

VU LEARNING HUB

vulearninghub.xyz

HIGHLIGHTED HANDOUT

STA636

Statistical Computing in R

STA636 by VU Learning HUB

Prepared for VU students • Free study resources

<https://vulearninghub.xyz>

STA636 - Statistical Computing in R

A structured study handout compiled from the course's lecture topics. Read each topic's objectives first, study the key concepts, then use the summary and exam tips for revision. ☰

STA636

Statistical Computing in R

Based on 45 lecture topics

Compiled for VU students as a structured study guide

*Concepts are summarized in the writer's own words;
use alongside your lecture videos for best results.*

Contents

Unit 1: R Fundamentals	4
Topic 1: Introduction to R and Statistical Computing	4
Topic 2: Objects, Vectors, and Data Types	5
Topic 3: Matrices, Arrays, and Lists	6
Topic 4: Control Flow and Functions	6
Topic 5: Getting Help and Packages	7
Unit 2: Data Import, Cleaning, and Manipulation	8
Topic 6: Importing Data into R	8
Topic 7: Data Cleaning Essentials	9
Topic 8: Manipulating Data with dplyr	9
Topic 9: Reshaping Data: Long vs. Wide	10
Topic 10: Dates, Strings, and Factors	11
Unit 3: Descriptive Statistics and Graphics	12
Topic 11: Numerical Summaries	12
Topic 12: Base R Graphics	12
Topic 13: Grammar of Graphics with ggplot2	13
Topic 14: Contingency Tables and Association	14
Topic 15: Correlation Analysis	15
Unit 4: Probability Distributions and Simulation	15
Topic 16: Probability Distributions in R	15
Topic 17: Random Number Generation and Seeds	16
Topic 18: The Central Limit Theorem in Action	17
Topic 19: Monte Carlo Simulation	18
Topic 20: Bootstrapping	18
Unit 5: Hypothesis Testing in R	19
Topic 21: t-Tests: One, Two Sample, and Paired	19
Topic 22: Nonparametric Alternatives	20
Topic 23: ANOVA in R	21
Topic 24: Proportions and Power Analysis	22
Topic 25: Goodness of Fit and Normality Checks	22
Unit 6: Regression and Reporting	23
Topic 26: Simple Linear Regression	23
Topic 27: Multiple Regression	24
Topic 28: Logistic Regression	25

Topic 29: Reproducible Reports with R Markdown 25

Topic 30: Functions, Debugging, and Best Practices 26

Glossary of Key Terms 27

Course Introduction

Statistical Computing in R teaches how to perform statistical analysis using R - the open-source programming language that has become the standard tool for statisticians and data analysts worldwide. The course bridges statistical theory and practical computation.

It begins with R fundamentals: installation, the console, objects, vectors, and data structures. Students then learn data import, cleaning, and manipulation - the unglamorous but essential majority of real analysis work - followed by descriptive statistics and publication-quality graphics.

Later units cover probability distributions and simulation in R, hypothesis testing, regression modeling, and writing reusable functions and reports with R Markdown. By the end, students can take a raw dataset, analyze it statistically, visualize the results, and present findings reproducibly - the core workflow of applied statistics.

How to use this handout: Each topic lists its learning objectives, explains the key concepts in simple words, defines important terms, gives an example where helpful, and closes with a summary plus exam tips. Topics marked **Very important** deserve extra revision.

Unit 1: R Fundamentals

Topic 1: Introduction to R and Statistical Computing

Learning Objectives

- Explain what R is and why statisticians use it
- Describe the R environment
- Compare R with other statistical software

Key Concepts

R is a free, open-source programming language and environment for statistical computing and graphics, created by Ross Ihaka and Robert Gentleman. It dominates academic statistics and is widely used in industry data science.

Why R: it is free (unlike SPSS/SAS licenses), has thousands of packages for every statistical method, produces excellent graphics, and supports reproducible research - entire analyses can be scripted and rerun. Its community continuously extends it.

The R environment includes the console (interactive prompt), scripts (saved code files), the workspace (objects in memory), and packages (add-on libraries). RStudio - the popular IDE - adds an editor, plots pane, and help viewer that make learning far easier.

Key Points

- R: free, open-source language for statistics and graphics.
- Advantages: cost, packages, graphics, reproducibility.
- Environment: console, scripts, workspace, packages; RStudio as IDE.

Key Terms

Console: R's interactive command prompt.

Package: An add-on library extending R's functions.

RStudio: The popular integrated development environment for R.

Reproducible research: Analyses fully captured in rerunnable code.

Example

- *Installing R and RStudio. then running `2 + 2` in the console as a first step.*

Summary: R is the standard free platform for statistical computing, best used through RStudio.

Exam Tips

- ★ Very important: why R (free, packages, graphics, reproducibility).
- ★ Remember console vs. script distinction.

Topic 2: Objects, Vectors, and Data Types

Learning Objectives

- Explain R objects and assignment
- Describe atomic vector types
- Perform basic vector operations

Key Concepts

Everything in R is an object stored with the assignment operator `<-` (e.g. `x <- 5`). Objects have types: numeric (numbers), integer, character (text in quotes), logical (TRUE/FALSE), and complex.

The vector is R's fundamental structure - an ordered collection of same-type elements created with `c()` (e.g. `marks <- c(78, 85, 92)`). R is vectorized: operations apply to whole vectors at once (`marks + 5` adds 5 to each element; `mean(marks)` averages all). This replaces loops for most tasks.

Key behaviors: combining types coerces to the most flexible (`c(1, 'a')` becomes character); indexing uses square brackets (`marks[2]` is the second element; `marks[marks > 80]` selects conditionally); and missing values are NA, handled with `na.rm = TRUE` in functions.

Key Points

- Objects assigned with `<-`; types: numeric, character, logical.
- Vectors via `c()`; operations are vectorized across elements.
- Indexing with `[]`; NA marks missing values.

Key Terms

Vector: An ordered collection of same-type elements.

Vectorization: Operations applying to whole vectors at once.

Coercion: Automatic conversion to a common type.

NA: R's missing-value indicator.

Example

- *`scores <- c(70, 85, 90)`; `mean(scores)` returns 81.67.*

Summary: Vectors and vectorized operations are the core of R thinking.

Exam Tips

- ★ Very important: vectorized operations with examples.
- ★ Remember `<-` assignment and NA handling.

Topic 3: Matrices, Arrays, and Lists

Learning Objectives

- Create and manipulate matrices
- Explain lists and data frames
- Distinguish the data structures

Key Concepts

Matrices are two-dimensional vectors: `matrix(1:6, nrow = 2)` fills column-wise by default. Matrix algebra (`t()` transpose, `%*%` multiplication, `solve()` inverse) supports multivariate statistics directly.

Arrays extend to more dimensions. Lists are R's flexible containers holding mixed types - even other lists and functions: `mylist <- list(name = 'Ali', marks = c(80, 90))`. Model results in R are lists, accessed with `$` (`model$coefficients`).

Data frames are the workhorse: list-like columns of possibly different types but equal length, behaving like matrices - the standard format for datasets. `df$page` accesses a column; `df[1, ]` a row. Factors store categorical variables with levels, crucial for correct statistical modeling. 

Key Points

- Matrices: 2D vectors with algebra operators (`%*%`, `solve()`).
- Lists hold mixed types; model outputs are lists accessed via `$`.
- Data frames: dataset standard; factors encode categories.

Key Terms

Matrix: A two-dimensional vector.

List: A container for mixed-type objects.

Data frame: A table-like structure: columns of equal length.

Factor: A categorical variable with defined levels.

Example

- `students <- data.frame(name = c('Ali', 'Sara'), age = c(20, 21))`

Summary: Matrices compute, lists flex, data frames organize - factors categorize.

Exam Tips

- ★ Key point: when to use each structure.
- ★ Remember `$` for list/data-frame access.

Topic 4: Control Flow and Functions

Learning Objectives

- Use if/else and loops
- Explain the apply family
- Write custom functions

Key Concepts

Control flow directs execution: `if (x > 0) {...} else {...}` branches; `for (i in 1:10)` and `while()` loop. But R favors vectorized alternatives - `ifelse()` and the apply family (`apply`, `lapply`, `sapply`, `tapply`) process

structures without explicit loops, running faster and reading cleaner.

Custom functions package reusable logic: `mymean <- function(x) { sum(x) / length(x) }`. Functions take arguments (with defaults), return the last evaluated value, and keep variables local - avoiding workspace clutter.

Good practice: vectorize first, loop only when necessary; write small tested functions; use meaningful names. Debugging uses `traceback()`, `debug()`, and RStudio's debugger.

Key Points

- if/else, for, while control flow; prefer vectorized apply family.
- Functions: `function(args) { body }`; local scope.
- Vectorize first; keep functions small and tested.

Key Terms

apply family: Functions applying operations across structures without loops.

Scope: Where a variable is visible; function variables are local.

Vectorization: Replacing loops with whole-object operations.

Example

- `sapply(mvlist, mean)` computes means without a loop.

Summary: R style means vectorized operations and small custom functions.

Exam Tips

- ★ Key point: apply family vs. loops.
- ★ Remember function syntax.

Topic 5: Getting Help and Packages

Learning Objectives

- Use R's help system
- Install and load packages
- Find the right package for a task

Key Concepts

R's help: `?mean` opens a function's documentation; `??'topic'` searches; `vignette('name')` shows long-form guides. Reading help - especially the Arguments and Examples sections - is a core skill.

Packages extend R: `install.packages('ggplot2')` downloads once; `library(ggplot2)` loads each session. CRAN hosts 20,000+ vetted packages; key ones include dplyr (data manipulation), ggplot2 (graphics), tidyr, readr, and MASS.

Choosing packages: check CRAN task views, download counts, recent updates, and documentation quality. Cite packages in work (`citation('ggplot2')`). Reproducibility needs `sessionInfo()` recording versions.

Key Points

- Help: `?function`, `??topic`, `vignette()`; read Arguments and Examples.
- `install.packages()` once; `library()` each session.
- Choose maintained, documented packages; record versions.

Key Terms

CRAN: The Comprehensive R Archive Network hosting packages.

Vignette: A long-form package tutorial.

Namespace: A package's collection of functions.

Example

- *library(dplyr) then ?filter to learn data filtering.*

Summary: Help systems and packages make R infinitely extensible.

Exam Tips

- ★ Remember install vs. library distinction.
- ★ Key point: using ? and vignettes.

Unit 2: Data Import, Cleaning, and Manipulation

Topic 6: Importing Data into R

Learning Objectives

- Read CSV and Excel files
- Handle common import problems
- Describe other data sources

Key Concepts

Most analyses start by importing data: `read.csv('data.csv')` for comma-separated files; `readxl::read_excel()` for spreadsheets. Set the working directory (`setwd()`) or use RStudio projects with relative paths - never hard-code personal paths in shared code.

Common problems: headers in wrong rows (`skip =`), decimal commas, encoding issues (`fileEncoding = 'UTF-8'`), and automatic type guessing converting IDs to numbers (`colClasses` fixes). Always inspect after import: `head()`, `str()`, `summary()`.

Other sources: `read.table()` for delimited text, `haven` for SPSS/Stata files, `RMySQL`/`odbc` for databases, and `jsonlite`/`httr` for web APIs. `saveRDS()/readRDS()` store R objects efficiently between sessions.

Key Points

- `read.csv()` and `readxl` for common files; check with `head()`, `str()`, `summary()`.
- Fix headers, encodings, and type guessing explicitly.
- Use projects and relative paths, never hard-coded paths.

Key Terms

Working directory: The folder R reads/writes by default.

Delimiter: The character separating fields in text files.

Encoding: How text characters are stored.

Example

- *`df <- read.csv('survey.csv', stringsAsFactors = FALSE); str(df)`*

Summary: Careful importing with immediate inspection prevents garbage-in analysis.

Exam Tips

- ★ Key point: the inspect-after-import habit (head, str, summary).
- ★ Remember relative paths.

Topic 7: Data Cleaning Essentials

Learning Objectives

- Handle missing values
- Detect and treat outliers
- Standardize messy data

Key Concepts

Real data is messy. Missing values (NA): understand why missing (MCAR/MAR/MNAR affects handling), then delete (`na.omit()`), or impute (means, medians, or model-based via `mice`). Never silently ignore - report missingness.

Outliers: detect with boxplots and z-scores; investigate before deleting - an outlier may be the discovery (fraud!) or a data-entry error (age 250). Winsorizing caps extremes; transformations reduce skew.

Standardizing: consistent date formats (`as.Date()`), uniform categories ('Male' vs 'male' vs 'M'), trimmed whitespace, correct types (numbers stored as text). Document every cleaning step in code - cleaning must be reproducible too.

Key Points

- Missing data: understand mechanism, then delete or impute; always report.
- Outliers: investigate first - error or discovery?
- Standardize formats, categories, and types reproducibly.

Key Terms

Imputation: Filling missing values with estimated ones.

Outlier: An extreme value warranting investigation.

Winsorizing: Capping extreme values at percentiles.

Example

- `table(is.na(df$income))` quantifies missingness before deciding.

Summary: Cleaning is detective work: investigate anomalies, standardize systematically, document everything.

Exam Tips

- ★ Very important: outlier investigation before deletion.
- ★ Remember imputation vs. deletion.

Topic 8: Manipulating Data with dplyr

Learning Objectives

- Use the core dplyr verbs
- Explain the pipe operator
- Group and summarize data

Key Concepts

dplyr provides intuitive verbs: `filter()` (rows by condition), `select()` (columns), `mutate()` (new columns), `arrange()` (sort), `summarise()` (aggregate), and `group_by()` (per-group operations). The pipe `%>%` (or `|>`) chains them into readable workflows.

Example: `sales %>% filter(year == 2024) %>% group_by(region) %>% summarise(total = sum(amount))` reads as a sentence. Joins (`left_join()`, `inner_join()`) combine tables by keys - the database operations of analysis.

Grouped summaries answer most business questions: average marks by class, total sales by month. dplyr's consistency - first argument always the data - makes code predictable and teachable.

Key Points

- Verbs: `filter`, `select`, `mutate`, `arrange`, `summarise`, `group_by`.
- Pipes chain verbs into readable workflows.
- Joins combine tables; grouped summaries answer business questions.

Key Terms

Pipe: The `%>%` operator passing data through steps.

Join: Combining tables by matching keys.

Tidy data: Each variable a column, each observation a row.

Example

- `df %>% group_by(aender) %>% summarise(mean_income = mean(income, na.rm = TRUE))`

Summary: dplyr's verbs plus pipes make data manipulation readable and reliable.

Exam Tips

- ★ Very important: the six verbs and what each does.
- ★ Remember the pipe concept.

Topic 9: Reshaping Data: Long vs. Wide

Learning Objectives

- Distinguish long and wide formats
- Reshape with `tidyr`
- Explain when each format suits analysis

Key Concepts

Wide format has one row per subject with repeated measures in columns (`mark1`, `mark2`); long format stacks them (subject, test, mark). Most R modeling and `ggplot2` want long format; human reports often prefer wide.

`tidyr` reshapes: `pivot_longer()` melts wide to long; `pivot_wider()` spreads long to wide. `separate()/unite()` split and combine columns.

Choosing: keep data long for analysis and plotting, pivot wide for presentation tables. Recognizing format mismatches explains many beginner errors - e.g. plotting failures when data is wide.

Key Points

- Wide: measures in columns; Long: measures stacked in rows.

- `pivot_longer()` and `pivot_wider()` convert between them.
- Analyze long; present wide.

Key Terms

Long format: Repeated measures stacked in rows.

Wide format: Repeated measures spread across columns.

`pivot_longer/wider`: tidy's reshaping functions.

Example

- `pivot_longer(df, cols = c(mark1, mark2), names_to = 'test')`

Summary: Reshaping fluency lets data meet each tool's expected format.

Exam Tips

- ★ Key point: long for analysis, wide for presentation.
- ★ Remember `pivot_longer` vs. `pivot_wider`.

Topic 10: Dates, Strings, and Factors

Learning Objectives

- Work with dates in R
- Manipulate strings
- Manage factor levels

Key Concepts

Dates: `as.Date('2024-01-15')` creates proper dates supporting arithmetic (differences in days) and extraction (`format(date, '%Y')`). `lubridate` simplifies parsing (`ymd()`, `dmy()`) and arithmetic - essential for time-based analysis.

Strings: `paste()` joins, `substr()/nchar()` extract and measure, `grep()/grepl()` search patterns, and `stringr` offers consistent verbs. Cleaning text columns (trimming, case-folding) precedes analysis.

Factors: `factor(x, levels = c('Low','Medium','High'))` sets meaningful order - critical because R orders alphabetically by default, scrambling ordinal plots and models. `relevel` sets reference categories in regression.

Key Points

- Dates support arithmetic; `lubridate` eases parsing.
- String functions clean and extract text.
- Set factor levels explicitly - alphabetical default misleads.

Key Terms

`lubridate`: Package simplifying date handling.

`stringr`: Package for consistent string operations.

Levels: The defined categories of a factor.

Example

- `factor(grade, levels = c('F','D','C','B','A'))` preserves grade order.

Summary: Dates, strings, and factors need explicit handling to behave correctly.

Exam Tips

- ★ Key point: factor level ordering.
- ★ Remember date arithmetic.

Unit 3: Descriptive Statistics and Graphics

Topic 11: Numerical Summaries

Learning Objectives

- Compute measures of center and spread
- Explain quantiles and the five-number summary
- Compare mean vs. median

Key Concepts

Center: `mean(x)` (sensitive to outliers), `median(x)` (robust middle value). Spread: `sd(x)` standard deviation, `var(x)` variance, `IQR(x)` interquartile range (robust), `range(x)` min-max. `summary()` gives min, quartiles, median, mean compactly.

The five-number summary (min, Q1, median, Q3, max) underlies boxplots and resists outliers - preferred for skewed data like incomes. Skewness direction: $\text{mean} > \text{median}$ indicates right skew.

Grouped summaries via `aggregate()` or `dplyr` reveal structure: performance by department, sales by region. Always pair numbers with plots - summaries hide shapes (Anscombe's quartet warns).

Key Points

- Center: mean (sensitive), median (robust); Spread: sd, IQR, range.
- Five-number summary resists outliers; use for skewed data.
- Group summaries reveal structure; pair numbers with plots.

Key Terms

Five-number summary: Min, Q1, median, Q3, max.

IQR: Interquartile range: Q3 minus Q1.

Skewness: Asymmetry of a distribution.

Example

- *`summary(income)` instantly shows skew via mean vs. median gap.*

Summary: Numerical summaries compress data; robust measures handle skew and outliers.

Exam Tips

- ★ Very important: mean vs. median under skew.
- ★ Remember the five-number summary.

Topic 12: Base R Graphics

Learning Objectives

- Create histograms, boxplots, scatterplots
- Customize plots
- Explain when each plot suits

Key Concepts

Base R graphics are immediate: `hist(x)` shows distribution shape; `boxplot(y ~ group)` compares groups; `plot(x, y)` scatterplots relationships; `barplot(table(x))` counts categories. Arguments `main`, `xlab`, `col` customize.

Choosing: histograms for single-variable shape, boxplots for group comparison, scatterplots for bivariate relationships, barplots for categories - never pie charts for serious comparison (humans judge angles poorly).

Customization: `par(mfrow = c(1,2))` arrays panels; `abline()` adds reference lines; `legend()` explains. Base graphics are fast for exploration; `ggplot2` (later) for presentation.

Key Points

- `hist()`, `boxplot()`, `plot()`, `barplot()` cover core needs.
- Match plot to question: shape, comparison, relationship, counts.
- Customize labels and layout; avoid pie charts.

Key Terms

Histogram: Bars showing a variable's distribution.

Boxplot: Five-number summary as a box with whiskers.

Scatterplot: Points showing two variables' relationship.

Example

- `boxplot(marks ~ section, data = df, main = 'Marks by Section')`

Summary: Base graphics give fast, honest exploratory plots.

Exam Tips

- ★ Key point: which plot for which question.
- ★ Remember why pie charts are discouraged.

Topic 13: Grammar of Graphics with ggplot2

Learning Objectives

- Explain the grammar of graphics
- Build layered plots
- Customize themes and labels

Key Concepts

`ggplot2` builds plots from components: data, aesthetics (`aes()` mappings like `x`, `y`, `color`), geometries (`geom_point()`, `geom_bar()`, `geom_line()`), plus scales, facets, and themes. Layers combine: `ggplot(df, aes(x = age, y = income)) + geom_point() + geom_smooth()`.

This grammar makes complex plots systematic: faceting (`facet_wrap(~ gender)`) splits by group; aesthetic mappings reveal dimensions (color by category, size by value). Scales control axes and legends; `labs()` titles professionally; `theme_minimal()` cleans styling.

The payoff: publication-quality graphics from composable parts, and code that documents exactly how each figure was made - reproducibility for visuals.

Key Points

- Plots = data + aesthetics + geometries + scales + facets + themes.
- Layers and mappings build complexity systematically.
- Facets split by group; themes polish professionally.

Key Terms

Aesthetics: Mappings from data to visual properties.

Geometry: The visual mark: points, bars, lines.

Faceting: Splitting a plot into subplots by group.

Example

- `aaplot(df, aes(x=hours, v=marks, color=aender)) + aeom_point() + aeom_smooth(method='lm')`

Summary: ggplot2's grammar turns plot-building into systematic layering.

Exam Tips

- ★ Very important: the grammar components.
- ★ Remember aes() mappings.

Topic 14: Contingency Tables and Association

Learning Objectives

- Create cross-tabulations
- Compute chi-square tests
- Interpret association measures

Key Concepts

`table(a, b)` cross-tabulates categories; `prop.table()` converts to proportions (margins 1 = rows, 2 = columns). Mosaic plots visualize associations.

The chi-square test (`chisq.test()`) assesses independence: does gender associate with product preference? Expected counts assume independence; large deviations give small p-values. Check assumptions: expected counts ≥ 5 .

Effect size matters beyond significance: Cramer's V (via `assocstats`) measures association strength. For 2x2 tables, odds ratios quantify effect directly. Report both significance and size - a significant but tiny association rarely matters practically.

Key Points

- `table()` and `prop.table()` summarize categorical relationships.
- Chi-square tests independence; check expected counts.
- Report effect size (Cramer's V), not just p-values.

Key Terms

Contingency table: Counts cross-classified by categories.

Chi-square test: Test of independence for categorical data.

Cramer's V: A 0-1 measure of association strength.

Example

- `chisq.test(table(gender, preference))` tests if preference differs by gender.

Summary: Tables, tests, and effect sizes together describe categorical associations.

Exam Tips

- ★ Key point: chi-square assumptions.
- ★ Remember effect size alongside p-values.

Topic 15: Correlation Analysis

Learning Objectives

- Compute correlations in R
- Interpret correlation strength
- Explain correlation vs. causation

Key Concepts

`cor(x, y)` gives Pearson's r (-1 to 1); `use = 'complete.obs'` handles missingness; `method = 'spearman'` for ranked/monotonic relationships. `cor(df)` matrices summarize many variables; `corrplot` visualizes them.

Interpretation: $|r|$ near 1 strong linear, near 0 weak/none - but always plot first, since r misses curves and is outlier-sensitive. `cor.test()` adds significance and confidence intervals.

The eternal warning: correlation is not causation - confounders, reverse causality, and coincidence create spurious links. R quantifies association; subject knowledge judges causation.

Key Points

- `cor()` for Pearson; `spearman` for monotonic; plot before trusting.
- Matrices and `corrplot` summarize many variables.
- Correlation never proves causation.

Key Terms

Pearson's r : Linear correlation between -1 and 1.

Spearman correlation: Rank-based, capturing monotonic relationships.

Spurious correlation: A misleading association without causal link.

Example

- *`cor.test(hours_studied, marks)` with a scatterplot showing the trend.*

Summary: Correlation quantifies linear association; plots and judgment complete the story.

Exam Tips

- ★ Very important: correlation vs. causation.
- ★ Remember to plot first.

Unit 4: Probability Distributions and Simulation

Topic 16: Probability Distributions in R

Learning Objectives

- Use the `d/p/q/r` distribution functions
- Explain the naming system

- Apply common distributions

Key Concepts

R's elegant system: for each distribution, d = density (dnorm), p = cumulative probability (pnorm), q = quantiles (qnorm), r = random generation (rnorm). Prefixes combine with distribution roots: norm, binom, pois, exp, t, chisq, f, unif.

Examples: `pnorm(1.96) = 0.975` ($P(Z < 1.96)$); `qnorm(0.975) = 1.96`; `dbinom(3, 10, 0.5) = P(X=3)`; `rnorm(1000, mean = 50, sd = 10)` simulates data. This quartet covers every distribution uniformly.

Applications: computing p-values, critical values, confidence intervals, and simulating sampling distributions - the computational backbone of inference.

Key Points

- d/p/q/r + distribution root: density, CDF, quantile, random.
- One system covers normal, binomial, t, chi-square, etc.
- Basis for p-values, critical values, and simulation.

Key Terms

Density function: d*: probability density/mass at a point.

Quantile function: q*: value at a given cumulative probability.

CDF: Cumulative distribution function: p*.

Example

- *at(0.975, df = 29) gives the t critical value for n = 30.*

Summary: The d/p/q/r system makes every distribution computable with one pattern.

Exam Tips

- ★ Very important: what d, p, q, r each do.
- ★ Remember qnorm/qt for critical values.

Topic 17: Random Number Generation and Seeds

Learning Objectives

- Explain pseudorandom generation
- Set seeds for reproducibility
- Sample from data

Key Concepts

Computers generate pseudorandom numbers - deterministic sequences that behave randomly. `set.seed(123)` fixes the sequence, making 'random' work reproducible - essential for homework, research, and debugging. Always set seeds before simulations.

r* functions simulate distributions; `sample()` draws from data: `sample(1:10, 3)` picks randomly, `replace = TRUE` allows repetition, `prob = weights` choices. Sampling underlies bootstrapping and cross-validation.

Check simulations: replicate means of rnorm samples to see the Central Limit Theorem emerge - simulation as a teaching and verification tool.

Key Points

- Pseudorandom sequences; `set.seed()` makes them reproducible.
- `sample()` draws from data with/without replacement.
- Simulation verifies theory (e.g. CLT).

Key Terms

Seed: The starting value fixing a pseudorandom sequence.

Pseudorandom: Deterministic but statistically random-like.

Sampling: Random selection from a population or dataset.

Example

- `set.seed(42); means <- replicate(1000, mean(rnorm(30))); hist(means)`

Summary: Seeds make randomness reproducible; simulation makes theory visible.

Exam Tips

- ★ Very important: why `set.seed()` matters.
- ★ Remember `sample()` arguments.

Topic 18: The Central Limit Theorem in Action

Learning Objectives

- State the CLT
- Demonstrate it by simulation
- Explain its practical importance

Key Concepts

The Central Limit Theorem: sample means approach normality as n grows, regardless of the population's shape - the foundation of confidence intervals and t-tests. R makes it visible: draw many samples from a skewed exponential population, plot their means, watch normality emerge by $n = 30$.

Simulation code: `means <- replicate(5000, mean(rexp(30, rate = 1))); hist(means)` shows the bell curve. Increase n to see tightening around the true mean (standard error = $sd/\sqrt{n}$).

Practical import: the CLT justifies normal-based inference for means with moderate samples even when data is skewed - but check: heavy tails and tiny samples need alternatives (bootstrap, nonparametric tests).

Key Points

- CLT: sample means become normal as n grows, any population shape.
- Simulation demonstrates it convincingly.
- Justifies normal inference for means; mind small samples.

Key Terms

Central Limit Theorem: Sample means' distribution tends to normal.

Standard error: The standard deviation of a statistic's sampling distribution.

Sampling distribution: The distribution of a statistic over repeated samples.

Example

- *Even from exponential data, means of 30 look normal.*

Summary: The CLT, seen through simulation, underwrites classical inference.

Exam Tips

- ★ Very important: CLT statement and its use.
- ★ Remember standard error formula.

Topic 19: Monte Carlo Simulation

Learning Objectives

- Explain Monte Carlo methods
- Design a simulation study
- Estimate probabilities by simulation

Key Concepts

Monte Carlo simulation estimates quantities by random sampling: simulate thousands of scenarios, average the outcomes. Uses: probabilities too complex for formulas, verifying theoretical results, and power analysis.

Design: define the data-generating process, choose replications (10,000+ for stable estimates), set seed, vectorize for speed (`replicate()` over loops), and summarize with means and quantiles. Report Monte Carlo error ($\text{sd}/\sqrt{B}$).

Example: estimate $P(\text{dice sum} > 9)$ by simulating 100,000 rolls. Power analysis: simulate experiments under assumed effects to find the sample size achieving 80% power - rigorous study design.

Key Points

- Monte Carlo: estimate by simulating many scenarios.
- Design: process, replications, seed, vectorize, summarize.
- Uses: complex probabilities, verification, power analysis.

Key Terms

Monte Carlo: Estimation via repeated random simulation.

Replication: One simulated experiment.

Power analysis: Finding sample size for desired detection probability.

Example

- `mean(replicate(100000, sum(sample(1:6, 2, TRUE)) > 9))`

Summary: Monte Carlo turns computing power into statistical answers.

Exam Tips

- ★ Key point: simulation design steps.
- ★ Remember Monte Carlo error.

Topic 20: Bootstrapping

Learning Objectives

- Explain the bootstrap idea
- Construct bootstrap confidence intervals
- Discuss bootstrap limitations

Key Concepts

Bootstrapping estimates sampling distributions by resampling the data with replacement: treat the sample as a pseudo-population, draw thousands of resamples, compute the statistic each time. The resamples' spread estimates the standard error - no formulas needed.

Percentile confidence intervals take the 2.5th and 97.5th percentiles of bootstrap statistics. It works for medians, correlations, and custom statistics where theory is hard.

Code pattern: `boot_means <- replicate(5000, mean(sample(x, replace = TRUE))); quantile(boot_means, c(0.025, 0.975))`. Limitations: needs representative samples (fails for tiny n or extremes), and assumes the sample reflects the population.

Key Points

- Bootstrap: resample with replacement to estimate sampling distributions.
- Percentile intervals from bootstrap quantiles.
- Needs decent samples; fails for extremes and tiny n.

Key Terms

Bootstrap: Resampling-based inference without formulas.

Resampling: Drawing samples from the observed data.

Percentile interval: CI from bootstrap distribution quantiles.

Example

- *Bootstrap CI for the median income when no formula exists.*

Summary: Bootstrapping computes inference from the data itself.

Exam Tips

- ★ Very important: the bootstrap idea in words.
- ★ Remember its limitations.

Unit 5: Hypothesis Testing in R

Topic 21: t-Tests: One, Two Sample, and Paired

Learning Objectives

- Perform t-tests in R
- Choose the right t-test
- Interpret output correctly

Key Concepts

`t.test()` handles all: `t.test(x, mu = 100)` (one-sample vs. a value), `t.test(a, b)` (two independent samples; `var.equal = TRUE` for pooled), `t.test(before, after, paired = TRUE)` (matched pairs). Output gives the statistic, df, p-value, and confidence interval - report the interval, not just p.

Choosing: one sample (vs. benchmark), independent (two groups), paired (same units twice). Check assumptions: approximate normality (robust for moderate n via CLT), independence; use `var.test()` or default Welch's test for unequal variances.

Interpretation: $p < 0.05$ rejects H_0 - but report effect size (mean difference with CI) and practical significance. A significant 0.1-mark difference rarely matters.

Key Points

- `t.test()` for one-sample, two-sample, and paired designs.
- Welch's default handles unequal variances; check normality.
- Report CIs and effect sizes, not just p-values.

Key Terms

p-value: Probability of data as extreme under H_0 .

Welch's t-test: Two-sample t-test without equal-variance assumption.

Effect size: The magnitude of the observed difference.

Example

- *`t.test(marks_A, marks_B)` compares two sections' performance.*

Summary: t-tests compare means; correct choice and full reporting make them trustworthy.

Exam Tips

- ★ Very important: which t-test for which design.
- ★ Remember Welch's default.

Topic 22: Nonparametric Alternatives

Learning Objectives

- Explain when parametric tests fail
- Perform Wilcoxon and Kruskal-Wallis tests
- Compare with parametric results

Key Concepts

When normality fails badly (heavy skew, outliers, tiny samples) or data is ordinal, nonparametric tests use ranks: `wilcox.test()` (Mann-Whitney for two samples; paired version for matched pairs), `kruskal.test()` (three+ groups - the nonparametric ANOVA).

These test medians/distributions rather than means and resist outliers. Power is slightly lower than t-tests when normality holds - the price of robustness.

Workflow: plot data, check assumptions (`shapiro.test()` for normality, though visual checks matter more), choose parametric or rank-based, and report medians with IQRs for nonparametric analyses. R makes switching one argument away.

Key Points

- Rank-based tests for non-normal, ordinal, or tiny samples.
- `wilcox.test()` for two samples; `kruskal.test()` for 3+ groups.
- Report medians/IQRs; slight power cost vs. t-tests.

Key Terms

Nonparametric test: A test without distributional assumptions.

Mann-Whitney test: Rank-based two-sample comparison.

Kruskal-Wallis test: Rank-based comparison of 3+ groups.

Example

- `kruskal.test(income ~ region, data = df)` for skewed incomes across regions.

Summary: Nonparametric tests trade a little power for robustness.

Exam Tips

- ★ Key point: when to go nonparametric.
- ★ Remember the test names.

Topic 23: ANOVA in R

Learning Objectives

- Perform one-way ANOVA
- Explain the F-test logic
- Run post-hoc comparisons

Key Concepts

ANOVA compares 3+ group means: `model <- aov(marks ~ section, data = df)`; `summary(model)` gives the F-test - the ratio of between-group to within-group variance. A significant F says some means differ, not which.

Post-hoc tests identify pairs: `TukeyHSD(model)` controls the family-wise error across all pairwise comparisons - never run many t-tests instead (inflated Type I error). Plot with boxplots plus the Tukey intervals.

Assumptions: normality within groups, equal variances (`bartlett.test()`), independence. For repeated measures, use `aov` with `Error()` terms or mixed models (`lme4`). Two-way ANOVA (`aov(y ~ A * B)`) tests interactions - whether effects combine non-additively.

Key Points

- `aov()` + `summary()` for the F-test; `TukeyHSD()` for pairwise comparisons.
- $F = \text{between-group} / \text{within-group variance}$.
- Check normality and equal variances; never replace with many t-tests.

Key Terms

ANOVA: Analysis of variance comparing 3+ means.

F-test: Variance-ratio test for group differences.

Tukey HSD: Post-hoc pairwise test controlling error rate.

Interaction: Non-additive combined effects.

Example

- `aov(yield ~ fertilizer, data = farms)`; `TukeyHSD(model)`

Summary: ANOVA tests overall group differences; Tukey pinpoints them honestly.

Exam Tips

- ★ Very important: why not many t-tests (error inflation).
- ★ Remember F-test logic.

Topic 24: Proportions and Power Analysis

Learning Objectives

- Test proportions in R
- Explain statistical power
- Compute sample sizes

Key Concepts

`prop.test()` tests proportions: `prop.test(60, 100, p = 0.5)` checks if 60% differs from half; two-sample versions compare groups. `binom.test()` gives exact small-sample tests with Clopper-Pearson intervals.

Power - the probability of detecting a real effect - depends on effect size, sample size, significance level, and variability. Underpowered studies waste resources; `power.t.test()` computes needed n : `power.t.test(delta = 5, sd = 10, power = 0.8)` returns the per-group sample size.

Design workflow: specify the minimum meaningful effect, estimate variability from pilots, set power 0.8, compute n , then collect data. R's power functions cover t-tests, proportions, ANOVA, and more.

Key Points

- `prop.test()` for proportions; `binom.test()` exact for small n .
- Power depends on effect, n , alpha, and variability.
- `power.t.test()` sizes studies before data collection.

Key Terms

Power: Probability of detecting a true effect.

Effect size: The magnitude to be detected.

Type II error: Missing a real effect; $1 - \text{power}$.

Example

- `power.t.test(delta = 5, sd = 10, power = 0.8)` for study planning.

Summary: Power analysis designs studies that can actually answer their questions.

Exam Tips

- ★ Key point: the four determinants of power.
- ★ Remember to size studies before collecting data.

Topic 25: Goodness of Fit and Normality Checks

Learning Objectives

- Test distributional fit
- Check normality visually and formally
- Decide on transformations

Key Concepts

Goodness-of-fit tests compare data to a distribution: `chisq.test()` for categorical fit; `shapiro.test()` for normality (best for $n < 5000$); `ks.test()` general two-sample or vs. theoretical.

But formal tests are oversensitive in large samples (trivial deviations 'significant') and weak in small ones - so lead with plots: `hist()` with normal curve overlay, and especially `qqnorm()`; `qqline()` - points

following the line mean normal. Interpretation skill beats test output.

Remedies for non-normality: transformations (log for right skew, sqrt for counts), or nonparametric methods. Document the check-and-decide process - it is part of honest analysis.

Key Points

- Lead with Q-Q plots; use tests as backup.
- `shapiro.test()` for normality; `qqnorm()` to see it.
- Transform (log/sqrt) or go nonparametric when needed.

Key Terms

Q-Q plot: Quantiles plotted against theoretical quantiles.

Shapiro-Wilk test: A formal test of normality.

Transformation: Re-expressing data to meet assumptions.

Example

- `aanom(loo(income))`: `aaline(loo(income))` checks *loo-normality*.

Summary: Visual checks plus judgment beat blind test reliance.

Exam Tips

- ★ Very important: Q-Q plot interpretation.
- ★ Remember log for right skew.

Unit 6: Regression and Reporting

Topic 26: Simple Linear Regression

Learning Objectives

- Fit regressions with `lm()`
- Interpret coefficients and R-squared
- Check regression assumptions

Key Concepts

`model <- lm(marks ~ hours, data = df)` fits $\text{marks} = a + b \cdot \text{hours}$. `summary(model)` reports coefficients (intercept, slope with t-tests), R-squared (variance explained), and the F-test. `predict(model, newdata)` forecasts.

Interpretation: the slope is the expected marks change per extra hour - with its confidence interval (`confint(model)`). R-squared near 1 means tight fit; but prediction intervals (wider than confidence intervals) honestly show forecast uncertainty.

Assumption checks via `plot(model)`: residuals vs. fitted (linearity, equal spread), Q-Q (normality), scale-location, and leverage plots. Violations suggest transformations or robust methods - diagnostics are not optional.

Key Points

- `lm(y ~ x)`; `summary()` gives coefficients, R-squared, tests.
- Slope = expected y change per x unit; report with CIs.
- `plot(model)` diagnostics check all assumptions.

Key Terms

R-squared: Proportion of variance explained by the model.

Residual: Observed minus fitted value.

Leverage: An observation's influence on the fit.

Example

- *lm(sales ~ advertising, data = df): slope = extra sales per ad rupee.*

Summary: Regression quantifies relationships; diagnostics keep it honest.

Exam Tips

- ★ Very important: slope interpretation.
- ★ Remember the four diagnostic plots.

Topic 27: Multiple Regression

Learning Objectives

- Fit and interpret multiple regression
- Explain multicollinearity
- Select variables sensibly

Key Concepts

`model <- lm(y ~ x1 + x2 + x3, data = df)` estimates each predictor's effect holding others constant - the power of multiple regression. Coefficients are partial effects; factor predictors enter as dummy variables automatically.

Multicollinearity - correlated predictors - inflates standard errors (check `vif()` from `car`); remedies: drop, combine, or regularize. Never interpret correlated predictors' coefficients naively.

Variable selection: theory first, then compare models with adjusted R-squared, AIC (`AIC(model)` - lower better), and cross-validation. Avoid stepwise fishing - it overfits. Report the final model with CIs and diagnostics, and validate on held-out data when possible.

Key Points

- Coefficients are partial effects, holding others constant.
- Multicollinearity inflates SEs; check VIF.
- Select via theory + AIC/cross-validation, not stepwise fishing.

Key Terms

Partial effect: A predictor's effect controlling for others.

Multicollinearity: Correlated predictors destabilizing estimates.

AIC: A criterion balancing fit against complexity.

Example

- *lm(price ~ area + bedrooms + location, data = houses)*

Summary: Multiple regression isolates effects; careful selection keeps models credible.

Exam Tips

- ★ Very important: partial-effect interpretation.

★ Remember VIF for multicollinearity. 

Topic 28: Logistic Regression

Learning Objectives

- Explain when logistic regression applies
- Fit models with `glm()`
- Interpret odds ratios

Key Concepts

For binary outcomes (pass/fail, buy/not), `lm()` fails - use `glm(y ~ x, family = binomial)`: it models log-odds linearly. Coefficients exponentiate to odds ratios: $\exp(\text{coef})$ = how odds multiply per unit x .

Output: z-tests for coefficients, null vs. residual deviance (fit improvement), and AIC. `predict(model, type = 'response')` gives probabilities; classify with thresholds (usually 0.5) and evaluate with confusion matrices.

Assumptions: linear log-odds, independent observations, no perfect separation. Logistic regression powers credit scoring, medical diagnosis, and churn prediction - classification everywhere.

Key Points

- `glm(y ~ x, family = binomial)` for binary outcomes.
- $\exp(\text{coefficients})$ = odds ratios.
- Evaluate with confusion matrices, not just significance.

Key Terms

Odds ratio: Multiplicative change in odds per unit predictor.

Deviance: Lack-of-fit measure for GLMs.

Confusion matrix: Predicted vs. actual classification counts.

Example

- `glm(admit ~ are + apa, family = binomial, data = arad)`

Summary: Logistic regression models probabilities for yes/no outcomes.

Exam Tips

- ★ Very important: odds ratio interpretation.
- ★ Remember family = binomial.

Topic 29: Reproducible Reports with R Markdown

Learning Objectives

- Explain reproducible reporting
- Write R Markdown documents
- Generate HTML/PDF reports

Key Concepts

R Markdown weaves code, output, and prose: `{r}` chunks execute and embed results; text uses Markdown (`#` headings, `**bold**`). Knitting renders to HTML, PDF, or Word - one source, many outputs, always in sync with the data.

YAML headers set title, author, output format. Chunk options control behavior: `echo = FALSE` hides code, `warning = FALSE` suppresses messages, `fig.width` sizes plots. This ends copy-paste errors - numbers in the report literally come from the analysis.

Workflows: parameterized reports (knit per region), scheduled rendering, and version control (Git) complete reproducibility. An analysis nobody can rerun is barely an analysis.

Key Points

- R Markdown: code + prose knitted to HTML/PDF/Word.
- Chunks with options control code/output display.
- Reproducibility ends copy-paste errors.

Key Terms

R Markdown: A format combining R code with narrative text.

Chunk: A code block executed during knitting.

YAML header: Document metadata at the file's top.

Example

- *A monthly sales report knitted fresh from the database each month.*

Summary: R Markdown makes reports that are always in sync with their analysis.

Exam Tips

- ★ Key point: why reproducibility matters.
- ★ Remember chunk options.

Topic 30: Functions, Debugging, and Best Practices

Learning Objectives

- Write robust functions
- Debug systematically
- Follow coding best practices

Key Concepts

Robust functions validate inputs (`stopifnot()`), handle edge cases (empty vectors, NAs), and return consistent types. Document with roxygen2 comments (`#' @param`) - future-you will be grateful.

Debugging: `traceback()` after errors, `debug(fun)` stepping, `browser()` breakpoints, and RStudio's debugger. Most bugs are wrong object types (`str()` reveals), scoping surprises, or vector-length mismatches - check assumptions early.

Best practices: the tidyverse style guide (`snake_case`, spacing), projects with `here::here()` paths, version control, small tested functions, and commenting why (not what). Professional R code is readable, tested, and shared.

Key Points

- Validate inputs; handle edge cases; document functions.
- Debug with `traceback()`, `debug()`, `str()` checks.
- Style guides, projects, and version control professionalize code.

Key Terms

roxygen2: Documentation system for R functions.

traceback(): Shows the call stack after an error.

Style guide: Conventions for readable code.

Example

- *stopifnot(is.numeric(x), length(x) > 0) guards a function.*

Summary: Professional R means robust, debugged, readable, and versioned code.

Exam Tips

- ★ Key point: debugging workflow.
- ★ Remember input validation.

Glossary of Key Terms

Vector: R's fundamental ordered collection of same-type elements.

Data frame: The standard table-like dataset structure.

dplyr: Package for intuitive data manipulation.

ggplot2: Package implementing the grammar of graphics.

p-value: Probability of data as extreme as observed under H_0 .

t-test: A test comparing means using the t-distribution.

ANOVA: Analysis of variance comparing three or more means.

Regression: Modeling a response as a function of predictors.

Bootstrap: Resampling-based inference without distributional formulas.

R Markdown: Reproducible reports combining code and narrative.

CRAN: The Comprehensive R Archive Network of packages.

Seed: The value fixing a pseudorandom sequence for reproducibility.

lm(): R's linear modeling function.

Factor: A categorical variable with defined levels.

Pipe: The `%>%` operator chaining data operations.