Combined Workbook: R Lectures

Table of Contents

[Lecture 1: Introduction to Programming 1](#_Toc210077109)

[Key Concepts 1](#_Toc210077110)

[Worked Coding Examples 1](#_Toc210077111)

[Assignment 1](#_Toc210077112)

[Vector 1](#_Toc210077113)

[Matrix 1](#_Toc210077114)

[Dataframe 2](#_Toc210077115)

[Quick Cheatsheet 3](#_Toc210077116)

[Key Concepts 4](#_Toc210077117)

[Worked Coding Examples 4](#_Toc210077118)

[Reading Data 4](#_Toc210077119)

[Subsetting 4](#_Toc210077120)

[New column 5](#_Toc210077121)

[Tidyverse mutate 5](#_Toc210077122)

[Regex 5](#_Toc210077123)

[Quick Cheatsheet 6](#_Toc210077124)

[Regular Expressions (Regex) in R 7](#_Toc210077125)

[String manipulation functions: 7](#_Toc210077126)

[Example — grep (search; return row numbers) 7](#_Toc210077127)

[Example — grepl (search; return TRUE/FALSE per element) 7](#_Toc210077128)

[Example — gsub (replace ALL matches) 7](#_Toc210077129)

[Example — sub (replace FIRST match only) 7](#_Toc210077130)

[Anchors 7](#_Toc210077131)

[Wildcards 7](#_Toc210077132)

[Alternation (OR) 8](#_Toc210077133)

[Escaping Special Characters 8](#_Toc210077134)

[Perl-Compatible Regex 8](#_Toc210077135)

[grep / grepl / gsub / sub 9](#_Toc210077136)

[Anchors & Wildcards 9](#_Toc210077137)

[How to write a regular expression 9](#_Toc210077138)

[Regex Practice Exercises 10](#_Toc210077139)

[Extract sample numbers 10](#_Toc210077140)

[Find RNA-seq files 10](#_Toc210077141)

[Extract treatment type 10](#_Toc210077142)

[Extract timepoints 10](#_Toc210077143)

[Identify BAM vs FASTQ 10](#_Toc210077144)

‘

Lecture 1: Introduction to Programming

Date: 29-09-2025

## Key Concepts

|  |  |
| --- | --- |
| Cues / Keywords | Notes |
| What is programming? | Giving the computer step-by-step instructions. |
| History | Babbage’s Analytical Engine; Ada Lovelace’s algorithm; evolution from machine code → R. |
| R language | Created in 1993, based on S. Open-source, built for statistics and data visualization. |
| Data types | Logical (TRUE/FALSE), Numeric, Integer, Character (text). |
| Data structures | Vector (1D same type), Matrix (2D same type), Dataframe (2D mixed), List (flexible). |
| Assignment | Use <- or = to assign values. |
| Environment | Workspace in RStudio showing your objects, dataframes, and plots. |

# Worked Coding Examples

## Assignment

Storing values for BMI calculation

height <- 170  
weight <- 65  
bmi <- weight / (height/100)^2  
bmi

Output:

[1] 22.49

👉 Variables let you reuse values (like height/weight) to calculate new things (BMI).

## Vector

Storing ages of patients

ages <- c(34, 56, 45, 29, 61)  
ages[ages > 50]

Output:

[1] 56 61

👉 Vectors hold one type of data (numbers, characters, etc.). Here we filter patients > 50.

## Matrix

Gene expression counts across samples

expr\_matrix <- matrix(c(5,2,8, 3,6,1, 7,4,9), nrow=3, byrow=TRUE)  
rownames(expr\_matrix) <- c('Gene1','Gene2','Gene3')  
colnames(expr\_matrix) <- c('SampleA','SampleB','SampleC')  
expr\_matrix

Output:

SampleA SampleB SampleC  
Gene1 5 2 8  
Gene2 3 6 1  
Gene3 7 4 9

👉 Each row is a gene, each column a sample. Only numeric values allowed.

## Dataframe

Patient clinical data

patients <- data.frame(ID=c('P1','P2','P3'), Age=c(45,62,30), Diagnosis=c('NSCLC','SCLC','NSCLC'), Smoker=c(TRUE,TRUE,FALSE))  
patients

Output:

ID Age Diagnosis Smoker  
1 P1 45 NSCLC TRUE  
2 P2 62 SCLC TRUE  
3 P3 30 NSCLC FALSE

👉 Dataframes allow mixed data: numbers, text, logical values. Perfect for clinical data.

# Quick Cheatsheet

|  |  |
| --- | --- |
| Cues / Keywords | Code |
| Assignment | x <- 5 |
| Vector | ages <- c(34, 56, 45) |
| Matrix | matrix(data, nrow, ncol) |
| Dataframe | data.frame(ID, Age, Diagnosis) |

Common Beginner Mistakes

⚠️ Using '=' instead of '<-'  
✔️ Both work, but '<-' is standard style.

⚠️ Forgetting quotes for strings  
✔️ Always wrap text in quotes, e.g., "NSCLC".

⚠️ Mixing types in vectors  
✔️ All values will be coerced to the same type.

Lecture 2: Data Manipulation

Date: 29-09-2025

## Key Concepts

|  |  |
| --- | --- |
| Cues / Keywords | Notes |
| Data in R | Most data comes from external files (.csv, .tsv, .xlsx, .RDS, .RData). |
| Why manipulate? | Raw data needs cleaning, filtering, combining before analysis. |
| Data formats | CSV/TSV (plain text), Excel (manual checking), RDS/RData (R-native). |
| Tidyverse | A set of packages (dplyr, ggplot2) for tidy, readable data workflows. |
| Regex | Text-matching patterns, used to extract info from filenames or IDs. |

# Worked Coding Examples

## Reading Data

Importing patient CSV file

patients <- read.csv('patients.csv')  
head(patients)

Output:

Shows first 6 patients from file.

👉 R can easily read CSV/TSV files, which are the most common biomedical data format.

## Subsetting

Patients aged ≥ 50

subset(patients, Age >= 50)

Output:

ID Age Diagnosis Smoker  
2 P2 62 SCLC TRUE

👉 Subset filters rows based on logical conditions.

## New column

Add BMI column

patients$BMI <- c(22.5, 28.1, 19.8)  
head(patients)

Output:

ID Age Diagnosis Smoker BMI  
1 P1 45 NSCLC TRUE 22.5  
2 P2 62 SCLC TRUE 28.1  
3 P3 30 NSCLC FALSE 19.8

👉 Dataframes can be extended with new columns, e.g., calculated measures like BMI.

## Tidyverse mutate

Adding risk classification

library(dplyr)  
patients <- patients %>% mutate(Risk = ifelse(Age>50,'High','Low'))  
patients

Output:

ID Age Diagnosis Smoker BMI Risk  
1 P1 45 NSCLC TRUE 22.5 Low  
2 P2 62 SCLC TRUE 28.1 High  
3 P3 30 NSCLC FALSE 19.8 Low

👉 Mutate adds new columns cleanly using conditions.

## Regex

Extract treatment info from filenames

files <- c('RNAseq\_Sample1\_Control','RNAseq\_Sample2\_Drug')  
gsub('.\*\_(Control|Drug).\*','\1',files)

Output:

[1] "Control" "Drug"

👉 Regex lets you extract metadata from filenames (e.g., Control vs Drug).

# Quick Cheatsheet

|  |  |
| --- | --- |
| Cues / Keywords | Code |
| Read CSV | read.csv('file.csv') |
| Subset | subset(df, Age > 50) |
| New column | df$new <- ... |
| Mutate | df %>% mutate(new = ...) |
| Regex | gsub('pattern','replace',text) |

Common Beginner Mistakes

⚠️ Forgetting delimiter in read.delim  
✔️ Use sep=',' for CSVs.

⚠️ Mismatched row/col names when merging  
✔️ Always align colnames with rownames.

⚠️ Wrong dimensions in cbind/rbind  
✔️ Check number of rows/columns match.

⚠️ Using mutate without tidyverse  
✔️ Run library(tidyverse) first.

# Regular Expressions (Regex) in R

Regular expressions are text-matching patterns. They are essential for extracting information from filenames, sample IDs, or metadata.

## String manipulation functions:

### Example — grep (search; return row numbers)

Code: grep('\\.bam$', files)

grep → **search filenames;** return the row numbers of matches (name comes from Unix g/re/p).

\\. → **wildcard** – a literal dot (.).

bam → the letters "bam".

$ → **anchor** for end of string.

files → **vector of filenames** being searched.

👉 Finds which files end with .bam and returns their positions.

### Example — grepl (search; return TRUE/FALSE per element)

Code: grepl('^RNAseq', files)

grepl → like grep, but **returns** **TRUE/FALSE for each filename**.

**^** → **anchor** for start of string.

RNAseq → literal text.

files → **vector of filenames** being checked.  
👉 Tells you whether each filename starts with “RNAseq”.

### Example — gsub (replace ALL matches)

Code: gsub('.\*\_(Drug|Control)\_.\*', '\\1', files)

*gsub → global substitution* ***(replace all matches).***

*.\*\_ →* ***wildcard*** *- any text up to an underscore.*

*(Drug|Control) → capture the word “Drug” or “Control”.*

*\_.\* →****wildcard*** *- another underscore and the rest of the text.*

*\\1 →* ***wildcard*** *- replace with the first captured group.*

*files →* ***filenames being processed.***

*👉 Extracts the treatment type (Drug or Control) from each filename.*

### Example — sub (replace FIRST match only)

Code: sub('Sample', 'S', 'RNAseq\_Sample12')

*sub →* ***replace*** *only the* ***first match.***

*Sample → text to* ***find****.*

*S →* ***replacement****.*

*👉 Changes "RNAseq\_Sample12" into "RNAseq\_S12".*

## Anchors

grep('^RNAseq', files) # ^ = starts with  
grep('bam$', files) # $ = ends with

[1] 1 2 # files starting with RNAseq  
[1] 3 # file ending with bam

👉 Anchors match positions, not characters. ^ = beginning, $ = end.

## Wildcards

^ start • $ end • . any char • \\. literal dot • [0-9] digit • [0-9]+ 1+ digits • (A|B) A or B • ( ) capture • ? 0/1 • \* 0+ • + 1+ • {n} exactly n

Example:

files <- c(  
 'RNAseq\_Sample1\_Control\_24h.fastq.gz',  
 'RNAseq\_Sample12\_Drug\_72h.fastq.gz',  
 'ChIPseq\_Sample3\_Control\_24h.bam',  
 'ATACseq\_Sample4\_Drug\_48h.fastq.gz'  
)

|  |  |  |  |
| --- | --- | --- | --- |
| Pattern | Meaning | Example (R code) | Matches (per file) |
| . | Any single character | grepl('Sample.', files) | TRUE, TRUE, TRUE, TRUE |
| .\_ | One char + underscore | grepl('Sample.\_', files) | TRUE, FALSE, TRUE, TRUE |
| .\* | Any sequence | grepl('Sample.\*Drug', files) | FALSE, TRUE, FALSE, TRUE |
| [0-9] | One digit | grepl('Sample[0-9]', files) | TRUE, TRUE, TRUE, TRUE |
| [0-9]+ | One or more digits | grepl('Sample[0-9]+', files) | TRUE, TRUE, TRUE, TRUE |
| [0-9]{2} | Exactly two digits | grepl('Sample[0-9]{2}', files) | FALSE, TRUE, FALSE, FALSE |
| \_? | Zero or one underscore | grepl('Sample[0-9]\_?', files) | TRUE, FALSE, TRUE, TRUE |
| ^ | Start of string | grepl('^RNAseq', files) | TRUE, TRUE, FALSE, FALSE |
| $ | End of string | grepl('bam$', files) | FALSE, FALSE, TRUE, FALSE |

## Alternation (OR)

gsub('.\*\_(Drug|Control)\_.\*', '\\1', files)

[1] 'Control' 'Drug' 'Control' 'Drug'

👉 `(Drug|Control)` matches either word.

## Escaping Special Characters

grep('\\.bam$', files)

[1] 3

👉 Use `\\.` to match a literal dot, since `.` means any character.

## Perl-Compatible Regex

gsub('.\*Sample(\\d+).\*','\\1', files, perl=TRUE)

[1] '1' '12' '3' '4'

👉 With `perl=TRUE`, you can use shortcuts: `\\d` (digit), `\\s` (space), `\\w` (word chars).

**Regular Expression Summary**

Purpose: pull out bits of information from text eg. Finding out whether patient 1 was drug or control group.

## grep / grepl / gsub / sub

|  |  |  |  |
| --- | --- | --- | --- |
| Function | Meaning | Typical use | Example |
| grep() | Returns indices of matches | Which entries match? | grep('SCLC', patients$Diagnosis) # 2 |
| grepl() | TRUE/FALSE for each entry | Filter rows | patients[grepl('NSCLC', patients$Diagnosis), ] |
| gsub() | Replace all matches | Clean labels | gsub('NSCLC','Lung Cancer', patients$Diagnosis) |
| sub() | Replace first match only | Rename partial strings | sub('Sample','S','RNAseq\_Sample12') |
|  |  |  |  |

## Anchors & Wildcards

^ = start • $ = end • . = any char • \\. = literal dot • [0-9] = one digit • [0-9]+ = one/more digits • ( ) = capture group • (A|B) = A or B • ? = zero/one • \* = zero/more • + = one/more • {n} = exactly n

## How to write a regular expression

Grep/grepl/gsub/sub ( location targeted with wildcards directing text within or ensuing , first / second / third of its kind , file selected from )

# Regex Practice Exercises

files <- c(  
 'RNAseq\_Sample1\_Control\_24h.fastq.gz',  
 'RNAseq\_Sample12\_Drug\_72h.fastq.gz',  
 'ChIPseq\_Sample3\_Control\_24h.bam',  
 'ATACseq\_Sample4\_Drug\_48h.fastq.gz'  
)

## Extract sample numbers

gsub('.\*Sample([0-9]+).\*','\\1', files)

[1] '1' '12' '3' '4'

## Find RNA-seq files

grep('^RNAseq', files)

[1] 1 2

## Extract treatment type

gsub('.\*\_(Drug|Control)\_.\*','\\1', files)

[1] 'Control' 'Drug' 'Control' 'Drug'

## Extract timepoints

gsub('.\*\_([0-9]+h).\*','\\1', files)

[1] '24h' '72h' '24h' '48h'

## Identify BAM vs FASTQ

grep('\\.bam$', files)  
grep('\\.fastq\\.gz$', files)

[1] 3  
[1] 1 2 4

# Installing files and manipulating data frames

#### Setting up – Save your external data and create an ‘Output’ folder to local downloads folder on Mac (where R is stored).

#### ```{r} Install and load required packages -

A screen shot of a computer code

Description automatically generated

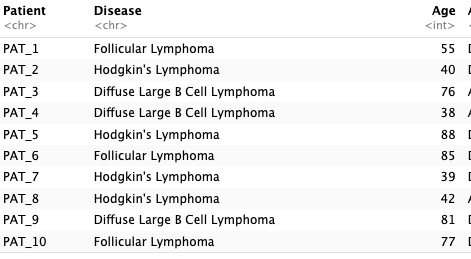
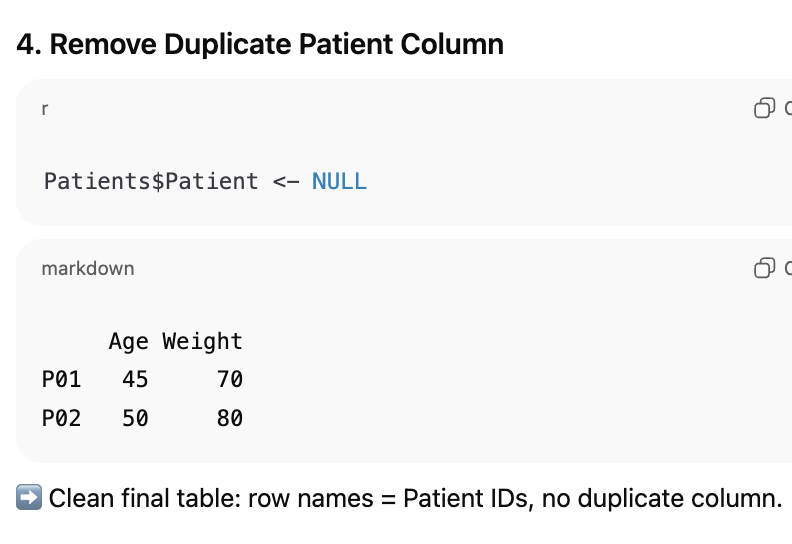
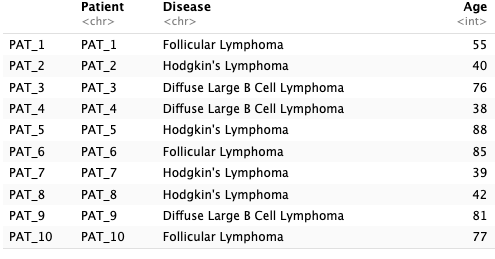
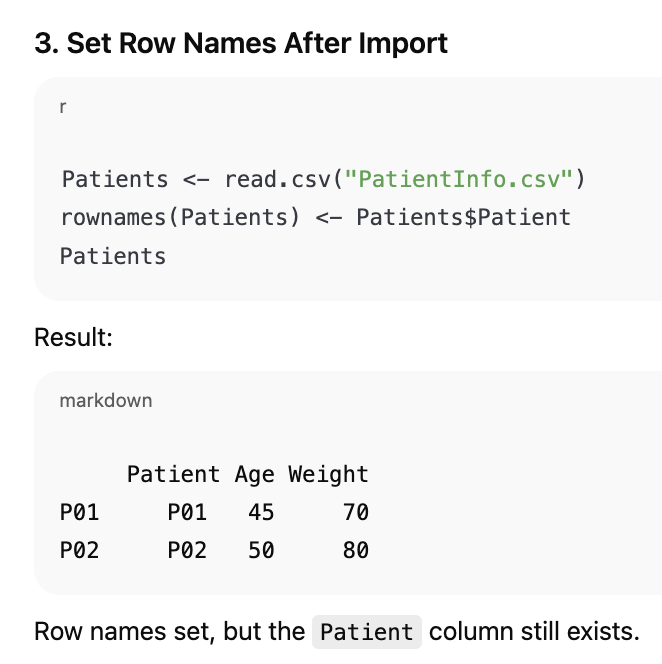
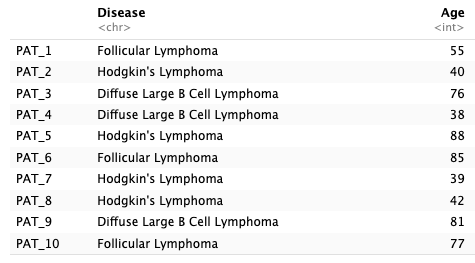
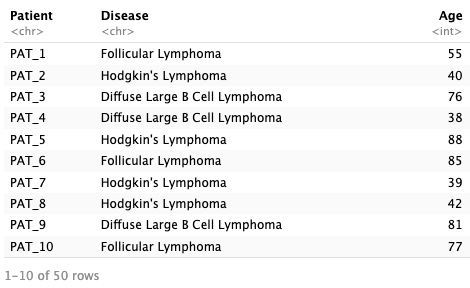
#### A screenshot of a computer code Description automatically generatedA screenshot of a computer Description automatically generatedLoad files: make sure you add sep “,” or “;” etc, depending on what separates columns if load csv file in rea.delim or just use read.csv("Data/PatientInfo.csv") -> will produce dataframe (different datatypes) -

#### Manipulate your loaded dataframes:

#### Setting row/column names (or assigning row/column names) when you import a dataset

patients <- read.csv("Data/PatientInfo.csv", row.names = 1)

* read.csv → “read a CSV file”
* "Data/PatientInfo.csv" → the file you want to read (inside a folder called Data)
* row.names = 1 → tells R: “Use the first column in the file (Patient IDs) as the row names (labels), not as normal data.”
* <- means “save it into”
* patients → the name we’re giving the data table in R.



### Given data with no proper column (header) names

read.delim("PatientInfo.csv", sep = ",", header = FALSE, skip = 1)

A screenshot of a phone

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

### Given an Excel file (not txt, rmd or csv)

PatientsXLSX <- read.xlsx("Data/PatientInfo.xlsx", sheet = "Patients")) #read xlsx file and call on sheet 1 (patients) if data stored here

PatientsXLSX #call the file

Patients2XLSX <- read.xlsx("Data/PatientInfo.xlsx", sheet = "Patients2") #Patients2 is much bigger than Patients

Patients2XLSX

Start with **read.xlsx()**.

If your file takes ages to load or R throws memory warnings → switch to **read.xlsx2()**. (for bigger packages)

### Given a RDS file (r data object)

PatientsRDS <- readRDS("Data/Patients.rds")



Check the r data object matches the imported data from the csv file exactly



#### Save your loaded manipulated data files:

When you’ve finished your analysis, you’ll often want to save results. The **format** depends on what you (or your collaborators) will use next:

* **Text files** → .csv, .tsv, .txt (good for sharing, small, easy to open in Excel/Notepad).
* **Spreadsheets** → .xlsx, .ods (handy if collaborators want to open directly in Excel/LibreOffice).
* A screenshot of a computer

  Description automatically generated**R objects** → .rds, .RData (best for your own work, because they preserve R-specific structures).

A screenshot of a computer

Description automatically generatedA screenshot of a computer code

Description automatically generated

A screenshot of a computer program

Description automatically generated

✅ **Summary**

* Use **CSV/TSV** if you’re sharing with others or want to open in Excel.
* Use **Excel (.xlsx)** if collaborators prefer spreadsheets.
* Use **RDS/RData** if you plan to load it back into R later.