

APAC 2026

Beyond Basics: Essential R Code Tips and Tricks for Clinical Programmers

Sanjana Jayacumar,
Pfizer Healthcare India Pvt. Ltd.



Disclaimer

I confirm that, the opinions and thoughts discussed in this presentation are subject to my own independent views and are not subject to the opinions of the organization that I represent

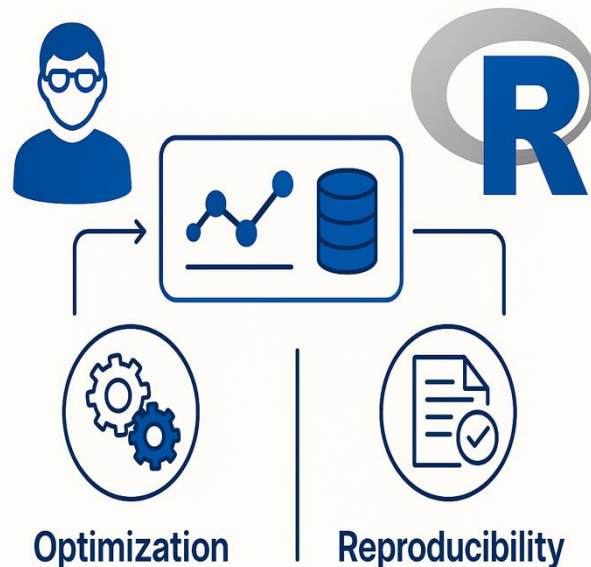


Introduction

- The clinical research industry is increasingly embracing **open-source** tools.
- Among these, **R** has emerged as a powerful solution for statistical programming.

Challenges

- Managing diverse datasets and implement complex decision logic with precision with efficiency.
- Often face challenges with syntax **optimization** and **reproducibility**.





Agenda



Dynamic Data Loading



Handling Joins & Column Operations



Regex for Dynamic Column Selection



Version Control for Reproducibility



Functional Iteration for Scalability



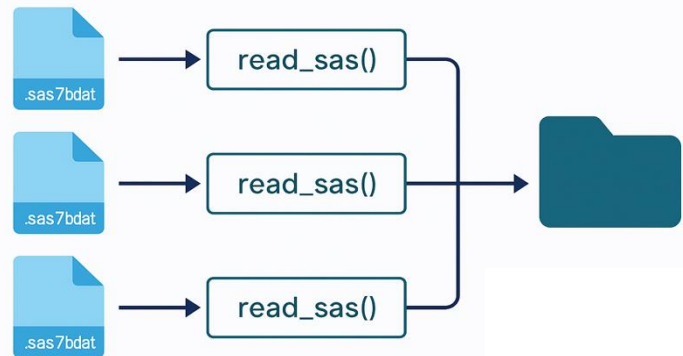
Dynamic Data Loading

Problem Statement

In SAS, LIBNAME statement can point to a folder, and all datasets in that library are accessible by name without additional code

R Code

```
dm <- read_sas("<library path>/dm.sas7bdat")  
adsl <- read_sas("<library path>/adsl.sas7bdat")  
adsl_s002 <- read_sas("<library path>/adsl_s002.sas7bdat")
```



Manually reading each file with separate read_sas() calls



Time-consuming, error-prone, and doesn't scale as new datasets are added

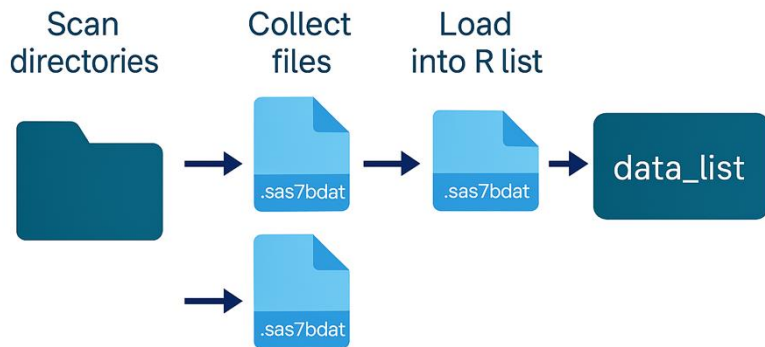


Dynamic Data Loading

Code Tip

```
paths <- c( "<library path>", "<library path>", "<library path>" )  
  
all_files <- unlist(lapply(paths, list.files, pattern = "\\..sas7bdat$", full.names = TRUE))  
  
data_list <- setNames(lapply(all_files, read_sas), tools::file_path_sans_ext(basename(all_files)))
```

How it Works



Advantages



New datasets
No code changes



Avoids repetition
& typos



Scans multiple
library paths



Handling Joins & Column Operations

Problem Statement

After merging two datasets, duplicate columns with suffixes .x and .y are created for common variables

SUBJID.x	AGE.x	SEX.x	RACE.x	SUBJID.y	AGE.y	SEX.y	RACE.y
Subject Identifier for the Study	Age	Sex	Race	Subject Identifier for the Study	Age	Sex	Race
10011001	43	M	WHITE	NA	NA	NA	NA
10011002	53	M	WHITE	10011002	53	M	WHITE
10011002	53	M	WHITE	10011002	53	M	WHITE

How it works



I
Identify

C
Coalesce

X
Remove

- **Identify:** columns ending with .x → strip suffix to get base names
- **Coalesce:** for each base, merge .x and .y into a single column
- **Remove:** drop the original .x and .y columns

Handling Joins & Column Operations

Code Tip

```
<input dataframe> <- <input dataframe> |>
  (\(df) {bases <- names(df) |>
    keep(~ str_ends(.x, ".x")) |>
    str_remove("\\.x$")
  } df |>
  mutate(!!!set_names(
    map(bases, ~ coalesce(df[[paste0(.x, ".x")]], df[[paste0(.x, ".y")]])), bases))
  )() |>
  select(-ends_with(".x"), -ends_with(".y"))
```

Objectives



Consolidate **duplicate columns** created by joins (e.g., var.x and var.y) into a single variable



Use values from the left dataset and fallback to the right when left is missing



Avoids manual rename/mutate/drop steps for each variable.



Regex for Dynamic Column Selection

Problem Statement

To select columns in a dataset that represent different periods, and flags regardless of zero-padding.

USUBJID Unique Subject Identifier	TRT01P Planned Treatment for Period 01	TRT02P Planned Treatment for Period 02	TRT03P Planned Treatment for Period 03	TRT01A Actual Treatment for Period 01	TRT02A Actual Treatment for Period 02	TRT03A Actual Treatment for Period 03	ANL01FL Analysis Flag 01:Window Record	ANL02FL Analysis Flag 02:In TRT Period	ANL03FL Analysis Flag 03:Last Per Value	ANL04FL Analysis Flag 04:Largest Increase
--------------------------------------	---	---	---	--	--	--	---	---	--	--

R Code

```
advs <- advs %>%  
  select(USUBJID, TRT01P, TRT02P, TRT03P, TRT01A, TRT02A, TRT03A, ANL01FL,  
         ANL02FL, ANL03FL, ANL04FL)
```



Need to re-write selections for each study with different period and flag counts.



Manual updates every time a new column appears.



Regex for Dynamic Column Selection

Code Tip

```
advs <- advs |>
  select(USUBJID, matches("^TRT\\d+P$"), matches("^TRT\\d+A$"), matches("^ANL\\d+FL$"))
```

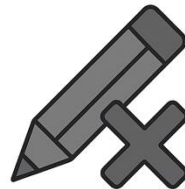
How it works

- ✓ **^** → start of the string
- ✓ **TRT/ANL** → Prefix
- ✓ **\\d+** → one or more digits (handles zero-padded numbers too)
- ✓ **P\$/A\$/FL\$** → Ends with P (or) A (or) FL

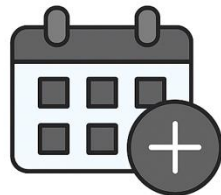
Objectives



Regex-based **automatically** captures all period-specific treatment and analysis flag columns



Reduces **manual updates** and typos.



If additional periods are added **automatically** includes them



Version Control for Reproducibility

Problem Statement

- ❖ Package versions and dependencies change frequently.
- ❖ Without version control, analyses may produce different results over time or fail to run on another machine and audit risks in clinical workflows.

Why It Matters

- 🔄 Ensures **reproducibility** across teams and time.
- 📋 Meets regulatory expectations for **traceability** in clinical workflows.
- ⚠️ Prevents **unexpected changes** due to package updates.
- 👥 Improves **collaboration** and **auditability**.

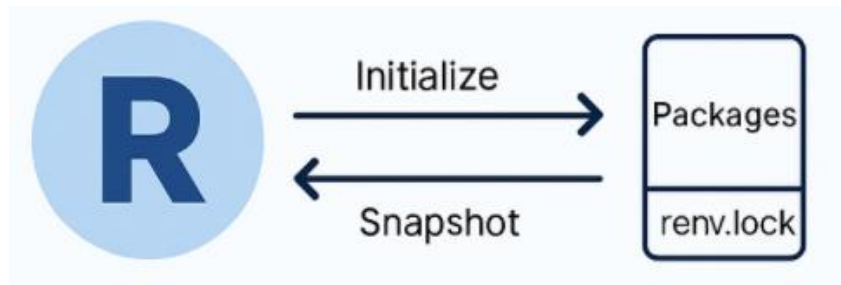




Version Control for Reproducibility

Code

```
renv::init()  
renv::install()  
renv::snapshot()  
renv::restore()
```



How It Works



`renv::init()` – Initializes a **project-specific environment** and creates a lockfile for reproducibility.



`renv::install()` – **Installs packages** into the project-local library managed by renv.



`renv::snapshot()` – Captures the current package versions into **renv.lock** for version control.



`renv::restore()` – **Restores** the project environment to match the versions recorded in renv.lock.



Functional Iteration for Scalability

Problem Statement

- ❖ Iterating over data frames with repetitive logic increases complexity and reduces readability in data workflows.

Why It Matters



Efficiency

Reduces repetitive code and speeds up iteration across datasets.



Consistency

Ensures the same transformation is applied uniformly across all elements.



Readability & Maintainability

Produces cleaner, more concise code.



Scalability

Handles growing datasets and integrates with complex workflows



Functional Iteration for Scalability

Adverse Events by Maximum Severity

SYSTEM ORGAN CLASS Preferred Term	Drug A			Drug B			Drug C		
	Mild	Moderate	Severe	Mild	Moderate	Severe	Mild	Moderate	Severe
CARDIAC DISORDERS	xx (xx.x)	xx (xx.x)	xx (xx.x)	xx (xx.x)	xx (xx.x)	xx (xx.x)	xx (xx.x)	xx (xx.x)	xx (xx.x)
Atrioventricular block	xx (xx.x)	xx (xx.x)	xx (xx.x)	xx (xx.x)	xx (xx.x)	xx (xx.x)	xx (xx.x)	xx (xx.x)	xx (xx.x)
GASTROINTESTINAL DISORDERS	xx (xx.x)	xx (xx.x)	xx (xx.x)	xx (xx.x)	xx (xx.x)	xx (xx.x)	xx (xx.x)	xx (xx.x)	xx (xx.x)
Salivary hypersecretion	xx (xx.x)	xx (xx.x)	xx (xx.x)	xx (xx.x)	xx (xx.x)	xx (xx.x)	xx (xx.x)	xx (xx.x)	xx (xx.x)
Glossodynia	xx (xx.x)	xx (xx.x)	xx (xx.x)	xx (xx.x)	xx (xx.x)	xx (xx.x)	xx (xx.x)	xx (xx.x)	xx (xx.x)
INFECTIONS AND INFESTATION	xx (xx.x)	xx (xx.x)	xx (xx.x)	xx (xx.x)	xx (xx.x)	xx (xx.x)	xx (xx.x)	xx (xx.x)	xx (xx.x)
Influenza	xx (xx.x)	xx (xx.x)	xx (xx.x)	xx (xx.x)	xx (xx.x)	xx (xx.x)	xx (xx.x)	xx (xx.x)	xx (xx.x)
Gastroenteritis	xx (xx.x)	xx (xx.x)	xx (xx.x)	xx (xx.x)	xx (xx.x)	xx (xx.x)	xx (xx.x)	xx (xx.x)	xx (xx.x)

Code Tip

```
df <- pmap_dfr(<param grid>, <function>)
```

How it Works

- **pmap_dfr():** A function for mapping a function over rows of a parameter grid in a data frame
- **param_grid:** A data frame defining combinations of parameters
- **function:** A function created to generate count

	TRTA	TRTAN	SEVERITY
1	Drug A	1	MILD
2	Drug A	1	MODERATE
3	Drug A	1	SEVERE
4	Drug B	2	MILD
5	Drug B	2	MODERATE
6	Drug B	2	SEVERE
7	Drug C	3	MILD
8	Drug C	3	MODERATE
9	Drug C	3	SEVERE

Parameter Grid



Conclusion

Key Takeaways



Efficiency

Automate repetitive tasks using functional programming and regex.



Reproducibility

Use renv for version control to meet regulatory and audit requirements.



Scalability

Dynamic dataset handling and coalescing techniques reduce manual effort.



Performance

Avoid redundancy and optimize workflows for large-scale clinical data.



Final Thought

Mastering these tips ensures smoother transition from SAS and empowers clinical programmers



Future Scope

- Normalizing these tips and tricks as a **helper functions** in CRAN package which will act as standardized building blocks for clinical programming workflows.





References

- Wickham H, Henry L (2026). purrr: Functional Programming Tools. R package version 1.2.1, <https://purrr.tidyverse.org/>.
- Wickham H, Henry L (2026). renv: Introduction to renv. R package version 1.1.6, [Introduction to renv • renv](#)
- Pharmaverse: <https://pharmaverse.org/>
- R for Clinical Study Reports: <https://r4csr.org/>



Acknowledgement

- Debjit Biswas
- Priya Venkatesan Iyer
- Jeba Kumar
- Karthik Palani
- Divya Kanagaraj
- Alex Jegadeep Raja A

THANK YOU



**The Global Healthcare
Data Science Community**

Contact Channels

📞 UK +44 1843 609600
✉ office@phuse.global
🌐 phuse.global

Social Media

🐦 [@phusetwitta](https://twitter.com/phusetwitta)
📘 [/phusebook](https://facebook.com/phusebook)
▶ [/phusetube](https://youtube.com/phusetube)
🌐 [/company/phuse](https://linkedin.com/company/phuse)