Complete Guide to R Package Development

Overview

This tutorial will teach you how to create an R package from scratch, using the FuzzyDelphiJmv package as inspiration. We'll cover everything from basic setup to advanced features like creating Jamovi modules.

Prerequisites

Before starting, ensure you have:

- R (version 4.0 or higher)
- RStudio (recommended IDE)
- Required packages: `devtools`, `usethis`, `roxygen2`

```
r

install.packages(c("devtools", "usethis", "roxygen2", "testthat"))
```

Step 1: Setting Up Your Package Structure

Creating a New Package

```
r

# Option 1: Using usethis (recommended)
library(usethis)
create_package("~/path/to/your/FuzzyDelphi")

# Option 2: Using devtools
library(devtools)
create("FuzzyDelphi")
```

This creates the basic package structure:

```
FuzzyDelphi/
├── DESCRIPTION
├── NAMESPACE
├── R/
├── man/
└── .Rbuildignore
```

Understanding Key Files

DESCRIPTION: Package metadata

```
Package: FuzzyDelphi
Title: Fuzzy Delphi Method Implementation
Version: 0.0.0.9000
Authors@R:
    person("Your", "Name", email = "your.email@example.com", role = c("aut", "cre"))
Description: Implementation of the Fuzzy Delphi method for consensus building
    and decision making under uncertainty.
License: GPL (>= 3)
Encoding: UTF-8
LazyData: true
Roxygen: list(markdown = TRUE)
RoxygenNote: 7.2.3
Imports:
    stats,
    utils
Suggests:
    testthat (>= 3.0.0)
```

NAMESPACE: Controls function exports (auto-generated by roxygen2)

Step 2: Writing R Functions

Basic Function Structure

Create your first function in `R/fuzzy_delphi.R`:

r

```r
#' Calculate Fuzzy Delphi Consensus
#'
#' This function implements the Fuzzy Delphi method to analyze expert opinions
#' and determine consensus levels.
#'
#' @param data A data frame containing expert ratings
#' @param threshold Consensus threshold (default: 0.75)
#' @param alpha Alpha cut level for fuzzy numbers (default: 0.5)
#'
#' @return A list containing:
#'   \item{consensus}{Logical vector indicating consensus for each item}
#'   \item{fuzzy_values}{Matrix of fuzzy values}
#'   \item{defuzzified}{Defuzzified values}
#'   \item{ranking}{Item rankings}
#'
#' @examples
#' # Sample data
#' expert_data <- data.frame(
#'   Item1 = c(4, 5, 4, 5, 4),
#'   Item2 = c(3, 4, 3, 4, 3),
#'   Item3 = c(5, 5, 4, 5, 5)
#' )
#'
#' result <- fuzzy_delphi_analysis(expert_data)
#' print(result$consensus)
#'
#' @export
fuzzy_delphi_analysis <- function(data, threshold = 0.75, alpha = 0.5) {
  # Validate inputs
  if (!is.data.frame(data)) {
    stop("Data must be a data frame")
  }

  if (threshold < 0 || threshold > 1) {
    stop("Threshold must be between 0 and 1")
  }

  # Convert to matrix for calculations
  data_matrix <- as.matrix(data)

  # Calculate triangular fuzzy numbers for each item
  fuzzy_values <- calculate_triangular_fuzzy(data_matrix)

  # Defuzzification using centroid method
  defuzzified <- defuzzify(fuzzy_values)
```

```r
  # Calculate consensus percentage
  consensus_pct <- calculate_consensus_percentage(fuzzy_values)

  # Determine consensus (typically >75%)
  consensus <- consensus_pct >= threshold

  # Rank items by defuzzified values
  ranking <- rank(-defuzzified, ties.method = "average")

  # Return results
  list(
    consensus = consensus,
    fuzzy_values = fuzzy_values,
    defuzzified = defuzzified,
    consensus_percentage = consensus_pct,
    ranking = ranking,
    threshold = threshold
  )
}

#' Calculate Triangular Fuzzy Numbers
#'
#' @param data_matrix Numeric matrix of expert ratings
#' @return Matrix with min, mode, max for each item
#' @keywords internal
calculate_triangular_fuzzy <- function(data_matrix) {
  n_items <- ncol(data_matrix)
  fuzzy_matrix <- matrix(0, nrow = n_items, ncol = 3)
  colnames(fuzzy_matrix) <- c("min", "mode", "max")
  rownames(fuzzy_matrix) <- colnames(data_matrix)

  for (i in 1:n_items) {
    item_data <- data_matrix[, i]
    fuzzy_matrix[i, "min"] <- min(item_data, na.rm = TRUE)
    fuzzy_matrix[i, "mode"] <- median(item_data, na.rm = TRUE)
    fuzzy_matrix[i, "max"] <- max(item_data, na.rm = TRUE)
  }

  return(fuzzy_matrix)
}

#' Defuzzify Triangular Fuzzy Numbers
#'
#' @param fuzzy_values Matrix with min, mode, max columns
#' @return Vector of defuzzified values
#' @keywords internal
defuzzify <- function(fuzzy_values) {
```

```r
  # Centroid method: (min + mode + max) / 3
  (fuzzy_values[, "min"] + fuzzy_values[, "mode"] + fuzzy_values[, "max"]) / 3
}

#' Calculate Consensus Percentage
#'
#' @param fuzzy_values Matrix with fuzzy number components
#' @return Vector of consensus percentages
#' @keywords internal
calculate_consensus_percentage <- function(fuzzy_values) {
  # Calculate fuzzy spread
  spread <- fuzzy_values[, "max"] - fuzzy_values[, "min"]

  # Convert to consensus percentage (smaller spread = higher consensus)
  max_spread <- max(spread, na.rm = TRUE)
  if (max_spread == 0) {
    return(rep(1, nrow(fuzzy_values)))
  }

  consensus_pct <- 1 - (spread / max_spread)
  return(consensus_pct)
}
```

Adding Utility Functions

Create `R/utils.R` for helper functions:

```r
#' Validate Delphi Data
#'
#' @param data Input data frame
#' @return Logical indicating if data is valid
#' @export
validate_delphi_data <- function(data) {
  if (!is.data.frame(data)) return(FALSE)
  if (nrow(data) < 3) return(FALSE)  # Need at least 3 experts
  if (ncol(data) < 1) return(FALSE)  # Need at least 1 item

  # Check if all columns are numeric
  numeric_cols <- sapply(data, is.numeric)
  if (!all(numeric_cols)) return(FALSE)

  return(TRUE)
}

#' Generate Sample Delphi Data
#'
#' @param n_experts Number of experts (default: 10)
#' @param n_items Number of items (default: 5)
#' @param scale_min Minimum scale value (default: 1)
#' @param scale_max Maximum scale value (default: 5)
#' @return Data frame with sample expert ratings
#' @export
generate_sample_data <- function(n_experts = 10, n_items = 5,
                                 scale_min = 1, scale_max = 5) {
  # Generate item names
  item_names <- paste0("Item", 1:n_items)

  # Generate random ratings
  data <- data.frame(
    matrix(
      sample(scale_min:scale_max, n_experts * n_items, replace = TRUE),
      nrow = n_experts,
      ncol = n_items
    )
  )

  colnames(data) <- item_names
  return(data)
}
```

Step 3: Documentation with roxygen2

The `#'` comments above functions create documentation. Key roxygen2 tags:

- `@param`: Parameter descriptions
- `@return`: Return value description
- `@examples`: Usage examples
- `@export`: Makes function available to users
- `@keywords internal`: Marks internal functions

Generate documentation:

```
r

devtools::document()
```

Step 4: Adding Tests

Create tests in `tests/testthat/test-fuzzy-delphi.R`:

```r
test_that("fuzzy_delphi_analysis works correctly", {
  # Create sample data
  test_data <- data.frame(
    Item1 = c(4, 5, 4, 5, 4),
    Item2 = c(3, 4, 3, 4, 3),
    Item3 = c(5, 5, 4, 5, 5)
  )

  result <- fuzzy_delphi_analysis(test_data)

  # Test structure
  expect_type(result, "list")
  expect_named(result, c("consensus", "fuzzy_values", "defuzzified",
                         "consensus_percentage", "ranking", "threshold"))

  # Test dimensions
  expect_length(result$consensus, 3)
  expect_equal(nrow(result$fuzzy_values), 3)
  expect_equal(ncol(result$fuzzy_values), 3)
})

test_that("validation works", {
  expect_true(validate_delphi_data(data.frame(x = 1:5, y = 2:6)))
  expect_false(validate_delphi_data("not a dataframe"))
  expect_false(validate_delphi_data(data.frame(x = 1:2)))  # Too few rows
})
```

Run tests:

```r
devtools::test()
```

Step 5: Adding Data

Include sample datasets in `data/`:

```r
# Create sample dataset
sample_delphi_data <- generate_sample_data(n_experts = 15, n_items = 8)

# Save to data/ directory
usethis::use_data(sample_delphi_data)
```

Document the data in `R/data.R`:

```r
#' Sample Delphi Survey Data
#'
#' A dataset containing expert ratings for 8 items from 15 experts
#' using a 5-point Likert scale.
#'
#' @format A data frame with 15 rows and 8 columns:
#' \describe{
#'   \item{Item1-Item8}{Expert ratings on 1-5 scale}
#' }
#' @source Generated for demonstration purposes
"sample_delphi_data"
```

Step 6: Building and Checking

Build the Package

```r
devtools::build()
```

Check for Issues

```r
devtools::check()
```

Install and Test

```r
devtools::install()
library(FuzzyDelphi)

# Test your functions
data(sample_delphi_data)
result <- fuzzy_delphi_analysis(sample_delphi_data)
print(result)
```

Step 7: Advanced Features

Adding Visualization Functions

Create `R/plotting.R`:

r

```r
#' Plot Fuzzy Delphi Results
#'
#' @param result Output from fuzzy_delphi_analysis
#' @param type Plot type: "consensus", "ranking", or "fuzzy"
#' @return ggplot object
#' @export
plot_delphi_results <- function(result, type = "consensus") {
  if (!requireNamespace("ggplot2", quietly = TRUE)) {
    stop("ggplot2 package is required for plotting")
  }

  if (type == "consensus") {
    # Plot consensus percentages
    df <- data.frame(
      Item = names(result$consensus_percentage),
      Consensus = result$consensus_percentage,
      Meets_Threshold = result$consensus
    )

    ggplot2::ggplot(df, ggplot2::aes(x = Item, y = Consensus,
                                     fill = Meets_Threshold)) +
      ggplot2::geom_col() +
      ggplot2::geom_hline(yintercept = result$threshold,
                          linetype = "dashed", color = "red") +
      ggplot2::labs(title = "Consensus Analysis",
                    y = "Consensus Percentage",
                    fill = "Meets Threshold") +
      ggplot2::theme_minimal() +
      ggplot2::theme(axis.text.x = ggplot2::element_text(angle = 45, hjust = 1))
  }
  # Add other plot types as needed
}
```

Creating S3 Methods

```r
#' Print method for fuzzy delphi results
#' @param x Result from fuzzy_delphi_analysis
#' @param ... Additional arguments
#' @export
print.fuzzy_delphi <- function(x, ...) {
  cat("Fuzzy Delphi Analysis Results\n")
  cat("============================\n\n")

  cat("Items reaching consensus (>", x$threshold, "):\n")
  consensus_items <- names(x$consensus)[x$consensus]
  if (length(consensus_items) > 0) {
    cat(paste(consensus_items, collapse = ", "), "\n\n")
  } else {
    cat("None\n\n")
  }

  cat("Top 3 ranked items:\n")
  top_items <- names(sort(x$ranking)[1:min(3, length(x$ranking))])
  cat(paste(1:length(top_items), top_items, sep = ". ", collapse = "\n"), "\n\n")

  invisible(x)
}

# Modify your main function to return the correct class
fuzzy_delphi_analysis <- function(data, threshold = 0.75, alpha = 0.5) {
  # ... (existing code) ...

  result <- list(
    consensus = consensus,
    fuzzy_values = fuzzy_values,
    defuzzified = defuzzified,
    consensus_percentage = consensus_pct,
    ranking = ranking,
    threshold = threshold
  )

  class(result) <- "fuzzy_delphi"
  return(result)
}
```

Step 8: Creating a Jamovi Module (Advanced)

For Jamovi integration like FuzzyDelphiJmv, you'll need:

1. **jamovi module structure**: Uses `.a.yaml` files for analysis definitions

2. **jmvcore integration**: R package that interfaces with jamovi
3. **JavaScript UI components**: For the jamovi interface

Basic jamovi module files:

- `jamovi/fuzzydelphi.a.yaml`: Analysis definition
- `jamovi/fuzzydelphi.u.yaml`: UI definition
- `R/fuzzydelphi.b.R`: Backend R implementation

Step 9: Distribution

CRAN Submission

1. Ensure all checks pass: `devtools::check()`
2. Update version in DESCRIPTION
3. Submit to CRAN following their guidelines

GitHub Distribution

```r
# Install from GitHub
devtools::install_github("yourusername/FuzzyDelphi")
```

Creating a Vignette

```r
usethis::use_vignette("introduction-to-fuzzydelphi")
```

Best Practices

1. **Code Style**: Use consistent naming and formatting
2. **Error Handling**: Always validate inputs and provide clear error messages
3. **Testing**: Aim for >80% code coverage
4. **Documentation**: Every exported function should be well-documented
5. **Versioning**: Use semantic versioning (major.minor.patch)
6. **Dependencies**: Minimize external dependencies
7. **Performance**: Profile and optimize critical functions

Common Issues and Solutions

1. **NAMESPACE errors**: Run `devtools::document()` to regenerate

2. **Test failures**: Check your test assumptions and edge cases
3. **Check warnings**: Address all NOTEs, WARNINGs, and ERRORs before submission
4. **Documentation issues**: Ensure all @param and @return tags are complete

This comprehensive guide should help you create a robust R package similar to FuzzyDelphiJmv. Start with the basic structure and gradually add more sophisticated features as you become comfortable with the development process.