Writing Your First R Program, Just the Basics
Sunil Gupta, Founder of SASSavvy.com
Sponsored by ACL Digital Life Science

Are you ready to install R and start writing your first R program? Are you ready to start using one of the most powerful and common R package, Tidyverse? This paper assumes you have no prior knowledge of R programming.

Explore and test drive basic steps to install R and Tidyverse. This paper shows and tells basic R syntax to get data into R, combine data frames, query data, create new variables and finally summarize data. These key tasks are required for all projects. They show how R programming is direct with function-based tasks that are executed in logic sequence to get data into R, perform data management and cleaning and then to summarize results.

The upcoming webinar series 'R Programming for Biometrics Professionals' is a logically prepared step-by-step crash course that will immerse you in the world of R programming. This upcoming series will be sponsored by ACL Digital Life Science.

Outline

- R Programming is Not for Everyone
- TIDYVERSE Package
- Installing R Packages and Libraries
- Starting to Program in R
- Changing the Default R Working Folder
- Submitting R Commands
- R Data Object Process Flow, Structure, Rules and Scope
- R Syntax – Basics
- Common R Data Frame Operations
- R Examples
- Data Access Functions to Create Data Frames
- R Assignments, Keep, Drop, Subset and Sorting Operations
- SQL Example
- Appending and Merging R Data Frames
- Summarize R Data Frames
- Summary
- Your First R Script

R Programming Is Not for Everyone

While R is getting popular with more exposure and interest, R may not be for everyone because R is a very technical. R is mostly a 'short-cut language' since one-line R commands are common and concise. R is similar to SAS's advanced macros which contain many parameters with options including default settings. R syntax is not intuitive since programmers need to remember keywords and syntax.

The primary objective of these R-Guru webinars is to cut through the vast knowledge and focus on key concepts to uniquely learn R with simple and productive examples and visual illustrations. This paper introduces you to the different flavors of R syntax so you can appreciate the full power of R.

- sort(unique(csv_file[,56]))
- In CSV file, Unique and then Sort by Column # 56

- adsl$sexn[adsl$sex=="m"] <- 1 adsl$sexn[adsl$sex=="f"] <- 2
- In ADSL, assign numeric values to SEXN variable based on SEX values

- dm_ex <- left_join(dm, ex, by = "usubjid")
- Create DM_EX by Left Joining DM and EX by USUBJID

TIDYVERSE Package

Programming in R world is similar to navigating in space. For many programmers, creating and working with objects in memory represent the new frontier in space. One of the first R packages all programmers should learn is the Tidyverse package. The Tidyverse package is very popular because it is one package that contains libraries with many utilities. Some of the important features include: Import Data, Data Manipulation, Program Language, Visualize, Statistical Models, Publish and Web Applications. More importantly, Tidyverse is a validated package which means that it is universally accepted and produces reliable results. Programmers do not have to worry about the accuracy of results when using Tidyverse packages.

Installing R Packages and Libraries

The four steps to install R are very simple and straight forward.

- Step 1. install R windows (32 or 64 bit)
- # https://cran.rstudio.com

- Step 2. download R studio
- # https://www.rstudio.com/products/rstudio/download

- Step 3. Double click on R icon

- Step 4. Please select a CRAN mirror for use in this session
- # Select US option for internal setting.

R runs on R packages. R programmers need to identify and run R packages to run R functions.

- install.packages('tidyverse')
- library(tidyverse)
- library(dplyr)
- library(sqldf)
- library() # display loaded libraries

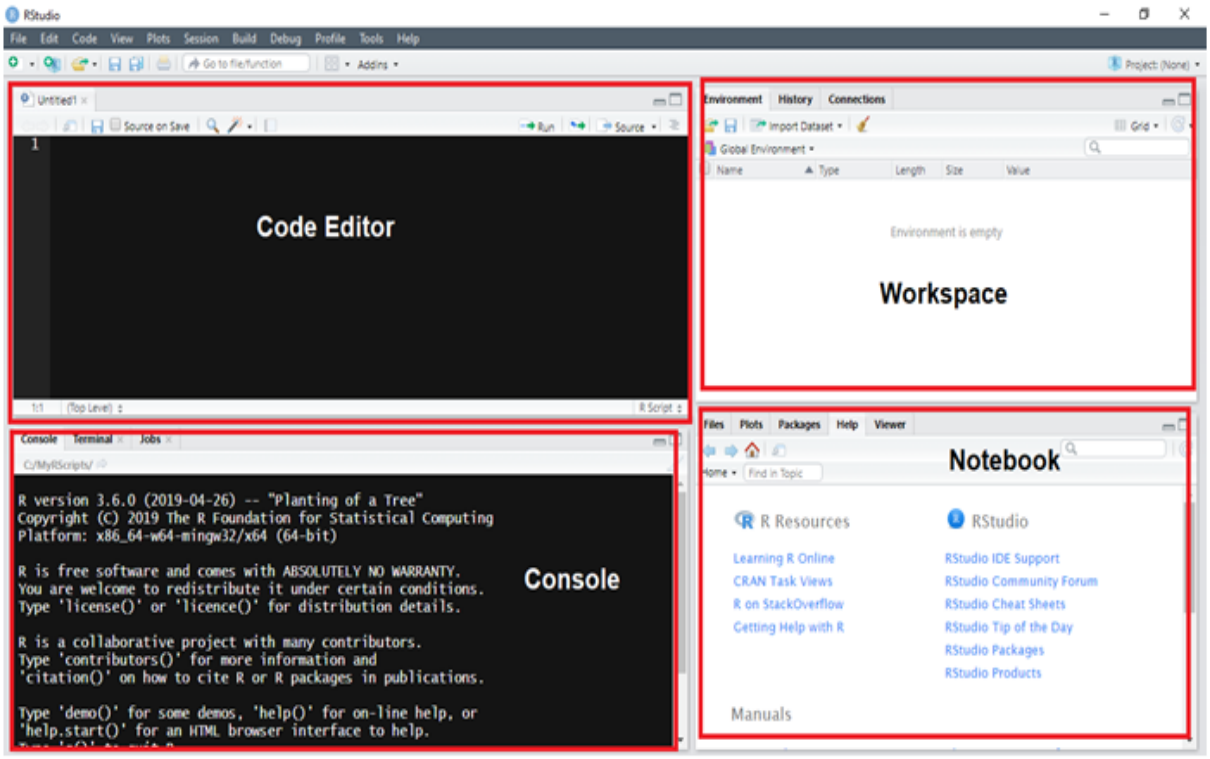
Below is a list of common R packages that perform essential data access, data management and reporting.
An overview of R packages is a topic for future webinars.

R Package	Features / Libraries / Functions
Base	c(), data.frame()
tidyverse	Common data management package for reshaping, transforming and plotting data tidyr – useful set of functions such as .data[[]] dplyr – useful for data management stringr – string functions purr – functional programming tibble – modern and effective table system ggplot2 - Popular graphic package readr – read csv files
Plyr	ddplyr() – useful to create intermediate steps

Sqldf	sql syntax
flextable	Proc Report and ODS type control for report formatting
Sassy	logr(), fmtr(), datastep(), reporter(), libr()
officer	Create rtf and powerpoint files
magtrittr	Allow %>% for piping operations from one function to another function

Starting to Program in R

The R interface shows four panels: Code Editor, Workspace, Console and Notebook. Each panel has a purpose so that the programmer has full control of his working environment.



Just like SAS Enterprise Guide, some training is required to start using the R interface.

Code Editor window
- Area for code development
- Code will not be evaluated or executed until you hit the 'Run' button

Workspace window
- Environment / History window
- List objects that exist in the working space
- View comment history (like SAS log)

Console window
- Here code from the script source is evaluated by R
- Also allows you to perform quick calculations that you don't need to save

Notebook window
- Files/Plots/Packages/Help
- Here you can see file folders, plot output, browse and install available packages, access R help

Changing the Default R Working Folder

One of the first tasks programmers do is reset the default R working folder (C:/Users/XXX/Documents) to paths that contain data or saved files. This is important since several R functions will read and write to the default working folder

Working Folder

- Commands: setwd("C:/study/analysis/output") # create working folder in advance
- getwd() # confirm new working folder, default C:/Users/Sunil/Documents
- Menu: File > Change dir > browse to C:/study/analysis/output
- Important for using saveRDS and readRDS functions for storing & reading files

Write, Save and Read Functions

- write.csv(dm, "C:/study/analysis/output/mydm.csv", row.names = FALSE)
- saveRDS(asl2, file = "asl2.RDS") # save asl2 as permanent data frame
- myasl <- readRDS("asl2.RDS") # read permanent asl2 data frame

Submitting R Commands

Submitting R commands is a simple process. Once an R script file is opened, you just select one or more lines and enter Control-R to execute them. The alternative is to select Edit from the menu and then Run Line or Selection or Run All. The cursor can be placed anywhere on the R command to execute it. Note that if partial R command is submitted, then R will insert '+' in the log to recognize partial R commands and will then expect the remaining R command. See the log window for results after each R comment.

The R script is essentially a text file with an .R extension. The .R extension enables R to recognize it. A tip is to copy R comments from a text file into an R script file. Best practices are to create an R setup file to install packages and load libraries used on regular tasks similar to SASAUTOS.

File > Open R Script > Just the R Basics.R

To Run R commands:

Place cursor on R command line in R script
Menu: Edit > a) Run line or selection (Control-R to run)
b) Run all (To run whole R script)

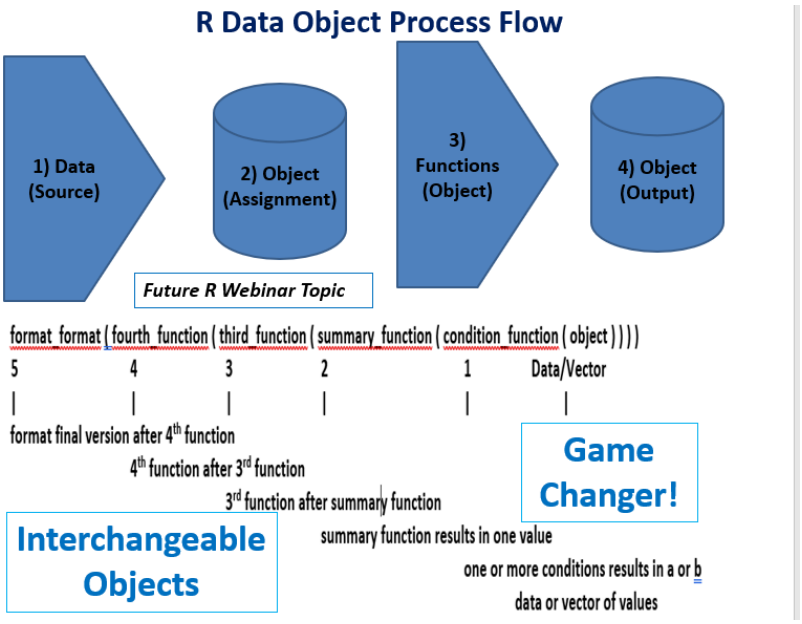
R Data Object Process Flow, Structure, Rules and Scope

R processes are basically a collection of R functions that create and read R objects until the final object is created. The output of the first R function is the input to the second R function. Output of the second R function is input to the third R function. Assure open and closed brackets: [], (), ''. Close bracket defines end of R-command. %>% saves time from creating intermediate objects.

In the diagram below, this continues to the fifth outer R function. This means that R functions are used to access, manage, transform and then summarize data. In addition, because of this unique software architecture, R functions can have multiple nested levels.

- One 'Function' Away
- my_data <- cbind(usubjid, age, date, in_study)
- New_R_Object <- R_Function(R_Objects)

- R_Function(Parameter 1, Parameter 2, etc.)
- R Keyword(Existing R_Objects)



R programming is unique since R objects store data which must be valid data types so that R functions can be applied to create new R objects. So, in a sense, given the software's unique architecture the prior level of data within an object must be fully validated in order to advance to the next level of data contained within the subsequent object. With symbols, direct references can be made to variables as independent objects. This makes R programming more flexible. Along with R base, Tidyverse and ggplots are popular R packages. R data object process flow and structure, rules and scope are topics for future webinars.

- <-
- # left assignment, best practice to make assignments, similar to = (equal) or copy object

- 'Object 2' <- 'R Function Creates Object 1'
- 'R Function Reads Object 1 to Create Object 2'

- One Line R-Commands do the job
- t <- c("Sun","Mon","Tue","Wed","Thurs","Fri","Sat")

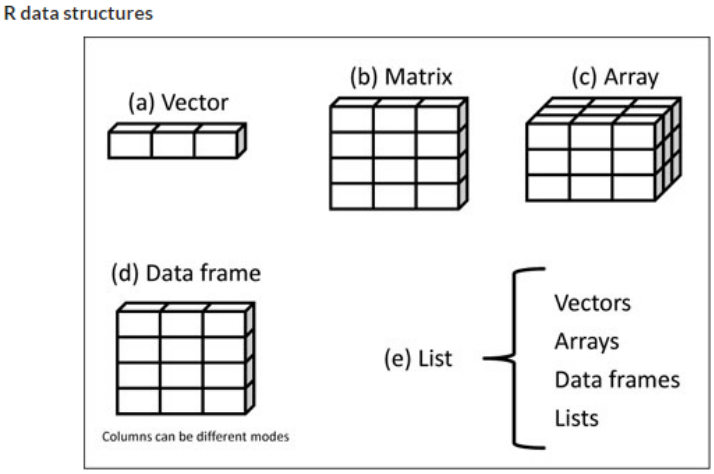
- Compared to several lines in SAS Procedures
- proc print data=dm; var x y; run;

- iris %>% head() # x %>% f(y) is similar to f(x, y)
- %>% # enables running R functions in sequence to save you time from creating intermediate objects, 'and then or pipe to continue statement', best to unit test each function before sequencing functions

- objects() # to display all objects in workspace (data frames, vectors, etc.)

R Syntax – Basics

R is different from SAS in that there are five essential R data structures – **vectors, matrix, array, data frames and lists**. For SAS programmers, I think the vectors and data frames are the most important. As a basic concept, vectors are similar to values in a dataset with only one variable. Data frames contain a collection of vectors so contain many variables and records. Most all R processing is performed on data frames. All R data structures and any outputs created from R functions are R objects.



Common R Data Frame Operations

As mentioned earlier, R leverages symbols to perform common tasks. Below is a list of common R data frame operations.

Data Frame Operations/Functions	Symbol / Syntax
Create Data in Objects	c()
Data Frame Variable Reference	$
Index Subset (Variable or Records)	[]
Format	format() to format variables and data types sprintf() to format reporting columns str_glue() to format char, num and dates as string
Combine Data Frames by Rows	rbind() to append data frames
Combine Data Frames by Columns	cbind() to merge vectors
Merge Data Frames	merge(), left_join(), sqldf()
SQL Clauses	select(), mutate() & chg=, ifelse(), filter(), group_by(), arrange()
SQL Operations	ddplyr(), dplyr, sqldf()
Transpose Data	pivot_wider() – tidyverse pivot_longer() - tidyverse Old - melt(), spread()
Data Type Conversion	as.character(), as.date(), as.integer(), as.data.frame(), as.logical(), as.table(), as.array()
Missing Values	is.na(), is.null()
Object Metadata	is.list(), is.matrix(), is.vector(), is.data.frame(), is.array()
Characterize Data / Variable Metadata	View(), attributes(), length(), ncol(), nrow(), names(), str(), typeof(), class(), head(), print(), summary(), table()
View Data Frame	View() – upper case 'V', base R view() – lower case 'v', tidyverse
Loop through List or Variable	for (i in <data_frame>$<variable_name>)

Custom Functions	function((condition) {true} else {false})

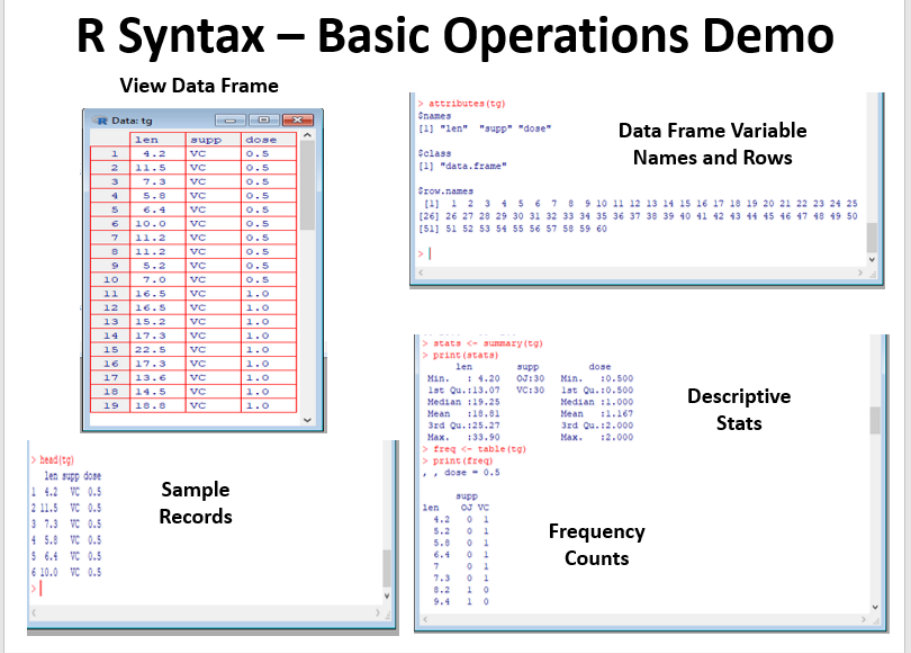
R Examples

Below are useful metadata type R functions to describe data frames. Information from R functions displays data frame variable names, number of records, sample records as well as unique frequency counts and descriptive statistics. The '<=' symbol assigns the tg object the contents of the ToothGrowth data frame. All R functions in the remaining statements process the tg data frame. Any text after the '#' symbol is a comment so ignored by R. These R examples show how R objects are processed by R functions. Simple one function call performs specific tasks. For any of these R functions since the '<=' symbol is not applied, the results are displayed instead of being saved to another R object.

You can run R statements individually from the Edit > Run Line or Selection option or run all R statements in an R script file from the Edit > Run All option. Results will be displayed in the console window and the tg data frame window will open.

> **tg** <- ToothGrowth # save sample data frame to tg data frame
> **View**(tg) # browse tg
> **str**(tg) # display tg attributes and sample records
> **attributes**(tg) # display tg attributes, names (tg) is alternative to display variable names
> **head**(tg) # display tg sample records
> **print**(tg) # display tg all records, similar to proc print
> **summary**(tg) # display stats object of continuous variables
> **table**(tg) # display freq of categorical variables

Below are outputs from the R functions.



Data Access Functions to Create Data Frames

Tidyverse has a rich set of R functions to convert CSV, excel or SAS datasets into R data frames. In addition, permanent R data frames can also be loaded in memory. Below are several useful examples.

- CSV files
  - dm <- read.csv("c:/mydata.csv")
  - dm = read.table("c:/mydata.csv") # data tables are similar to data frames
- Excel files
  - library(readxl) # required for read_excel function
  - l_cancer <- read_excel("C:/data/l_survey.xlsx") # forward slashes
  - View(l_survey) # display object in grid mode containing excel file which you can filter
- Libnames
  - library(haven) # required to read SAS datasets
  - sdtm <- "/study/sdtm" # assign libname to object named sdtm
- SAS Datasets
  - dm <- read_sas(file.path(sdtm, "dm.sas7bdat")) # read_sas from haven package within tidyverse to read DM dataset in sdtm library
  - dm <- read_sas("C:/study/sdtm/dm.sas7bdat") # read DM directly
- R Data Frames
  - load("dm.RData") # loads dm in R
  - attach(dm) # R sets dm as the default dataset
  - detach(dm) # R releases dm dataset

R Assignments, Keep, Drop, Subset and Sorting Operations

Below are examples of R assignment, keep, drop, subset and sorting commands:

- Left Variable Assignment (Default)
- patient <- 1 # assign number patient = 1

- Right Variable Assignment
- 1 -> patient # right assign number patient = 1
- Variable Assignment Options
- assign('ae', 'Headache') # assign char ae = 'Headache'
- ae = "Headache" # assign char ae = 'Headache'

- Keeping Variables
- dm2 = dm1[c("sex", "race")]
- # data dm2; set dm1; keep sex race; run;

- Dropping Variables
- dm2 = subset(dm1, select = -c(sex, race))
- # subset and select functions are similar
- # data dm2; set dm1; drop sex race; run;
- dm2 = dm1[-c(1)] # drop by column index number

- Subsetting Records
- # = = required for comparison
- dm2 <- subset(dm, sex == 'M')
- dm2 <- filter(dm, sex == 'M')

- dm2 <- filter(dm, sex == 'M', age > 21)
- # subset and filter functions are similar
- # data dm2; set dm; where sex = 'M'; run;

- Sorting Records
- dm2 <- dm[order(sex, race),]
- dm2 <- arrange(dm, sex, race)
- # order and arrange functions are similar
- # proc sort data=dm2; by sex race; run;

SQL Example

As in SAS, R's SQL provides powerful query and multitasking functionality. SQL will be a standalone topic covered in a later webinar during this series. Below is an SQL example.

- dm3 = dm1 %>% # create dm3 from dm1 and then or pipe
- mutate(idvarval_ = as.numeric(idvarval)) %>% # create new var
- idvarval_ = numeric(idvarval)
- select(usubjid, idvarval_, qnum, qval) %>% # select four vars
- spread(., qnam, qval) # transpose three vars

Appending and Merging R Data Frames

Appending and merging is very direct with R. Examples are below.

- demo3 <-bind_rows(demo1, demo2)
- # data demo3; set demo1 demo2; run;

- vitals2<-inner_join(demo1, vitals, by="subjid")
- # data vitals2;
- merge demo1 (in=a)
- vitals (in=b);
- by subjid; if a and b;
- run;

Summarizing R Data Frames

Once data is in R data frames, summarizing your categorical or numeric data is very easy with the TABLE() and SUMMARISE() R functions.

- table(cars$Type)
- # proc freq data=cars; tables type; run;

- table(cars$Type, cars$Cylinders)
- # proc freq data=cars; tables cars*Cylinders; run;

- ASL_mean <- ASL %>%
- group_by(ARMCD) %>%
- summarise(avg_age = mean(AGE), avg_bmi = mean(BMI))

- # proc means data=ASL;
- by ARMCD;

- var mean bmi;
- run;

Summary

While learning a new language requires some patience, I think the rewards of R programming are well worth it! With the simple steps to install and examples to get started in R in this paper, I hope your journey is clearer. The best method to prevent errors and minimize trouble-shooting is to start with the correct and working R example and then copy and customize it with your data frame name, variables, values and conditions. Test run each R program update to assure correct execution before making more customization. You will soon start to feel more comfortable with R programming!

Your First R Script

```
# Name First R Program
# Purpose is to install and show basic R commands
# Author Sunil Gupta, SASSavvy.com/R-Guru
# Date Oct 25 2021
# Copyright - 2021, This macro has copyright protection by the author.
Application of this program for its intended use only in its original form is authorized by the author. All other
applications are at users risk.
R program is not to be used in commercial software product without the written approval by the author.
Any reference to the R program must include reference to SASSavvy.com/R-Guru.
No support or guarantee is provided with the R program or results.
User takes responsibility for R program and results and is encouraged to test the program.

# 1. install R windows (32 or 64 bit)
# https://cran.rstudio.com

# 2. download R studio
# https://www.rstudio.com/products/rstudio/download

# 3. Double click on R icon

# 4. Please select a CRAN mirror for use in this session
# Select US option for internal setting.

setwd("C:/study/analysis/output")
# change default working folder
getwd()
# confirm working folder

# option b
install.packages('tidyverse')
library(tidyverse) # load popular data management package
library(dplyr) # load common data management and sql package
library(sqldf) # load sql package

library() # display loaded libraries

.libPaths() # display path of each package

lsf.str("package:dplyr") # display dplyr functions
```

```
lsf.str("package:tidyverse") # display tidyverse functions

t <- c("Sun","Mon","Tue","Wed","Thurs","Fri","Sat")
view(t)

x= c(2, 4, 6)
mean(x)
y=c(1, 1, 1)
z = x + y
view(z)

mydata <- data.frame(
  class = c("1st", "2nd", "3rd", "Crew"),
  n = c(325, 285, 706, 885),
  prop = c(14.8, 12.9, 32.1, 40.2)
)
mydata

tg <- ToothGrowth # save sample data frame to tg data frame
View(tg) # browse tg
str(tg)  # display tg attributes and sample records
attributes(tg) # display tg attributes, names (tg) is alternative to display variable names
head(tg) # display tg sample records
print(tg) # display tg all records, similar to proc print
summary(tg) # display stats object of continuous variables
table(tg) # display freq of categorical variables

objects() # to display all objects in workspace (data frames, vectors, etc.)

library(readxl) # read excel files
dm_specs <- read_excel("D:/Sunil/GP/Business/ACL Digital/r programming/DM_Specs.xlsx")
View(dm_specs)

dm <- read.csv("D:/Sunil/GP/Business/ACL Digital/r programming/input.csv")
dm1 <- read.table("D:/Sunil/GP/Business/ACL Digital/r programming/input.csv")
dm
dm1

write.csv(dm, "C:/study/analysis/output/mydm.csv", row.names = FALSE)
# save permanent csv

sdtm <- "D:/Sunil/GP/Business/ACL Digital/r programming" # assign libname to object named sdtm
list.files
dir

library(haven) # read SAS datasets
asl1 <- read_sas(file.path(sdtm, 'adsl.sas7bdat'))  #  read_sas from haven package within tidyverse to read DM dataset in
sdtm library
asl1

asl2 <- read_sas("D:/Sunil/GP/Business/ACL Digital/r programming/adsl.sas7bdat") # read DM dataset directly
asl2
View(asl2)
```

```
saveRDS(asl2, file = "asl2.RDS")
# save permanent R data frame

myasl <- readRDS("asl2.RDS")
# load permanent R data frame
myasl

# enter data values into data frame
vs <- read.table(header = TRUE,
 stringsAsFactors = FALSE,
 text = "
 SUBJID HT WT PERIOD
001 180 NA 1
001 181 77 2
002 173 85 1
002 173 83 2");
View(vs)

test_df <- data.frame(
  class = c("1st", "2nd", "3rd", "Crew"),
  n = c(325, 285, 706, 885),
  prop = c(14.8, 12.9, 32.1, 40.2)
)
View(test_df)

patient <- 1  # assign number patient = 1
is(patient)
patient

Right Variable Assignment
1 ->  patient  # right assign number patient = 1
is(patient)
patient

# Variable Assignment Options
assign('ae', 'Headache') # assign char ae = 'Headache'
ae = 'Headache' # assign char ae = 'Headache'
ae

# Keeping Variables
test_df2=test_df[c('class', 'n')]
View(test_df2)

# Dropping Variables
test_df3= subset(test_df, select = -c(class))
View(test_df3)
test_df4= test_df[-c(3)]
View(test_df4)

df1 <- subset(test_df, class == '1st')
df1 <- filter(test_df, class == '1st')
df1
```

```
# sorting examples using the mtcars dataset
attach(mtcars)

# sort by mpg
newdata <- mtcars[order(mpg),]

# sort by mpg and cyl
newdata <- mtcars[order(mpg, cyl),]
newdata

# pipe example
query1 <- ToothGrowth %>%
select(len, supp, dose) %>%
mutate(dose2 = (dose*2)) %>%
filter(supp == 'VC') %>%
arrange(supp, dose)
View(query1)

# intermediate objects
query1 <- ToothGrowth

query2 <- select(query1, len, supp, dose)
View(query2)

query3 <- mutate(query2, dose2 = (dose*2))
View(query3)

query4 <- filter(query3, supp == 'VC')
View(query4)

query5 <- arrange(query4, supp, dose)
View(query5)

objects()

ASL_mean <- asl1 %>%
 group_by(SEX) %>%
 summarise(avg_age = mean(AGE))
asl1
ASL_mean

table(asl1$SEX)
# proc freq data=asl1; tables sex; run;

table(asl1$SEX, asl1$SAFFL)
# proc freq data=asl1; tables sex*saffl; run;
```

Biography

Sunil Gupta, MS, is an international speaker, best-selling author of 5 SAS books, and a global SAS and CDISC corporate trainer. Sunil has over twenty-five years of experience in the pharmaceutical industry. Most recently, Sunil is teaching a CDISC online class at the University of California at San Diego and classes on Data Science using SAS at UCLA and UCSD

Extension. In 2019, Sunil published his fifth book, Clinical Data Quality Checks for CDISC Compliance Using SAS and in 2011, Sunil launched his unique SAS mentoring blog, SASSavvy.com, for smarter SAS searches. Sunil has MS in Bioengineering from Clemson University and a BS in Applied Mathematics from the College of Charleston.