where lists every element has same length.

See that it is actually a special list:

```
> is.list(iris)
[1] TRUE

> class(iris)
[1] "data.frame"
> --
```

Naming objects

Other R objects can also have names not just true for data.frames.
Adding names is helpful since it's useful for readable code and self describing objects

```
x <- 1:3
names[x] <- c("rich", "daniel", "diego")

x ↵
```

List can also have names

```
x <- as.list[1:10]
names(x) <- letters[seq(x)]
x
```

Finally matrices can haves names and these are called dimnames.

```
m <- matrix(1:4, nrow = 2)
dimnames(m) <- list(c("a", "b", "c"), c("c", "d"))
# first element = rownames
# Second element = colnames.
```

Missing Values

Denoted by NA and/or NAN for undefined mathematical operations.

is.na()
is.nan()

check for both.

NA values have a class. So you can have both an integer NA and a missing character NA.

NaN is also NA. But not the other way around.

x <- c(1, 2, NA, 4, 5)

is.na(x) returns logical. shows third

is.nan(x) # none are NaN

x <- c(1,2, NA, NaN, 4, 5)

is.na(x) shows 2 TRUE. is.nan(x) shows 1 TRUE

Missing values are very important in R, but can be very frustrating for new users.

What do these do? What should they do?
~~~ 1 == NA NA == NA ~~~
~~~

Diagonistic functions in R:

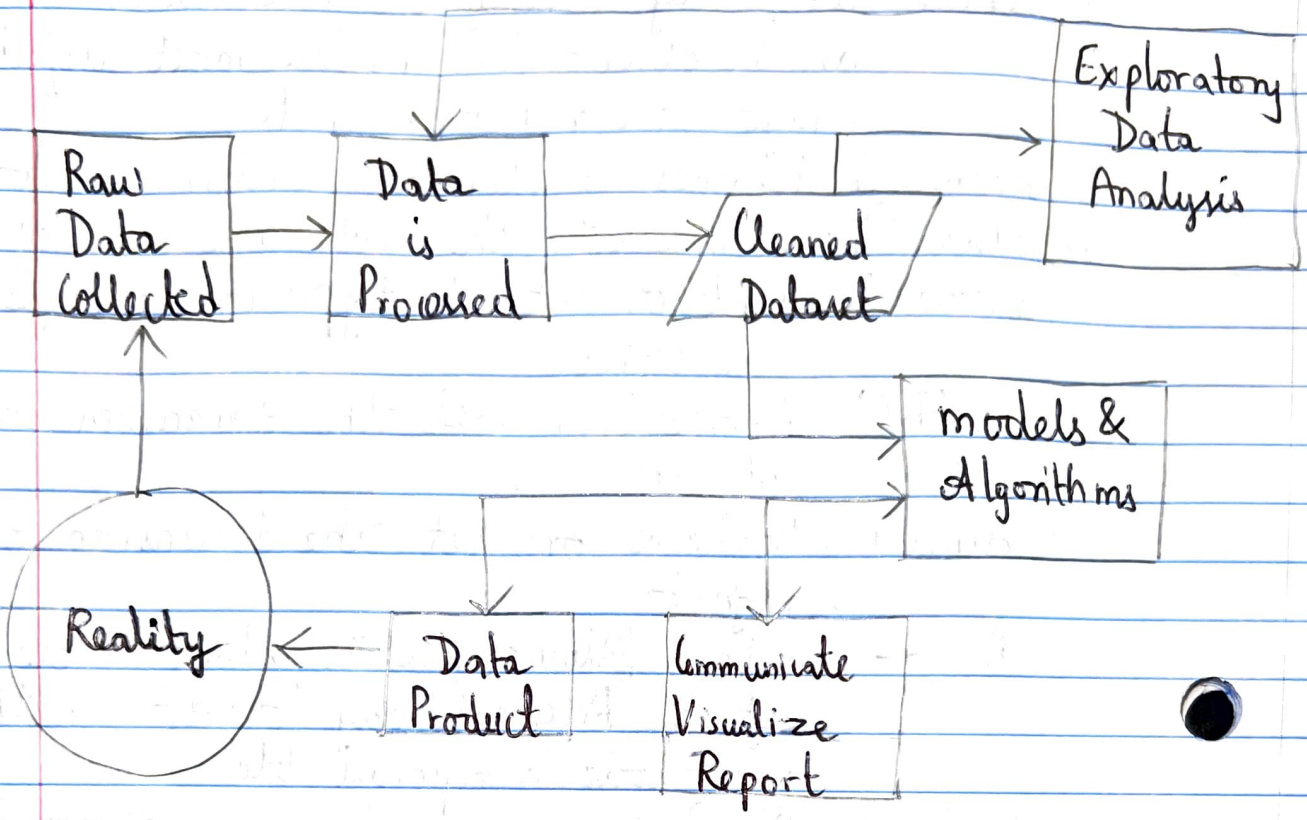
1. str(): Compactly display the internal structure of an R object. Perhaps the most useful diagonistic function in R.
2. names(): Names of elements within an object
3. class(): Retrieves the internal class of an object
4. mode(): Get or set the type or storage mode of an object
5. length(): Retrieve or set the dimension of an object
6. dim(): Retrieve or set the dimension of an object
7. R --vanilla- Allows you to start a clean session of R. A great way to test whether your code is reproducible.
8. sessionInfo(): Print version information about R and attached or loaded packages
9. options()- Allow the user to set and examine a variety of global options which affect the way in which R computes and displays its result.
10. sessionInfo() - Print version information about R and attached or loaded packages.

str() is our best friend indeed

str is short for structure. You can use it, on any object. Try the following:

```
x <- 1:10
class(x)
mode(x)
str(x)
```

Data Science Process



What all visualizations we use for EDAV

1. Histogram
2. Box Plot
3. Violin Plot
4. Ridgeline Plot
5. QQ - Plot
6. Bar Chart
7. Cleveland Dot Plot
8. Scatterplot
9. Chart: Parallel Coordinate Plots
10. Chart: Mosaic
11. Chart: Heatmap

... to be continued