MODULE 3: PROGRAMMING FOR BIOLOGICAL DATA

Session 2: The Data Frame & Digital Sample Management

Moving from Single Tubes (Vectors) to Master Worksheets

Instructor: Dr. Gu Haogao

January 7, 2026

Recap: The Vector (Single Tube)

- **Concept:** A sequence of data points of the *same type*.
- **Great for:** Batch processing a single list of numbers (e.g., QC values, Ct scores).
- **The Flaw:** You cannot mix text (Patient ID) and numbers (Viral Load) in the same vector.

```
# Numeric Vector
vl <- c(1500, 0, 450000)

# Character Vector
ids <- c("P1", "P2", "P3")
```

The Danger of Loose Vectors

What happens if you sort one vector but forget to sort the others?

Sample Swap

Patient P1 gets P2's diagnosis.

Disconnected Data

Vectors have no inherent link to each other.

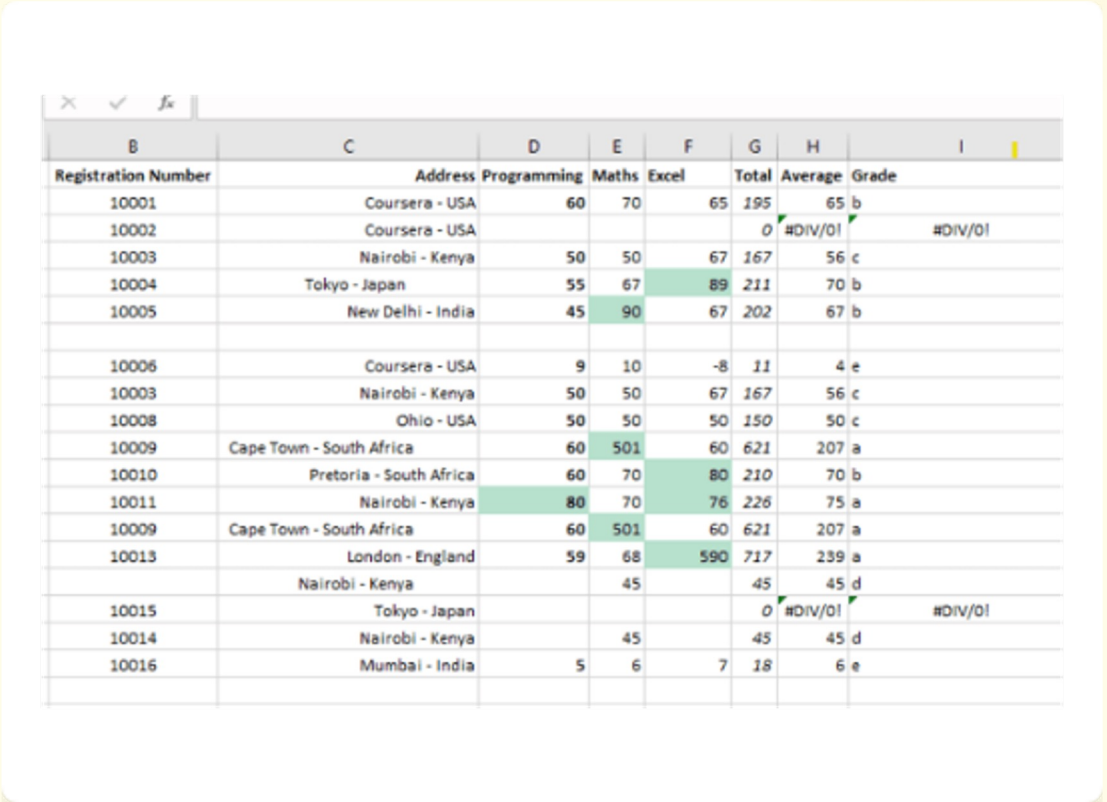
Clinical Error

High risk in diagnostics.

The Solution: The Data Frame

A **Data Frame** is the R equivalent of an Excel Worksheet or a Lab Bench Log.

- It glues vectors together.
- It enforces that rows stay together.
- It allows mixed data types (Text columns + Numeric columns).



B	C	D	E	F	G	H	I
Registration Number	Address	Programming	Maths	Excel	Total	Average	Grade
10001	Coursera - USA	60	70	65	195	65	b
10002	Coursera - USA				0	#DIV/0!	#DIV/0!
10003	Nairobi - Kenya	50	50	67	167	56	c
10004	Tokyo - Japan	55	67	89	211	70	b
10005	New Delhi - India	45	90	67	202	67	b
10006	Coursera - USA	9	10	-8	11	4	e
10003	Nairobi - Kenya	50	50	67	167	56	c
10008	Ohio - USA	50	50	50	150	50	c
10009	Cape Town - South Africa	60	501	60	621	207	a
10010	Pretoria - South Africa	60	70	80	210	70	b
10011	Nairobi - Kenya	80	70	76	226	75	a
10009	Cape Town - South Africa	60	501	60	621	207	a
10013	London - England	59	68	590	717	239	a
	Nairobi - Kenya		45		45	45	d
10015	Tokyo - Japan				0	#DIV/0!	#DIV/0!
10014	Nairobi - Kenya		45		45	45	d
10016	Mumbai - India	5	6	7	18	6	e

Anatomy of a Data Frame

↔Rows (Observations)

Each row represents a single **Patient** or **Sample**.

If you filter a row, all data for that patient moves together.

↕Columns (Variables)

Each column is a **Vector**.

Each column must contain only one data type (e.g., Age is numeric, Gender is text).

Visualizing the Structure

Patient_ID (Chr)	Age (Num)	Gender (Chr)	Viral_Load (Num)
P001	45	M	15000
P002	32	F	0
P003	58	F	450000

This looks like Excel, but it is a programmable object in R.

Concept 2: The "Read-Only" Principle

- **Excel Risk:** You open a file, accidentally delete a cell, and hit "Save". The original clinical data is corrupted forever.
- **R Safety:** You "Read" the file into R's memory. The original CSV on your hard drive is **never touched**.
- **Audit Trail:** Any changes happen in code, not on the original file.

Reading Data: Sample Reception

The read.csv() function is your sample intake window.

```
# Assign the data to a variable named 'lis_data'
lis_data <- read.csv("mock_patient_data.csv")

# Note: The file must be in your working directory
```

This creates a Data Frame object in R's environment.

Inspecting the Manifest

Always check your data immediately after loading.

head(data)

Shows the first 6 rows. Checks headers and formatting.

dim(data)

Shows dimensions (Rows x Columns). Did we receive all 100 samples?

The MRI Scan: str()

str() stands for Structure. It reveals the data types of every column.

```
str(lis_data)

# Output:
# $ Patient_ID : chr  "P001" "P002" ...
# $ Age        : int  45 32 ...
# $ Result     : num  15000 0 ...
```

Crucial Check: Ensure your numeric results are actually labeled as int or num, not chr.

Concept 3: The Coordinate System

To find data in a Data Frame, we use coordinates, just like a 96-well plate.

```
data[Row, Column]
```

Example: data[1, 2] gets the value at Row 1, Column 2.

Indexing: Slicing the Plate

═Get a Row

Leave the column coordinate **blank** to get all columns.

```
data[1, ]
```

(Pulling one patient's full file)

‖Get a Column

Leave the row coordinate **blank** to get all rows.

```
data[ , 2]
```

(Pulling the 'Age' vector for the batch)

The Dollar Sign $

The "Quick Grab" Operator

Instead of memorizing column numbers, grab them by name.

```
lis_data$Viral_Load
```

Practice: Basic Stats on Frames

Now we can combine the $ operator with the functions we learned in Session 1.

- **Calculate Average Age:**
  mean(lis_data$Age)
- **Find Max Viral Load:**
  max(lis_data$Viral_Load)
- **Summary of all columns:**
  summary(lis_data)

Concept 4: Logical Subsetting (The Filter)

We rarely analyze *all* patients. We usually want a specific subgroup (e.g., "Positives" only).

The Logic: "Select all rows WHERE Status is Positive".

This creates a reproducible filter, unlike manually hiding rows in Excel.

Step 1: Creating the Mask

First, we ask R a Question. This creates a logical vector of TRUE/FALSE.

```
is_positive <- lis_data$Status == "Positive"
```

TRUE FALSE TRUE TRUE

"Keep" vs "Discard"

Step 2: Applying the Mask

We put the mask inside the row coordinate of the brackets.

```
# data[ROWS, COLUMNS]

positive_patients <- lis_data[is_positive, ]
```

Note the comma! We leave the column coordinate blank because we want to keep *all columns* for the positive patients.

Complex Logic (AND / OR)

AND (&)

Both conditions must be TRUE.

```
criteria <- (lis_data$Gender == "M") &
            (lis_data$Viral_Load > 1000)
```

OR (|)

At least one condition is TRUE.

```
criteria <- (lis_data$Status == "High") |
            (lis_data$Status == "Critical")
```

Handling Missing Data (NA)

⚠The Broken Tube

If a vector contains NA, math functions will return NA by default.

mean(c(10, 20, NA)) -> NA

The Fix: Tell the function to remove NAs.

```
mean(lis_data$Viral_Load, na.rm = TRUE)
```

Concept 5: Writing Data (Output)

Once you have filtered the "Critical" patients, you need to send a list to the doctor.

```
write.csv(positive_patients,
          "Critical_Report.csv",
          row.names = FALSE)
```

This creates a new CSV file. The raw data remains untouched.

Session 2 Summary

- read.csv(): Import data from a file.
- head() / str(): Inspect data structure.
- $: Select a column by name.
- [Row, Col]: Select by coordinates.
- data[mask,]: Filter rows using logic.
- write.csv(): Export results to a file.

Tutorial 2: The LIS Clean-Up

Goal: Import a messy LIS dump, fix data type errors, filter for critical female patients, and export a report.

1. Import from URL
2. Clean Data Types
3. Logical Filtering

Step 1: Load the Data

We will read the raw data directly from the GitHub repository.

```
# 1. Define the URL
url <- "https://raw.githubusercontent.com/.../mock_LIS_data.csv"

# 2. Read the data
df <- read.csv(url)

# 3. Check the receipt
head(df)
```

Step 2: The Inspection

Run the MRI scan on your data.

```
str(df)
```

Observation: Look at the Result_Value column. Does it say chr or num?

If it says chr, it means someone typed text (e.g., "Error") into a number column.

Step 3: Data Cleaning (Coercion)

We need to force the column to be numeric.

```
df$Result_Value <- as.numeric(df$Result_Value)
```

Warning: "NAs introduced by coercion"

This is good! It means R turned the text "Error" into NA.

Step 4: Stats with NAs

Calculate the mean Result Value. Don't forget to handle the new NAs.

```
# This will return NA
mean(df$Result_Value)

# This will return the actual mean
mean(df$Result_Value, na.rm = TRUE)
```

Step 5: Complex Filtering

We want: **Female** AND **High Result (>40)** AND **Not NA**.

```
# Create the complex mask
fem_high_mask <- (df$Gender == "F") &
                 (df$Result_Value > 40) &
                 !is.na(df$Result_Value)

# Apply the mask
critical_females <- df[fem_high_mask, ]
```

Step 6: Export

Save your work to a clean CSV file.

```
write.csv(critical_females,
          "Urgent_Review_Females.csv",
          row.names = FALSE)
```

Check your file pane to see the new report!

https://github.com/Koohoko/MSc_Module3_Programming_BioData_HKUSpacce_2026

README

MSc Module 3: Programming for Biological Data (2026)

Instructor: Dr. Gu Haogao
Institution: SPH HKU / HKU SPACE **Date:** January 2026

Course Overview

This repository contains the comprehensive preparation package for **Module 3**, tailored for Postgraduate Certificate in Bioinformatics for Medical Laboratory Technologists (MLTs). This plan integrates the "Code-as-Protocol" pedagogical approach.

Repository Structure

- `lectures/` : Slides and R scripts used during the lecture.
- `tutorials/` : Interactive notebooks for hands-on practice.
- `data/` : Raw datasets used in exercises.
- `setup/` : Scripts to initialize the R environment.

Quick Start

No software installation is required. We will use Google Colab for our "Dry Lab" sessions. Remember to **change runtime to R** while using colab (**Runtime -> Change runtime type -> R**).

- Session 1: Introduction to R (Part 1)
  - Lecture Script
  - Tutorial Notebook: Open in Colab
- Session 2: Introduction to R (Part 2) - Data Frames & I/O
  - Lecture Script
  - Tutorial Notebook: Open in Colab