Rural Scotland Data Dashboard

Complete User Guide for Modifications

Table of Contents

Project Structure

The project is split across multiple folders and files allowing modular modifications to the dashboard structure. Each of the 9 policy areas has its designated folder where all the data sits, with a separate file for each of the metrics and classifications where applicable.

Main Files and Folders:

- **`app.R`** - Main dashboard file combining all other modules

- **`modules/`** - Contains individual policy area modules
  - 9 module files (e.g., `population_module.R`, `housing_module.R`)
  - `config.R` - Universal information like classifications and colors
- **Data folders** - Policy-specific folders containing CSV/Excel files
  - `population/`, `housing/`, `transport/`, etc.
- **`www/`** - Contains images, photos, and logos (must keep this name as it's R's default for web assets)

Module-Specific Notes: For modules with complex metrics (e.g., environment), define sub-metric vectors separately and reference them in `switch()` blocks in the server. Example: Environment module handles non-standard classifications and Scotland-only metrics differently from other modules.

General Tips

---

1. **File Access**: If you try to run the dashboard with one of the data files open on your machine, that file will fail to load. Always close data files before running the dashboard.

2. **Change One Thing at a Time**: Only modify one element at a time and test that it works before moving on to another change.

3. **Hero Sections**: This guide refers to "hero sections" - these are essentially blocks of code that control major visual elements.

4. **Finding Text to Modify**: Use Ctrl+F to search for text as it appears in the dashboard - this will take you directly to where it needs to be changed in the code.

5. **Documentation**: If any coding elements are unclear (like FontAwesome icons), Google the documentation for complete information on implementation. Use `cat()` statements in loading/aggregation functions for debugging, as seen in environment module.

6. **Connecting to GitHub**: Follow the instructions on this page: https://resources.github.com/github-and-rstudio/.

Main App.R File Guide

This covers what you can safely modify in the main app.R file without breaking the dashboard. More complex changes like adding new data sources or metrics are handled in individual module files.

What You Can Change in App.R

1. Homepage Content and Branding

Change the main title and tagline: Look for the hero section labeled `#Main UI`:

- Main heading “Rural Scotland Data Dashboard”
- Subtitle “Supporting the Rural Delivery Plan”
- Descriptive text in the “About the Dashboard” box

Update homepage background image: Find `background-image: url('home_page_img.jpg')` in the hero section. Replace `home_page_img.jpg` with your new image file path in the `www` folder.

Modify action buttons: The buttons “Explore Key Policy Areas” and “Explore Key Policy Metrics” can have their text changed by editing the `actionButton` labels in the hero section.

2. Navigation Menu Structure

Add or remove main navigation tabs: In the `sidebarMenu` section:

- Add new `menuItem()` entries for additional main sections
- Change display names of existing menu items (Home, Policy Areas, etc.)
- Change icons using different FontAwesome icon names

Reorder navigation tabs: Rearrange the `menuItem()` entries in your preferred order.

3. Visual Styling and Color Scheme

Change the main color palette: The dashboard uses two primary RESAS colors:

- Green (#0E450B) for headers, data table pagination, and hover effects

- Orange (#FDBE41) for main action buttons

Search for these hex codes in the CSS section and replace with your preferred colors.

Modify category tile appearance: In the CSS section, look for `.category-card` styles to change:

- Background colors of policy area tiles
- Hover effects and animations
- Border styles and spacing
- Text alignment for specific categories

4. Policy Area Categories

Update category names and descriptions: Find the `COMPLETE_METRICS` list at the beginning where each policy area is defined:

- Change display names (e.g., “Population & Skills” instead of “Population, Education and Skills”)
- Update FontAwesome icons for each category
- Modify which categories are marked as `data_available = TRUE`

5. Classification Information

Update rural classification explanations: In the home tab’s classification section:

- Introductory text about rural Scotland statistics
- Detailed explanations for each classification type (2-fold, 3-fold, etc.)
- Add or remove classification tabs
- Update definitions and descriptions for each area type

This is particularly useful if classification definitions change or if you want to emphasize different aspects of rural/urban distinctions.

6. About and Help Content

Modify the About section: In the “about” `tabItem`:

- Description of the dashboard’s purpose
- List of data sources
- Technical architecture information

- Contact or attribution information

Add new informational sections: Create additional `tabItem` entries for user guides, data methodology, or contact information.

7. Layout and Spacing

Adjust the category selection grid: Look for the grid layout in the categories section:

- `grid-template-columns: repeat(auto-fit, minmax(300px, 1fr))` to change tile sizing
- Gap spacing between tiles
- Padding within each tile

Modify header behavior: Category headers are set to be sticky/fixed at the top:

- Header height by changing `min-height` values
- Positioning behavior by modifying sticky CSS classes
- Background overlay opacity and colors

Warning: Sizing has been carefully designed - changing one element size will likely require cascade changes to positioning across other dashboard elements.

8. Warning and Error Handling

Customize error messages: The dashboard includes CSS and JavaScript to suppress certain Shiny warnings:

- Remove warning suppression if you want to see all messages during development
- Customize what types of warnings are hidden
- Add custom error messaging for specific scenarios

Note: Currently suppressed warnings relate to data table column naming mismatches and don't affect functionality.

9. Module Integration

When adding new categories to `COMPLETE_METRICS`, ensure module files handle special cases like Scotland-only metrics. Extend `reset_module_states()` for new modules to clear inputs on category switch.

What NOT to Change in App.R

- **Module loading and server functions**: Don't modify `source()` statements or `observe()` blocks
- **Reactive value structure**: The values reactive structure tracks navigation state
- **Core navigation logic**: The `observeEvent()` handlers for navigation should remain intact
- **JavaScript functionality**: Custom JavaScript for sidebar toggling and sticky headers is complex

---

Modifying Dropdown Buttons

Adding New Metric Options

Location: In each module file (e.g., `population_module.R`), find the metrics list:

```
population_metrics <- list(
  "Median age" = list(
    file_6fold = "population/median_age.xlsx",
    classifications = c("6-fold"),
    full_name = "Median age by urban rural classification"
  ),
  # Add new metric here:
  "Your New Metric" = list(
    file_6fold = "population/your_new_file.xlsx",
    classifications = c("2-fold", "6-fold"),
    full_name = "Full descriptive name for your metric"
  )
)
```

Key Points:

- First part in quotes becomes the dropdown option name
- `file_6fold` points to your data file
- `classifications` determines which geographic breakdowns are available
- `full_name` appears in titles and descriptions

Modifying Classification Options

What this controls: The second dropdown for geographic breakdowns (“Urban/Rural”, “6-fold”, etc.)

Available options:

- `"2-fold"` = Urban/Rural
- `"3-fold"` = Urban, Accessible Rural, Remote Rural
- `"6-fold"` = Six detailed categories
- `"4-fold"` = RESAS classification (some modules)

Adding Sub-Metrics

What this does: Creates additional dropdowns for metrics with subcategories.

Location: Find sub-metrics definitions:

```
participation_sub_metrics <- c(
  "Education" = "Participating (Education)",
  "Employment" = "Participating (Employment)",
  "Training" = "Participating (Training)"
)
```

Enable with: `has_sub_metrics = TRUE` in metric definition

Environment Examples: For non-urban/rural metrics (e.g., environment’s ‘Fresh water’), set classifications to sub-type names and add to `switch()` in server for choices. Add flags like `include_scotland=TRUE` in metrics list for Scotland averages.

Customizing Charts

Changing Chart Colors

Location: `modules/config.R` - Look for `CLASSIFICATION_COLORS`:

```
CLASSIFICATION_COLORS <- list(
  "2-fold" = c("Urban" = "#FDBE41", "Rural" = "#0E450B"),
  "6-fold" = c(
    "Large_Urban_Areas" = "#FDBE41",
```

```
    "Other_Urban_Areas" = "#F4E470"
    # Replace hex codes with your preferred colors
  )
)
```

Modifying Chart Types and Labels

Location: Find chart rendering functions (look for `renderPlotly`):

```
output$population_trend_chart <- renderPlotly({
  p <- ggplot(data, aes(x = Year, y = Value)) +
    geom_line() +  # Change to geom_bar() for bar charts
    labs(x = "Year", y = "Your Custom Label")
})
```

Common changes:

- `geom_line()` → `geom_bar(stat = "identity")` for bar charts
- `labs()` for custom axis labels
- `theme_minimal()` → `theme_classic()` for different styling

Value Formatting in Charts

Location: Find formatting functions:

```
format_population_value <- function(value, metric_name) {
  if (metric_name == "Your Metric") {
    return(paste0("$", scales::comma(value)))  # Currency format
  }
  return(paste0(round(value, 1), "%"))  # Percentage format
}
```

Special Chart Flags and Behaviors

In metrics list, add `no_bar_chart=TRUE` to disable comparison charts (e.g., for Scotland-only). Use `units='your unit'` for custom labels. For multi-line metrics, aggregation stacks sub-metrics as separate lines.

Modifying Data Tables

Changing Column Names

Location: In data table output section:

```
colnames = c("Year", "Area", "Your Custom Column Name")
```

Adding or Removing Columns

Location: Table preparation code:

```
table_data <- filtered_data %>%
  select(Year, Area, Value, New_Column) %>%  # Add/remove columns
  arrange(Year, Area)
```

Modifying Table Filters

What this controls: Dropdowns above tables for filtering by year, area, etc.

Location: Find filter UI outputs:

```
output$population_table_year_filter <- renderUI({
  choices <- list(
    "All Years" = "all",
    "Recent 5 Years" = "recent5"
  )
  selectInput("filter_name", "Filter Label:", choices = choices)
})
```

Unit-Based Metrics

For unit-based metrics (e.g., environment's Renewable Electricity), add custom `Value_Display` in `renderDataTable` with `paste0(round(value), ' unit')`.

Customizing Key Insights

Key Insights are the colored summary statistic boxes.

Adding New Value Boxes

UI section: Add to fluidRow containing valueBoxOutput:

```
fluidRow(
  valueBoxOutput("existing_box1", width = 3),
  valueBoxOutput("your_new_box", width = 3),  # Add this
  valueBoxOutput("existing_box2", width = 3)
)
```

Server logic:

```
output$your_new_box <- renderValueBox({
  valueBox(
    value = "Your Value",
    subtitle = "Your Description",
    icon = icon("chart-line"),
    color = "blue"
  )
})
```

Changing Value Box Layout

Modify width parameters (must total 12):

```
fluidRow(
  valueBoxOutput("box1", width = 4),
  valueBoxOutput("box2", width = 4),
  valueBoxOutput("box3", width = 4)
)
```

Note: Update placeholder insights in module-specific `key_insights` lists with real data summaries.

Adding Sources and Notes

Adding Static Notes

Location: Create or find notes lists in modules:

```
population_notes <- list(
  "Median age" = "Data collected every 5 years from census.",
  "Your Metric" = "Important data limitations note."
)

population_key_insights <- list(
  "Median age" = "Rural areas tend to have older populations.",
  "Your Metric" = "Key finding for this metric."
)
```

Adding Data Source Information

Location: In data summary rendering:

```
source_info <- switch(input$population_metric,
  "Your New Metric" = list(
    text = "Your Data Source Name",
    url = "https://your-source-website.com"
  ),
  list(text = "Scottish Government", url = "https://www.gov.scot/")
)
```

Environment Module: For environment, add `Data_Source` in loading functions; consider adding `source_url` flag like in health module.

Adding/Removing Dashboard Sections

Removing Sections

Comment out sections in UI:

```
# fluidRow(
#   box(title = "Trend Analysis", ...)
# ),
```

Adding New Sections

UI addition:

```
fluidRow(
  box(
    title = "Your New Section",
    status = "primary",
    width = 12,
    plotlyOutput("your_new_chart")
  )
)
```

Server function:

```
output$your_new_chart <- renderPlotly({
  plot_ly(data = your_data, x = ~x_var, y = ~y_var)
})
```

Conditional Sections

Show sections only for specific metrics:

```
conditionalPanel(
  condition = "input.population_metric == 'Your Specific Metric'",
  fluidRow(
    box(title = "Special Analysis", ...)
  )
)
```

Environment Considerations: For new environment metrics, must handle special types (Scotland-only, multi-line) in aggregation/charts. E.g., add to switch for `sub_metrics_choices`.

Adding New Data Files and Classifications

Data File Format Requirements

Required structure (Excel or CSV):

- Column 1: Geographic area names
- Remaining columns: Years as headers
- Numeric data values
- Consistent area names matching existing files

Example:

```
Region          | 2020 | 2021 | 2022
Large Urban     | 45.2 | 46.1 | 47.0
Rural           | 38.5 | 39.2 | 40.1
```

Adding Files for Existing Classifications

Step 1: Place file in correct folder (`population/your_metric.xlsx`)

Step 2: Update metrics list:

```
"Your New Metric" = list(
  file_6fold = "population/your_metric.xlsx",
  classifications = c("6-fold"),
  full_name = "Descriptive name"
)
```

Step 3: Add loading function:

```
load_your_metric_data <- function() {
  tryCatch({
    raw_data <- read_excel("population/your_metric.xlsx")
    # Process to standard format
    processed_data <- raw_data %>%
      gather(key = "Year", value = "Value", -Region) %>%
      mutate(
        Year = as.numeric(Year),
        Area = Region,
        Data_Source = "Your Source"
      )
    return(processed_data)
  }, error = function(e) {
    return(data.frame())
```

```
  })
}
```

Step 4: Connect to main loader:

```
load_population_data_simple <- function(metric_name, classification_type) {
  if (metric_name == "Your New Metric") {
    return(load_your_metric_data())
  }
}
```

Environment-Like Modules: For environment-like modules, update aggregation for new types (e.g., if `classifications='Scotland'`, return Scotland data only). Add to `sub_metrics` switch if `has_sub_metrics=TRUE`.

Adding Completely New Metrics

Complete Process

Step 1: Add to metrics list

```
population_metrics <- list(
  "New Metric Name" = list(
    file_6fold = "population/new_metric.xlsx",
    classifications = c("2-fold", "6-fold"),
    full_name = "Full descriptive name"
  )
)
```

Step 2: Create loading function

```
load_new_metric_data <- function() {
  # Your data processing logic
}
```

Step 3: Add to main dispatcher

```
load_population_data_simple <- function(metric_name, classification_type) {
  if (metric_name == "New Metric Name") {
```

```
    return(load_new_metric_data())
  }
}
```

Step 4: Add formatting rules

```
format_population_value <- function(value, metric_name) {
  if (metric_name == "New Metric Name") {
    return(paste0(round(value, 2), " units"))
  }
}
```

Step 5: Add insights and notes

```
population_key_insights <- list(
  "New Metric Name" = "Key insight about this metric"
)
```

Environment Module Considerations: For new environment metrics, must handle special types (Scotland-only, multi-line) in aggregation/charts. E.g., add to switch for `sub_metrics_choices`.

Module-Specific Variations (e.g., Environment)

The environment module represents the most complex implementation in the dashboard, with several special features that differ from standard modules:

Special Metric Flags

Scotland-only metrics: Use `no_bar_chart=TRUE` to disable comparison charts when data only exists for Scotland as a whole.

Custom units: Add `units='MW'` or similar to display custom units in charts and tables.

Accentuate differences: Use `accentuate_difference=TRUE` for metrics where small changes are significant.

Repurposed Classifications

Instead of standard urban/rural classifications, environment uses a number of different metrics and subclassifications.

Multi-line Handling

For metrics with multiple data series:

```
# In aggregation functions
if (has_sub_metrics) {
  # Stack sub-metrics as separate lines
  aggregated_data <- data %>%
    group_by(Year, Sub_Metric) %>%
    summarise(Value = sum(Value, na.rm = TRUE))
}
```

Custom Switch Blocks

```
sub_metrics_choices <- switch(input$environment_metric,
  "Fresh water" = c("Rivers" = "Rivers", "Lochs" = "Lochs"),
  "Renewable electricity" = c("Wind" = "Wind", "Hydro" = "Hydro"),
  NULL
)
```

Integration with Standard Framework

Despite complexity, environment integrates with the standard dashboard through:

- Consistent data loading patterns
- Standard chart rendering with custom parameters
- Unified error handling and debugging

Common Customizations

Layout Changes

Two-column to one-column:

```
# From:
fluidRow(
  column(6, box(title = "Chart 1")),
  column(6, box(title = "Chart 2"))
)

# To:
fluidRow(
  column(12, box(title = "Chart 1")),
  column(12, box(title = "Chart 2"))
)
```

Adding Download Functionality

UI button:

```
tags$a(
  class = "excel-download-btn",
  onclick = "Shiny.setInputValue('download_data', Math.random());",
  icon("download"), "Download"
)
```

Server handler:

```
observeEvent(input$download_data, {
  openxlsx::write.xlsx(your_data, "filename.xlsx")
  showNotification("Downloaded!", type = "message")
})
```

Note: Adapt downloads to CSV if preferred; test special metrics (Scotland-only) separately.

Testing Your Changes

Basic Testing Workflow

1. **Save all files** before testing
2. **Test locally**: Run `shiny::runApp()` in R console

3. **Check functionality**: Test all dropdowns and features
4. **Check browser console**: Press F12, look for errors
5. **Start small**: Make one change at a time

Common Issues and Solutions

"Object not found" errors: Check variable names match between UI and server

Charts don't display: Ensure Year and Value columns are numeric

Dropdowns are empty: Verify file paths and file existence

Colors don't change: Clear browser cache (Ctrl+F5) and restart R

New metrics don't appear: Check exact name matching in lists vs. loading functions

Debugging Techniques

Add debug messages:

```
cat("Debug: Loading", metric_name, "\n")
cat("Data dimensions:", nrow(data), "x", ncol(data), "\n")
```

Test data loading separately:

```
test_data <- load_your_new_metric_data()
print(head(test_data))
```

Pre-Help Checklist

Before requesting assistance, verify:

- [ ] File paths are correct and files exist
- [ ] Column names match code expectations
- [ ] Metric names are consistent throughout
- [ ] Required R packages are installed
- [ ] No typos in variable/function names
- [ ] Data is in expected format
- [ ] Browser cache cleared after CSS changes

Important: Always keep backups of working versions and make incremental changes. The modular structure makes it relatively safe to modify individual components, but test frequently to catch issues early. The guide covers generic cases well, but modules like environment require special attention to flags (`no_bar_chart` , `units` , `accentuate_difference`), repurposed classifications, and multi-line handling.