

Metaprogramming in R

Mar 2025 | PHUSE US Connect 2025

Sudhir Kedare

Raj Chennamaneni

Integrated Data Analytics and Statistical Programming



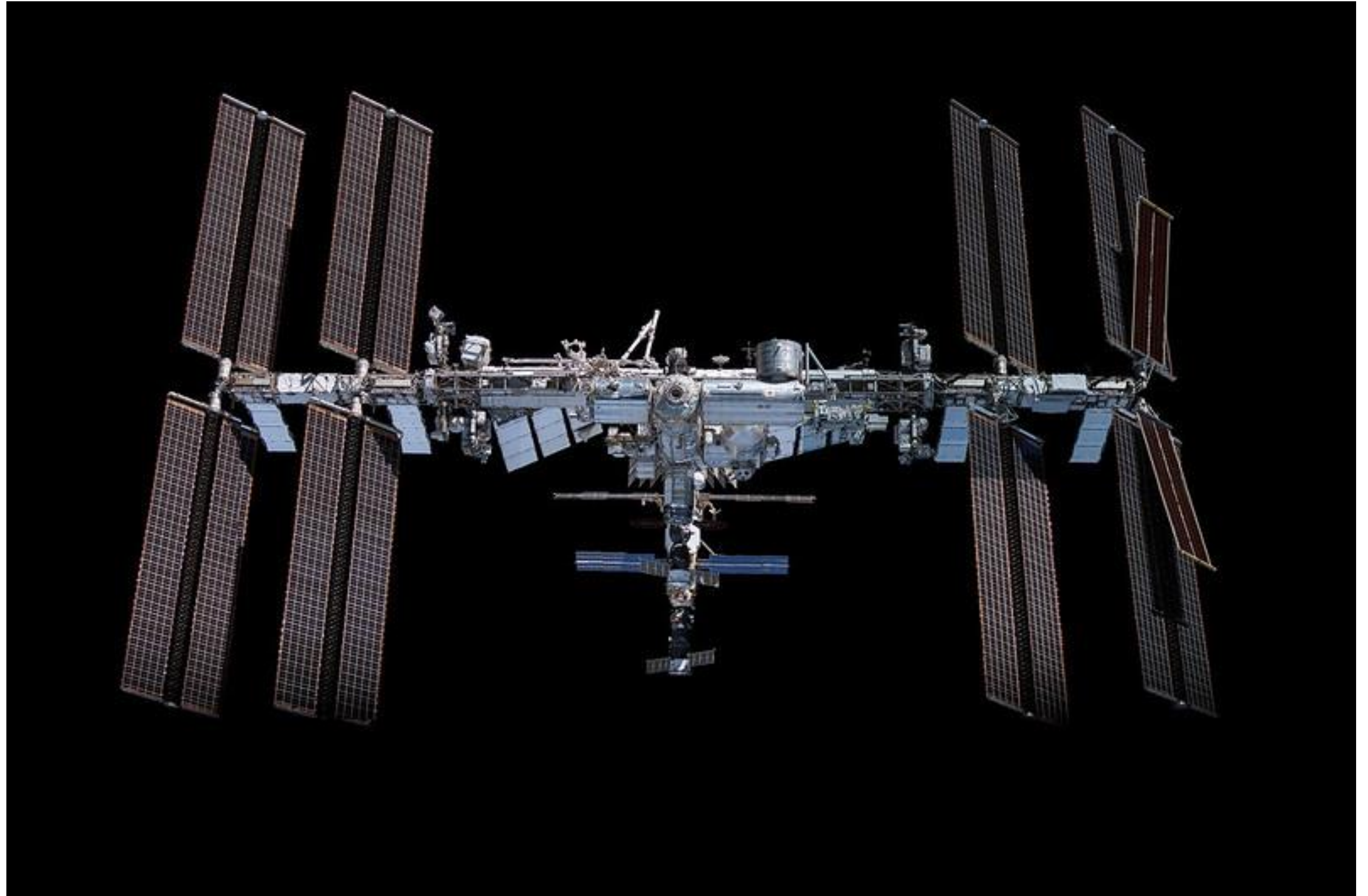
Agenda

- Challenges & Opportunity
- Defusion/Quotation
- Environment And Data Variables
- Unquotation
- Map Function
- Practical Example



Challenges and Opportunity

- Why
- What
- How



Defusion/Quotation

- Standard Code

```
1 + 1  
#> [1] 2
```

- Code Capture

```
expr(1 + 1)  
#> 1 + 1
```

- Captured code is called an expression



Defusion/Quotation

- `expr()` – Lets you capture the code

```
custom_function <- function(x) {  
  expr(x)  
}
```

```
custom_function(a + b + c)  
#> x
```

- `enexpr()` – Lets you capture the code passed to a function – ‘enrich’

```
custom_function <- function(x) {  
  enexpr(x)  
}
```

```
custom_function(a + b + c)  
#> a + b + c
```



Code is Data

- Big Idea: Code can be inspected and modified.

```
func <- expr(f(x = 100, y = 'b'))
```

```
#Add a new argument  
func$z <- 50
```

```
func  
#> f(x = 100, y = 'b', z = 50)
```

- Code can generate code

```
call2("+", 1, call2("*", 2, 3))  
#> 1 + 2 * 3
```



Environment And Data Variables

- Global Environment / User Defined Environment

```
x <- 10  
x  
#> [1] 10
```

- Variable Collision

```
# Function defines 'x' and uses it  
funct <- function() {  
  x <- 30  
  print(paste0("x = ", x))  
}
```

```
funct()  
#> [1] "x = 30"
```

```
X  
#> 10
```



Environment And Data Variables

•Data Masking

```
salary <- 50000
```

```
df <- tibble(  
  age = c(25, 30, 35, 40),  
  salary = c(50000, 60000, 70000, 80000)  
)
```

```
# Use data masking to refer to column names directly  
df %>%  
  mutate(salary_increase = salary * 1.1)
```



Unquotation

- Unquotation - process of evaluating a previously captured expression in R, reverse of Quotation.
- Defuse And Inject Separately

```
my_summarise <- function(data, arg) {  
  arg <- enquos(arg)  
  data |> dplyr::summarise(mean = mean(!!arg, na.rm = TRUE))  
}
```

- Defuse And Inject in One Step

```
my_summarise <- function(data, arg) {  
  data |> dplyr::summarise(mean = mean({{ arg }}, na.rm = TRUE))  
}
```



MAP FUNCTION

- Dynamic alternative to traditional way to read data using loops
- No explicit loops, improved readability and efficient
- Cleaner and Concise

Example Using Loops:

```
import_csv <- function(file) {  
  data <- read.csv(file, header = TRUE, stringsAsFactors = FALSE)  
  return(data)  
}  
all_data <- list()  
for (file in file_list) {  
  all_data[[file]] <- import_csv(file)  
}
```

Example:

```
import_csv <- map_df(file_list, extract_data_from_file)
```



Practical Example - clinical trial analysis

- **Problem:**

- Descriptive statistics (e.g., mean, median, standard deviation) for different variables across treatment groups.

- **Challenge:**

- Avoid repetitive code and less flexible analysis.

- **Efficient Solution:**

- **Quotation:** Capture column names (like `group_var` and `sum_var`) without evaluating them.
- **Unquotation:** Dynamically evaluate those column names at the appropriate time using `!!`.



Practical Example - clinical trial analysis

Function to generate summary statistics for a given variable

```
Custom_summary <- function(df, group_var, sum_var) {  
  df %>%  
    group_by(!!group_var) %>%  
    summarize(  
      n      = sum(!is.na(!!sum_var)),  
      mean   = mean(!!sum_var, na.rm = TRUE),  
      .groups = "drop"  
    )  
}
```

Combined function to process multiple variables dynamically

```
adsl_summary_combined <- function(data, group_var) {  
  purrr::map_df(  
    .x = dplyr::vars(AGE, BMI), # List of variables to summarize  
    .f = \(x) custom_summary(df = data, group_var = group_var, sum_var = !!x)  
  )  
}
```

Function Call:

```
adsl_summary <- adsl_summary_combined(df, group_var = TRTA)
```



Conclusion

- **Dynamic code generation**
 - Enabling more flexible and adaptable programs.
- **Quotation and Unquotation**
 - Powerful ways to handle expressions, allowing you to work with variables and column names dynamically
- **Efficiency and Flexibility**
 - Reduces redundancy and enhances code scalability
- **Automation in Data Analysis**
 - Facilitates the automation of repetitive tasks
- **Future learnings**
 - Translation - Substitute(), modify expression etc.



Contact us:

Sudhir Kedare

- sudhir.kedare@jazzpharma.com
- Jazz Pharmaceuticals
- Palo Alto, CA

Raj Chennamaneni

- Rajprakash.Chennamaneni@jazzpharma.com
- Jazz Pharmaceuticals
- Wilmington, NC



Thank you!

