Exploratory data analysis with R

Exploratory Data Analysis

- Exploratory Data Analysis (EDA) is a method used to analyze and summarize the main characteristics of a dataset, often employing statistical graphics and data visualization techniques.

- It's a crucial preliminary step before more formal data analysis or modeling, helping to understand the data, identify patterns, and formulate hypotheses.

What EDA covers

- Data management (Data wrangling…)
- Descriptive analytics (Central tendency, dispersion)
- Diagnostic analytics (correlation….)
- Hypothesis testing
- Visualization (bar graphs, box plots, scatter plots…)

Operators in R

- R has many operators to carry out different mathematical and logical operations.

Type of operators in R

Arithmetic operators
Relational operators
Logical operators
Assignment operators

Arithmetic operators

- These operators are used to carry out mathematical operations like addition and multiplication.

- Examples of arithmetic operators available in R are in the table.

Arithmetic Operators in R	
Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
^	Exponent
%%	Modulus (Remainder from division)
%/%	Integer Division

R Relational Operators

- Relational operators are used to compare between values. Here is a list of relational operators available in R.

Relational Operators in R	
Operator	Description
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	Equal to
!=	Not equal to

R Logical Operators

- Logical operators are used to carry out Boolean operations like AND, OR etc.

- & represents AND

- | represents OR

R Assignment Operators

- These operators are used to assign values to variables.

- The operators <- and = can be used, almost interchangeably, to assign to variable in the same environment.

The pipe operator in R

- The pipe operator **(%>%) or (|>)** is a binary operator that forwards the value of its left-hand side to the first argument of the function on its right-hand side.

- This enables a natural and readable way of chaining multiple functions together, reducing the need for intermediate variables and improving code clarity.

- %>% is a function in dplyr package

- |> is a function in base R.

Pipe operator usage

```r
# Load required library
library(dplyr)
# Create a data frame
data <- data.frame(x = 1:10, y = rnorm(10))
# Filter rows, mutate column, and summarize data using pipe operator
summary <- data %>%
  filter(x > 5) %>%
  mutate(z = x + y) %>%
  summarize(mean_z = mean(z))
# Print summary
print(summary)
```

Part 1

Data wrangling with R

Filtering rows in data sets

filter() selects the rows of a data frame that meet a column criteria.

Example: Create a new dataset containing the maize crop only

```
maize <- filter(crop_recommendation, Crop == "maize")
```

Example2: Create a new dataset containing humidity values above 50

```
hum_above50 <- filter(crop_recommendation, humidity > 50)
```

Example3: Create a new dataset containing humidity values above 50 for maize crop

```
hum_maize_above50 <- filter(crop_recommendation, humidity > 50 & Crop == "maize")
```

	Crop	Nitrogen	Phophorous	Patassium	temperature
1	maize	75.0	32.3	130.6	25.7
2	maize	115.0	35.4	87.4	18.5
3	maize	121.1	40.1	134.6	25.0
4	maize	73.5	36.0	112.4	22.1
5	maize	109.4	26.1	80.3	18.6
6	maize	81.5	25.7	135.6	23.4
7	maize	93.6	27.4	125.3	23.2
8	maize	81.0	54.4	107.5	27.4
9	maize	70.9	49.8	131.4	22.5
10	maize	126.8	34.0	114.7	26.4
11	maize	120.2	54.7	113.4	28.1
12	maize	112.1	44.4	119.1	21.8

Select columns

select() is used to select columns that you want to **retain**.

Example: Select Crop and yield_category then store them.

```
data <- select(crop_recommendation, Crop, yield_category )
```

Example Using pipe operator

```
data1 <- crop_recommendation  %>%
         select(Crop, yield_category )
```

#deselect/remove columns from the data set

```
data2 <- select(crop_recommendation, -Crop, -yield_category )
```

Filter

	Crop	yield_category
1	maize	High
2	cotton	High
3	rice	High
4	coffee	Low
5	cotton	High
6	coffee	High
7	chickpea	Low
8	maize	High
9	cotton	High
10	wheat	Low
11	chickpea	High
12	maize	Low

Combining functions to select and filter

Example Using pipe operator

```
data3 <- crop_recommendation  %>%
          select( Crop, yield_category ) %>%
           filter(Crop == "rice ")
```

Filter

	Crop	yield_category
1	rice	High
2	rice	High
3	rice	Low
4	rice	Low
5	rice	Low
6	rice	High
7	rice	High
8	rice	High
9	rice	Low
10	rice	Low
11	rice	Low
12	rice	Low

arrange

arrange() orders the rows of a data frame by the values of selected columns.

arrange() sorts values in ascending order or descending order.

Example: Sort the yield in ascending order

```
data4 <- arrange(crop_recommendation, yield_kg_ha)
```

Sort yield in descending order

```
data5 <- arrange(crop_recommendation, -yield_kg_ha)
```

idity	pH	rainfall	yield_kg_ha	yi
71.7	5.61	193.1	1347	Lo
89.3	6.31	242.6	1433	Lo
60.7	6.04	180.3	1433	Lo
88.5	5.93	131.8	1433	Lo
87.4	6.01	201.4	1451	Lo
60.1	5.35	208.5	1467	Lo
70.3	5.34	243.2	1470	Lo
40.0	6.86	102.4	1525	Lo
50.3	6.06	72.1	1542	Lo
72.5	5.42	223.3	1544	Lo
33.0	7.01	110.5	1549	Lo
87.0	6.55	211.6	1575	Lo

idity	pH	rainfall	yield_kg_ha	y
72.7	6.83	200.1	84071	H
85.2	6.21	243.9	83453	H
78.3	6.38	220.1	82620	H
68.5	7.01	265.7	81222	H
66.2	6.63	148.2	80924	H
73.7	6.36	223.2	80729	H
87.7	7.26	276.5	80657	H
69.6	7.12	248.2	80031	H
86.8	6.97	147.5	79779	H
82.6	6.76	240.0	79705	H
69.7	6.82	198.4	79257	H
60.7	6.60	260.0	79225	H

mutate

mutate() creates new variables in the data set

It adds the new variable to existing data

#Example: creates a new variable "new_yield" by getting half the yield_kg_ha variable

```
half_yield <- mutate(crop_recommendation , new_yield = yield_kg_ha/2)
```

#Alternative

```
half_yield <- crop_recommendation %>%
                mutate(new_yield = yield_kg_ha/2)
```

rainfall	yield_kg_ha	yield_category	new_yield
171.5	6062	High	3031.0
63.6	1789	High	894.5
156.6	5697	High	2848.5
126.4	1579	Low	789.5
92.4	1930	High	965.0
144.4	1675	High	837.5
67.4	1726	Low	863.0
198.3	6558	High	3279.0
71.6	1750	High	875.0
128.8	4465	Low	2232.5
86.0	1780	High	890.0
139.9	6004	Low	3002.0

rename

rename() creates new names for variables in the data set

#Example: rename Nitrogen to N

```
nit <- rename(crop_recommendation, N = Nitrogen)
```

	Crop	Nitrogen	Phophorous
1	maize	75.0	32.
2	cotton	78.9	48.
3	rice	101.3	58.
4	coffee	108.8	26.
5	cotton	70.5	32.
6	coffee	98.7	34.
7	chickpea	43.6	26.
8	maize	115.0	35.
9	cotton	50.7	43.
0	wheat	97.0	25.
1	chickpea	48.7	39.
2	maize	121.1	40.

	Crop	N	Phophorou
1	maize	75.0	32.3
2	cotton	78.9	48.8
3	rice	101.3	58.8
4	coffee	108.8	26.6
5	cotton	70.5	32.2
6	coffee	98.7	34.9
7	chickpea	43.6	26.9
8	maize	115.0	35.4
9	cotton	50.7	43.9
10	wheat	97.0	25.1
11	chickpea	48.7	39.2
12	maize	121.1	40.1

relocate

relocate() gives a new order to column names

Example: relocate the rainfall and yield_category from their original position

New_order <- relocate(crop_recommendation, rainfall, yield_category)

	Crop	Nitrogen	Phosphorous	Potassium
1	maize	75.0	32.3	130.6
2	cotton	78.9	48.8	105.7
3	rice	101.3	58.8	142.5
4	coffee	108.8	26.6	100.9
5	cotton	70.5	32.2	90.8
6	coffee	98.7	34.9	102.2
7	chickpea	43.6	26.9	43.7
8	maize	115.0	35.4	87.4
9	cotton	50.7	43.9	96.4
10	wheat	97.0	25.1	93.9
11	chickpea	48.7	39.2	79.9
12	maize	121.1	40.1	134.6

	rainfall	yield_category	Crop	Nitrogen
1	171.5	High	maize	75.0
2	63.6	High	cotton	78.9
3	156.6	High	rice	101.3
4	126.4	Low	coffee	108.8
5	92.4	High	cotton	70.5
6	144.4	High	coffee	98.7
7	67.4	Low	chickpea	43.6
8	198.3	High	maize	115.0
9	71.6	High	cotton	50.7
10	128.8	Low	wheat	97.0
11	86.0	High	chickpea	48.7
12	139.9	Low	maize	121.1

Part 2

Descriptive analytics

Data summary and missingness

- Data summary gives a comprehensive understanding of large datasets and variables.
- Categorical data can be summarized using frequency distribution tables.
- Frequency distribution tables are obtained using the table function: table()
- Missingness can be detected using the table function: table()
- Descriptive summaries are obtained using the summary function: summary()

❖Demonstration in RStudio

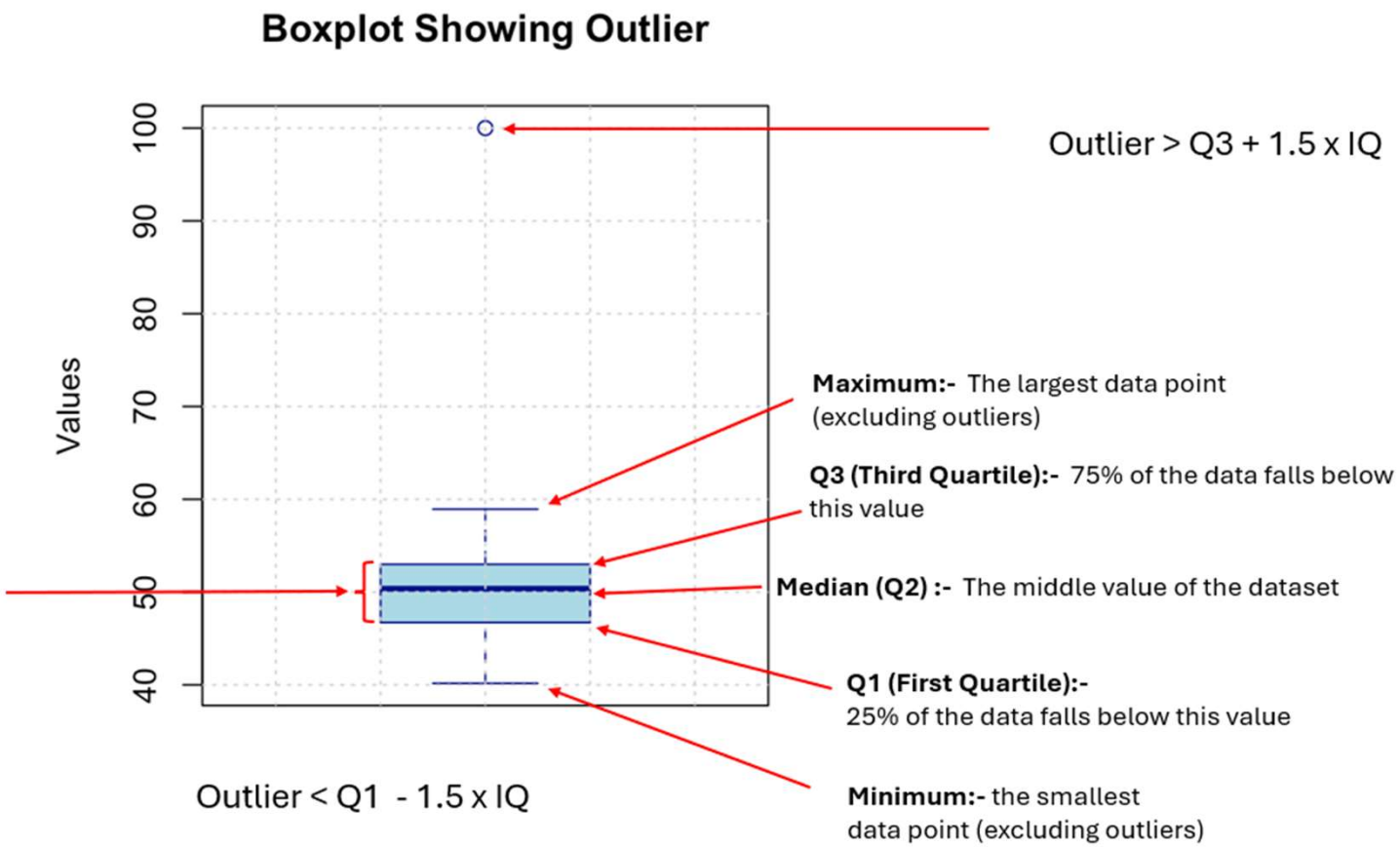
Outlier detection

- Outliers are values that are considered out of range in a given numerical variable.
- Outliers affect data modelling and can cause drastic changes in results.
- Outliers can be quickly detected visually using box plots.

Box plots

- Box plots are figures that contain important information about the summary of numeric variable.
- The box represents the interquartile range (IQR), from Q1 (25th percentile) to Q3 (75th percentile).
- The line inside the box is the median.
- Whiskers extend up to 1.5 × IQR from Q1 and Q3.
- Points beyond the whiskers are considered outliers.
- Minimum:- the smallest data point (excluding outliers)
- Maximum:- The largest data point (excluding outliers)

Box plots showing outlier



Distribution of data

- Understanding of distribution of numeric data is key.
- It involves testing for normality of key variables
- Distributions can be assessed in two ways:

➢ Visually using graphs such as the histogram

➢ Performing a statistical test of normality such as Shapiro's wilk test.

Data visualization with ggplot2

What is ggplot2?

- ggplot2 is part of the collection of packages within tidyverse.
  - It is used for visualization of data in R
  - "gg" stands for grammar of graphics

#load the library of ggplot2 to access its functions
library(ggplot2)

How ggplot2 works….

- ggplot2 works by adding different layers of information to a graph.
- Layers are added to the graph using the plus sign (+).
- Different layers perform different functions within ggplot2 package.
- Layers can be optionally added onto the graph.

Syntax of ggplot2

```
ggplot(dataframe, aes(x=variable, y=variable) +
geom_object() +
labs(title= "title of graph", subtitle = "subtitle of graph", x= "xlabel",y= "ylabel")+
coord_cartesian(xlim=c(a,b),ylim=c(a,b))
```

- **Importance of different functions above:**
  - ggplot() function specifies the data frame.
  - aes() specifies the variables to be plotted,color etc. It is the aestetics function.
  - geom() function specifies the type of graph to be plotted.
  - labs() function specifies the title of the graph and axis labels.
  - coord_cartesian() specifies the limits on the axes of a graph.
- **There are many other layers that can be added to graphs**.

Types of graphs

- Graphs are specified using the geom() function.
- You can have more than one type of graph on the same visualization.

Examples of graphs created by the geometric objects:

geom_point() : to draw points on a graph e.g scatter plots.

geom_smooth(): to draw smooth lines on a graph.

geom_histogram(): to draw a histogram on a graph

geom_line(): to draw a line graph

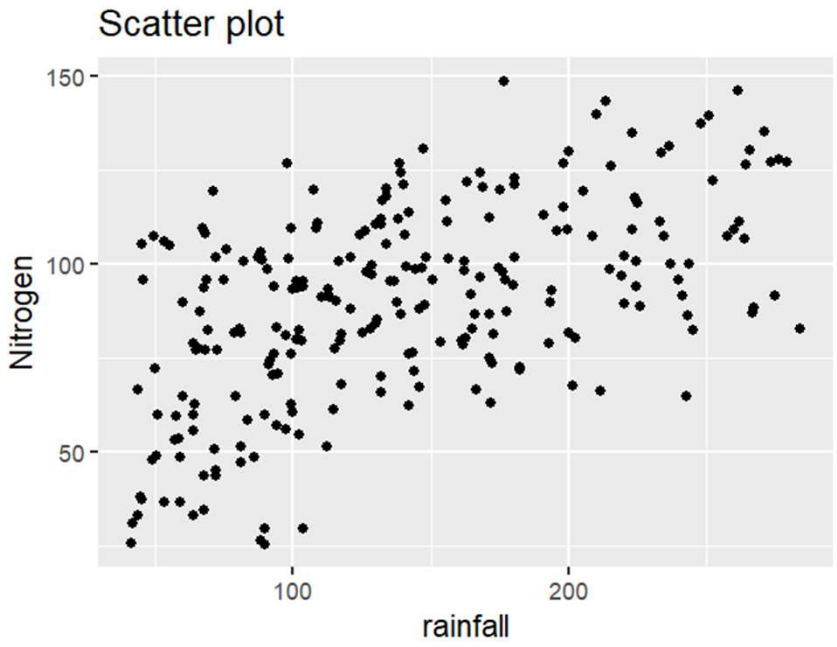
geom_bar(): to draw a bar graphs

geom_boxplot() :to draw a boxplot on a graph

Scatter plot

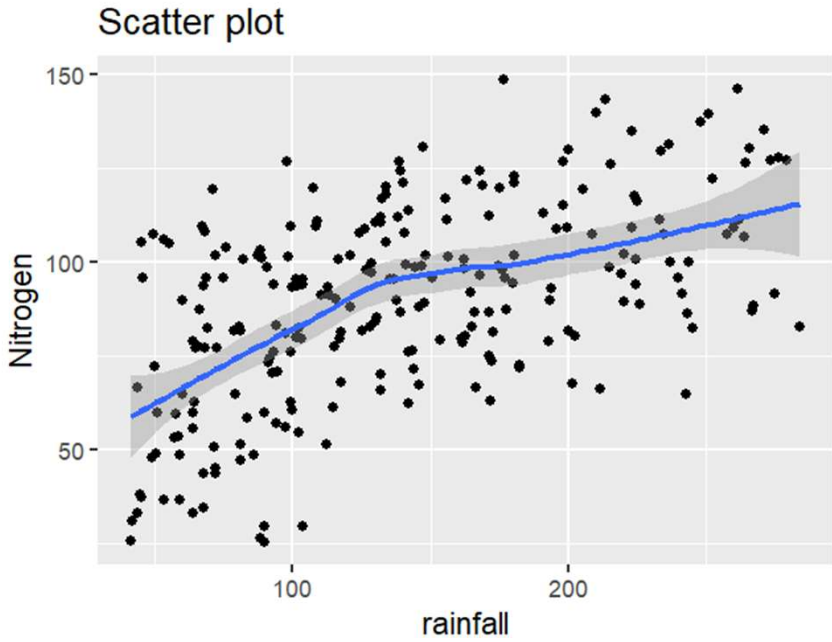
A scatterplot is a visualization of two continuous/quantitative variables.

```
#draw a scatter plot
ggplot(crop_recommendation, aes(x=rainfall, y=Nitrogen)) +
geom_point() +
labs(title= "Scatter plot", x= "rainfall", y= "Nitrogen")
```



Add a smooth line to the scatter plot.

```
ggplot(crop_recommendation, aes(x=rainfall, y=Nitrogen)) +
geom_point() +
geom_smooth() +
labs(title= "Scatter plot", x= "rainfall", y= "Nitrogen")
```



Add a straight line to the scatter plot.

```
ggplot(crop_recommendation, aes(x=rainfall, y=Nitrogen)) +
geom_point() +
geom_smooth(method="lm")  +
labs(title= "Scatter plot", x= "rainfall", y= "Nitrogen")
```