Creating Paramterized Reports with {R Markdown}

Introduction . . . . .
Learning Objectives . . . . .
Packages . . . . .
Dataset . . . . .
Build a Single Report for One Country (e.g., “Angola”) . . . . .
DRY Principle: Do Not Repeat Yourself! . . . . .
Building a Parametrized Report . . . . .
The Whole Game . . . . .
WRAP UP! . . . . .

Introduction

Parametrizing reports represents a significant task in epidemiological reporting, offering the flexibility to dynamically generate content based on specific parameters. This technique is invaluable, particularly in scenarios requiring report generation across varying factors, such as different countries, time periods, or disease incidences. The essence of parameterization lies in its ability to utilize a single, versatile template to produce a multitude of reports.

This not only ensures consistency and accuracy but also significantly enhances the efficiency of epidemiological data communication. With parameterized reports, complex data can be presented in a more accessible and interpretable format, aiding in the effective dissemination of critical health information.

Learning Objectives

1. **Understand the Importance of Parameterization in R Markdown**: Grasp the fundamental concept of parameterization, its significance in epidemiological reporting, and how it transforms the way data is presented and analyzed.

2. **Learn to Create Dynamic Reports Based on Specified Parameters**: Develop the skill to craft reports in R Markdown that automatically adjust their content based on given parameters, such as geographical regions or time frames.

3. **Develop Skills in Creating Functions for Report Parameterization**: Acquire proficiency in writing functions in R that enable the parameterization of reports, thereby simplifying the process of report generation.

4. **Explore Functional Programming with `map()` and `pwalk()`**: Delve into the functional programming aspects of R, specifically focusing on the usage of `map()` and `pwalk()` functions from the {purrr} package. Understand how these functions can be utilized to efficiently handle multiple sets of data inputs for report generation.

By achieving these objectives, you will be well-equipped to handle epidemiological datasets and present them in an informative, organized, and impactful manner. The ability to parameterize reports not only streamlines the process of data analysis but also greatly enhances the versatility and applicability of the findings presented.

Packages

This code snippet bellow shows how to load necessary packages using `p_load()` from the {pacman} package. If not installed, it installs the package before loading.

```
pacman::p_load(readr, ggplot2, dplyr, knitr, kableExtra, purrr)
```

```
## Error in download.file(url, destfile, method, mode = "wb", ...) :
##   download from
'https://cran.rstudio.com/bin/windows/contrib/4.3/kableExtra_1.3.4.zip'
failed
```

Dataset

```
hiv_data <- read_csv(here::here("data/clean/hiv_incidence.csv"))
```

The dataset `hiv_incidence.csv` contains three columns:

1. `country`: The name of the country.
2. `year`: The year of the record.
3. `new_cases`: The number of new HIV cases reported in that country for the given year.

With this understanding, let's proceed to create the reports as outlined earlier. We will adapt the steps to fit this specific dataset. The tutorial will demonstrate how to build reports focusing on HIV prevalence data for different countries across various years.

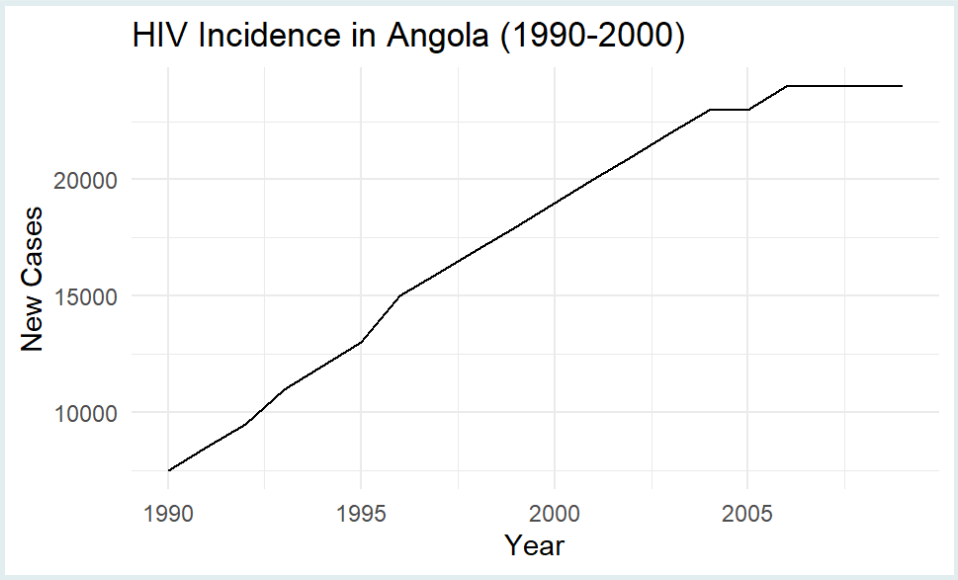
Build a Single Report for One Country (e.g., "Angola")

In this section, we focus on creating a visual report for a specific country, Angola, using R programming language. Our goal is to analyze and visualize the trend in new HIV cases in Angola over a set of years.

Output 1: Line Plot for "Angola"

The objective is to create a line plot that represents the trend in new HIV cases in Angola over the years. A line plot is an appropriate choice for this task as it clearly shows changes over time.

```
angola_data <- subset(hiv_data, country == "Angola")
ggplot(angola_data, aes(x = year, y = new_cases)) +
  geom_line() +
  theme_minimal() +
  labs(title = "HIV Incidence in Angola (1990-2000)",
       x = "Year",
       y = "New Cases")
```



Output 2: Statistical Table for “Angola”

This code snippet bellow is designed to generate a summary table for Angola, focusing on HIV cases by year. The table will highlight the most recent year’s data for easier identification. Let’s break down and explain each part of the code.

```
library(dplyr)
library(knitr)
library(kableExtra)
```

```
## Error: package or namespace load failed for 'kableExtra':
##  .onLoad failed in loadNamespace() for 'kableExtra', details:
##   call: !is.null(rmarkdown::metadata$output) && rmarkdown::metadata$output
%in%
##   error: 'length = 2' in coercion to 'logical(1)'
```

```r
# Filter for Angola data
angola_data <- hiv_data %>% filter(country == "Angola")

# Summarize data by year
angola_summary <- angola_data %>%
    group_by(year) %>%
    summarise(Total_Cases = sum(new_cases))

# Identify the most recent year
most_recent_year <- max(angola_summary$year)

# Display the table using kable and highlight the most recent year
angola_summary %>%
    kable("html", caption = "Summary of HIV Cases in Angola by Year") %>%
    kable_styling(bootstrap_options = c("striped", "hover")) %>%
    row_spec(which(angola_summary$year == most_recent_year), background =
        "lightblue")
```

```
## Error in row_spec(., which(angola_summary$year == most_recent_year),
background = "lightblue"): could not find function "row_spec"
```

Build a simple report using R makrdown file

Now that you have two nicely prepared outputs – a line plot and a statistical table for Angola's HIV data – you can compile them into a simple, cohesive report using R Markdown. This report will not only showcase your analytical skills but also your ability to communicate findings effectively.

Steps to Follow:

1. Create a new RMarkdown file
2. Write the YAML header
3. Add a data preparation section
4. Add a line plot and statistical table
5. Knit the RMarkdown file

Build a Report for Multiple Countries (e.g., "Angola", "Nigeria", "Mali")

In addition to Angola, we want to create the same report for Nigeria and Mali. This challenge requires you to extend the analytical approach used for Angola to these additional countries.

Is there a more streamlined method to do this without duplicating our lines of code?

DRY Principle: Do Not Repeat Yourself!

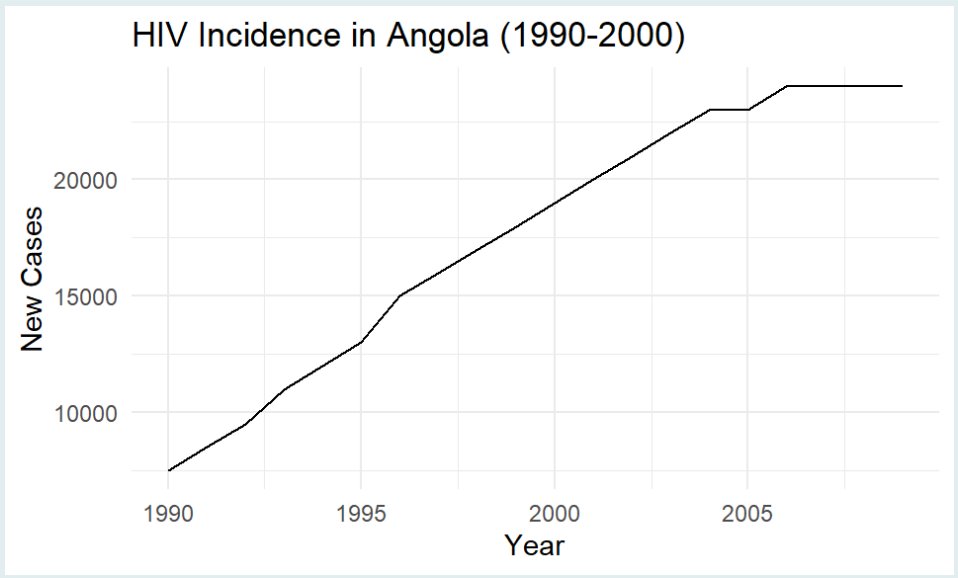
Approach 1: Produce Output Through a Customized Function

The function `generate_country_report()` bellow, is designed to automate the process of generating a report for a given country from a dataset. This approach follows the DRY (Don't Repeat Yourself) principle by consolidating repetitive tasks into a single function.

DRY (Don't Repeat Yourself): This is a principle of software development aimed at reducing repetition of software patterns, replacing them with abstractions or using data normalization to avoid redundancy. In essence, DRY promotes the use of functions or methods instead of repetitive blocks of code. This principle is fundamental in various programming languages and design practices.

```r
generate_country_report <- function(data, country_name) {

  # Plotting the data
  country_data_plot <- subset(data, country == country_name)
  p <- ggplot(country_data_plot, aes(x = year, y = new_cases)) +
    geom_line() +
    theme_minimal() +
    labs(title = paste("HIV Incidence in", country_name, "(1990-2000)"),
         x = "Year",
         y = "New Cases")
  print(p)

  # Creating the summary table
  country_data_table <- data %>%
    filter(country == country_name) %>%
    group_by(year) %>%
    summarise(Total_Cases = sum(new_cases))

  # Identify the most recent year
  most_recent_year <- max(country_data_table$year)

  # Display the table using kable and highlight the most recent year
  table_output <- country_data_table %>%
    kable("html", caption = paste("Summary of HIV Cases in", country_name, "by
        Year")) %>%
    kable_styling(bootstrap_options = c("striped", "hover")) %>%
    row_spec(which(country_data_table$year == most_recent_year), background =
        "lightblue")

  print(table_output)
}

# Example usage of the function
generate_country_report(hiv_data, "Angola")
```

```
## Error in row_spec(., which(country_data_table$year == most_recent_year), :
could not find function "row_spec"
```



Let's explain this code:

1. **Function Declaration**: The function `generate_country_report()` takes two parameters: `data` (the dataset) and `country_name` (the name of the country for which the report is to be generated).

2. **Data Plotting**: The function begins by subsetting the data for the specified country. It then creates a line plot of new HIV cases over the years using {ggplot}.

3. **Summary Table Creation**: Next, it filters the data for the specified country, groups it by year, and calculates the total number of new cases for each year.

4. **Highlighting Recent Data**: The function identifies the most recent year in the data and highlights it in the summary table.

5. **Table Display**: The summary table is displayed using `kable()` and `kable_styling()` for a clean and interactive HTML output. The row corresponding to the most recent year is highlighted.

6. **Printing Outputs**: Both the plot (`p`) and the table (`table_output`) are printed to the output.

Functional Programming Using {purr}:

For scaling this approach to handle multiple countries efficiently, consider using functional programming paradigms, specifically with the {purr} package in R. This allows for iterating over a list of

countries and applying the `generate_country_report()` function to each, streamlining the process and adhering to the DRY principle.

This function-based approach makes the code more organized and readable and also enhances its reusability and scalability, crucial aspects of efficient programming practices.

Using `map()`

Suppose you have a list of countries and you need to generate reports for each one. The `map()` function from the {purrr} package is an effective tool for this task, allowing you to apply the `generate_country_report()` function to each country in the list.

Here's an example of how you can use `map()` with `generate_country_report()`:

```
library(purrr)
# Vector of countries
countries <- c("Angola", "Nigeria", "Mali")

# Assuming 'hiv_data' is your dataset
# Apply 'generate_country_report' to each country
map(countries, ~generate_country_report(hiv_data, .))
```

Explanation

- `countries` is a vector containing the names of the countries for which you want to generate reports.
- `map()` takes this vector and applies the `generate_country_report()` function to each element (country name). The `~` symbol is used to define a formula where `.` represents the current element in the vector.
- For each country in `countries`, `generate_country_report(hiv_data, .)` is called, thus generating a report for that country.

This approach is advantageous when creating similar reports for multiple countries. It automates the process and efficiently applies the function to each country, ensuring consistency and saving time.

Building a Parametrized Report

YAML Header Configuration

- **Defining Parameters**: Parameters are defined in the YAML header. This is the section at the beginning of your R Markdown file where you specify various settings for your document, such as its title, output format, and parameters.

Example with an HIV Dataset

Consider an R Markdown file designed to generate a report on HIV data for different countries. You can set a parameter in the YAML header to dynamically change the country for which the report is generated.

YAML in the R Markdown file:

```
---
title: "Country HIV Report"
output: html_document
params:
  country: "Angola"
---
```

Explanation

1. **Document Title and Output**: In the YAML header, the document is titled "Country HIV Report", and the output format is specified as an HTML document.
2. **Parameter Definition**:
   - `params`: This field is used to define parameters.
   - `country`: Under `params`, a parameter named `country` is declared. This parameter will control which country's data is used in the report.
   - `"Angola"`: The default value for the `country` parameter is set to "Angola". This means that when the report is generated without specifying a different country, it will automatically use data for Angola.

Using the Parameter in the Document

Inside the R Markdown document, you can reference this parameter using `params$country`. For example, if you have a function like `generate_country_report()`, you can call it within a code chunk as `generate_country_report(data, params$country)`. This will use the value of the `country` parameter to determine which country's data to report on.

By using parameters, your R Markdown document becomes more adaptable. If you need to generate a report for a different country, you simply change the parameter value when knitting the document, rather than altering the code itself. This makes your document more efficient and versatile, especially when dealing with reports that require frequent updates or modifications based on different data subsets.

Implementing Parameters in the Analysis

- **Filtering Data Based on Parameters:** Use the parameters to dynamically filter the TB dataset. For example, if the report is for a specific country and year, you can filter the data accordingly using `dplyr`.

Example code snippet

```
filtered_data <- tb_data %>%
                  filter(country == params$country, year == params$year)
```

Explanation of Parameter Mapping

- **Accessing Parameters**: Parameters defined in the YAML header of your R Markdown document can be accessed in the code using `params$parameter_name`. This is similar to how you access columns in a dataframe using `dataframe$column_name`.
- **Dynamic Reporting**: When you create visualizations or summaries, you use `params$country` and `params$year` to ensure that the output reflects the specific country and year set as parameters. This means the content of the report will adapt based on the values provided for these parameters at the time of knitting the document.

In the context of R Markdown with parameters, `params$parameter_name` accesses the value of a parameter named `parameter_name` that was defined in the YAML header. This enables dynamic and interactive report generation based on these parameter values.

Overall, the `$` operator is a key part of R syntax that facilitate manipulating elements in R pbjects.

Knitting with Different Parameters

- **Knitting from RStudio:** Explain how to knit the document with different parameter values directly from RStudio's knit button, where you can specify the parameter values in a dialog box.
- **Command Line Knitting:** For automated report generation, you can use R commands to knit the document with different parameters. This is particularly useful for batch processing or automated report generation.
    - Example command:

```
rmarkdown::render("TB_Report.Rmd", params = list(country = "Angola"))
```

Knitting with Different Parameters

Creating dynamic and flexible reports in R Markdown often involves knitting the document with varying parameters. There are two main ways to do this: directly through RStudio's interface and via the command line for automation.

Knitting from RStudio

- **Using RStudio's Knit Button**: When you knit an R Markdown document in RStudio that contains parameters, a dialog box will appear. This dialog allows you to specify values for each parameter before knitting.
- **Interactive Parameter Selection**: This method is user-friendly and interactive, ideal for ad-hoc report generation where you can manually enter or change parameter values each time you knit the document.
- **Customizing Parameters**: In the dialog box, you can customize the parameters according to your needs. For instance, if you have a parameter for country, you can change its value to generate a report for a different country.

Command Line Knitting

- **Automated Report Generation**: For scenarios where you need to automate the report generation process, such as in batch processing or scheduled reports, you can knit documents using R commands.
- **Example Command**:

```
rmarkdown::render("TB_Report.Rmd", params = list(country = "Angola"))
```

This command uses the `render` function from the `rmarkdown` package. It specifies the file to be rendered (`"TB_Report.Rmd"`) and sets the `params` argument to a list of desired parameter values. In this example, the country parameter is set to "Angola".

- **Flexibility and Scripting**: Command line knitting allows for more flexibility and can be integrated into scripts for automated workflows. This is particularly useful when dealing with large datasets or needing to generate multiple reports periodically.

Both methods provide the means to leverage the power of parametrized reports in R Markdown, catering to different needs — from interactive, user-driven report generation to automated, script-driven processes. This flexibility is one of the key strengths of using R Markdown for data analysis and reporting.

The Whole Game

Creating a parametrized report in R involves several key steps, from setting up your R project to automating the report generation process. Let's go through these steps in detail:

Step 0: Create an R Project

- **.Rproj File**: Start by creating an R project with a `.Rproj` file. This file helps manage paths and settings, making it easier to work with files and data within the project.

Step 1: YAML Header

Here's an example of the YAML header for your R Markdown document:

```
---
title: "Report - `r stringr::str_to_title(params$state)`"
date: "`r format(Sys.time(), '%B %d, %Y')`"
output: html_document
editor_options:
  chunk_output_type: console
params:
  state: "Alabama"
---
```

This header sets up the document title, date, output format, and defines a parameter `state` with a default value of "Alabama".

Step 2: Set-Up Your Document Environment

In the initial code chunk, set up your environment by loading required packages and your dataset:

```
# Set-up global options
knitr::opts_chunk$set(warning = FALSE, message = FALSE, echo = FALSE)

# Load packages
pacman::p_load(ggplot2, dplyr, knitr, kableExtra)

# Load the dataset
hiv_data <- read.csv("hiv_incidence.csv")
```

Step 3: The Core of Your Report

Now, add sections with outputs, dynamically generated based on the selected state.

HIV Incidence Trend

```
hiv_incidence_data <- subset(hiv_data, country == params$country)
ggplot(hiv_incidence_data, aes(x = year, y = new_cases)) +
  geom_line() +
  theme_minimal() +
  labs(title = paste("HIV Incidence in", params$country, "(1990-2000)"),
       x = "Year",
       y = "New Cases")
```

HIV Cases Summary Table

```
## HIV Cases Summary Table

# Filter for selected country data
country_data <- hiv_data %>% filter(country == params$country)

# Summarize data by year
country_summary <- country_data %>%
    group_by(year) %>%
    summarise(Total_Cases = sum(new_cases))

# Identify the most recent year
most_recent_year <- max(country_summary$year)

# Display the table using kable and highlight the most recent year
country_summary %>%
    kable("html", caption = paste("Summary of HIV Cases in", params$country,
        "by Year")) %>%
    kable_styling(bootstrap_options = c("striped", "hover")) %>%
    row_spec(which(country_summary$year == most_recent_year), background =
        "lightblue")
```

Step 4: Knitting the Report

The document can now be knitted to generate a report for the default or specified state.

CHALLENGE

Create a new R Markdown with these components and knit it to see the output.

Step 5: Create an R Script to Parameterize the Whole Process

For automation, use an R script to parameterize the process:

```r
# Load necessary packages
pacman::p_load(rmarkdown, purrr, stringr)

# Define a vector of states
states <- c("Alabama", "Alaska", "Arizona")

# Generate a tibble for reports
reports <- tibble(
  filename = str_c("state_report_", states, ".html"),
  params = map(states, ~list(state = .))
)

# Use pwalk to render each report
reports %>%
  select(output_file = filename, params) %>%
  pwalk(rmarkdown::render, input = "state_report.Rmd", output_dir = "output/")
```

This script uses `map()` and `pwalk()` to iterate over each state, rendering an individualized report for each.

The `map()` function creates a list of parameters for each state, and `pwalk()` applies the `rmarkdown::render()` function to each set of parameters, generating the reports.

With these steps, you can efficiently generate customized reports for multiple states or criteria, streamlining your data analysis and reporting workflow in R.

WRAP UP!

To wrap up, we've explored a comprehensive workflow for creating parametrized reports using R Markdown. This process allows for dynamic and efficient report generation tailored to specific criteria, such as geographic regions or time frames.

Learning outcomes

At the conclusion of this lesson, you must be able to:

- **Efficiently Manage R Projects**: Organize and manage R projects for various data analysis tasks.
- **Produce Dynamic R Markdown Documents**: Generate R Markdown documents that adapt to different data inputs.
- **Conduct Data Analysis and Visualization**: Analyze data and create visualizations, focusing on real-world datasets like HIV incidence.
- **Develop Automation Scripts for Reports**: Create scripts to automate the generation of parameter-based reports, enhancing consistency and efficiency.
- **Adopt Best Practices in R Programming**: Implement best practices for more efficient, readable, and maintainable R programming.

- **Enhance Report Creation Skills**: Improve capabilities in crafting comprehensive, informative, and visually appealing reports for diverse audiences.
- **Utilize Functional Programming Techniques**: Apply functional programming methods in R for streamlined data processing and reporting.

Happy rendering!

Contributors

The following team members contributed to this lesson:

IMAD EL BADISY
Data Science Education Officer
Deeply interested in health data

KENE DAVID NWOSU
Data analyst, the GRAPH Network
Passionate about world improvement

JOY VAZ
R Developer and Instructor, the GRAPH Network
Loves doing science and teaching science

References

- Johnson, Paul. “R Markdown: The Definitive Guide: Yihui Xie, JJ Allaire, and Garrett Grolemund. Boca Raton, FL: Chapman & Hall/CRC Press, 2018, xxxiv+ 303 pp., $38.95 (P), ISBN: 978-1-13-835933-8.” (2020): 209-210.