R Bootcamp for E2M2

We R
R User Group
Madagascar

Taratra D. Raharison
CEO - We R
taratra.d.raharison@gmail.com

Sponsored by

R consortium

Overview

- Introduction to R and RStudio
- R basics: Variables, Data Types, Vectors
- Data Frames in R
- Tidy Data Manipulation with **dplyr**
- Introduction to Data Visualization with **ggplot2**
- Introduction to Statistical Modeling with **stats**
- Introduction to Dynamical Modeling with **deSolve**
- Q&A

Introduction to R and RStudio

What is R?

What is R?

- R is a **free** software environment for **statistical computing** and **graphics**
- Used for:
    - Statistical Analysis
    - Data Manipulation
    - Scientific Computation and Simulation
    - Interfacing with other software (QGIS, GDAL, …)
- R is a programming **language** used to communicate with your computer
- As with any other language, it can **be learned**

Why R?

- Open-source, free and widely used in academia and research

SPSS

	Modules	Price per authorized user Save 10% with Yearly Monthly / Yearly	Select the number of authorized users	Price for selected configuration
Base Subscription Required	Data Preparation Bootstrapping Statistics Base	USD 99/month	1	USD 99/month
Add-on 1: Custom Tables & Advanced Statistics Optional Most Popular	Regression Advanced Statistics Custom Tables	add USD 79/month	0 Cannot be greater than quantity selected for Base subscription	USD 0/month
Add-on 2: Forecasting & Decision Trees Optional	Forecasting Decision Trees Direct Marketing Neural Networks	add USD 79/month	0 Cannot be greater than quantity selected for Base subscription	USD 0/month
Add-on 3: Complex Sampling and Testing	Complex Samples			

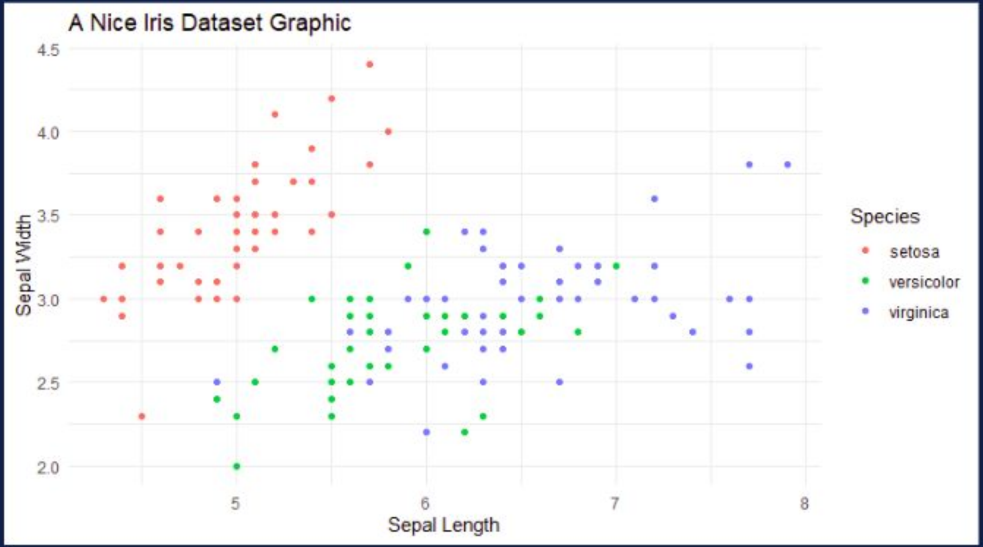
https://www.ibm.com/products/spss-statistics/pricing

Why R?

- Open-source, free and widely used in academia and research
- Powerful for Statistical Analysis, Visualization and modeling

R: ggplot2

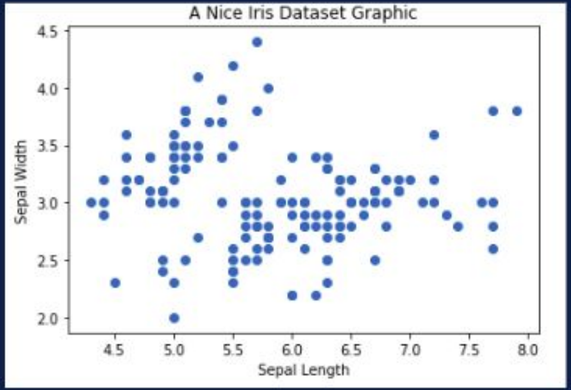
```
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width, col = Species)) +
  geom_point() +
  labs(title = "A Nice Iris Dataset Graphic", x = "Sepal Length", y = "Sepal Width") +
  theme_minimal()
```



https://www.inwt-statistics.com/blog/data-visualization-r-versus-python

Python: Matplotlib

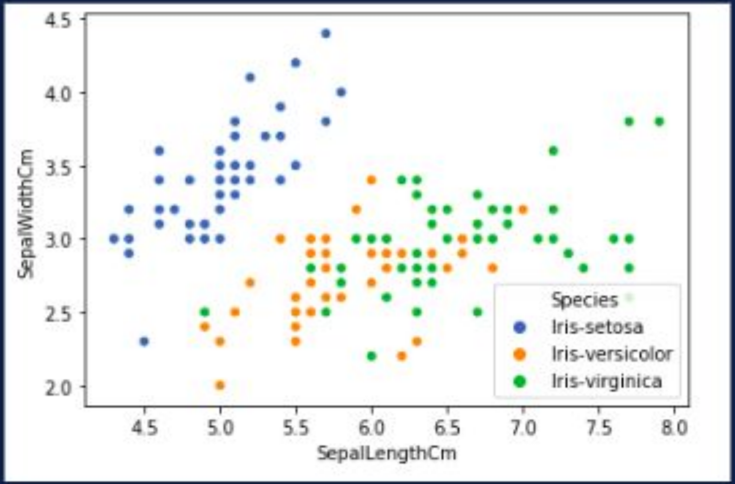
```
import pandas as pd
import matplotlib.pyplot as plt

iris = pd.read_csv('iris.csv')
plt.scatter(x = 'SepalLengthCm', y = 'SepalWidthCm', data = iris)
plt.title('A Nice Iris Dataset Graphic')
plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')
```



https://www.inwt-statistics.com/blog/data-visualization-r-versus-python

Python: Seaborn

```
import seaborn as sns
sns.scatterplot(x = 'SepalLengthCm', y = 'SepalWidthCm', hue = 'Species', data = iris)
```



https://www.inwt-statistics.com/blog/data-visualization-r-versus-python

Why R?

- Open-source, free and widely used in academia and research
- Powerful for Statistical Analysis, Visualization and modeling
- Large community

R Communities

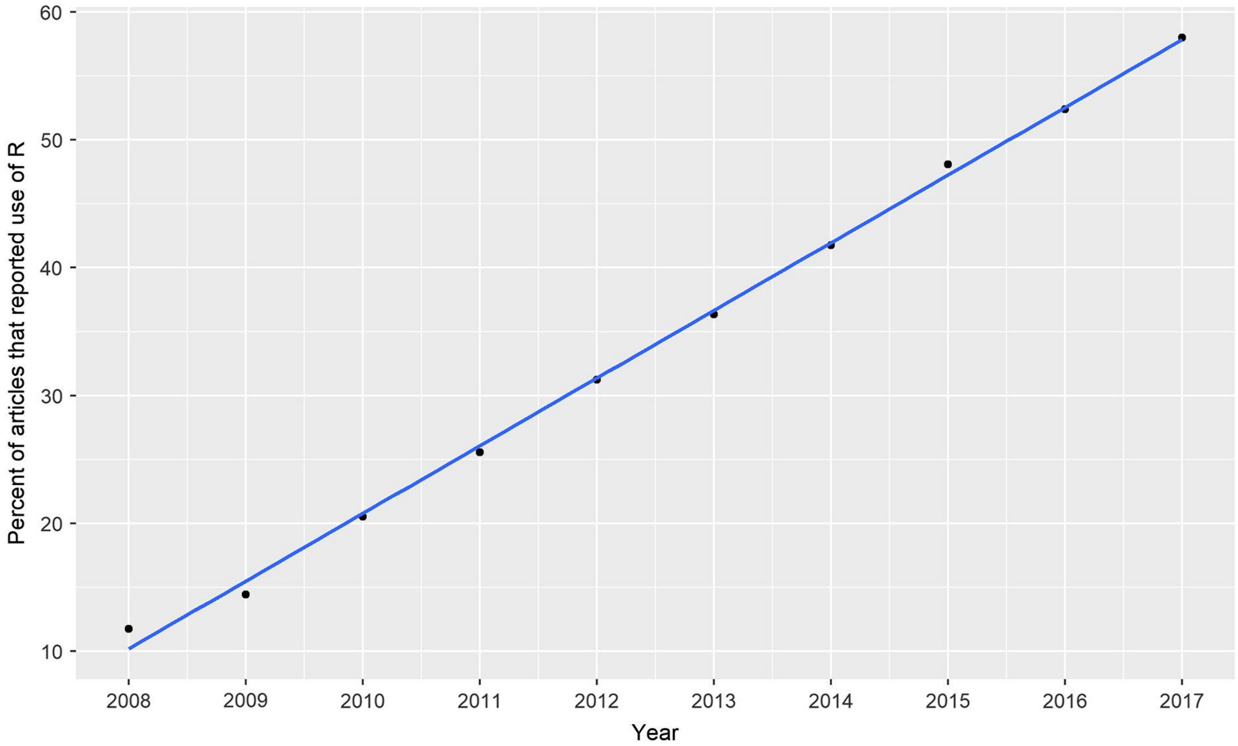
Why R?

- Open-source, free and widely used in academia and research
- Powerful for Statistical Analysis, Visualization and Modeling
- Large community
- Comes with tools for specific use-cases and extensive package ecosystem (CRAN):
  - Epidemiology: epitools, EpiEstim
  - Dynamical models: deSolve

Why R for Epidemiological and Ecological modeling?

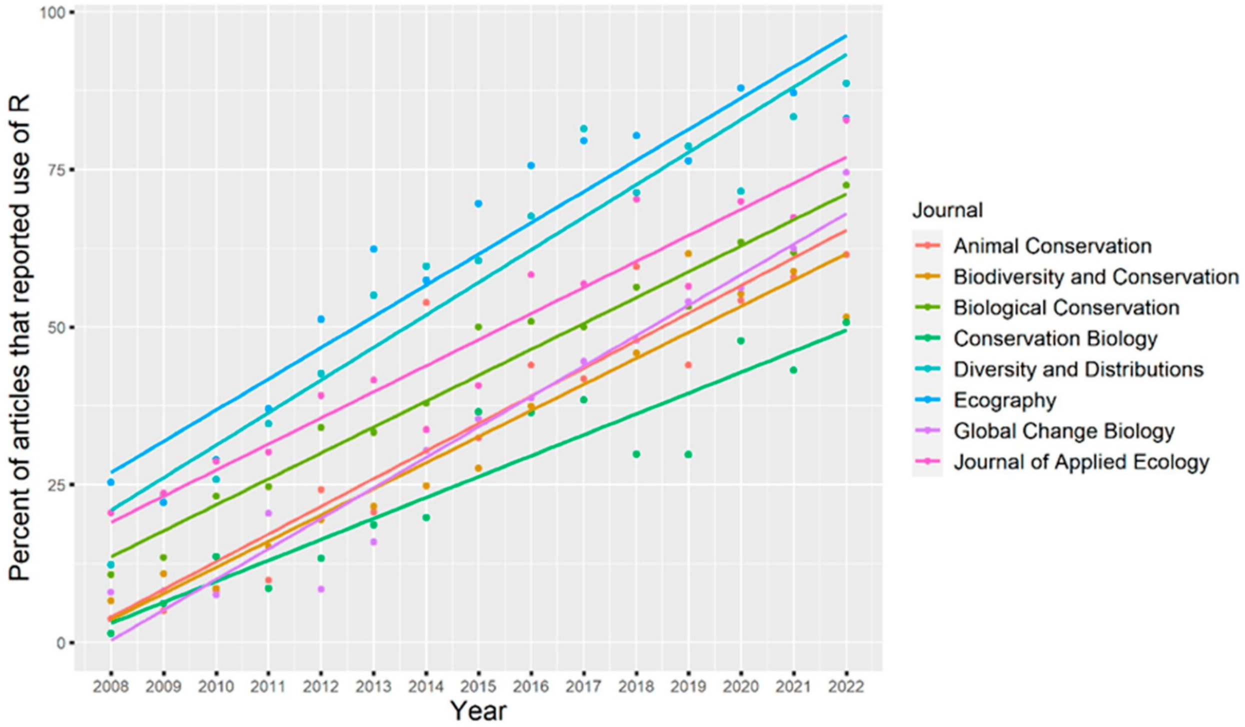
- **Data Handling:** works with case-data, time-series, and spatial datasets (tidyverse, dplyr, sf)
- **Statistical Analysis:** regression, hypothesis testing, GLMs with **stats**
- **Dynamical Models: deSolve** for epidemic models (SIR, SEIR) and ecological models (prey_predator, population growth)

Popularity of R in Ecology



Source: Lai, J., C. J. Lortie, R. A. Muenchen, J. Yang, and K. Ma. 2019. "Evaluating the Popularity of R in Ecology." *Ecosphere* 10: e02567.
https://doi.org/10.1002/ecs2.2567

Popularity of R in Ecology

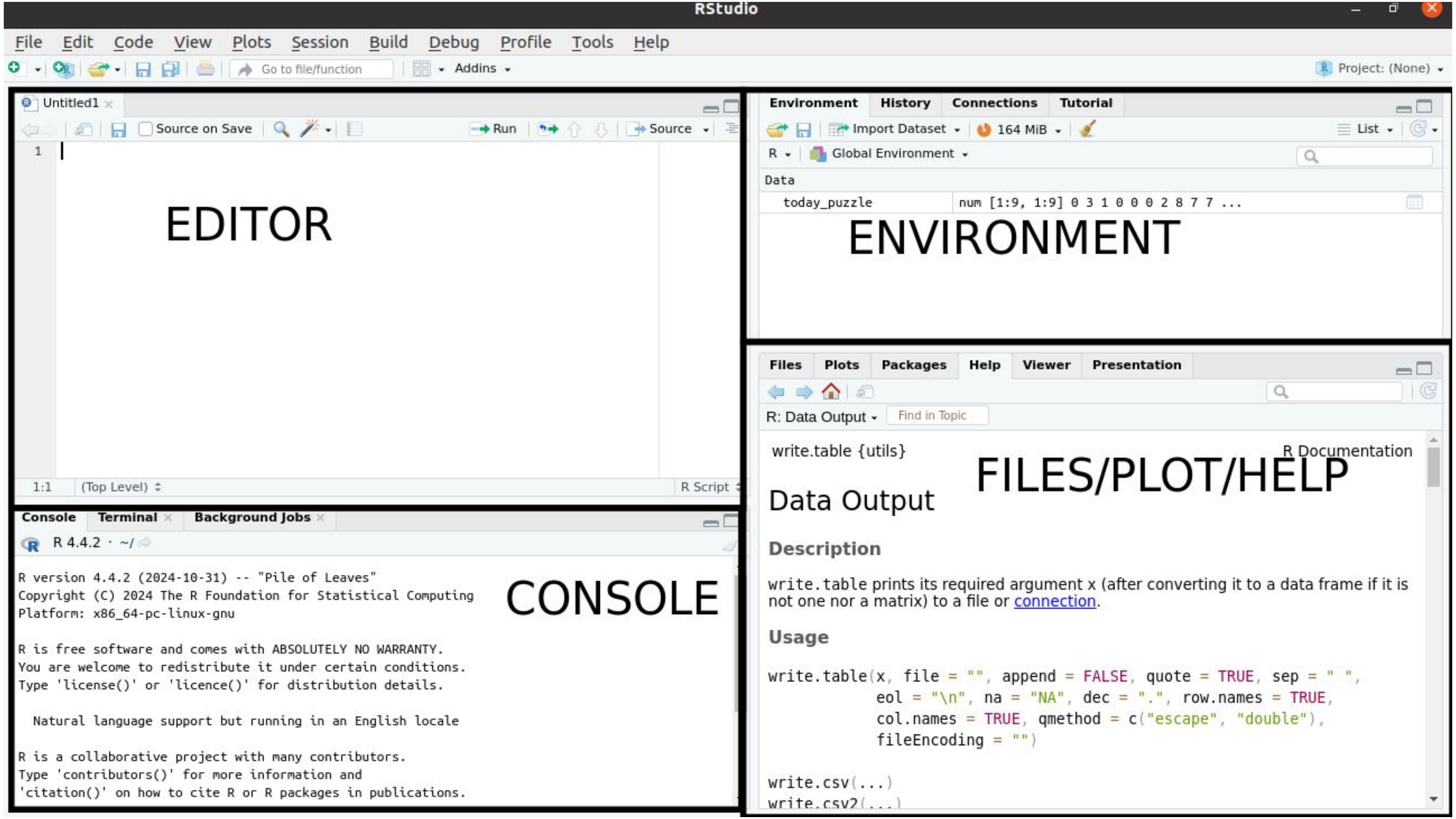


Source: Lai, J.; Cui, D.; Zhu, W.; Mao, L. The Use of R and R Packages in Biodiversity Conservation Research. *Diversity* **2023**, *15*, 1202. https://doi.org/10.3390/d15121202

Overview of R Interface

```
R version 4.4.2 (2024-10-31) -- "Pile of Leaves"
Copyright (C) 2024 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Previously saved workspace restored]

> 
```

Overview of RStudio Interface



Installing and Loading Packages

- From CRAN: install.packages(“Name Of the Package”) e.g:
  - install.packages(“stats”)
  - exercise: install the following packages: **tidyverse**, **stats**, **deSolve**
- Load the package with: library(Name of the Package) e.g:
  - library(tidyverse)
- Using devtools and github:
  - First, install the package devtools
  - load devtools
  - type devtools::install_github(“Repository”)
  - This won’t be covered here!

R basics: Variables, Data Types and Vectors

R Syntax and Basic Operations

- Assigning values:
  - `<-` is the common way to assign values to variables in R, = is also a valid way, though it is more used for arguments in functions

```r
PA_population_size <- 123486
PA_mean_population_age <- 13
read.csv(file = "path/to/csv", header = TRUE, sep = ";")
```

- Variables' name should be **descriptive** and **cannot start with a number**
- rm(variable_name) to remove a variable from memory

Data Types in R

Type	Example	Description
Numeric	pi <- 3.1415	Numbers (integer or decimal)
Integer	age <- 42L	Whole numbers (L specifies integer)
Character	name <- “John”	Text data (strings)
Logical	Infected <- TRUE	Boolean (TRUE or FALSE)
Factor	factor(c("low", "medium", "high"))	Categorical data

Arithmetic Operations in R

Operator	Description	Example
+	Addition	2 + 3 # 5
-	Subtraction	2 - 3 # -1
*****	Multiplication	2 * 3 # 6
/	Division	4 / 2 # 2
^ or **	Exponentiation	2^3 or 2**3 # 8

Logical Operations in R

<table>
<tr><th>Operator</th><th>Description</th><th>Example</th></tr>
<tr><td>==</td><td>Equal to</td><td>3 == 3 # TRUE</td></tr>
<tr><td>!=</td><td>Not equal to</td><td>2 != 3 # TRUE</td></tr>
<tr><td>></td><td>Greater than</td><td>3 > 2 # TRUE</td></tr>
<tr><td>>=</td><td>Greater than or equal to</td><td>2 >= 3 # FALSE</td></tr>
<tr><td><</td><td>Less than</td><td>2 < 3 # TRUE</td></tr>
<tr><td><=</td><td>Less than or equal to</td><td>3 <= 2 # FALSE</td></tr>
<tr><td>&</td><td>AND</td><td>TRUE & FALSE # FALSE</td></tr>
<tr><td>|</td><td>OR</td><td>TRUE | FALSE # TRUE</td></tr>
<tr><td>!</td><td>NOT</td><td>!TRUE # FALSE</td></tr>
</table>

Live Coding: Practice R basics

- Assign Variables
  - Create a variable `my_age` and assign it your age.
  - Create a variable `height_cm` for your height in centimeters.
- Work with Data Types
  - Create a character variable `my_name` **with your first name.**
  - Create a logical variable `is_student` (TRUE/FALSE).
- Perform Basic Calculations
  - Compute your age in **10 years** (`my_age + 10`).
  - Convert height from cm to meters (`height_cm / 100`).
- Test Logical Comparisons
  - Check if your age is greater than 18.
  - Compare two numbers and check if they are equal.

Vectors

- What is a Vector?
    - A **basic data structure** in R, used to store multiple values of the **same data type**.
    - Examples: Numeric, Character, Logical.
- .Why use Vectors?
    - Efficient way to store and manipulate data.
    - Foundation for **data frames and matrices**.
    - Supports **vectorized operations** (faster than loops).

```
ages <- c(25, 30, 35, 40)  # Numeric vector
first_names <- c("Alice", "Bob", "Charlie")  # Character vector
students <- c(TRUE, FALSE, TRUE)  # Logical vector
```

Vectors

- How to create Vectors?
  - `c()` function (c stands for combine value
  - Sequences using `:` or `seq()`.
  - Repeating elements using `rep()`.

```r
x <- c(1, 2, 3, 4, 5)  # Basic vector
y <- 1:10  # Sequence from 1 to 10
z <- seq(1, 10, by=2)  # Sequence with step size 2
w <- rep(5, times=3)  # Repeat 5 three times
```

- Accessing Vector Elements
  - Use **indexing** with `[ ]` (R is **1-based**).
  - Negative indexing removes elements.

```r
ages[2]  # Second element
ages[-1]  # All elements except the first
ages[1:3]  # First three elements
```

Vector Operations in R

Operator	Description	Example
+	Addition	x+2 # adds 2 to all element
-	Subtraction	x-2 # subtract 2 to all element
*****	Multiplication	x* 2 # multiplies each element by 2
/	Division	x/ 3 # divides each element by 2

Vector Operations in R

- R applies operations **directly to all elements** (vectorized).

```
x <- c(10, 20, 30)
x + 5  # Adds 5 to each element
x * 2  # Multiplies each element by 2
y <- c(2, 4, 6)
x + y  # Element-wise addition
```

- Applying functions to vectors: **sum()**, **mean()**, **length()**, **min()**, **max()**

```
sum(x)  # Total sum
mean(x)  # Average value
length(x)  # Number of elements
```

Logical Indexing & Filtering of Vectors

- Filtering with Logical Conditions
  - Use logical comparisons to **select elements**.
  - Can be combined with `which()` or `subset()`.

```
ages <- c(20, 25, 30, 35, 40)
ages > 30  # Returns TRUE/FALSE
ages[ages > 30]  # Select values greater than 30
which(ages > 30)  # Returns indices where condition is TRUE
```

- Handling Missing Values (`NA`)
  - Use `is.na()` to check for missing value
  - Remove `NA` values using `na.omit()`.

```
x <- c(10, NA, 30, 40)
is.na(x)  # Checks for missing values
x[!is.na(x)]  # Keeps only non-missing values
```

Live Coding – Practice with Vectors

1. Create Vectors

- Create a numeric vector `temps` with values: `30, 32, 28, 25, 29`.
- Create a character vector `cities` with names of 3 cities.

2. Perform Operations

- Find the mean of `temps`.
- Multiply all elements of `temps` by 1.8 and add 32 (convert to Fahrenheit).

3. Filtering Data

- Select temperatures greater than 28.
- Use `which()` to find the index of the **coldest** temperature.

Data Frames in R

What is a Data Frame?

1. Definition

- A data frame is a **table-like structure** in R.
- Each **column** is a vector; each **row** is an observation.
- Can contain different data types (numeric, character, logical).

2. Why is it important?

- Standard way to store and analyze tabular data.
- Used in modeling, plotting, and statistical analysis.

```
data.frame(
  name = c("Alice", "Bob"),
  age = c(25, 30),
  infected = c(TRUE, FALSE)
)
```

Creating Data Frames from Vectors

From Scratch

- Use `data.frame()` to combine vectors into a table.
- Vectors must be **equal length**.

```
names <- c("Alice", "Bob", "Charlie")
ages <- c(22, 30, 28)
infected <- c(TRUE, FALSE, TRUE)

df <- data.frame(name = names, age = ages, infected = infected)
```

Importing Data (CSV & Excel)

From CSV Files

- Use `read.csv("filename.csv")`

```
data <- read.csv("cases.csv")
```

From Excel Files

- Use `readxl` package
- First: `install.packages("readxl"), then load it library(readxl)`
- Then: `read_excel("file.xlsx")`

```
library(readxl)
data <- read_excel("cases.xlsx")
```

Basic Data Frame Manipulation

Accessing elements

- Use `$`, column name, or `df[row, column]` syntax.

```
df$age
df[1, "name"]
```

Filtering Rows

```
df[df$age > 25, ]  # Select rows where age > 25
```

Selecting Columns

```
df[, c("name", "age")]
df$name
```

Basic Data Frame Manipulation

Adding/Modifying Columns

```
df$risk_group <- df$age > 60
```

Sorting Data

```
df[order(df$age), ]
```

Exploring and Summarizing Data Frames

Understand the Data

- `head(df)` : First rows
- `str(df)`: Structure
- `summary(df)`: Summary stats
- `dim(df): dimension` or `nrow(df): number of rows`, `ncol(df): number of columns`

Renaming columns

```
names(df) <- c("Name", "Age", "InfectionStatus")
```

Best Practices for Entering Field Data in Excel/Notebooks

Column Naming

- Use **short, meaningful, lowercase names**:
  - ✅ `date`, `location`, `species`, `count`, `temp_c`
  - ❌ `Date of Observation`, `# of Birds`, `Temp.`
- Avoid spaces or special characters → use `_` or camelCase

Data Consistency

- Use the **same format** in each column:
  - Dates: `YYYY-MM-DD` (e.g., `2024-02-28`)
  - Text: consistent spelling and case (`Forest`, not `forest`, `FORest`)
  - Numbers: no commas or text (e.g., `1000`, not `1,000` or `"1000 cases"`)

Best Practices for Entering Field Data in Excel/Notebooks

One Observation per Row

- Each row = one observation or measurement
- Don’t merge cells or use multiple headers

File Format

- Save as **CSV** or **XLSX**
- Example filename: `species_observations_2024.csv`
- Avoid non-English characters in filenames and column names

	A	B	C	D
1	**date**	**site**	**species**	**count**
2	2024-02-01	Andasibe	lemur	3
3	2024-02-01	Ranomafana	chameleon	1
4	2024-02-01	Isalo	chameleon	2
5				

Live Coding – Practice with Data Frames

1. Create a data frame

- Use vectors for `country`, `cases`, `deaths`, and `recovered`.

2. Explore and filter the data

- Find rows where `cases > 1000`
- Add a column `fatality_rate <- deaths / cases`

3. Filtering Data

- Select temperatures greater than 28.
- Use `which()` to find the index of the **coldest** temperature.

Tidy Data Manipulation with dplyr

What is dplyr ?

1. Definition

- Part of the `tidyverse`: tools for clean data analysis.
- Works with **data frames** and **tibbles**.
- Makes data manipulation **faster** and **easier to read**.

2. Core idea

- Use **verbs**: `filter()`, `select()`, `mutate()`, `arrange()`, `summarize()`, `group_by()`.
- Combine them with `%>%` (pipe operator) to chain operations.

`filter()` – Keep Rows That Match a Condition

1. Syntax

```
filter(data, condition)
```

2. Examples

```
filter(df, ages >= 30)
# Equivalent to
df %>% filter(ages >= 30)
```

`select()` – Pick Specific Columns

1. Syntax

```
select(data, column1, column2)
```

2. Examples

```
select(df, first_names,ages)
# Equivalent to
df %>% select(first_names, ages)
# To select every column except one:
select(df, -column)
```

`mutate()` – Add or Modify Columns

1. Syntax

```
mutate(data, new_column = calculation)
```

2. Examples

```
mutate(df, age_in_months = age * 12)
# Equivalent to
df %>% mutate(age_in_months = age * 12)
```

`arrange()` – Sort Rows

1. Syntax

```
arrange(data, column)
arrange(data, desc(column))
```

2. Examples

```
arrange(df, age)
arrange(df, desc(age))
```

`summarize()` and `group_by()` – Summary Statistics

1. Group then summarize

```
group_by(data, column) %>% summarize(mean_age = mean(age))
```

2. Example

```
df %>%
  group_by(infected) %>%
  summarize(avg_age = mean(age))
```

Live Coding – Practice with dplyr

1. **Filter**: Extract all patients who are still in the hospital (i.e., those with `NA` in `date_of_removal`).
2. **Select**: Extract the columns `patient_id`, `unit`, and `status` for patients in the `ICU`.
3. **Create**: Add a new column `clinical_risk`, where:

    - `“High”` if the patient is in `Reanimation` or has `ECMO` in the `machines`.
    - `“Moderate”` if the patient is in **ICU** or **General** and does **not** have ECMO
    - `"Unknown"` otherwise.

4. **Arrange**: Sort by `age` in ascending order to see younger patients first.
5. **Group by**: Group by `unit` and summarize the number of `Critical` patients in each unit.
6. **Count**: Count the number of patients in each `unit`.

Introduction to Data Visualization with ggplot2

Why Data Visualization ?

1. **install.packages("datasauRus")**

2. **Run the following commands**

- `library(ggplot2)`
- `library(datasauRus)`

```
# Load the data
data("datasaurus_dozen")

# Summary statistics
summaries <- datasaurus_dozen %>%
  group_by(dataset) %>%
  summarize(
    n_points = n(),
    mean_x = mean(x), sd_x = sd(x), min_x = min(x), max_x = max(x), IQR_x = IQR(x),
    mean_y = mean(y), sd_y = sd(y), min_y = min(y), max_y = max(y), IQR_y = IQR(y)
  )

# View all results
print(summaries, n = Inf)
```

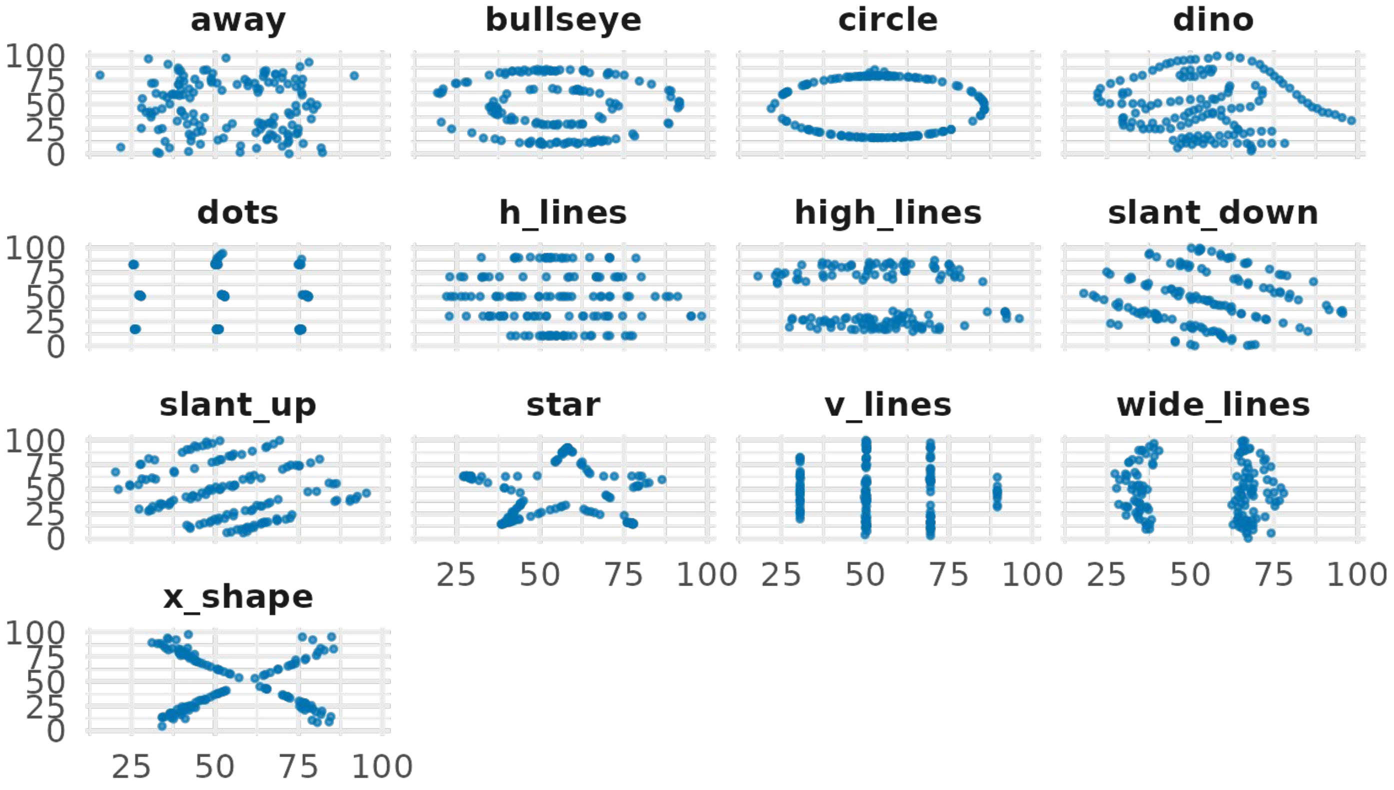
Why Data Visualization ?

1. Run the following commands

```
# Plot all datasets
datasaurus_plot <- ggplot(datasaurus_dozen, aes(x = x, y = y)) +
  geom_point(color = "#0072B2", size = 0.5, alpha = 0.7) +
  theme_minimal() +
  facet_wrap(~ dataset, ncol = 4) +
  labs(
    title = "Same Stats, Different Stories",
    subtitle = "Each dataset has nearly identical means, SDs\n, and correlations",
    caption = "Source: Datasaurus Dozen by Alberto Cairo & Justin Matejka"
  ) +
  theme(
    plot.title = element_text(size = 16, face = "bold"),
    plot.subtitle = element_text(size = 12),
    strip.text = element_text(face = "bold"),
    axis.title = element_blank()
  )
datasaurus_plot
```

Same Stats, Different Stories

Each dataset has nearly identical means, SDs
, and correlations



Source: Datasaurus Dozen by Alberto Cairo & Justin Matejka

Why Data Visualization ?

- Summary statistics alone can be misleading
- Visual exploration helps detect trends, outliers, and patterns.
- Essential for hypothesis generation and communicating findings.
- ggplot2 is the standard for high-quality plots in R.

The Grammar of Graphics

- **gg**plot2: **g**rammar of **g**raphics (**layered grammar**)
    - **Data**: the dataset
    - **Aesthetics (`aes`)**: mapping variables to visual properties
    - **Geometries (`geom_`)**: the shape of the plot (points, bars, lines, etc.)
- Plots are built by **adding layers**
- with `+`.

Basic Syntax

- General syntax

```
ggplot(data = <DATA>, aes(x = <X>, y = <Y>)) +
  geom_<TYPE>()
```

- Example:

```
admission_plot <- ggplot(number_of_admission_by_day, aes(x = date_of_admission, y = count)) +
  geom_point(color = "#0072B2", size = 1.2, alpha = 0.7) +
  labs(
    title = "Daily Hospital Admissions During Outbreak",
    x = "Date of Admission",
    y = "Number of Admissions"
  ) +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```

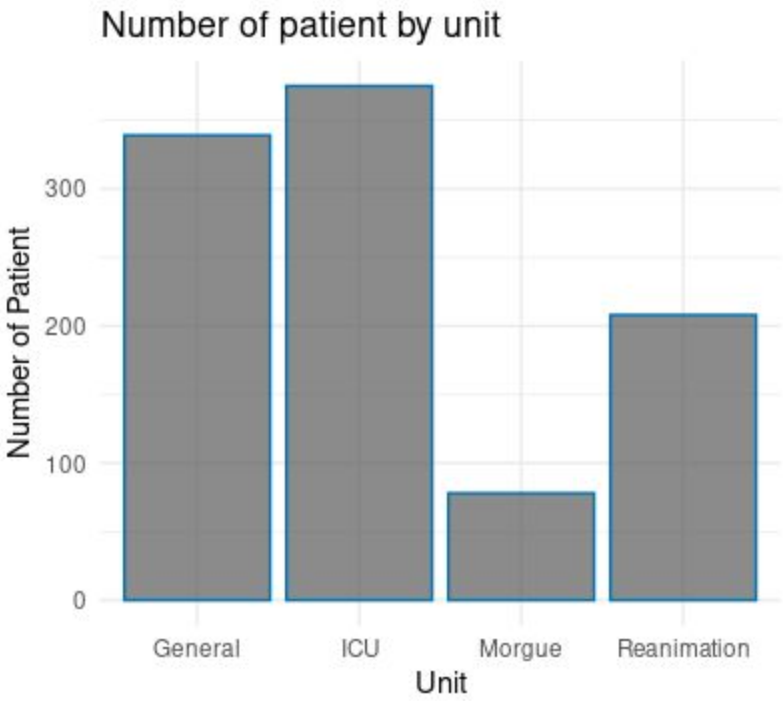
Plot Types

- Bar Plot:

```r
unit_plot <- ggplot(hospital_data, aes(x = unit)) +
  geom_bar(color = "#0072B2", width = 0.9, alpha = 0.7, stat= "count") +
  labs(
    title = "Number of patient by unit",
    x = "Unit",
    y = "Number of Patient"
  ) +
  theme_minimal()

unit_plot
```

Plot Types

- Bar Plot:



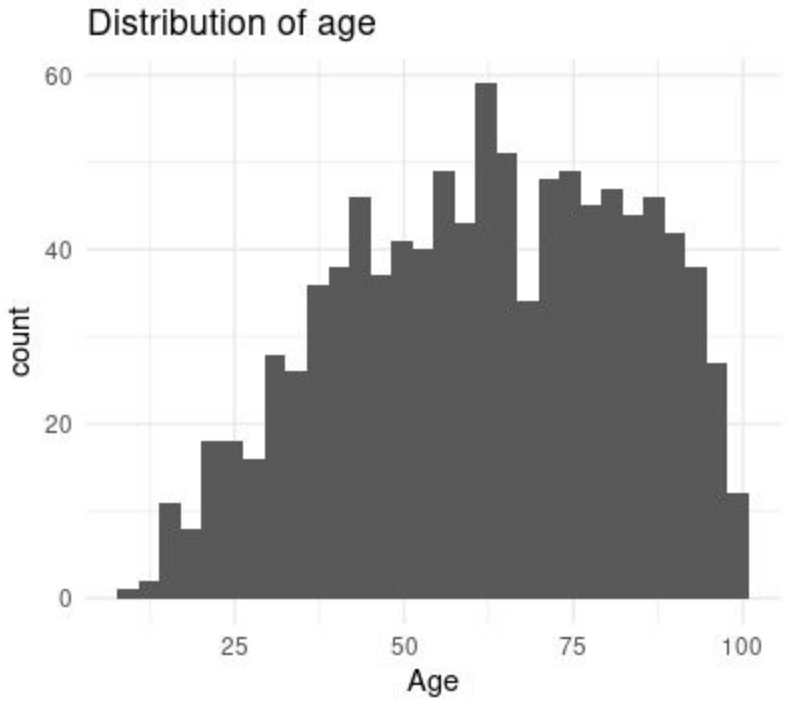
Plot Types

- Histogram:

```
# Histogram
age_plot <- ggplot(hospital_data, aes(x = age)) +
  geom_histogram() +
  labs(
    title = "Distribution of age",
    x = "Age"
  ) +
  theme_minimal()

age_plot
```

Plot Types

- Histogram:



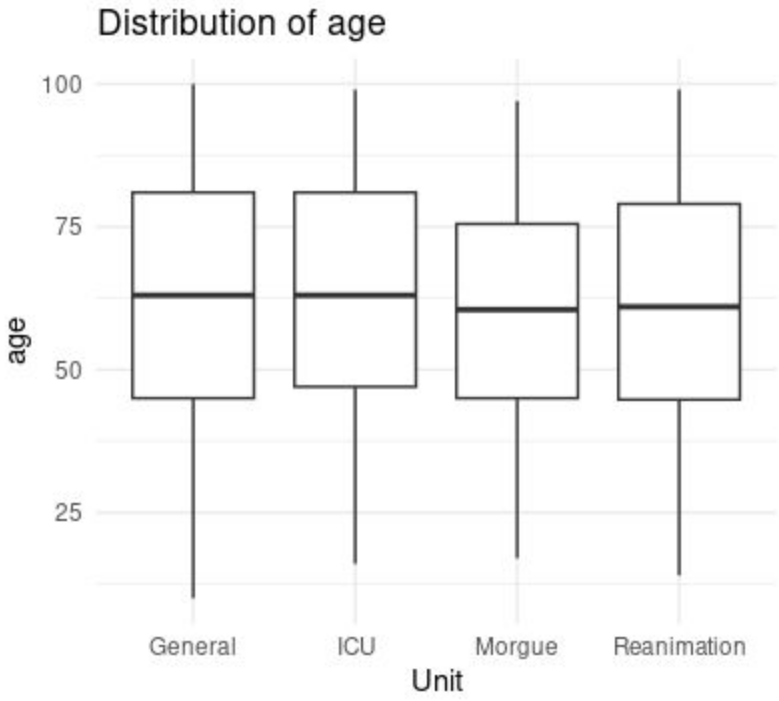
Plot Types

- Boxplot:

```
# Boxplot
ageBox_plot <- ggplot(hospital_data, aes(x = unit, y=age)) +
  geom_boxplot() +
  labs(
    title = "Distribution of age",
    x = "Unit"
  ) +
  theme_minimal()

ageBox_plot
```

Plot Types

- Boxplot:



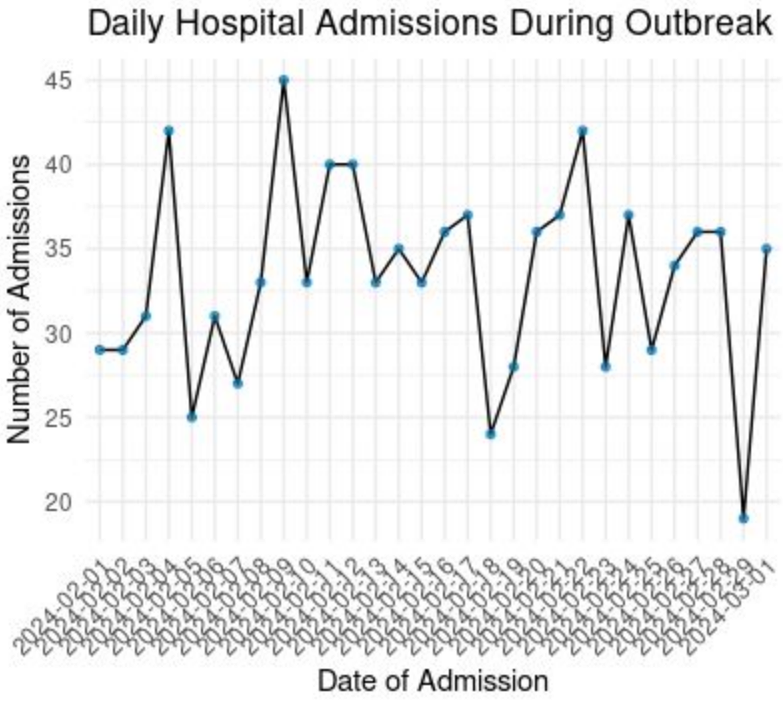
Plot Types

- Line Plot:

```r
#Line plot
admission_plot_line <- ggplot(number_of_admission_by_day, aes(x = date_of_admission, y = count)) +
  geom_line(aes(group = 1)) +
  geom_point(color = "#0072B2", size = 1.2, alpha = 0.7) +
  labs(
    title = "Daily Hospital Admissions During Outbreak",
    x = "Date of Admission",
    y = "Number of Admissions"
  ) +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

admission_plot_line
```

Plot Types

- Line Plot:



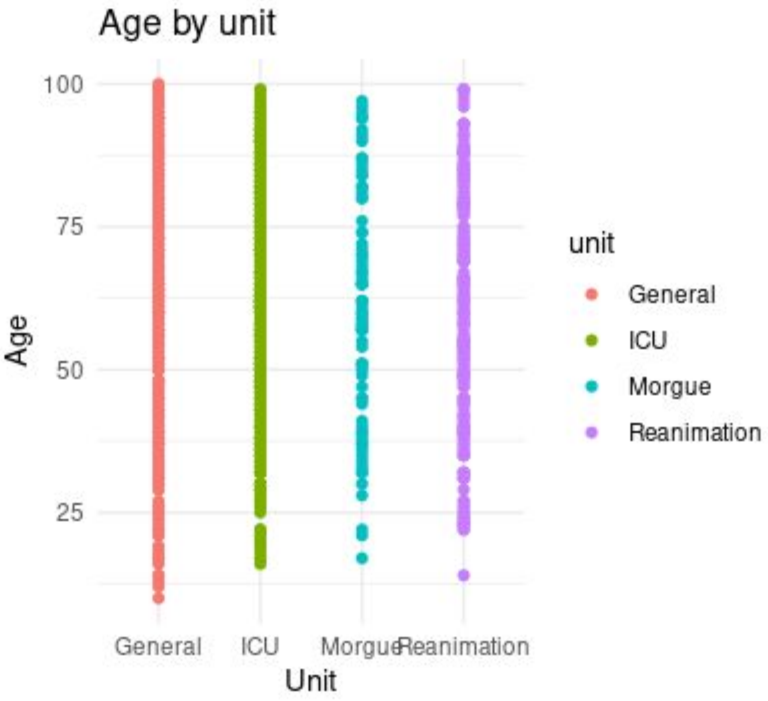
Aesthetics and Customization

- Mapping color/shape/size to variables

```r
# Color mapping
ggplot(hospital_data, aes(x = unit, y = age, color = unit)) +
  geom_point()
```

- Titles, labels, and themes:

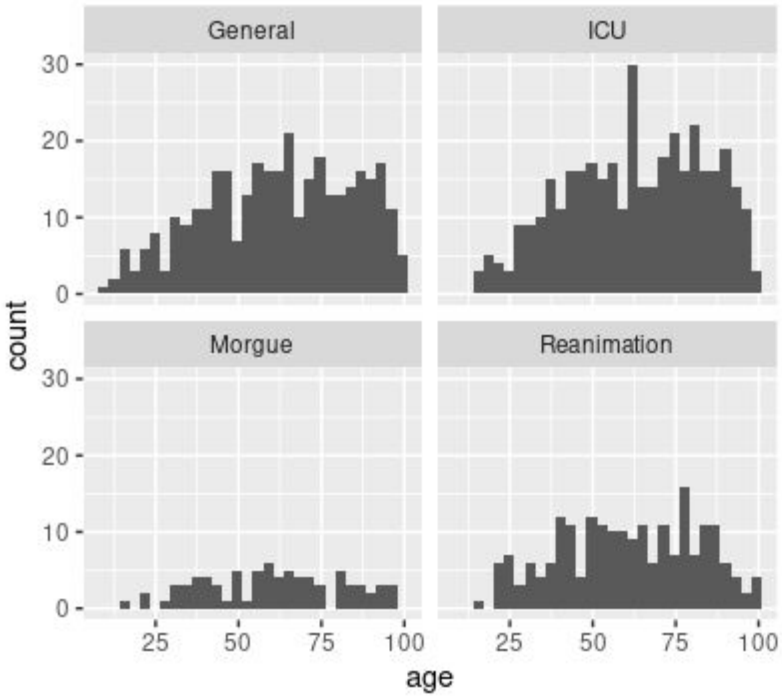
```r
labs(
  title = "Age by unit",
  x = "Unit",
  y = "Age"
) +
theme_minimal()
```



Faceting (for subgroup plots)

- code:

```
# Faceting
ggplot(hospital_data, aes(x = age)) +
  geom_histogram() +
  facet_wrap(~ unit)
```



Live Coding – Data Visualization with ggplot2

- Create a **bar plot** of patient count by unit.
- Show a **histogram** of age distribution.
- Create a **boxplot** of age by unit.
- Use **color** to show patient status in scatterplot of age vs patient_id.
- Facet by `status` or `unit`.

Introduction to Statistical Modeling with stats

What is statistical modeling ?

1. Definition

- Formally: the mathematical relationship between random and non-random variables.
- the different mathematical methodologies data analysts and data scientists use to interpret data
- Helps make predictions, test hypotheses, or understand patterns.
- There are a variety of statistical models, and the one you apply to a data set will depend on the question you're attempting to answer.

2. Core idea

- **Model = Equation + Assumptions**: We describe how a response variable depends on one or more predictors.
- **Fit to Data**: Use observed data to estimate model parameters (e.g., slope, intercept).
- **Assess Validity**: Evaluate how well the model explains the data (e.g., using p-values, R^2, residuals).

Types of models

Linear Models (LM)

Use when the response variable is continuous and there is a linear relationship

→ `lm(y ~ x1 + x2, data = ...)`

Generalized Linear Models (GLM)

Extend linear models for other types of outcomes

→ `glm(y ~ x, family = binomial, data = ...)`

- *Binomial*: for binary outcomes (e.g., ICU or not)
- *Poisson*: for counts (e.g., number of cases)

Nonlinear Models (NLS)

For explicitly nonlinear relationships

→ `nls(y ~ a * exp(b * x), data = ...)`

Linear Regression

```r
# Loading data
hospital_data <- read.csv(file = "simulated_hospital_dataset.csv", sep = ",")
head(hospital_data)

# Count the number of symptoms
hospital_data$number_symptoms <- sapply(strsplit(hospital_data$symptoms, ";"), length)

# Compute the stay period
hospital_data$stay_length <- as.numeric(as.Date(hospital_data$date_of_removal) - as.Date(hospital_data$date_of_admission))

# Linear Regression model using age and number of symptoms as predictors
lm_model <- lm(stay_length ~ age + number_symptoms, data = hospital_data)
summary(lm_model)
```

Logistic Regression

```r
# Binary classification on ICU admission
hospital_data$in_icu <- ifelse(hospital_data$unit == "ICU", 1, 0)
# Step 1: Split each symptom string into a list
symptom_lists <- strsplit(hospital_data$symptoms, ";\\s*")
# Step 2: Flatten the list and get unique symptom names
all_symptoms <- unique(unlist(symptom_lists))
# Step 3: Create dummy variables for each symptom dynamically
for (symptom in all_symptoms) {
  hospital_data[[paste0("has_", symptom)]] <- sapply(symptom_lists, function(sym_list) symptom %in% sym_list)
}
glm_model <- glm(in_icu ~ age + number_symptoms + has_cough + has_fever + has_diarrhea, data = hospital_data, family = "binomial")
summary(glm_model)
exp(coef(glm_model))
install.packages("margins")
library(margins)
margins(glm_model)
```

Live Coding – Statistical Modeling

- Fit a linear model to explore how stay length depends on age or ICU status.
- Fit a logistic model predicting ICU admission from age and the fever symptom.
- Interpret coefficients: What increases the chance of ICU admission?

Introduction to Dynamical Modeling with deSolve

What is dynamical modeling ?

- Describes how a system evolves **over time**.
- Based on **differential equations** that track rates of change (e.g., infections, recoveries).
- Essential for modeling epidemics and ecosystems with **time-varying behavior**.

Why `deSolve` in R?

- `deSolve` lets you **solve differential equations numerically** in R.
- Widely used in **epidemiology (SIR, SEIR models)** and **ecological systems (population models)**.
- Integrates smoothly with tidy workflows for simulation and plotting.

Why `deSolve` in R?

- **Initial values**: state of the system at time = 0

  e.g., `S = 999, I = 1, R = 0`
- **Parameters**: fixed model inputs (e.g., infection rate β)
- **Time sequence**: when to solve (e.g., `times = seq(0, 100, by = 1)`)
- **ODE function**: returns derivatives `dS/dt`, `dI/dt`, `dR/dt`

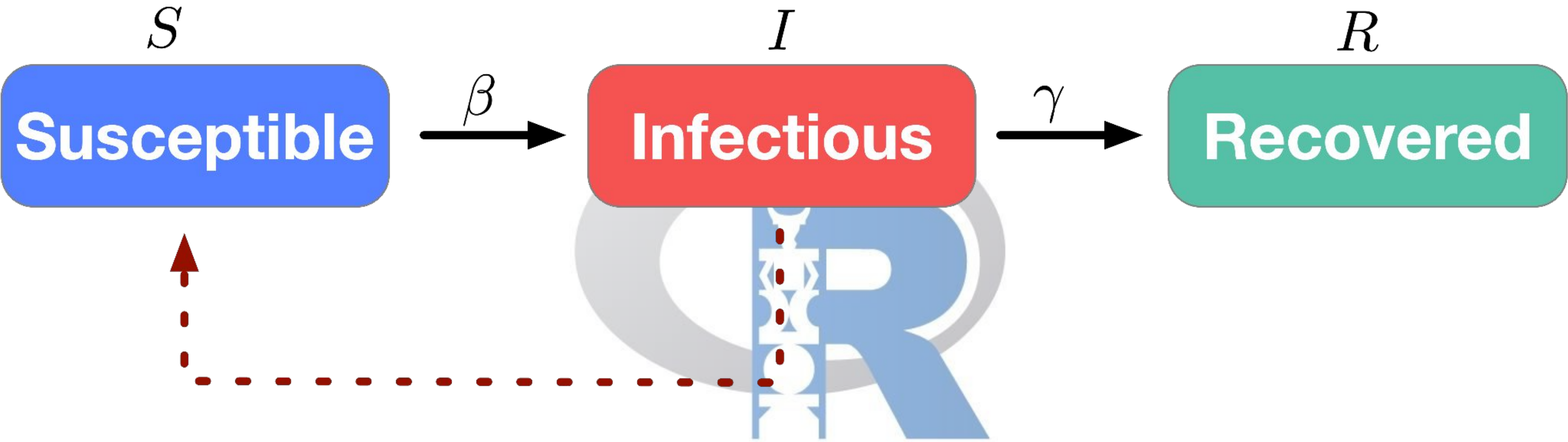
Example - SIR model

- **S**: Susceptible, i.e., people who can be infected.
- **I**: Infected , i.e., currently infected people.
- **R**: Recovered, i.e., people who recovered or died and no longer spread disease.
- time.

Equations:

$$\frac{dS}{dt} = -\beta \frac{SI}{N}, \quad \frac{dI}{dt} = \beta \frac{SI}{N} - \gamma I, \quad \frac{dR}{dt} = \gamma I$$

Example - SIR model



Source: https://covid19.uclaml.org/model.html

Example - SIR model

```r
# SIR model definition
sir_model <- function(time, state, parameters) {
  with(as.list(c(state, parameters)), {
    N <- S + I + R
    dS <- -beta * S * I / N
    dI <- beta * S * I / N - gamma * I
    dR <- gamma * I
    return(list(c(dS, dI, dR)))
  })
}
```

```r
# Parameters and initial state definition
params <- c(beta = 0.3, gamma = 0.1)
state <- c(S = 999, I = 1, R = 0)

# Time span for simulation:
times <- seq(0, 100, by = 1)

# Solving the model
output <- ode(y = state, times = times, func = sir_model, parms = params)
```

Example - SIR model

Explanation:

- You define a function that returns the *rate of change* for S, I, and R.
- `with(as.list(...))`: unpacks the values inside `state` and `parameters`.
- Calculates each derivative using the model equations.
- beta: infection rate (**how fast people get infected**)
- gamma: recovery rate (**how fast people recover**)
- state: initial number of Susceptible, Infected, and Recovered

Example - SIR model

Explanation:

`ode()` is the core function from `deSolve`.

It takes:

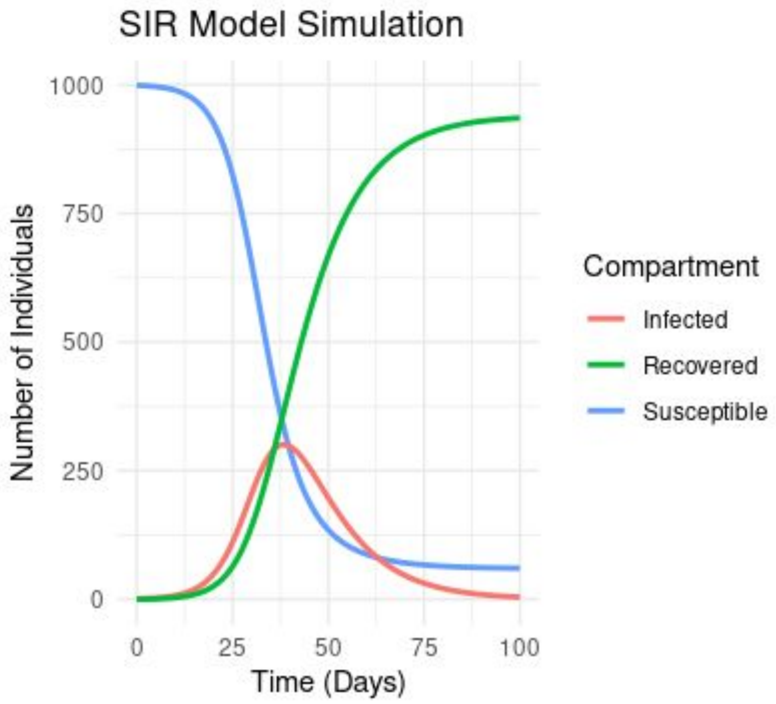
- `y`: initial state,
- `times`: time sequence,
- `func`: the model you defined,
- `parms`: parameters like `beta` and `gamma`.

It returns a time series of `S`, `I`, and `R`.

Plotting - SIR model

```r
# Converting output to a Dataframe
output_df <- as.data.frame(output)

#Plotting
library(ggplot2)

ggplot(output_df, aes(x = time)) +
  geom_line(aes(y = S, color = "Susceptible"), size = 1) +
  geom_line(aes(y = I, color = "Infected"), size = 1) +
  geom_line(aes(y = R, color = "Recovered"), size = 1) +
  labs(title = "SIR Model Simulation",
       x = "Time (Days)",
       y = "Number of Individuals",
       color = "Compartment") +
  theme_minimal()
```

Plotting - SIR model



What You Should Understand

- How to define a system with **initial values and parameters**.
- How to implement the **ODEs** in R using `deSolve`.
- How to **run the simulation** and **plot the results**.

🎓 R Bootcamp: Key Takeaways

✅ **R Foundations**: Navigated RStudio, mastered R syntax, and explored vectors, data frames, and tidy data principles.

📊 **Data Manipulation**: Used `dplyr` to clean, transform, and summarize data.

📈 **Visualization with ggplot2**: Learned how to tell compelling stories with data through layered, customizable plots.

📉 **Statistical Modeling**: Built and interpreted basic models (linear, logistic) using the `stats` package.

🔁 **Dynamical Modeling**: Simulated epidemic processes using compartmental models with `deSolve`.

🧠 **Hands-On Practice**: Worked with realistic hospital datasets and outbreak scenarios to reinforce each concept.

🚀 **Workshop Ready**: Equipped with all essentials to engage in statistical and mechanistic modeling in R.

Thank you, all! You should now be ready for E2M2!