sas2rnotes

Ram

2026-06-12

Table of contents

About

Welcome to R for SAS Programmers

Course Overview

This comprehensive course is designed specifically for **SAS programmers** who want to learn R programming. We'll leverage your existing knowledge of data manipulation, statistical analysis, and programming concepts to help you become proficient in R.

Learning Objectives

By the end of this course, you will be able to:

- **Understand R fundamentals**: Master R syntax, data types, and programming concepts
- **Data manipulation**: Perform complex data transformations using `dplyr` and `base R`
- **Statistical analysis**: Apply statistical methods and create models in R
- **Data visualization**: Create compelling visualizations using `ggplot2`
- **Reporting**: Generate dynamic reports with R Markdown and Quarto
- **Bridge knowledge**: Map your SAS skills to equivalent R functions and workflows
- **Best practices**: Write clean, efficient, and reproducible R code

Course Structure

Module 1: R Fundamentals for SAS Users

- **Getting Started**
  - R and RStudio installation and setup
  - Understanding the R environment vs SAS environment
  - Package management
- **Basic R Syntax**
  - R syntax compared to SAS syntax
  - Variables and assignment operators
  - Data types and structures
  - Functions and help system

Module 2: Data Import and Export

- CSV, Excel and text files (`readr`, `readxl` packages)
- SAS dataset import (`haven` package) and export xpt files
- other formats (JSON, XML)

Module 3: Data Manipulation - dplyr

- **Core dplyr Functions**
  - `select()` vs `KEEP/DROP` statements
  - `filter()` vs `WHERE` clause
  - `mutate()` vs `assignment` statements
  - `summarise()` vs `PROC MEANS`
  - `group_by()` vs `BY` statement
- **Advanced Data Manipulation**
  - Joins equivalent to `PROC SQL` joins
  - Reshaping data (`tidyr` vs `PROC TRANSPOSE`)
  - String manipulation (`stringr` vs SAS string functions)

Module 4: Creation of ADSL dataset

- **ADSL and ADVS Dataset Creation**
  - Creating analysis variables
  - Handling missing data and derivations
  - creating ADSL xpt dataset

Module 5: Statistical Analysis

- **Descriptive Statistics**
  - Summary statistics (equivalent to `PROC UNIVARIATE`)
  - Frequency tables (equivalent to `PROC FREQ`)
  - Cross-tabulations and chi-square tests
  - creation of tables like DM and AE outputs
- **Statistical Modeling basics**
  - Linear regression (equivalent to `PROC REG`)
  - Logistic regression (equivalent to `PROC LOGISTIC`)

  - ANOVA (equivalent to `PROC ANOVA`)
  - Mixed models and advanced techniques

Module 6: Package Develpment and testing

- **Creating R Packages**
  - Package structure and essential files
  - Documenting functions with `roxygen2`
  - Building and installing packages
- **Testing with testthat**
  - Writing unit tests for R functions
  - Running tests and interpreting results

Module 7: Data Visualization

- **Base R Graphics**
  - Basic plots and customization
  - Comparison with SAS/GRAPH
- **ggplot2 - Grammar of Graphics**
  - Understanding the layered approach
  - Creating publication-ready plots
  - Advanced visualization techniques

Module 8: Reporting and Documentation

- **R Markdown and Quarto**
  - Creating dynamic reports (equivalent to ODS output)
  - Integrating code, results, and narrative
  - Output formats: HTML, PDF, Word
- **Reproducible Research**
  - Project organization
  - Version control with Git
  - Best practices for code documentation

SAS to R Translation Guide

SAS Concept	R Equivalent	Package
DATA step	`dplyr::mutate()`	dplyr
PROC SQL	`dplyr` verbs	dplyr
PROC MEANS	`dplyr::summarise()`	dplyr
PROC FREQ	`table()`, `xtabs()`	base R
PROC REG	`lm()`	base R
PROC LOGISTIC	`glm()`	base R
PROC TRANSPOSE	`tidyr::pivot_*()`	tidyr
ODS OUTPUT	R Markdown/Quarto	rmarkdown/quarto

Prerequisites

- **SAS Experience**: Familiarity with SAS programming, data steps, and procedures
- **Statistical Knowledge**: Basic understanding of statistical concepts
- **Programming Basics**: Understanding of programming logic and data structures

Course Format

- **Interactive Learning**: Hands-on exercises with real datasets
- **Comparative Examples**: Side-by-side SAS and R code comparisons
- **Practical Projects**: Real-world scenarios mimicking typical SAS workflows
- **Reference Materials**: Quick reference guides and cheat sheets

Getting Started

Required Software

1. **R** (version 4.3+): Download from CRAN
2. **RStudio**: Download from Posit
3. **Essential Packages**: We'll install these as needed

```
install.packages(c("tidyverse", "haven", "readxl", "rmarkdown"))
```

Tip: If you don't have sample SDTM/ADaM data yet, the chapters generate **small synthetic data** as a fallback so everything runs end-to-end. ## contact For questions or feedback, reach out to **r2sas2025@gmail.com**

Part I

datatype & structure

1 datatype and structure

1.1 R as a calculator

R can be used as a calculator following the order of operations using the basic arithmetic operators, although, the arithmetic equal sign (=) is the equivalent of ==.

```
# simple calculations
3*2
```

```
[1] 6
```

```
(59 + 73 + 2) / 3
```

```
[1] 44.66667
```

```
# complex calculations
pi/8
```

```
[1] 0.3926991
```

1.2 Storing outputs

An object can be created to assign the value of your operation to a specific variable name, which can be reused later in the R session. Using the `object_name <- value` naming convention, you can assign (<-) the value (`(59 + 73 + 2) / 3`) to an `object_name` simple_cal to look like `simple_cal <- (59 + 73 + 2) / 3` to store the evaluation of that calculation.

```
x <- 1:10

y <- 2*x

simple_cal <- (59 + 73 + 2) / 3
```

1.3 Loading data into R

Depending on the formats for the files containing your data, we can use different base R functions to read and load data into memory

R has two native data formats, **Rdata** (sometimes call Rda) and **RDS**.

Rdata can be selected R objects or a workspace, and **RDS** are single R object. R has base functions available to read the two native data formats, and some delimited files.

```
# saving rdata
save(x, file = "data/intro_1.RData")
# Save multiple objects
save(x, y, file = "data/intro_2.RData")

# Saving the entire workspake
save.image(file="data/intro_program.RData")

# We can follow the syntax for saving single Rdata object to save Rds files
# saveRDS(object, file = "my_data.rds")


# loading Rdata or Rda files
load(file = "data/intro_program.RData")

# loading RDS
# We can follow the syntax for read Rdata object to sread Rds files using the readRDS()


# Comma delimited
adsl_CSV <- read.csv("data/adsl.csv", header = TRUE)

# Save CSV
adsl_csv_save <- write.csv(adsl_CSV, "data/save_data/adsl.csv", row.names=TRUE)


adsl_TAB_save <- write.table(
  adsl_CSV,
  "data/save_data/adsl.txt",
  append = FALSE,
  sep = "\t",
  dec = ".",
  row.names = TRUE,
```

```
  col.names = TRUE
)

# Tab-delimited
adsl_TAB <- read.table("data/save_data/adsl.txt", header = TRUE, sep = "\t")
```

1.4 R Packages

R packages are a collection of reusable functions, compiled codes, documentation, sample data and tests. Some formats of data require the use of an R package in order to load that data into memory. Share-able R packages are typically stored in a repository such as the Comprehensive R Archive Network (CRAN), Bioconductor, and GitHub.

1.5 Installing R packages

```
# From CRAN
#install.packages("insert_package_name")
# {haven} is used to import or export foreign statistical format files (SPSS, Stata, SAS)
install.packages("haven")

# {readxl}
install.packages("readxl")


# From Github
remotes::install_github("pharmaverse/admiral", ref = "devel")
```

1.6 Using R packages, functions from an R package, and accessing help pages

Since an R packages are a collection of functions, you can choose to load the entire package within R memory or just the needed function from that package. Usually, the order you choose to load your package does not make a difference, unless you are loading two or more packages that has functions with the same name. If you are loading two or more packages with a common function name, then the package loaded last will hide that function in the earlier packages, so in that case it is important to note the order you choose to load the packages.

```r
# read file using library call
library(haven)
adsl_sas <- read_sas("data/adsl.sas7bdat")


# Reading Excel xls|xlsx files
# read_excel reads both xls|xlsx files but read_xls and read_xlsx can also be used to read re

# if NA are represented by another something other than blank then you can specified the NA v
# within the read_excel() function
```

1.7 Data types

R has different types of **Datatype**
* Integer * numeric * Character * Logical * complex * raw

But we will focus on the top 4.

```r
set.seed(1234)

type_int <- (1:5)
type_num <- rnorm(5)
type_char <- "USUBJID"
type_logl_1 <- TRUE
type_logl_2 <- FALSE


class(type_int)
```

```
[1] "integer"
```

```r
class(type_num)
```

```
[1] "numeric"
```

```r
class(type_logl_1)
```

```
[1] "logical"
```

```r
class(type_logl_2)
```

```
[1] "logical"
```

```r
class(type_char)
```

```
[1] "character"
```

1.8 Date formats

There are base R functions that can be used to format a date object similar to the Date9 formatted date variable from SAS. In addition, there are R packages available, such as {lubridate}, for more complex date/date time formatted objects.

```r
# using adsl_sas RFSTDTC
class(adsl_sas$RFSTDTC)

# Convert the date from that adsl_sas into a date variable
adsl_date <- as.Date(adsl_sas$RFSTDTC, "%m/%d/%Y")
class(adsl_date)


library(lubridate)
date9 <- as_date(18757)
adsl_sas$RFSTDTC <- ymd(adsl_sas$RFSTDTC)
class(adsl_sas$RFSTDTC)
```

1.9 Structures

Data structures are dimensional ways of organizing the data. There are different data structures in R, let's focus on **vectors** and **dataframe**

Vectors are 1 dimensional collection of data that can contain one or more element of the same data type

```r
vect_1 <- 2
vect_2 <- c(2, "USUBJID")

class(vect_1)
class(vect_2)

# Saving vectors from a dataset to a specific variable
usubjid <- adsl_sas$USUBJID
subjid <- adsl_sas[, 3]
```

Dataframe is similar to SAS data sets and are 2 dimensional collection of vectors. Dataframe can store vectors of different types but must be of the same length

```r
df <- data.frame(
  age = c(65, 20, 37,19,45),
  seq = (1:5),
  type_logl = c(TRUE,FALSE, TRUE, TRUE, FALSE),
  usubjid = c("001-940-9785","002-950-9726","003-940-9767","004-940-9795","005-940-9734")
)

# str() provides the data structure for each object in the dataframe
str(df)
```

```
'data.frame':   5 obs. of  4 variables:
 $ age      : num  65 20 37 19 45
 $ seq      : int  1 2 3 4 5
 $ type_logl: logi  TRUE FALSE TRUE TRUE FALSE
 $ usubjid  : chr  "001-940-9785" "002-950-9726" "003-940-9767" "004-940-
9795" ...
```

```r
# In addition to the data structure per variable, also get some descriptive statistics
summary(df)
```

```
      age            seq    type_logl         usubjid
 Min.   :19.0   Min.   :1   Mode :logical   Length:5
 1st Qu.:20.0   1st Qu.:2   FALSE:2         Class :character
 Median :37.0   Median :3   TRUE :3         Mode  :character
 Mean   :37.2   Mean   :3
 3rd Qu.:45.0   3rd Qu.:4
 Max.   :65.0   Max.   :5
```

2 datatype and structure exercise

3 Exercise 1

Install and load the following packages

{tidyverse} {admiral} {dplyr} {tidyr} {admiral.test}

```
#installing the packages
install.packages(c("tidyverse", "admiral", "dplyr", "tidyr"))

library(tidyverse)
library(admiral)
library(admiral.test)
library(dplyr)
library(tidyr)
```

4 Exercise 2

Import `adsl.sas7bdat` as adsl

```
library(haven)
adsl <-  read_sas("data/adsl.sas7bdat")
```

Part II

data manipulation

5 Introduction

This is a book created from markdown and executable code.

See Knuth (1984) for additional discussion of literate programming.

```
1 + 1
```

```
[1] 2
```

6 Summary

In summary, this book has no content whatsoever.

```
1 + 1
```

```
[1] 2
```

References

Knuth, Donald E. 1984. "Literate Programming." *Comput. J.* 27 (2): 97–111. https://doi.org/10.1093/comjnl/27.2.97.