R: A Comprehensive Introduction
From Basics to Data Science

Diogo Ribeiro

ESMAD - Escola Superior de Média Arte e Design
Instituto Politécnico do Porto

November 13, 2025

Learning Objectives

By the end of this session, you will be able to:

- Understand R's core data structures and operations
- Perform data manipulation and analysis
- Create visualizations and statistical models
- Apply modern R practices and workflows
- Use the tidyverse ecosystem effectively
- Implement reproducible research practices

Course Outline

Outline

R: Language for Data Science

- **Statistical computing language** developed by Ross Ihaka and Robert Gentleman
- **Open-source** and free
- **Functional programming** paradigm
- **Interpreted language** with REPL interface
- **Cross-platform**: Linux, macOS, Windows
- **Active community** with 18,000+ packages

Current version: R 4.3.x

R vs Python vs Other Tools

Feature	**R**	**Python**	**SPSS/SAS**
Statistical Analysis	★★★	★★☆	★★★
Machine Learning	★★☆	★★★	★☆☆
Data Visualization	★★★	★★☆	★★☆
Learning Curve	★★☆	★★★	★☆☆
Cost	Free	Free	Expensive
Community	Large	Huge	Commercial

Getting Started: Installation

1. **Install R**: Download from `https://cran.r-project.org/`
2. **Install RStudio**: Download from `https://posit.co/products/open-source/rstudio/`
3. **Alternative IDEs**: VS Code with R extension, Jupyter notebooks

First commands to try:

```r
# Check R version
version

# Get help
?mean

# Install essential packages
install.packages(c("tidyverse", "here", "rmarkdown"))
```

RStudio Interface Overview

- **Console**: Interactive R session
- **Source**: Script editor
- **Environment**: Variables and objects
- **Files/Plots/Packages**: File browser, plots, help

Essential keyboard shortcuts:

```
# Execute current line/selection: Ctrl+Enter
# New R script: Ctrl+Shift+N
# Save: Ctrl+S
# Clear console: Ctrl+L
# Assignment operator: Alt+-
```

Outline

Atomic Data Types

```r
# Numeric (double)
x <- 42.5
y <- 1e-3

# Integer
n <- 42L

# Logical (boolean)
is_valid <- TRUE
is_missing <- FALSE

# Character (string)
name <- "Alice"
greeting <- 'Hello,␣World!'

# Check types
typeof(x)     # "double"
class(x)      # "numeric"
is.numeric(x) # TRUE
```

Type Coercion and Testing

```r
# Automatic coercion
x <- c(1, 2, "3")      # becomes character vector
y <- c(TRUE, FALSE, 1) # becomes numeric vector

# Explicit coercion
as.numeric("123")      # 123
as.character(123)      # "123"
as.logical(c(0, 1, 2)) # FALSE TRUE TRUE

# Type testing
is.numeric(x)   # FALSE
is.character(x) # TRUE
is.na(x)        # FALSE FALSE FALSE
```

Creating and Manipulating Vectors

```r
# Creating vectors
numbers <- c(10.4, 5.6, 3.1, 6.4, 21.7)
sequence <- 1:10
custom_seq <- seq(0, 1, by = 0.1)
repeated <- rep(c(1, 2, 3), times = 3)
repeated_each <- rep(c(1, 2, 3), each = 3)

# Vector operations (vectorized!)
numbers * 2
numbers + c(1, 2)  # recycling
sqrt(numbers)
log10(numbers)

# Length and summary
length(numbers)
summary(numbers)
```

Vector Indexing and Subsetting

```r
x <- c(10, 20, 30, 40, 50)
names(x) <- c("a", "b", "c", "d", "e")

# Positional indexing
x[1]         # first element: 10
x[c(1, 3)]   # elements 1 and 3: 10 30
x[-1]        # all except first: 20 30 40 50

# Logical indexing
x[x > 25]    # elements > 25: 30 40 50
x[x %% 20 == 0]  # divisible by 20: 20 40

# Named indexing
x["a"]       # element "a": 10
x[c("a", "c")]  # elements "a" and "c"
```

Missing Values and Special Values

```r
# Missing values
x <- c(1, 2, NA, 4, 5)
is.na(x)              # FALSE FALSE TRUE FALSE FALSE
sum(x)                # NA
sum(x, na.rm = TRUE)  # 12

# Special values
x <- c(0, 1/0, -1/0, 0/0)
is.finite(x)          # TRUE FALSE FALSE FALSE
is.infinite(x)        # FALSE TRUE TRUE FALSE
is.nan(x)             # FALSE FALSE FALSE TRUE

# Handling missing data
x[is.na(x)] <- 0    # replace NA with 0
complete.cases(x)   # check for complete observations
```

Matrices: 2D Homogeneous Data

```r
# Creating matrices
A <- matrix(1:12, nrow = 3, ncol = 4)
B <- matrix(1:12, nrow = 3, ncol = 4, byrow = TRUE)

# Matrix operations
t(A)          # transpose
A + B         # element-wise addition
A %*% t(B)    # matrix multiplication

# Indexing
A[2, 3]       # row 2, column 3
A[2, ]        # entire row 2
A[, 3]        # entire column 3
A[1:2, 2:4]   # submatrix

# Matrix properties
dim(A)        # dimensions
nrow(A); ncol(A)
```

Factors: Categorical Data

```r
# Creating factors
gender <- factor(c("M", "F", "F", "M", "F"))
education <- factor(c("High", "Low", "Medium", "High"),
                    levels = c("Low", "Medium", "High"),
                    ordered = TRUE)

# Factor properties
levels(gender)     # "F" "M"
nlevels(gender)    # 2
table(gender)      # frequency table

# Recoding factors
levels(gender) <- c("Female", "Male")
gender <- relevel(gender, ref = "Male")  # set reference level

# Converting factors
as.character(gender)
as.numeric(education)  # 3 1 2 3 (ordered levels)
```

Lists: Heterogeneous Containers

```r
# Creating lists
person <- list(
  name = "Alice",
  age = 30,
  scores = c(85, 92, 78),
  married = TRUE,
  children = c("Bob", "Carol")
)

# Accessing list elements
person$name           # "Alice"
person[["age"]]       # 30
person["scores"]      # returns list
person[["scores"]]    # returns vector

# Modifying lists
person$city <- "Lisbon"
person[["age"]] <- 31

```

Data Frames: The Workhorse

```r
# Creating data frames
students <- data.frame(
  id = 1:5,
  name = c("Alice", "Bob", "Carol", "David", "Eve"),
  age = c(22, 23, 21, 24, 22),
  grade = c("A", "B", "A", "C", "B"),
  passed = c(TRUE, TRUE, TRUE, FALSE, TRUE),
  stringsAsFactors = FALSE
)

# Exploring structure
str(students)          # structure
head(students)         # first few rows
tail(students, 2)      # last 2 rows
summary(students)      # summary statistics
dim(students)          # dimensions
```

Data Frame Manipulation

```r
# Accessing columns
students$name          # vector
students["name"]       # data frame
students[["name"]]     # vector
students[, "name"]     # vector

# Subsetting rows and columns
students[1:3, ]        # first 3 rows
students[, c("name", "age")]  # specific columns
students[students$age > 22, ] # conditional subsetting

# Adding/removing columns
students$gpa <- c(3.8, 3.2, 3.9, 2.1, 3.5)
students$grade <- NULL  # remove column

# Sorting
students[order(students$age), ]                        # by age
students[order(-students$gpa, students$age), ]         # by GPA (desc), then age
```

Hands-on Exercise #1 (15 min)

Practice with data structures:

1. Create a vector of 20 random normal values with mean=100, sd=15
2. Convert values below 85 or above 115 to NA
3. Create a factor for letter grades based on the values:
   - A: 110+, B: 95–109, C: 85–94, F: below 85
4. Build a data frame combining the original scores, grades, and pass/fail status
5. Calculate summary statistics for each grade level

Outline

Base R I/O Functions

```r
# CSV files
data <- read.csv("data.csv", header = TRUE, stringsAsFactors = FALSE)
write.csv(data, "output.csv", row.names = FALSE)

# Tab-delimited files
data <- read.delim("data.txt", sep = "\t")

# General text files
data <- read.table("data.txt", header = TRUE, sep = ",")

# R objects
save(data, file = "data.RData")       # save specific objects
save.image("workspace.RData")         # save entire workspace
load("data.RData")                    # load objects

# Other formats (requires packages)
library(readxl)
excel_data <- read_excel("data.xlsx", sheet = "Sheet1")
```

Modern Data Import with readr

```r
library(readr)

# Fast and consistent parsing
data <- read_csv("data.csv")          # tibbles by default
data <- read_tsv("data.txt")
data <- read_delim("data.txt", delim = "|")

# Specify column types
data <- read_csv("data.csv",
                 col_types = cols(
                   id = col_integer(),
                   name = col_character(),
                   score = col_double(),
                   date = col_date()
                 ))

# Handle problematic data
problems(data)   # show parsing problems
data <- read_csv("data.csv", na = c("", "NA", "NULL"))
```

Data Inspection and Cleaning

```r
# Load built-in dataset
data(mtcars)

# Basic inspection
glimpse(mtcars)      # dplyr version of str()
summary(mtcars)
head(mtcars, 10)

# Check for missing values
sum(is.na(mtcars))
colSums(is.na(mtcars))

# Check for duplicates
sum(duplicated(mtcars))

# Data types
sapply(mtcars, class)

# Quick visualization
```

Working Directories and Projects

```r
# Working directory management
getwd()                # current directory
setwd("~/Documents/R_projects")  # NOT recommended!

# Better approach: Use RStudio Projects or here package
library(here)
data_path <- here("data", "raw", "dataset.csv")
output_path <- here("output", "results.csv")

# File operations
list.files(".", pattern = "*.csv")
file.exists("data.csv")
file.info("data.csv")

# Create directories
dir.create("data")
dir.create("output")
dir.create(here("data", "processed"))
```

Hands-on Exercise #2 (10 min)

Data import and exploration:

1. Load the built-in `iris` dataset
2. Explore its structure, dimensions, and summary statistics
3. Check for missing values and duplicates
4. Create a subset with only Setosa and Versicolor species
5. Export the subset to a CSV file
6. Reload the CSV and verify it matches your subset

Outline

Conditional Statements

```r
# Simple if-else
x <- 5
if (x > 0) {
  print("Positive")
} else if (x < 0) {
  print("Negative")
} else {
  print("Zero")
}

# Vectorized conditional: ifelse()
scores <- c(85, 92, 67, 88, 95)
grades <- ifelse(scores >= 90, "A",
                 ifelse(scores >= 80, "B",
                        ifelse(scores >= 70, "C", "F")))

# Multiple conditions with case_when() (dplyr)
library(dplyr)
grades <- case_when(
```

Loops and Iteration

```r
# For loops
for (i in 1:5) {
  print(i^2)
}

# Iterate over elements
fruits <- c("apple", "banana", "orange")
for (fruit in fruits) {
  print(paste("I like", fruit))
}

# While loops
x <- 1
while (x <= 5) {
  print(x)
  x <- x + 1
}

# Apply family (vectorized operations)
```

Creating Functions

```r
# Basic function
square <- function(x) {
  return(x^2)
}

# Function with multiple arguments and defaults
calculate_bmi <- function(weight, height, units = "metric") {
  if (units == "imperial") {
    # Convert pounds and inches to kg and meters
    weight <- weight * 0.453592
    height <- height * 0.0254
  }

  bmi <- weight / (height^2)

  category <- case_when(
    bmi < 18.5 ~ "Underweight",
    bmi < 25 ~ "Normal",
    bmi < 30 ~ "Overweight",
```

Advanced Function Features

```r
# Functions with ... (dot-dot-dot)
my_summary <- function(x, ...) {
  list(
    mean = mean(x, ...),
    median = median(x, ...),
    sd = sd(x, ...)
  )
}

# Usage with additional arguments
data_with_na <- c(1, 2, NA, 4, 5)
my_summary(data_with_na, na.rm = TRUE)

# Input validation
safe_divide <- function(x, y) {
  if (!is.numeric(x) || !is.numeric(y)) {
    stop("Both␣arguments␣must␣be␣numeric")
  }
  if (any(y == 0)) {
```

Functional Programming Concepts

```r
# Anonymous functions
numbers <- 1:10
squared <- sapply(numbers, function(x) x^2)

# Map functions (purrr package - part of tidyverse)
library(purrr)
numbers <- 1:5
squared <- map_dbl(numbers, ~ .x^2)
cubed <- map_dbl(numbers, ~ .x^3)

# Working with lists
data_list <- list(a = 1:3, b = 4:6, c = 7:9)
means <- map_dbl(data_list, mean)
lengths <- map_int(data_list, length)

# Function composition
compose_functions <- function(f, g) {
  function(x) f(g(x))
}
```

Hands-on Exercise #3 (15 min)

Programming practice:

1. Write a function `standardize()` that:
   - Takes a numeric vector
   - Returns z-scores (mean=0, sd=1)
   - Has options for removing NAs and clipping outliers
2. Write a function `grade_analysis()` that:
   - Takes a vector of numeric scores
   - Returns a list with mean, median, grade distribution
   - Assigns letter grades based on customizable cutoffs
3. Test your functions with simulated data

Outline

Tidyverse: Modern R Data Science

Core principles:

- **Tidy data**: Each variable is a column, each observation is a row
- **Pipe operator**: Chain operations with
- **`Consistent API`**`: Similar function names and arguments`
- **`Human-readable`**`: Code that reads like English`

Core packages:

- `dplyr`: Data manipulation
- `ggplot2`: Visualization
- `tidyr`: Data reshaping
- `readr`: Data import

The Pipe Operator

```r
library(dplyr)

# Traditional nested approach (hard to read)
result <- summarise(
  filter(
    select(mtcars, mpg, hp, wt),
    hp > 100
  ),
  mean_mpg = mean(mpg),
  mean_wt = mean(wt)
)

# Pipe approach (readable)
result <- mtcars %>%
  select(mpg, hp, wt) %>%
  filter(hp > 100) %>%
  summarise(
    mean_mpg = mean(mpg),
    mean_wt = mean(wt)
```

Selecting and Filtering

```r
library(dplyr)

# Select columns
mtcars %>%
  select(mpg, hp, wt)           # by name

mtcars %>%
  select(1:3)                   # by position

mtcars %>%
  select(starts_with("m"))      # helper functions

mtcars %>%
  select(-gear, -carb)          # exclude columns

# Filter rows
mtcars %>%
  filter(mpg > 20)              # single condition

```

Creating and Modifying Variables

```r
# Create new variables with mutate()
mtcars %>%
  mutate(
    power_to_weight = hp / wt,
    efficiency_class = case_when(
      mpg >= 25 ~ "High",
      mpg >= 20 ~ "Medium",
      TRUE ~ "Low"
    ),
    # Create multiple variables
    log_mpg = log(mpg),
    mpg_squared = mpg^2
  )

# Conditional mutations
mtcars %>%
  mutate(
    fuel_efficiency = ifelse(mpg > median(mpg), "Efficient", "Inefficient"),
    performance = case_when(
```

Grouping and Summarizing

```r
# Summary statistics
mtcars %>%
  summarise(
    mean_mpg = mean(mpg),
    median_hp = median(hp),
    sd_wt = sd(wt),
    count = n()
  )

# Grouped operations
mtcars %>%
  group_by(cyl) %>%
  summarise(
    count = n(),
    avg_mpg = mean(mpg),
    avg_hp = mean(hp),
    min_wt = min(wt),
    max_wt = max(wt),
    .groups = "drop"  # ungroup after summarizing
```

Arranging and Ranking

```r
# Sorting data
mtcars %>%
  arrange(mpg)                    # ascending

mtcars %>%
  arrange(desc(hp))               # descending

mtcars %>%
  arrange(cyl, desc(mpg))         # multiple columns

# Window functions
mtcars %>%
  mutate(
    mpg_rank = rank(mpg),
    mpg_dense_rank = dense_rank(mpg),
    mpg_percentile = percent_rank(mpg),
    row_number = row_number()
  ) %>%
  arrange(desc(mpg))
```

Joins and Combining Data

```r
# Sample datasets
cars_info <- data.frame(
  model = rownames(mtcars)[1:10],
  manufacturer = c("Mazda", "Mazda", "Datsun", "Hornet", "Hornet",
                   "Valiant", "Duster", "Merc", "Merc", "Merc"),
  stringsAsFactors = FALSE
)

mtcars_with_names <- mtcars %>%
  mutate(model = rownames(mtcars))

# Different types of joins
inner_join(mtcars_with_names, cars_info, by = "model")
left_join(mtcars_with_names, cars_info, by = "model")
right_join(mtcars_with_names, cars_info, by = "model")
full_join(mtcars_with_names, cars_info, by = "model")

# Binding rows and columns
bind_rows(mtcars[1:5, ], mtcars[25:32, ])
```

Hands-on Exercise #4 (20 min)

Advanced dplyr practice:

1. Load the `starwars` dataset from dplyr
2. Clean the data:
   - Remove rows with missing height or mass
   - Create BMI variable: mass / height^2 $\times$ 10000
3. Analysis tasks:
   - Find the average height and mass by species (top 5 species by count)
   - Identify characters with extreme BMI values
   - Create a summary by homeworld showing character count and avg BMI
4. Export your results to CSV

Outline

Exploratory Data Analysis

```r
# Load and explore a dataset
library(datasets)
data("airquality")

# Basic descriptive statistics
summary(airquality)
sapply(airquality, function(x) c(mean = mean(x, na.rm = TRUE),
                                 sd = sd(x, na.rm = TRUE),
                                 min = min(x, na.rm = TRUE),
                                 max = max(x, na.rm = TRUE)))

# Correlation matrix
cor(airquality, use = "complete.obs")

# Advanced descriptive statistics
library(psych)
describe(airquality)
pairs.panels(airquality[1:4])  # correlation plot with histograms
```

Handling Missing Data

```r
# Identify missing patterns
library(VIM)
aggr(airquality, col = c('navyblue', 'red'),
     numbers = TRUE, sortVars = TRUE)

# Simple imputation strategies
# Mean imputation
airquality_mean <- airquality %>%
  mutate(
    Ozone = ifelse(is.na(Ozone), mean(Ozone, na.rm = TRUE), Ozone),
    Solar.R = ifelse(is.na(Solar.R), mean(Solar.R, na.rm = TRUE), Solar.R)
  )

# Multiple imputation
library(mice)
imputed_data <- mice(airquality, m = 5, method = 'pmm', seed = 123)
completed_data <- complete(imputed_data)
```

Hypothesis Testing

```r
# t-tests

# One-sample t-test
t.test(airquality$Temp, mu = 75)

# Two-sample t-test
# Split data by month
summer_temp <- airquality$Temp[airquality$Month %in% c(6, 7, 8)]
other_temp <- airquality$Temp[!airquality$Month %in% c(6, 7, 8)]
t.test(summer_temp, other_temp)

# Paired t-test (example with before/after data)
# before <- c(85, 78, 82, 79, 88)
# after <- c(87, 80, 85, 81, 92)
# t.test(before, after, paired = TRUE)

# Chi-square test
# Create categorical variables for example
temp_cat <- cut(airquality$Temp, breaks = 3, labels = c("Cool", "Moderate", "Hot"))
```

ANOVA and Non-parametric Tests

```r
# One-way ANOVA
anova_result <- aov(Temp ~ factor(Month), data = airquality)
summary(anova_result)

# Post-hoc tests
TukeyHSD(anova_result)

# Two-way ANOVA (if we had more factors)
# aov(Temp ~ Month * Wind_Category, data = airquality)

# Non-parametric alternatives
# Kruskal-Wallis test (non-parametric ANOVA)
kruskal.test(Temp ~ Month, data = airquality)

# Wilcoxon rank-sum test (non-parametric t-test)
wilcox.test(summer_temp, other_temp)

# Correlation tests
cor.test(airquality$Temp, airquality$Ozone, use = "complete.obs")
```

Simple and Multiple Regression

```r
# Simple linear regression
model1 <- lm(Ozone ~ Temp, data = airquality)
summary(model1)

# Multiple regression
model2 <- lm(Ozone ~ Temp + Wind + Solar.R, data = airquality)
summary(model2)

# Model with interactions
model3 <- lm(Ozone ~ Temp * Wind + Solar.R, data = airquality)

# Polynomial regression
model4 <- lm(Ozone ~ poly(Temp, 2) + Wind + Solar.R, data = airquality)

# Model comparison
anova(model1, model2)  # F-test for nested models
AIC(model1, model2, model3, model4)  # Information criteria
```

Model Diagnostics and Validation

```r
# Basic diagnostic plots
par(mfrow = c(2, 2))
plot(model2)
par(mfrow = c(1, 1))

# Residual analysis
residuals <- resid(model2)
fitted_vals <- fitted(model2)

# Check assumptions
# 1. Linearity
plot(fitted_vals, residuals)
abline(h = 0, col = "red")

# 2. Normality of residuals
qqnorm(residuals)
qqline(residuals)
shapiro.test(residuals)  # formal test

```

Prediction and Model Selection

```r
# Predictions with confidence intervals
new_data <- data.frame(
  Temp = c(70, 80, 90),
  Wind = c(10, 15, 5),
  Solar.R = c(200, 250, 300)
)

predictions <- predict(model2, newdata = new_data, interval = "confidence")

# Cross-validation
library(caret)
set.seed(123)
train_control <- trainControl(method = "cv", number = 10)
cv_model <- train(Ozone ~ Temp + Wind + Solar.R,
                  data = airquality,
                  method = "lm",
                  trControl = train_control,
                  na.action = na.omit)
print(cv_model)
```

Generalized Linear Models

```r
# Logistic regression
# Create binary outcome
airquality$high_ozone <- ifelse(airquality$Ozone > median(airquality$Ozone, na.rm = TRUE),
    1, 0)

logistic_model <- glm(high_ozone ~ Temp + Wind + Solar.R,
                      data = airquality,
                      family = binomial)
summary(logistic_model)

# Odds ratios
exp(coef(logistic_model))
exp(confint(logistic_model))

# Model evaluation
library(pROC)
predicted_prob <- predict(logistic_model, type = "response")
roc_curve <- roc(airquality$high_ozone, predicted_prob, na.rm = TRUE)
plot(roc_curve)
```

Hands-on Exercise #5 (25 min)

Statistical modeling project:

1. Use the `mtcars` dataset for regression analysis
2. Exploratory analysis:
   - Descriptive statistics and correlations
   - Identify outliers and missing values
3. Build models to predict `mpg`:
   - Simple regression with `wt`
   - Multiple regression with `wt`, `hp`, `disp`
   - Model with interactions
4. Model evaluation:
   - Check assumptions with diagnostic plots
   - Compare models using AIC/BIC
   - Calculate R^2 and RMSE

Outline

The Philosophy of ggplot2

Grammar of Graphics principles:

- **Data**: The dataset being plotted
- **Aesthetics**: Visual properties (x, y, color, size, shape)
- **Geometries**: The type of plot (points, lines, bars)
- **Scales**: How aesthetic values are displayed
- **Coordinate systems**: How data is positioned
- **Facets**: Subplots based on categorical variables
- **Themes**: Overall visual appearance

Basic structure: `ggplot(data, aes(x, y)) + geom_*() + theme() + ...`

Basic ggplot2 Syntax

```r
library(ggplot2)

# Basic scatter plot
ggplot(mtcars, aes(x = wt, y = mpg)) +
  geom_point()

# Add aesthetics
ggplot(mtcars, aes(x = wt, y = mpg, color = factor(cyl))) +
  geom_point(size = 3) +
  labs(title = "Fuel␣Efficiency␣vs␣Weight",
       x = "Weight␣(1000␣lbs)",
       y = "Miles␣per␣Gallon",
       color = "Cylinders")

# Add trend line
ggplot(mtcars, aes(x = wt, y = mpg)) +
  geom_point() +
  geom_smooth(method = "lm", se = TRUE)
```

Distribution Plots

```r
# Histogram
ggplot(mtcars, aes(x = mpg)) +
  geom_histogram(bins = 15, fill = "skyblue", alpha = 0.7) +
  labs(title = "Distribution␣of␣MPG")

# Density plot
ggplot(mtcars, aes(x = mpg, fill = factor(cyl))) +
  geom_density(alpha = 0.5) +
  labs(title = "MPG␣Distribution␣by␣Cylinders")

# Box plot
ggplot(mtcars, aes(x = factor(cyl), y = mpg)) +
  geom_boxplot(fill = "lightblue") +
  geom_jitter(width = 0.2, alpha = 0.6) +  # add data points
  labs(x = "Number␣of␣Cylinders", y = "Miles␣per␣Gallon")

# Violin plot
ggplot(mtcars, aes(x = factor(cyl), y = mpg)) +
  geom_violin(fill = "lightgreen", alpha = 0.7) +
```

Categorical Data Visualization

```r
# Bar chart
mtcars %>%
  count(cyl) %>%
  ggplot(aes(x = factor(cyl), y = n)) +
  geom_bar(stat = "identity", fill = "coral") +
  labs(x = "Cylinders", y = "Count")

# Grouped bar chart
mtcars %>%
  count(cyl, am) %>%
  ggplot(aes(x = factor(cyl), y = n, fill = factor(am))) +
  geom_bar(stat = "identity", position = "dodge") +
  labs(x = "Cylinders", y = "Count", fill = "Transmission")

# Stacked bar chart
mtcars %>%
  count(cyl, am) %>%
  ggplot(aes(x = factor(cyl), y = n, fill = factor(am))) +
  geom_bar(stat = "identity", position = "stack") +
```

Advanced Plot Types

```r
# Correlation heatmap
library(reshape2)
cor_matrix <- cor(mtcars)
melted_cor <- melt(cor_matrix)

ggplot(melted_cor, aes(Var1, Var2, fill = value)) +
  geom_tile() +
  scale_fill_gradient2(low = "blue", high = "red", mid = "white",
                       midpoint = 0, limit = c(-1,1)) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

# Scatter plot matrix
library(GGally)
ggpairs(mtcars[c("mpg", "disp", "hp", "wt")])

# Time series plot (example with built-in data)
economics %>%
  ggplot(aes(x = date, y = unemploy)) +
  geom_line(color = "steelblue") +
```

Scales and Coordinate Systems

```r
# Custom scales
ggplot(mtcars, aes(x = wt, y = mpg, color = hp)) +
  geom_point(size = 3) +
  scale_color_gradient(low = "blue", high = "red") +
  scale_x_continuous(breaks = seq(1, 6, by = 0.5)) +
  scale_y_continuous(limits = c(10, 35))

# Log scales
ggplot(mtcars, aes(x = disp, y = mpg)) +
  geom_point() +
  scale_x_log10() +
  annotation_logticks(sides = "b")

# Coordinate transformations
ggplot(mtcars, aes(x = factor(cyl), y = mpg)) +
  geom_boxplot() +
  coord_flip()  # horizontal box plot

# Polar coordinates (for pie charts)
```

Faceting and Multiple Plots

```r
# Facet wrap
ggplot(mtcars, aes(x = wt, y = mpg)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE) +
  facet_wrap(~ cyl, scales = "free")

# Facet grid
ggplot(mtcars, aes(x = wt, y = mpg)) +
  geom_point() +
  facet_grid(am ~ cyl, labeller = label_both)

# Multiple plots with patchwork
library(patchwork)
p1 <- ggplot(mtcars, aes(x = wt, y = mpg)) + geom_point()
p2 <- ggplot(mtcars, aes(x = hp, y = mpg)) + geom_point()
p3 <- ggplot(mtcars, aes(x = factor(cyl), y = mpg)) + geom_boxplot()

(p1 | p2) / p3  # combine plots
```

Themes and Styling

```r
# Built-in themes
base_plot <- ggplot(mtcars, aes(x = wt, y = mpg)) +
  geom_point(aes(color = factor(cyl))) +
  labs(title = "Fuel␣Efficiency␣vs␣Weight")

base_plot + theme_minimal()
base_plot + theme_classic()
base_plot + theme_dark()

# Custom theme
custom_theme <- theme(
  plot.title = element_text(size = 16, face = "bold", hjust = 0.5),
  axis.title = element_text(size = 12),
  axis.text = element_text(size = 10),
  legend.position = "bottom",
  panel.grid.minor = element_blank(),
  plot.background = element_rect(fill = "white", color = NA)
)

```

Hands-on Exercise #6 (20 min)

Data visualization project:

1. Use the `diamonds` dataset from ggplot2
2. Create a comprehensive visualization dashboard:
   - Price distribution histogram with faceting by cut
   - Scatter plot of carat vs price, colored by clarity
   - Box plot of price by cut quality
   - Correlation heatmap of numeric variables
3. Customize your plots:
   - Apply consistent color schemes
   - Add informative titles and labels
   - Use a professional theme
4. Combine plots into a single figure using patchwork

Outline

R Markdown for Reproducible Reports

~~~
# YAML header example
---
title: "Data␣Analysis␣Report"
author: "Your␣Name"
date: "`r␣Sys.Date()`"
output:
  html_document:
    toc: true
    toc_float: true
    code_folding: hide
    theme: flatly
---

# Analysis Overview

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE, warning = FALSE, message = FALSE)
library(tidyverse)
library(knitr)
~~~

Project Organization and here Package

```r
# Recommended project structure
# project/
#   data/
#      raw/
#      processed/
#   R/
#      functions.R
#      analysis.R
#   output/
#      plots/
#      tables/
#   docs/
#   project.Rproj

# Using the here package for robust file paths
library(here)

# Reading data
raw_data <- read_csv(here("data", "raw", "dataset.csv"))
```

Creating R Packages

```r
# Create a package skeleton
library(devtools)
library(usethis)

# Create new package
create_package("~/mypackage")

# Add functions
use_r("my_function")

# Example function with roxygen2 documentation
#' Calculate summary statistics
#'
#' @param x A numeric vector
#' @param na.rm Logical, should missing values be removed?
#' @return A named vector of summary statistics
#' @export
#' @examples
#' my_summary(c(1, 2, 3, NA), na.rm = TRUE)
```

Git Integration in RStudio

```r
# Initialize git in your project
use_git()

# Connect to GitHub
use_github()

# Basic git workflow in R console
# (Better to use RStudio's Git pane or terminal)

# Check status
system("git status")

# Add files
system("git add .")

# Commit changes
system('git commit -m "Add data analysis script"')

# Push to remote
```

Writing Efficient R Code

```r
# Vectorization vs loops
# Slow
result <- numeric(1000)
for (i in 1:1000) {
  result[i] <- i^2
}

# Fast
result <- (1:1000)^2

# Pre-allocate memory
# Slow: growing objects
x <- numeric(0)
for (i in 1:1000) {
  x <- c(x, i^2)  # Bad!
}

# Fast: pre-allocation
x <- numeric(1000)
```

Debugging and Profiling

```r
# Debugging functions
# (works only in an interactive R session)
debug(my_function)
undebug(my_function)

# Browser for interactive debugging
my_function <- function(x) {
  y <- x^2
  browser()  # Pause execution here
  z <- y + 1
  return(z)
}

# Profiling code performance
library(dplyr)
library(profvis)

profvis({
  data <- data.frame(x = rnorm(1000), y = rnorm(1000))
```

Final Exercise: Complete Project (30 min)

Comprehensive data science project:

1. **Setup**: Create an RStudio project with proper folder structure
2. **Data**: Load and clean a real dataset (e.g., from `datasets` package)
3. **Analysis**:
   - Exploratory data analysis with summary statistics
   - Statistical tests or regression modeling
   - Data visualization with multiple plot types
4. **Report**: Create an R Markdown document with:
   - Introduction and methodology
   - Results with embedded plots and tables
   - Conclusions and interpretation
5. **Reproducibility**: Ensure all code runs from scratch

Outline

Advanced R Topics to Explore

Statistical Methods:

- Machine learning with `caret`, `mlr3`
- Time series analysis with `forecast`
- Survival analysis with `survival`
- Bayesian analysis with `brms`, `rstanarm`

Specialized Domains:

- Bioinformatics with `Bioconductor`
- Spatial analysis with `sf`, `terra`
- Text mining with `tidytext`, `quanteda`
- Web scraping with `rvest`

Advanced Programming:

- Object-oriented programming (S3, S4, R6)
- Parallel computing with `parallel`, `future`
- C++ integration with `Rcpp`
- Shiny web applications

Data Engineering:

- Big data with `sparklyr`
- Databases with `DBI`, `dbplyr`
- APIs with `httr`, `jsonlite`
- Cloud computing integration

Learning Resources

Books:

- *R for Data Science* by Wickham & Grolemund
- *Advanced R* by Hadley Wickham
- *R Packages* by Wickham & Bryan
- *An Introduction to Statistical Learning with R* by James et al.

Online Resources:

- RStudio Education: `https://education.rstudio.com/`
- R-bloggers: `https://www.r-bloggers.com/`
- CRAN Task Views: `https://cran.r-project.org/web/views/`
- R Weekly: `https://rweekly.org/`

Communities:

- R Community on Twitter: `#rstats`
- Stack Overflow R tag
- Local R User Groups (R-Ladies, etc.)

Outline

What We've Covered

- **R Fundamentals**: Data types, structures, and basic operations
- **Data Management**: Import/export, cleaning, and manipulation
- **Programming**: Control flow, functions, and best practices
- **Modern R**: tidyverse ecosystem and pipe operator
- **Statistical Analysis**: Descriptive stats, hypothesis testing, regression
- **Visualization**: ggplot2 and the grammar of graphics
- **Workflows**: Reproducible research, project organization
- **Advanced Topics**: Package development, version control

Key Takeaways

1. **Think in vectors**: R is designed for vectorized operations
2. **Embrace the tidyverse**: Modern R is more readable and consistent
3. **Reproducibility matters**: Use projects, scripts, and R Markdown
4. **Visualization is key**: ggplot2 is powerful and flexible
5. **Practice regularly**: The more you use R, the more natural it becomes
6. **Join the community**: R has an incredibly supportive user base

Next Steps for Your R Journey

Immediate actions:

- Set up your R environment with RStudio
- Complete the exercises from today's session
- Start a small project with your own data

Medium-term goals:

- Master the tidyverse ecosystem
- Learn advanced statistical methods relevant to your field
- Contribute to open-source R packages

Long-term development:

- Attend R conferences (useR!, RStudio Conference)
- Mentor others in R
- Consider R package development

Questions and Discussion

Questions?

Contact Information:

`dfr@esmad.ipp.pt`

Resources from today:

All code examples and exercises available on GitHub

Thank you for your attention!