Paper CT09

$_aR_e$ You Ready!! Fun, Fast & Functional Coding Tips

Xeviers Koner, Merck Specialities Pvt. Ltd., Hyderabad, India

ABSTRACT

This paper presents a concise, practice-oriented overview of “fun, fast, and functional” coding techniques in R. The discussion focuses on a set of everyday tasks that frequently appear in data science and statistical programming: piping and function composition, conditional logic, data merging and sorting, string manipulation, handling missing values, numerical stability, and finally the use of playful but pedagogically useful syntax through the “genzplyr” package. Each topic is explored from the perspective of writing code that is not only correct, but also readable, efficient, and enjoyable to work with. By contrasting “old ways” with newer, more idiomatic approaches, the paper illustrates how small syntactic choices can substantially affect maintainability, clarity, and speed of development. The intent is educational; it aims to serve as a narrative companion to the slides, elaborating on the underlying ideas behind each background theme and slide title rather than merely repeating bullet points.

INTRODUCTION

In modern data science workflows, time-to-insight is a critical metric. Analysts and programmers are constantly searching for ways to make their code faster to write, easier to read, and simpler to maintain. The R ecosystem, enriched by the tidyverse and many community-driven innovations, offers an extensive toolbox for achieving these goals. Yet, for many practitioners coming from other environments, such as SAS or base R, newer idioms like the native pipe, functional string manipulation, or more expressive conditional constructs can initially appear unfamiliar or even intimidating.

The presentation “*aRe You Ready!! Fun, Fast & Functional Coding Tips”* is built around the idea that learning these idioms does not have to be dry or overly technical. Instead, the slides use playful language, vibrant backgrounds, and memorable analogies to introduce serious programming concepts. References to “fun”, “fast”, and “functional” are not superficial branding; they signal three pillars of good code: enjoyment while coding, efficiency in computation and development time, and functional correctness combined with clarity. This paper develops those pillars into a coherent narrative, connecting each slide’s background topic: pipes, conditional logic, sorting, string operations, missing data quirks, numerical details, and “Genzplyr” into a single storyline.

The remainder of the paper is organized as follows. The explanation section is divided into thematic subsections corresponding to the major slide backgrounds: (1) piping in R and its evolution, (2) expressive conditional logic (switch() versus ifelse()), (3) sorting and merging data, especially in contrast to SAS practices, (4) string manipulation using sub(), gsub(), and the relationship between glue() and paste(), (5) the idiosyncrasies of missing values in R, (6) numerical details highlighted by the “Great Square Root” slide, and (7) the playful but insightful re-imagining of dplyr through genzplyr. The paper concludes with reflections on coding style, pedagogy, and practical recommendations for everyday work.

EXPLANATION

1. Pipes and the evolution of readable workflows:

The slides first compare the native R pipe |> with the %>% pipe from the magrittr package. Both tools let you write code as a left to right story. The result of one step flows directly into the next step, so a chain of operations is much easier to read than nested function calls. Instead of having to read from the inside out, you can simply follow the pipeline in order and see how the data changes at each stage.

The native pipe |> is built into base R, so it does not need extra packages and is a good default for simple scripts. %>% is still very useful because it integrates tightly with tidyverse tools and provides extras like placeholders. The key idea in the slide is to think of data processing as a pipeline. Data enters at the start, passes through several clear steps, and exits ready for analysis or plotting. This mindset helps you write code that is both readable and easy to maintain.

2. Conditional logic: switch() vs ifelse():

Another slide contrasts switch() with ifelse(). Many R users start with ifelse() because it is a vectorized version of if and else. It works very well when there is only one condition and two outcomes. Problems appear when you chain several ifelse() calls to handle many cases. The code becomes hard to read and easy to break.

switch() is cleaner when you must choose between several named options based on one value, such as a method type or label. It lists each possible case in a compact way, making it obvious which options exist and what each one does. Using switch() in these situations reduces the risk of missing a case or creating overlapping conditions. The slide encourages you to pick the control structure that matches the problem, so that your logic is both clear and reliable.

3. Sorting and merging: from SAS PROC SORT to R's philosophy:

One slide compares the SAS habit of running PROC SORT before merges with the R philosophy that sorting is usually not required before joining tables. In SAS, merges often assume that data is sorted by the key variables. This leads to extra sorting steps which add time and complexity and can cause mistakes if sorting is not done correctly.

In R, especially in the tidyverse, joins are normally done with functions like dplyr::left_join() or base R merge(). These functions match rows based on key columns, not on row order. The message "No Sorting required in R" means that keys are more important than physical order. Data frames are treated as sets of observations identified by variables. This makes joins more declarative: you tell R which columns define the relationship, and R figures out the matching.

Skipping unnecessary sorts can speed up code and reduce memory use, especially for large clinical or real world data. It also matches how relational databases work, where joins are defined using keys and indexes instead of pre-sorted tables. For people moving from SAS to R, understanding this change in thinking opens the door to simpler and more expressive data wrangling.

4. String manipulation: glue(), paste(), sub(), and gsub():

Several slides cover string handling and use playful wording to make the topic engaging. paste() and paste0() are base R functions that join character vectors, with options for separators and how to treat missing values. They are flexible, but building complex sentences with many variables can become long and hard to read.

The glue() function offers string interpolation. You write a template with curly brace placeholders, and glue() fills in those placeholders using values from the current environment. This makes the code look very similar to the final text, which improves readability and reduces formatting mistakes.

Another slide explains sub() and gsub(). Both perform pattern-based replacement, often using regular expressions. sub() replaces only the first match of the pattern in each string, which is helpful when you know there is just one place to change, for example a prefix at the start. gsub() replaces every match, so it is used for tasks like removing all instances of a symbol or fixing a recurring typo. Choosing between these two is important both for correctness and for performance. Using gsub() when sub() is enough does extra work, while using sub() when you really need gsub() leaves some unwanted text unchanged.

5. The "Moody NA": understanding missingness in R:

The slide titled "Moody NA" focuses on how missing values behave in R. Many operations that include NA return NA. For example, arithmetic with NA usually gives NA, and logical comparisons with NA often return NA instead of TRUE or FALSE. Functions like mean() and sum() will also output NA if there are missing values, unless you explicitly ask them to remove NAs using na.rm = TRUE.

This behaviour is designed to force analysts to think carefully about missing data. In healthcare and life science work, a missing value can have different meanings, such as a test not performed, or a result not recorded. Automatically ignoring missing data can hide important information or introduce bias. The "Moody NA" slide reminds

programmers to be deliberate: decide when to remove missing values, when to flag them, and when to model them. Writing code that checks for NAs early and states clearly how they are handled leads to more trustworthy results.

6. R 4.5.1: “Great Square Root”:

The slide titled "R 4.5.1: Great Square Root" points to the current R release version used in the talk. The presenter notes that only functions relevant to everyday data science and statistical programming have been highlighted. Although the slide is brief, it reminds users that R is still evolving. New versions can change or improve numerical routines, such as the way square roots and other mathematical functions are calculated, especially for tricky input values.

Understanding which R version, you use and what changed in that version helps you trust your results. This is important when you work on sensitive applications like clinical trials or safety analyses, where small numerical differences may matter. The slide encourages programmers to stay aware of the software environment they rely on, instead of treating it as a black box.

7. Genzplyr: a playful re-imagining of dplyr:

The final topic is genzplyr, a package presented "only for fun". It introduces alternative names for familiar dplyr verbs using modern internet slang. For example, filter() becomes yeet(), select() becomes vibe_check(), mutate() becomes glow_up(), and joins are described using terms like "squads" and "mutuals". The idea is playful, but it serves a teaching purpose.

By mapping data operations to slang that some learners find intuitive, genzplyr helps them build mental pictures of what each verb does. For example, "squad_up()" for group_by() suggests that rows are grouped into squads. Once the concept is clear, users can move back to standard names with more confidence. The slide also shows that programming can be creative and enjoyable, while reminding us that production code should still use widely known verbs so that all team members can understand and maintain it.

CONCLUSION

The slide deck and this explanation show that writing R code that is fun, fast, and functional is realistic. Each topic pipes, conditional logic, sorting and merging, string handling, missing values, numerical details, and genzplyr highlights the same idea. Older styles tend to be longer and harder to read, while newer idioms aim for clarity, safety, and speed of development.

Using pipes helps you describe data workflows as clear sequences of steps. Choosing wisely between switch() and ifelse() leads to cleaner decision logic. Knowing that joins in R rely on keys instead of sorted rows simplifies merging and reduces unnecessary work. Strong skills with glue(), paste(), sub(), and gsub() give you powerful control over text. Treating NA as something that must be handled intentionally avoids hidden problems. Paying attention to R versions and numerical behaviour keeps your computations trustworthy. Finally, playful tools like genzplyr show that teaching and learning code can be enjoyable and memorable.

For everyday work, two lessons stand out. First, invest time in learning the idiomatic tools of R, because they will improve both speed and code quality. Second, use clear stories, metaphors, and good visual design in your explanations and documentation. When coding feels like an engaging craft rather than a tedious task, teams are more willing to share knowledge, review each other’s work, and keep improving their practices. That mindset is what it means to be ready for fun, fast, and functional coding.

REFERENCES

1. https://tidyverse.org/blog/2023/04/base-vs-magrittr-pipe/
2. https://www.geeksforgeeks.org/cpp/switch-vs-else/
3. https://stackoverflow.com/questions/53049986/r-functions-glue-vs-paste
4. https://communities.sas.com/t5/SAS-Programming/Is-it-a-must-to-do-proc-sort-right-before-merging-dataset-using/td-p/709497

5. https://stackoverflow.com/questions/44045220/data-need-to-be-sorted-before-merging-in-r
6. https://github.com/hadley/genzplyr/?tab=readme-ov-file

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Author Name: Xeviers Koner

Company: Merck Specialities Pvt. Ltd.

Email: Xeviers.koner@merckgroup.com