Comprehensive Guide to Testing Data Analysis Workflows in R

Table of Contents

1. Introduction
2. Testing Philosophy for Data Analysis
3. Testing Framework Overview
4. Types of Tests for Data Analysis
5. Setting Up Your Testing Environment
6. Unit Testing for Analysis Functions
7. Integration Testing for Data Pipelines
8. Data Validation and Quality Testing
9. Reproducibility Testing
10. Testing Analysis Scripts
11. Testing Report Generation
12. Continuous Integration for Data Analysis
13. Best Practices and Common Pitfalls
14. Tools and Packages
15. Examples and Templates
16. References

Introduction

Testing data analysis workflows presents unique challenges compared to traditional software development. While package development focuses on testing isolated functions with predictable inputs and outputs, data analysis involves testing complex pipelines, validating data quality, ensuring reproducibility, and verifying that analytical results are correct and meaningful (Wilson et al., 2017; Wickham, 2015).

This guide provides comprehensive strategies for testing data analysis workflows in R, building upon established software engineering practices while addressing the specific needs of computational research and data science (Poldrack, 2019; Wilson et al., 2014).

Testing Philosophy for Data Analysis

Core Principles

Data analysis testing serves two primary goals:

1. **Computational Reproducibility**: Ensuring another researcher can use your code and data to independently obtain identical results
2. **Result Correctness**: Validating that the results generated by the code are accurate and meaningful

As noted by Wilson et al. (2017), "An important aspect of this is to validate the code using software development practices that prevent errors and software testing methods that can help detect them when they occur."

Three-Phase Testing Approach

Following the reproducible research workflow principles (Poldrack, 2019), testing should align with the three phases of data analysis:

- **Explore Phase**: Rapid prototyping with lightweight validation

- **Refine Phase**: Comprehensive testing of stabilized analysis components
- **Produce Phase**: Full integration and reproducibility testing

Testing Framework Overview

The testthat Package

The `testthat` package (Wickham, 2011) provides the foundation for testing in R. It offers:

- Structured test organization with `describe()` and `test_that()`
- Rich expectation functions (`expect_equal()`, `expect_error()`, etc.)
- Integration with R package development workflows
- Clear, informative test output

```r
library(testthat)

test_that("data loading works correctly", {
  data <- load_data("sample.csv")
  expect_true(nrow(data) > 0)
  expect_true("outcome" %in% names(data))
})
```

Extended Testing Ecosystem

For data analysis, additional packages complement `testthat`:

- **testdat**: Data validation testing (Hope, 2022)
- **validate**: Rule-based data validation (van der Loo & de Jonge, 2021)
- **assertr**: Assertive programming for data analysis pipelines (Fischetti, 2021)
- **targets**: Pipeline testing and validation (Landau, 2021)

Types of Tests for Data Analysis

1. Unit Tests

Test individual analysis functions in isolation:

```r
test_that("outlier detection function works", {
  test_data <- c(1, 2, 3, 100, 4, 5)
  outliers <- detect_outliers(test_data, method = "iqr")
  expect_equal(outliers, 100)
  expect_length(outliers, 1)
})
```

2. Integration Tests

Test how different components work together:

```r
test_that("complete data pipeline runs successfully", {
  # Test entire workflow from raw data to results
  expect_no_error({
    raw_data <- load_raw_data()
    clean_data <- clean_data(raw_data)
    model <- fit_model(clean_data)
    results <- generate_results(model)
  })
  expect_s3_class(results, "analysis_results")
  expect_true(results$converged)
})
```

3. Data Validation Tests

Ensure data quality and consistency:

```r
test_that("data meets quality standards", {
  data <- load_analysis_data()

  # Check data structure
  expect_equal(ncol(data), 15)
  expect_true(all(c("id", "outcome", "treatment") %in% names(data)))

  # Check data ranges
  expect_true(all(data$age >= 0 & data$age <= 120))
  expect_true(all(data$outcome %in% c("success", "failure")))

  # Check for missing data patterns
  missing_rate <- sum(is.na(data)) / (nrow(data) * ncol(data))
  expect_true(missing_rate < 0.05)  # Less than 5% missing
})
```

4. Reproducibility Tests

Verify analysis can be reproduced:

```r
test_that("analysis produces identical results", {
  set.seed(12345)
  results1 <- run_analysis()

  set.seed(12345)
  results2 <- run_analysis()

  expect_equal(results1$estimates, results2$estimates)
  expect_equal(results1$p_values, results2$p_values)
})
```

Setting Up Your Testing Environment

Directory Structure

Organize tests following R package conventions with extensions for data analysis:

```
tests/
├── testthat/
│   ├── test-data-loading.R
│   ├── test-data-cleaning.R
│   ├── test-statistical-models.R
│   ├── test-visualization.R
│   └── helper-test-data.R
├── integration/
│   ├── test-data-pipeline.R
│   ├── test-analysis-scripts.R
│   └── test-report-rendering.R
└── data/
    ├── sample-data.csv
    └── expected-outputs.rds
```

Test Configuration

Create a `testthat.R` file in your `tests/` directory:

```r
library(testthat)
library(myproject)  # Your analysis package

# Set testing options
options(warn = 2)  # Convert warnings to errors during testing

# Run all tests
test_check("myproject")
```

Helper Functions

Create reusable test utilities in `helper-*.R` files:

```r
# helper-test-data.R
create_test_dataset <- function(n = 100) {
  data.frame(
    id = 1:n,
    treatment = sample(c("A", "B"), n, replace = TRUE),
    outcome = rnorm(n, mean = 50, sd = 10),
    age = sample(18:80, n, replace = TRUE)
  )
}

expect_valid_model <- function(model) {
  expect_s3_class(model, "lm")
  expect_true(length(coef(model)) > 0)
  expect_true(summary(model)$r.squared >= 0)
}
```

Unit Testing for Analysis Functions

Statistical Functions

Test statistical computations with known results:

```r
test_that("confidence interval calculation is correct", {
  # Test with known values
  data <- c(10, 12, 14, 16, 18, 20)
  ci <- calculate_ci(data, confidence = 0.95)

  # Verify structure
  expect_length(ci, 2)
  expect_named(ci, c("lower", "upper"))

  # Verify mathematical properties
  expect_true(ci$lower < mean(data))
  expect_true(ci$upper > mean(data))
  expect_true(ci$upper > ci$lower)
})
```

Data Transformation Functions

Test data manipulation and cleaning:

```r
test_that("missing value imputation works correctly", {
  test_data <- data.frame(
    x = c(1, 2, NA, 4, 5),
    y = c(10, NA, 30, 40, 50)
  )

  imputed <- impute_missing(test_data, method = "mean")

  # Check no missing values remain
  expect_false(any(is.na(imputed)))

  # Check imputation values are reasonable
  expect_equal(imputed$x[3], mean(c(1, 2, 4, 5)))
  expect_equal(imputed$y[2], mean(c(10, 30, 40, 50)))
})
```

Visualization Functions

Test plot generation and properties:

```r
test_that("scatter plot function creates valid plot", {
  test_data <- create_test_dataset()

  # Test plot creation
  p <- create_scatter_plot(test_data, x = "age", y = "outcome")

  # Check plot object
  expect_s3_class(p, "ggplot")
  expect_equal(length(p$layers), 1)

  # Check plot data
  plot_data <- ggplot_build(p)$data[[1]]
  expect_equal(nrow(plot_data), nrow(test_data))
})
```

Integration Testing for Data Pipelines

Complete Pipeline Testing

Test the entire data analysis workflow:

```r
test_that("full analysis pipeline executes successfully", {
  # Create temporary directory for outputs
  temp_dir <- tempdir()
  output_dir <- file.path(temp_dir, "analysis_output")
  dir.create(output_dir, recursive = TRUE)
  # Run complete pipeline
  expect_no_error({
    results <- run_full_analysis(
      input_file = "tests/data/sample-data.csv",
      output_dir = output_dir
    )
  })

  # Verify outputs exist
  expect_true(file.exists(file.path(output_dir, "results.csv")))
  expect_true(file.exists(file.path(output_dir, "plots.pdf")))

  # Verify results structure
  expect_s3_class(results, "analysis_results")
  expect_true("model_summary" %in% names(results))
  expect_true("diagnostics" %in% names(results))

  # Cleanup
  unlink(output_dir, recursive = TRUE)
})
```

Cross-Script Dependencies

Test that analysis scripts work together:

```r
test_that("analysis scripts run in sequence", {
  scripts <- c(
    "01_data_preparation.R",
    "02_exploratory_analysis.R",
    "03_statistical_modeling.R",
    "04_results_visualization.R"
  )

  # Create isolated environment for each script
  for (script in scripts) {
    script_path <- file.path("scripts", script)
    expect_true(file.exists(script_path))

    expect_no_error({
      source(script_path, local = new.env())
    }, info = paste("Script", script, "failed to execute"))
  }
})
```

Data Validation and Quality Testing

Using testdat for Data Validation

The `testdat` package provides specialized functions for testing data frames:

```r
library(testdat)

test_that("dataset meets quality requirements", {
  data <- load_analysis_data()
  # Test data completeness
  expect_data(data, has_all_names, c("id", "treatment", "outcome"))
  expect_data(data, has_nrows, min = 100)
  expect_data(data, has_ncols, 10)

  # Test data types
  expect_data(data, is_col_type, "id", "integer")
  expect_data(data, is_col_type, "treatment", "factor")
  expect_data(data, is_col_type, "outcome", "numeric")

  # Test value ranges
  expect_data(data, has_range, "age", min = 0, max = 120)
  expect_data(data, has_no_missing, "id")
})
```

Custom Validation Rules

Create domain-specific validation functions:

```
validate_clinical_data <- function(data) {
  test_that("clinical data validation", {
    # Patient ID format
    expect_true(all(grepl("^P\\d{6}$", data$patient_id)))

    # Visit dates are reasonable
    expect_true(all(data$visit_date >= as.Date("2020-01-01")))
    expect_true(all(data$visit_date <= Sys.Date()))

    # Biomarker values within expected ranges
    expect_true(all(data$biomarker >= 0 & data$biomarker <= 1000))

    # Treatment groups are balanced
    treatment_table <- table(data$treatment)
    expect_true(all(treatment_table >= 0.3 * nrow(data)))
  })
}
```

Reproducibility Testing

Seed Management Testing

Ensure random processes are reproducible:

```
test_that("random sampling is reproducible", {
  # Test bootstrap procedure
  test_data <- create_test_dataset(100)

  set.seed(42)
  bootstrap1 <- bootstrap_analysis(test_data, n_boots = 1000)

  set.seed(42)
  bootstrap2 <- bootstrap_analysis(test_data, n_boots = 1000)

  expect_equal(bootstrap1$estimates, bootstrap2$estimates)
  expect_equal(bootstrap1$confidence_intervals,
               bootstrap2$confidence_intervals)
})
```

Cross-Platform Reproducibility

Test that analysis works across different systems:

```r
test_that("analysis is platform independent", {
  # Test numerical stability
  test_data <- create_test_dataset(1000)

  # Run analysis multiple times
  results <- replicate(10, {
    model <- fit_complex_model(test_data)
    summary(model)$coefficients[1, 1]  # First coefficient
  })

  # Check consistency (allowing for floating point precision)
  expect_true(sd(results) < 1e-10)
})
```

Package Version Testing

Ensure results are stable across package versions:

```r
test_that("results consistent with package versions", {
  # Document package versions
  packages <- c("lme4", "ggplot2", "dplyr")
  versions <- sapply(packages, function(pkg) {
    as.character(packageVersion(pkg))
  })

  # Store with results for reproducibility tracking
  expect_true(all(packages %in% loadedNamespaces()))

  # Test that known results match
  known_results <- readRDS("tests/data/known_results_v1.rds")
  current_results <- run_standard_analysis()

  expect_equal(current_results$coefficients,
               known_results$coefficients,
               tolerance = 1e-6)
})
```

Testing Analysis Scripts

Script Execution Testing

Test that numbered analysis scripts run without error:

```r
test_that("analysis scripts execute in order", {
  script_dir <- "scripts"
  scripts <- list.files(script_dir, pattern = "^\\d+_.*\\.R$",
                       full.names = TRUE)
  scripts <- sort(scripts)  # Ensure numerical order
  # Create clean environment for each script
  for (script in scripts) {
    script_env <- new.env()

    expect_no_error({
      source(script, local = script_env)
    }, info = paste("Failed to execute:", basename(script)))
  }
})
```

Output Validation

Test that scripts produce expected outputs:

```r
test_that("scripts create expected outputs", {
  # Run data preparation script
  source("scripts/01_data_preparation.R", local = new.env())

  # Check that cleaned data was created
  expect_true(file.exists("data/derived_data/cleaned_data.rds"))

  # Validate cleaned data
  cleaned_data <- readRDS("data/derived_data/cleaned_data.rds")
  expect_true(nrow(cleaned_data) > 0)
  expect_false(any(is.na(cleaned_data$primary_outcome)))
})
```

Testing Report Generation

R Markdown Testing

Test that reports can be rendered:

```r
test_that("main report renders successfully", {
  report_path <- "analysis/report/report.Rmd"
  output_dir <- tempdir()

  # Test YAML parsing
  yaml_content <- rmarkdown::yaml_front_matter(report_path)
  expect_true("title" %in% names(yaml_content))
  expect_true("author" %in% names(yaml_content))

  # Test dependencies
  required_packages <- c("rmarkdown", "knitr", "ggplot2")
  for (pkg in required_packages) {
    expect_true(requireNamespace(pkg, quietly = TRUE))
  }

  # Test rendering (if LaTeX available)
  skip_if_not_installed("tinytex")
  expect_no_error({
    rmarkdown::render(report_path,
                     output_dir = output_dir,
                     quiet = TRUE)
  })
  # Verify PDF was created
  pdf_file <- file.path(output_dir, "report.pdf")
  expect_true(file.exists(pdf_file))
})
```

Content Validation

Test report content quality:

```
test_that("report contains required sections", {
  report_path <- "analysis/report/report.Rmd"
  content <- readLines(report_path)

  # Check for required sections
  expect_true(any(grepl("# Introduction", content)))
  expect_true(any(grepl("# Methods", content)))
  expect_true(any(grepl("# Results", content)))
  expect_true(any(grepl("# Discussion", content)))

  # Check for figure references
  expect_true(any(grepl("Figure \\\\@ref", content)))

  # Check for table references
  expect_true(any(grepl("Table \\\\@ref", content)))
})
```

Continuous Integration for Data Analysis

GitHub Actions Configuration

Example workflow for testing data analysis projects:

```
# .github/workflows/analysis-testing.yml
name: Analysis Testing

on: [push, pull_request]

jobs:
  test:
    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v3

    - uses: r-lib/actions/setup-r@v2
      with:
        r-version: '4.3.0'

    - uses: r-lib/actions/setup-r-dependencies@v2
      with:
        cache-version: 2

    - name: Run unit tests
      run: |
        Rscript -e "testthat::test_dir('tests/testthat')"

    - name: Run integration tests
      run: |
        Rscript -e "testthat::test_dir('tests/integration')"

    - name: Check reproducibility
      run: |
        Rscript -e "source('scripts/99_reproducibility_check.R')"

    - name: Validate environment
      run: |
        Rscript check_renv_for_commit.R --quiet --fail-on-issues
```

Quality Gates

Implement automated quality checks:

```r
# scripts/99_reproducibility_check.R
library(testthat)

test_that("analysis environment is reproducible", {
  # Check renv lockfile is current
  expect_true(renv::status()$synchronized)

  # Check R options are documented
  source("check_rprofile_options.R")

  # Check data integrity
  data_files <- list.files("data/raw_data", full.names = TRUE)
  for (file in data_files) {
    checksum <- digest::digest(file, file = TRUE)
    # Compare with stored checksums
    expect_true(verify_checksum(file, checksum))
  }
})
```

Best Practices and Common Pitfalls

Best Practices

1. **Test Early and Often**: Implement tests during the exploratory phase, not just at the end
2. **Use Meaningful Test Names**: Describe what the test validates, not just what it does
3. **Test Edge Cases**: Include tests for missing data, extreme values, and empty datasets
4. **Mock External Dependencies**: Use sample data instead of requiring database connections
5. **Document Expected Behavior**: Use informative error messages and comments

Common Pitfalls

1. **Testing Implementation Instead of Behavior**: Focus on what the function should do, not how
2. **Overly Specific Tests**: Avoid tests that break with minor, acceptable changes

3. **Ignoring Random Elements**: Always set seeds when testing functions with randomness
4. **Insufficient Data Validation**: Test data assumptions explicitly
5. **Skipping Integration Tests**: Unit tests alone don't catch pipeline failures

Performance Considerations

```r
test_that("analysis completes within reasonable time", {
  start_time <- Sys.time()

  results <- run_analysis(large_dataset)

  end_time <- Sys.time()
  duration <- as.numeric(difftime(end_time, start_time, units = "mins"))

  expect_true(duration < 5)  # Should complete within 5 minutes
})
```

Tools and Packages

Core Testing Framework

- **testthat**: Primary testing framework for R (Wickham, 2011)
- **devtools**: Development tools including test runners
- **usethis**: Automated test setup and configuration

Data Validation

- **testdat**: Specialized data frame testing (Hope, 2022)
- **validate**: Rule-based data validation (van der Loo & de Jonge, 2021)
- **assertr**: Pipeline assertions and data verification (Fischetti, 2021)
- **checkmate**: Fast argument checks and assertions

Reproducibility

- **targets**: Pipeline management and testing (Landau, 2021)
- **renv**: Environment reproducibility (Ushey, 2022)
- **here**: Path management for reproducible scripts
- **conflicted**: Managing package conflicts

Continuous Integration

- **rcmdcheck**: R CMD check automation
- **covr**: Code coverage analysis
- **lintr**: Code style checking

Examples and Templates

Simple Examples with Built-in Datasets

Basic Data Validation Tests with iris Dataset

```r
# tests/testthat/test-iris-analysis.R
library(testthat)
```

```r
test_that("iris dataset has expected structure", {
  data(iris)

  # Test basic structure
  expect_equal(nrow(iris), 150)
  expect_equal(ncol(iris), 5)
  expect_true(all(c("Sepal.Length", "Sepal.Width", "Petal.Length",
                    "Petal.Width", "Species") %in% names(iris)))

  # Test data types
  expect_true(is.numeric(iris$Sepal.Length))
  expect_true(is.numeric(iris$Sepal.Width))
  expect_true(is.factor(iris$Species))

  # Test value ranges
  expect_true(all(iris$Sepal.Length > 0))
  expect_true(all(iris$Petal.Length >= 0))
  expect_true(max(iris$Sepal.Length) <= 10)  # Reasonable upper bound

  # Test species levels
  expected_species <- c("setosa", "versicolor", "virginica")
  expect_equal(levels(iris$Species), expected_species)
  expect_equal(table(iris$Species), rep(50, 3))
})

test_that("iris measurements are biologically plausible", {
  data(iris)

  # Sepal measurements should be positive and reasonable
  expect_true(all(iris$Sepal.Length >= 4 & iris$Sepal.Length <= 8))
  expect_true(all(iris$Sepal.Width >= 1.5 & iris$Sepal.Width <= 5))

  # Petal measurements
  expect_true(all(iris$Petal.Length >= 1 & iris$Petal.Length <= 7))
  expect_true(all(iris$Petal.Width >= 0.1 & iris$Petal.Width <= 3))

  # No missing values
  expect_false(any(is.na(iris)))
})
```

Simple Statistical Functions with mtcars

```r
# tests/testthat/test-mtcars-stats.R
library(testthat)

# Helper function for testing
calculate_mpg_summary <- function(data) {
  list(
    mean_mpg = mean(data$mpg),
    median_mpg = median(data$mpg),
    sd_mpg = sd(data$mpg),
    range_mpg = range(data$mpg)
  )
}
```

```r
test_that("mpg summary statistics are correct", {
  data(mtcars)

  summary_stats <- calculate_mpg_summary(mtcars)

  # Test summary statistics properties
  expect_true(summary_stats$mean_mpg > 0)
  expect_true(summary_stats$median_mpg > 0)
  expect_true(summary_stats$sd_mpg > 0)
  expect_equal(length(summary_stats$range_mpg), 2)
  expect_true(summary_stats$range_mpg[1] <= summary_stats$range_mpg[2])

  # Test against known values (mtcars is static)
  expect_equal(summary_stats$mean_mpg, mean(mtcars$mpg))
  expect_equal(summary_stats$median_mpg, median(mtcars$mpg))
  # Approximate known value
  expect_true(abs(summary_stats$mean_mpg - 20.09) < 0.1)
})

test_that("correlation analysis works correctly", {
  data(mtcars)

  # Test correlation between mpg and weight
  cor_mpg_wt <- cor(mtcars$mpg, mtcars$wt)

  expect_true(cor_mpg_wt >= -1 & cor_mpg_wt <= 1)
  expect_true(cor_mpg_wt < 0)  # Should be negative correlation
  expect_true(abs(cor_mpg_wt) > 0.5)  # Should be strong correlation
})

test_that("linear model fitting works", {
  data(mtcars)

  # Fit simple linear model
  model <- lm(mpg ~ wt + hp, data = mtcars)

  # Test model object
  expect_s3_class(model, "lm")
  expect_equal(length(coef(model)), 3)  # Intercept + 2 predictors
  expect_true(summary(model)$r.squared > 0.5)
  expect_true(summary(model)$r.squared <= 1)

  # Test residuals
  residuals <- resid(model)
  expect_equal(length(residuals), nrow(mtcars))
  expect_true(abs(mean(residuals)) < 1e-10)  # Should be approximately zero
})
```

Basic Visualization Testing with penguins

```r
# tests/testthat/test-penguins-viz.R
library(testthat)
library(ggplot2)
library(palmerpenguins)

# Helper function
create_penguin_scatterplot <- function(data, x_var, y_var) {
  ggplot(data, aes_string(x = x_var, y = y_var, color = "species")) +
    geom_point() +
    theme_minimal() +
    labs(title = paste("Penguins:", y_var, "vs", x_var))
}

test_that("penguin scatterplot creation works", {
  data(penguins, package = "palmerpenguins")

  # Remove missing values for plotting
  clean_penguins <- penguins[complete.cases(penguins), ]
  expect_true(nrow(clean_penguins) > 0)

  # Create plot
  p <- create_penguin_scatterplot(clean_penguins, "bill_length_mm",
                                  "body_mass_g")

  # Test plot object
  expect_s3_class(p, "ggplot")
  expect_equal(length(p$layers), 1)  # One geom_point layer

  # Test plot data
  plot_data <- ggplot_build(p)$data[[1]]
  expect_equal(nrow(plot_data), nrow(clean_penguins))
  expect_true(all(c("x", "y", "colour") %in% names(plot_data)))
})

test_that("penguin data visualization validates species separation", {
  data(penguins, package = "palmerpenguins")
  clean_penguins <- penguins[complete.cases(penguins), ]

  # Create species summary for visualization validation
  species_summary <- aggregate(body_mass_g ~ species,
                               data = clean_penguins,
                               FUN = function(x) c(mean = mean(x),
                                                   sd = sd(x)))

  expect_equal(nrow(species_summary), 3)  # Three species
  expect_true(all(species_summary$body_mass_g[, "mean"] > 0))
  expect_true(all(species_summary$body_mass_g[, "sd"] > 0))

  # Gentoo penguins should be heaviest on average
  gentoo_mean <- species_summary[species_summary$species == "Gentoo",
                                 "body_mass_g"][1, "mean"]
  adelie_mean <- species_summary[species_summary$species == "Adelie",
                                "body_mass_g"][1, "mean"]
  expect_true(gentoo_mean > adelie_mean)
})
```

Medium Complexity Examples

Data Pipeline Testing with iris

```r
# tests/testthat/test-iris-pipeline.R
library(testthat)

# Analysis pipeline functions
prepare_iris_data <- function(data) {
  # Add derived variables
  data$sepal_ratio <- data$Sepal.Length / data$Sepal.Width
  data$petal_ratio <- data$Petal.Length / data$Petal.Width
  data$total_length <- data$Sepal.Length + data$Petal.Length
  return(data)
}

analyze_species_differences <- function(data) {
  # Perform ANOVA for each measurement
  results <- list()
  measurements <- c("Sepal.Length", "Sepal.Width", "Petal.Length",
                    "Petal.Width")

  for (measure in measurements) {
    formula_str <- paste(measure, "~ Species")
    aov_result <- aov(as.formula(formula_str), data = data)
    results[[measure]] <- summary(aov_result)
  }

  return(results)
}

test_that("iris data preparation pipeline works", {
  data(iris)

  # Test data preparation
  prepared_data <- prepare_iris_data(iris)

  # Check new variables were added
  expect_true("sepal_ratio" %in% names(prepared_data))
  expect_true("petal_ratio" %in% names(prepared_data))
  expect_true("total_length" %in% names(prepared_data))

  # Check derived variables are reasonable
  expect_true(all(prepared_data$sepal_ratio > 0))
  expect_true(all(prepared_data$petal_ratio > 0))
  expect_true(all(prepared_data$total_length >
                prepared_data$Sepal.Length))
  expect_true(all(prepared_data$total_length >
                prepared_data$Petal.Length))

  # Check no missing values introduced
  expect_false(any(is.na(prepared_data$sepal_ratio)))
  expect_false(any(is.na(prepared_data$petal_ratio)))
})
```

```r
test_that("species analysis produces valid results", {
  data(iris)
  prepared_data <- prepare_iris_data(iris)

  # Run analysis
  analysis_results <- analyze_species_differences(prepared_data)

  # Check structure of results
  expect_true(is.list(analysis_results))
  expect_equal(length(analysis_results), 4)
  expect_true(all(c("Sepal.Length", "Sepal.Width", "Petal.Length",
                    "Petal.Width") %in% names(analysis_results)))

  # Check each ANOVA result
  for (result in analysis_results) {
    expect_true(is.list(result))
    expect_true(length(result) == 1)  # One ANOVA table

    # Check F-statistic exists and is positive
    f_stat <- result[[1]][1, "F value"]
    expect_true(f_stat > 0)

    # Check p-value is between 0 and 1
    p_value <- result[[1]][1, "Pr(>F)"]
    expect_true(p_value >= 0 & p_value <= 1)
  }
})

test_that("complete iris analysis pipeline is reproducible", {
  data(iris)

  # Run pipeline twice
  set.seed(123)
  prepared1 <- prepare_iris_data(iris)
  results1 <- analyze_species_differences(prepared1)

  set.seed(123)
  prepared2 <- prepare_iris_data(iris)
  results2 <- analyze_species_differences(prepared2)

  # Results should be identical
  expect_equal(prepared1, prepared2)
  expect_equal(results1, results2)
})
```

Advanced Statistical Testing with mtcars

```r
# tests/testthat/test-mtcars-advanced.R
library(testthat)

# Advanced analysis functions
perform_regression_analysis <- function(data, response, predictors) {
  formula_str <- paste(response, "~", paste(predictors, collapse = " + "))
  model <- lm(as.formula(formula_str), data = data)
  # Add model diagnostics
  list(
    model = model,
    summary = summary(model),
    diagnostics = list(
      residuals = resid(model),
      fitted = fitted(model),
      r_squared = summary(model)$r.squared,
      adj_r_squared = summary(model)$adj.r.squared,
      aic = AIC(model),
      bic = BIC(model)
    )
  )
}

validate_model_assumptions <- function(model_result) {
  residuals <- model_result$diagnostics$residuals
  fitted <- model_result$diagnostics$fitted

  # Shapiro-Wilk test for normality (if sample size allows)
  if (length(residuals) <= 5000) {
    normality_test <- shapiro.test(residuals)
  } else {
    normality_test <- NULL
  }

  # Durbin-Watson test for autocorrelation
  if (requireNamespace("lmtest", quietly = TRUE)) {
    dw_test <- lmtest::dwtest(model_result$model)
  } else {
    dw_test <- NULL
  }

  list(
    normality = normality_test,
    durbin_watson = dw_test,
    residual_range = range(residuals),
    mean_residual = mean(residuals)
  )
}

test_that("regression analysis function works correctly", {
  data(mtcars)

  # Test basic regression
  result <- perform_regression_analysis(mtcars, "mpg",
                                       c("wt", "hp", "cyl"))

  # Check structure
  expect_true(is.list(result))
  expect_true(all(c("model", "summary", "diagnostics") %in% names(result)))

  # Check model object
  expect_s3_class(result$model, "lm")
  expect_equal(length(coef(result$model)), 4)  # Intercept + 3 predictors

  # Check diagnostics
  expect_true(result$diagnostics$r_squared >= 0 &
              result$diagnostics$r_squared <= 1)
  expect_true(result$diagnostics$adj_r_squared <=
              result$diagnostics$r_squared)
  expect_true(result$diagnostics$aic > 0)
  expect_true(result$diagnostics$bic > 0)
  expect_equal(length(result$diagnostics$residuals), nrow(mtcars))
})

test_that("model validation catches assumption violations", {
  data(mtcars)

  # Create a good model
  good_result <- perform_regression_analysis(mtcars, "mpg", c("wt", "hp"))
  good_validation <- validate_model_assumptions(good_result)

  # Check that validation runs without error
  expect_true(is.list(good_validation))
  expect_true(abs(good_validation$mean_residual) < 1e-10)  # Near zero

  # Test with a poor model (if we had one)
  # This tests that our validation function works
  expect_true(length(good_validation$residual_range) == 2)
  expect_true(good_validation$residual_range[1] <=
              good_validation$residual_range[2])
})

test_that("regression handles different model specifications", {
  data(mtcars)

  # Test different model complexities
  simple_model <- perform_regression_analysis(mtcars, "mpg", "wt")
  complex_model <- perform_regression_analysis(mtcars, "mpg",
                                               c("wt", "hp", "cyl",
                                                 "disp", "qsec"))

  # Simple model should have fewer parameters
  expect_true(length(coef(simple_model$model)) <
              length(coef(complex_model$model)))

  # Complex model might have higher R-squared but not necessarily better AIC
  expect_true(complex_model$diagnostics$r_squared >=
              simple_model$diagnostics$r_squared)

  # Both should be valid model objects
  expect_s3_class(simple_model$model, "lm")
  expect_s3_class(complex_model$model, "lm")
})
```

Integration Testing with penguins Dataset

```r
# tests/testthat/test-penguins-integration.R
library(testthat)
library(palmerpenguins)
library(ggplot2)

# Complete analysis workflow
run_penguin_analysis <- function(remove_na = TRUE) {
  # Load and prepare data
  data(penguins, package = "palmerpenguins")

  if (remove_na) {
    clean_data <- penguins[complete.cases(penguins), ]
  } else {
    clean_data <- penguins
  }

  # Descriptive statistics
  summary_stats <- list()
  numeric_vars <- c("bill_length_mm", "bill_depth_mm", "flipper_length_mm",
                   "body_mass_g")

  for (var in numeric_vars) {
    if (remove_na) {
      values <- clean_data[[var]]
    } else {
      values <- clean_data[[var]][!is.na(clean_data[[var]])]
    }

    summary_stats[[var]] <- list(
      mean = mean(values),
      sd = sd(values),
      median = median(values),
      range = range(values)
    )
  }

  # Statistical models
  models <- list()

  # Model 1: Body mass predicted by bill dimensions
  if (remove_na) {
    models$body_mass_model <- lm(body_mass_g ~ bill_length_mm +
                                   bill_depth_mm, data = clean_data)
  }

  # Model 2: Flipper length by species
  if (remove_na) {
    models$flipper_species_model <- aov(flipper_length_mm ~ species,
                                           data = clean_data)
  }

  # Visualizations
  plots <- list()
  if (remove_na && nrow(clean_data) > 0) {
    # Scatterplot
    plots$bill_scatter <- ggplot(clean_data,
                                 aes(x = bill_length_mm,
                                     y = bill_depth_mm,
                                     color = species)) +
      geom_point() +
      theme_minimal() +
      labs(title = "Bill Dimensions by Species")

    # Box plot
    plots$mass_boxplot <- ggplot(clean_data,
                                 aes(x = species, y = body_mass_g,
                                     fill = species)) +
      geom_boxplot() +
      theme_minimal() +
      labs(title = "Body Mass by Species")
  }

  # Return comprehensive results
  list(
    data = clean_data,
    summary_stats = summary_stats,
    models = models,
    plots = plots,
    sample_size = nrow(clean_data)
  )
}

test_that("complete penguin analysis workflow executes successfully", {
  # Run complete analysis
  expect_no_error({
    results <- run_penguin_analysis(remove_na = TRUE)
  })

  # Check overall structure
  expect_true(is.list(results))
  expect_true(all(c("data", "summary_stats", "models", "plots",
                  "sample_size") %in% names(results)))

  # Check data
  expect_s3_class(results$data, "data.frame")
  expect_true(results$sample_size > 300)  # Should have substantial sample
  expect_false(any(is.na(results$data)))  # No missing values after cleaning

  # Check summary statistics
  expect_equal(length(results$summary_stats), 4)
  for (stat in results$summary_stats) {
    expect_true(all(c("mean", "sd", "median", "range") %in% names(stat)))
    expect_true(stat$mean > 0)
    expect_true(stat$sd > 0)
  }

  # Check models
```

```r
  expect_s3_class(results$models$body_mass_model, "lm")
  expect_s3_class(results$models$flipper_species_model, "aov")

  # Check plots
  expect_s3_class(results$plots$bill_scatter, "ggplot")
  expect_s3_class(results$plots$mass_boxplot, "ggplot")
})

test_that("penguin analysis produces biologically meaningful results", {
  results <- run_penguin_analysis(remove_na = TRUE)

  # Check that Gentoo penguins are heaviest (known biological fact)
  species_masses <- aggregate(body_mass_g ~ species, data = results$data,
                             mean)
  gentoo_mass <- species_masses[species_masses$species == "Gentoo",
                               "body_mass_g"]
  adelie_mass <- species_masses[species_masses$species == "Adelie",
                               "body_mass_g"]
  chinstrap_mass <- species_masses[species_masses$species == "Chinstrap",
                                  "body_mass_g"]

  expect_true(gentoo_mass > adelie_mass)
  expect_true(gentoo_mass > chinstrap_mass)

  # Check that body mass model is significant
  model_summary <- summary(results$models$body_mass_model)
  expect_true(model_summary$r.squared > 0.1)  # At least some explanatory power
  expect_true(model_summary$fstatistic[1] > 1)  # F-statistic should be > 1

  # Check that species differences in flipper length are significant
  anova_result <- summary(results$models$flipper_species_model)
  f_stat <- anova_result[[1]][1, "F value"]
  p_value <- anova_result[[1]][1, "Pr(>F)"]
  expect_true(f_stat > 1)
  expect_true(p_value < 0.05)  # Should be significant
})

test_that("penguin analysis is reproducible with different options", {
  # Test reproducibility with NA removal
  results1 <- run_penguin_analysis(remove_na = TRUE)
  results2 <- run_penguin_analysis(remove_na = TRUE)

  # Data should be identical
  expect_equal(results1$data, results2$data)
  expect_equal(results1$sample_size, results2$sample_size)

  # Summary statistics should be identical
  expect_equal(results1$summary_stats, results2$summary_stats)

  # Model coefficients should be identical
  expect_equal(coef(results1$models$body_mass_model),
               coef(results2$models$body_mass_model))
})
```

Complete Test Suite Template

```r
# tests/testthat/test-complete-analysis.R
library(testthat)
library(myanalysis)

# Unit tests for core functions
test_that("data loading functions work", {
  # Test various data formats
  csv_data <- load_csv_data("tests/data/sample.csv")
  expect_s3_class(csv_data, "data.frame")

  xlsx_data <- load_excel_data("tests/data/sample.xlsx")
  expect_s3_class(xlsx_data, "data.frame")
})

test_that("statistical functions are correct", {
  test_data <- data.frame(
    group = rep(c("A", "B"), each = 50),
    value = c(rnorm(50, 10, 2), rnorm(50, 12, 2))
  )

  # Test t-test wrapper
  result <- compare_groups(test_data, "value", "group")
  expect_s3_class(result, "htest")
  expect_true(result$p.value > 0 & result$p.value <= 1)
})

# Integration tests
test_that("complete workflow executes", {
  # Test with sample dataset
  expect_no_error({
    data <- load_analysis_data()
    cleaned <- clean_and_validate(data)
    models <- fit_models(cleaned)
    results <- summarize_results(models)
    plots <- create_visualizations(results)
  })

  # Validate final outputs
  expect_s3_class(results, "analysis_summary")
  expect_true(length(plots) > 0)
})

# Reproducibility tests
test_that("analysis is reproducible", {
  set.seed(123)
  result1 <- run_bootstrap_analysis(n_bootstrap = 100)

  set.seed(123)
  result2 <- run_bootstrap_analysis(n_bootstrap = 100)

  expect_equal(result1, result2)
})
```

Data Validation Template

```r
# tests/testthat/test-data-validation.R
library(testdat)

test_that("clinical trial data meets requirements", {
  data <- load_trial_data()

  # Structure validation
  expect_data(data, has_all_names,
             c("patient_id", "treatment", "outcome", "baseline_score"))
  expect_data(data, has_nrows, min = 200)  # Minimum sample size

  # Data type validation
  expect_data(data, is_col_type, "patient_id", "character")
  expect_data(data, is_col_type, "treatment", "factor")
  expect_data(data, is_col_type, "outcome", "numeric")

  # Value validation
  expect_data(data, has_range, "outcome", min = 0, max = 100)
  expect_data(data, has_no_missing, "patient_id")

  # Business logic validation
  treatment_groups <- levels(data$treatment)
  expect_true(all(c("placebo", "active") %in% treatment_groups))

  # Check randomization balance
  group_sizes <- table(data$treatment)
  expect_true(all(group_sizes >= 90 & group_sizes <= 110))
})
```

References

Fischetti, T. (2021). assertr: Assertive Programming for R Analysis Pipelines. R package version 2.8. https://CRAN.R-project.org/package=assertr

Hope, R. (2022). testdat: Data Unit Testing for R. R package version 0.4.0. https://CRAN.R-project.org/package=testdat

Landau, W. M. (2021). The targets R package: a dynamic Make-like function-oriented pipeline toolkit for reproducibility and high-performance computing. Journal of Open Source Software, 6(57), 2959. https://doi.org/10.21105/joss.02959

Poldrack, R. A. (2019). The new mind readers: What neuroimaging can and cannot reveal about our thoughts. Princeton University Press.

Poldrack, R. A., et al. (2019). Reproducible data analysis. In Stanford Psychology Guide to Doing Open Science. https://poldrack.github.io/psych-open-science-guide/4_reproducibleanalysis.html

Ushey, K. (2022). renv: Project Environments. R package version 0.16.0. https://CRAN.R-project.org/package=renv

van der Loo, M., & de Jonge, E. (2021). Data validation infrastructure for R. Journal of Statistical Software, 97(10), 1-31. https://doi.org/10.18637/jss.v097.i10

Wickham, H. (2011). testthat: Get started with testing. The R Journal, 3(1), 5-10. https://journal.r-project.org/archive/2011-1/RJournal_2011-1_Wickham.pdf

Wickham, H. (2015). R packages: organize, test, document, and share your code. O'Reilly Media.

Wilson, G., Aruliah, D. A., Brown, C. T., Hong, N. P. C., Davis, M., Guy, R. T., ... & Wilson, P. (2014). Best practices for scientific computing. PLoS biology, 12(1), e1001745. https://doi.org/10.1371/journal.pbio.1001745

Wilson, G., Bryan, J., Cranston, K., Kitzes, J., Nederbragt, L., & Teal, T. K. (2017). Good enough practices in scientific computing. PLoS computational biology, 13(6), e1005510. https://doi.org/10.1371/journal.pcbi.1005510

This guide is part of the ZZCOLLAB framework for reproducible research collaboration. For more information, see the project documentation and workflow guides.