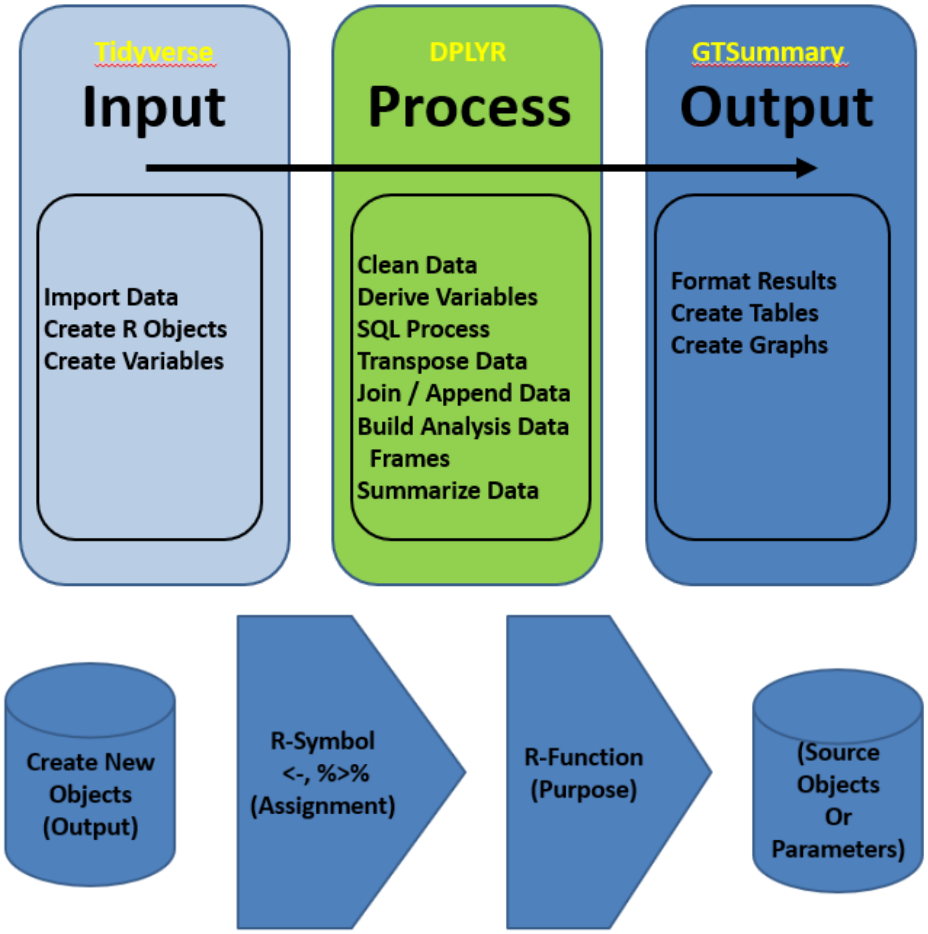
R-Guru.com Cheat Sheet for Statistical Programmers

R Process: Data Input to Statistical Analysis



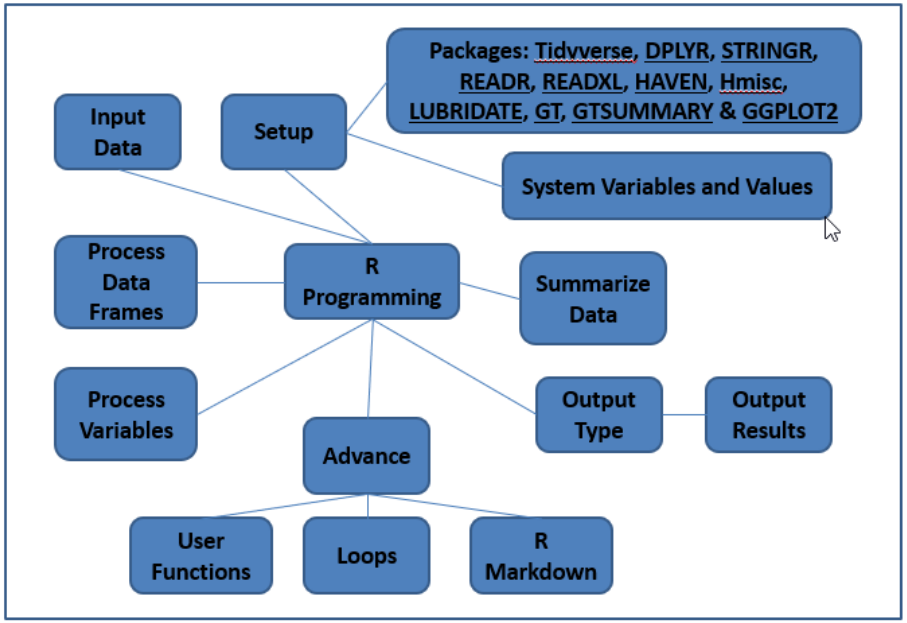
This guide contains common and best practice examples for creating, updating and reporting data frames in the pharma and medical device industries. This guide has sections for workspace setup, compare and contrast common R function and R and SAS and debugging which are ideal for SAS programmers making the transition to R. When possible, base R sample data frames are used in examples.

Tidyverse, DPLYR, DATA.TYPE, STRINGR, READR, READXL, HAVEN, Hmisc, arsenal, LUBRIDATE, PARSEDATE, GT, GTSUMMARY & GGPLOT2 are common and validated R packages by RStudio and the Pharma Industry.

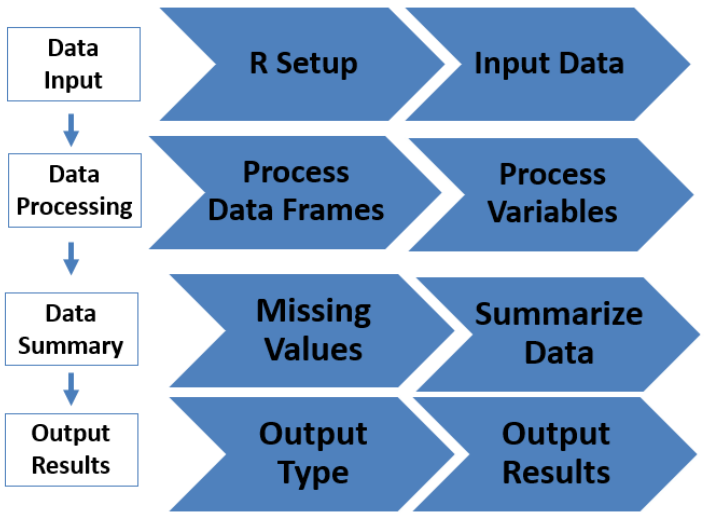
Mutate() function has five key features: case_when(), simple expression, summary functions, rowwise(), and group_by()/ungroup() with summary functions.

df# are data frame names & vr# are variable names. Character or numeric variables depend on the function and values. R functions may be nested for multiple tasks.

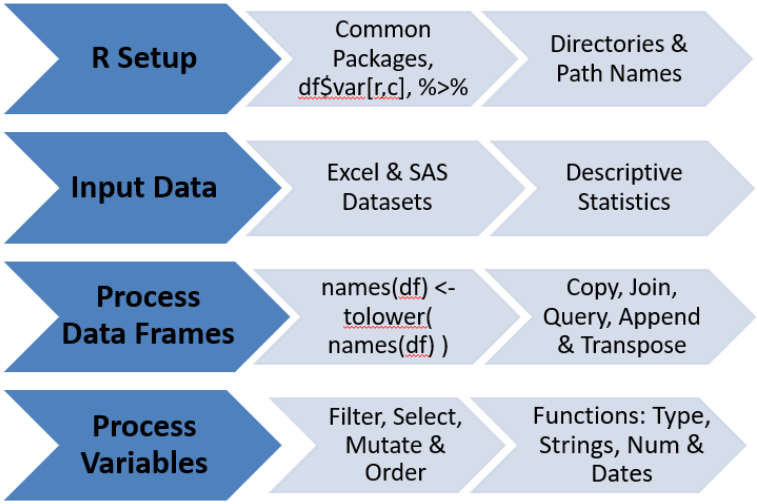
R-Guru Best Practices Mind Map



R Best Practices Checklist



R Best Practices Checklist for Data Input & Processing



PROCESS	TASK CATEGORY
Data Input	Importing Data, Creating Data Frames
Variable Process	Direct Variable Reference, Piping, Conditional and Group Processing, SQL components (Select, Filter, Mutate, Order), Character, Numeric and Date
	Operations, Variable Type Conversions, Summary Variables, Valid Values
Data Frame Process	Transposing, Joining, Appending
Reporting	Data Frames, Tables, Lists, Graphs

Outline

- Listings, Summary Tables and RTF Files
- Debugging R: Syntax, Logic, Data
- SDTMs: Metatools and SDTMChecks

Compare and Contrast R and SAS

While there are common features between R and SAS, there are also many unique features in R: Quick and Easy Graphs based on templates, Logic and Complex Variables, Piping, %>% for concatenating R functions, Object-based language since the concise code is based on special characters, Number of growing R packages and Shiny apps can be created in Days and enable Rapid Data Exploration and Visualization.

TASK	R	SAS
Language	Interpreter	Compiler and Interpreter
Character Var Length	N/A, no need to specify	length
Date Vars	# of days since 01JAN1970	# of days since 01JAN1960
Rounding 2.5	2 (even number)	3 (up)
Sorting Missing Values	'NA' is last obs unless converted to missing	Missing is first obs
Common Features **Data**: Input (Excel, CSV), Management, Analysis, & Reporting (RTF, PDF) **Var Type Conversion**: Character, Numeric and Date Variables **Other**: SQL, Do-Loops	R Studio Data Frames Direct Variable and Record References as.character() , as.numeric() vfmt[df$vr1] R Shiny App	Display Manager Datasets Dataset Options (Keep, Drop, Where) put(), input() proc format

Compare and Contrast Common R Functions

In R, there are multiple methods to accomplish tasks. For example, R functions for many common tasks exist in base R and Tidyverse. In general, using Tidyverse functions instead of base R functions offers more consistency in data frame results instead of vector results. For common R functions between two packages, it is best to specify package when using R functions as filter <- dplyr::filter, filter(mtcars, am & cyl == 8).

The following are best practices in R: %>% to streamline R functions, convert missing values to NA to leverage R's NA functions (is.na()) and options, convert to lower case names since case-sensitive (df, var, values), use data frame options for filtering and selecting variables, creating format objects to standardize values, mutate() to derive variables, case_when() to derive variables using multiple options and including otherwise as last option and including ungroup() when applying group() to prevent issues with downstream R functions.

Three common Data Frames: a. data.frame (simple tables, base R), b. data.table (complex and larger tables), c. tidy or tibble (tidyverse, dplyr or ggplot2) See R syntax comparison cheat sheet.

Just like SAS, R has multiple methods to accomplish similar tasks. The first table shows similar R functions between tidyverse and base R and the second table shows examples of multiple methods. See blog.

Tidyverse	Base R
select(df, var1, var2)	df[, c(var1, var2)]
filter(df, var1 == "name")	subset(df, var1 == "name")
arrange(df, desc(var1))	df[order(rev(df$var1)]
left_join(df1, df2, by = "var1")	merge(df1, df2, by = "var1", all.x = TRUE)
mutate(), summarize(), group_by(), gather(), spread()	mean(), by(), reshape()

TASK	df <- METHOD 1	df1 <- METHOD 2

Pipe or Concatenate R Functions	adsl <- dm %>% select(studyid, subjid, height, weight, scrfl) %>% mutate(bmi=(weight*703) /height^2) %>% filter(scrfl=='Y') %>% select(-scrfl) %>% arrange(studyid, subjid) # |> is an alternative to %>%	dm1 <- select(studyid, subjid, height, weight, scrfl) dm2 <- mutate(dm1, bmi=(weight*703)/height ^2) dm3 <- filter(dm2, scrfl=='Y') dm4 <- select(dm3, -scrfl) dm5 <- arrange(dm4, studyid, subjid) adsl <- dm5
Query, Add Variables	mutate(df, dose2 = (dose*2)) cbind(df, vr1=1, vr2='Drug A')	df1[df1$vr1 == 'male', c('vr1', 'vr2')] # df options
Add Variables by Conditions	mutate(df, vr2=case_when(grep("Yes ", vr1) ~ 'Yes'))	mutate(df, vr2=ifelse(data$vr1 >= 4, 1, 0)) # alternative if_else()
Add Summary Variables using summarize() [Overall, By Group]	mtcars_cyl_summary <- mtcars %>% summarize(mean_mpg = mean(mpg, na.rm = TRUE))	mtcars_cyl_summary <- mtcars %>% group_by(cyl) %>% summarize(mean_mpg = mean(mpg, na.rm = TRUE)) %>% ungroup()
Add Summary Variables using mutate() [Overall, By Group]	df2 <- df1 %>% filter(!is.na(vr1), vr1 > 5) %>% mutate(vr3 = mean(vr2, .1))	df2 <- df1 %>% filter(!is.na(vr1), vr1 > 5) %>% group_by(vr1) %>% mutate(vr3 = mean(vr2, .1)) %>% ungroup()
Variable Type Conversion	vr2=as.character(vr1) vr2=as.factors(vr1)	vr2=as.numeric(vr1)

		vr2=as.Date("2021-01-25")
Recode Values	vfmt <- c("M"="Male", "MALE"="Male", "F"="Female", "FEMALE"="Female") df$vr2 <- vfmt[df$vr1]	recode(vr1, 'val1'='val1a', 'val2'='val2a') recode(vr1, !!!vfmt$vr1))
Select Variables, Subset Records	subset(df1, select=c(vr1, vr2)) filter(df, vr=='X')	select(-scrfl) filter(year %in% c(2010, 2011))
Sort Records	df1[order(df1$vr1, df1$vr2, -df1$vr3),]	arrange(df, vr1, vr2)
Substring Vars	str_sub(vr1 , 3 , 6)	substr(vr1 , 3 , 6)
Concatenate Strings	str_c(vr1, sep="-", vr2)	paste(vr2,"(", format(round(vr3, digits = 2), nsmall = 2),")")
Search, Index String	select(-contains('vr1'))	str_detect(vr1 , "Health") starts_with(), ends_with(), word()
Record Position	vr2=lag(vr1)	vr2=lead(vr1)
Unique Records	unique(df) unique(df[, c('vr1', 'vr2')])	distinct(vr1) duplicated(vr1)
Join DFs	merge(x=df1, y=df2, by = 'vr1') # sort by vr1	left_join(df1, df2, by='vr1') right_join(), inner_join(), full_join()
Append DFs	bind_rows(df1, df2)	rbind(df1, df2) union(df1, df2)
Transpose DFs		pivot_wider(names_from = vr1, values_from = vr2)

	pivot_longer(c("vr1", "vr2"))	spread(., vr1, vr2)
Sample Data	head(df)	tail(df)
Descriptive Stats	summary(df)	table(df$vr)

Environmental Setup, Workspace and Missing Values

NA_character_, NA_integer_, NA_real_, NA_complex_

'NA' r epresents missing values for both character and numeric variables. R has two NULL-like values, NULL and NA. NA is a logical constant of length equal to 1 which contains a missing value indicator, and NULL represents the null object. NULL does not yield any response when evaluated in an expression and NA gives a response which is not available and hence unknown. NaN means not a number.

```
# '' and NA in char vectors are read as '' and NA in data frame but then converted to '' in SAS data sets and reread data frames
# NA in num vector is read as NA in data frame and in SAS
# In CSV files, missing char and num variables are saved as NA in data frames
# In SAS data sets, missing values in num and char variables may be read and saved as . and ‘’ instead of NA but it is important to double check

# one option to standardize missing values in all variables is to assign NA to leverage R’s NA syntax and then convert NA back to . and ‘’ before exporting

df$v1[df$v1== ''] <- NA
df$v2[df$v2== '.'] <- NA
df$v3[df$v3=='9'] <- NA
df$v4[df$v4==-1] = NA

# note that in R char variables internally store NA which impacts paste()

df[is.na(df)] <- ''                    # recode all char vars with NA to ‘’
df["vr1"][is.na(df["vr1"])] <- 0       # vr1 recode NA to 0
df %>% replace_na(list(vr1=0))         # vr1 recode NA to 0

Missing(== '', == '.') and non-missing (!='', !='.', na.rm=TRUE) conditions

ls()                                   # list all objects in workspace
remove(list=ls())                      # remove all objects, rm(df) to remove df

df$v1[df$v1 %in% c(‘’, ‘.’)] <- NA     # vr1 recode ‘’ and . to missing NA

df[is.na(vr1)] <- mean(vr1, na.rm = TRUE) # impute missing vr1 with mean
```

Metadata Functions for List Processing, Describing and Viewing

Metadata functions empower programmers to confirm data frame and variable attributes and scope of values. In addition, metadata functions support standardizing lower case variable names, types and missing values.

```
View(df) # View data frames
str(df) # Display object type and sample records: data frame, variable, etc.
class(vr) # Display variable type – character, numeric, integer, complex
ls(df) # display df variable names

head(df) # display first sample records
tail(df) # display last sample records

mutate(df, counter = row_number( ) ) # create sequence # based on sort order

summary(df) # display descriptive stats of all continuous variables
table(vr) # display freq of categorical variables

dim(df) # display total number of rows and columns in df
nrow(df) # display the total number of rows in df
ncol(df) # display total number of columns in df

print( df[ c(1,3), ] ) # display index record number 1 and 3 in df, print(df) for all
print(contents(df), maxlevels=10, levelType='table') # requires hmisc package

names(adsl)= tolower(names(adsl))      # lower case all var names, toupper()

df <- df %>% rename(vr_new = vr_old)  # rename variables
```

```
label(df$vr1) <- 'My Label'                    # assign variable labels

df2 <- df1 %>% select(-contains('vr1')) # drop variables names that contain vr1

df <- select (vr1, vr10:vr15, starts_with("L"))  # select vr1, vr10 to vr15
    variables by order and variables that start with L, ends_with()

mutate(df, across(where(is.character), replace_na(., ‘’) )) # replace all
character variables with NA to blanks

df(all_miss[all_miss > 0])       # display vars with all missing values
```

Lists: R Function Processing a list of Objects

```
thislist <- list("apple", "banana", "cherry")  # Create a lists of values
for (x in thislist) {                                  # loop through values in list
  print(x)                                  # apply R function to all values in the list
}

# lists are often processed by the family of apply() functions
apply(mtcars, 2, mean)      # overall summary of each column for all records

df_list <- list(df1, df2, df3)           # unite three data frames into a list of dfs

summarise_at( vars(mpg, wt), list(m=mean, sd=sd), na.rm=TRUE)
```

Factors: Data Validation and Non-Alpha Character Sort Order

In general, factors are character or integer variables with limited unique values. Factors are useful to validate values, create categories and in statistical modeling. In addition, factors are required to control row sort order.

```
vr1 <- c("Dec", "Apr", "Jan", "Mar")   # months in vr1 character type

month_levels <- c("Jan", "Feb", "Mar", "Apr", "May", "Jun",
  "Jul", "Aug", "Sep", "Oct", "Nov", "Dec" ) # list and order of all valid months

f1 <- factor(vr1, levels = month_levels) # create f1 using month_levels factor
sort(f1) # Jan Mar Apr Dec     # sort f1 by month order instead of alpha order

vr2 <- c("Dec", "Apr", "Jam", "Mar") # Jam typo
f2 <- factor(vr2, levels = month_levels) # assign NA to invalid data, Jam
f2        # Dec  Apr  <NA> Mar
```

Import Data: Data frames, CSV, Excel and SAS Datasets


```
install.packages('tidyverse')    # install packages from CRAN, repeat for each
package best to apply version control

installed.packages()             # confirm packages and versions installed

tidyverse_conflicts()        # Conflicts between tidyverse and other packages

packageVersion("ggplot2")       # confirm installed package version
packageDate("ggplot2")          # confirm installed package date
packageDescription("ggplot2") # display package description

library(readxl)                              # read excel files
library(haven)                               # read SAS datasets
library(tidyverse)                 # load popular data management package

file.edit("~/Desktop/foo/.Rprofile")     # In text file, add packages to autoload
library(ggplot2) library(scales) library(plyr) library(reshape2) # list packages
```

```
install.packages("librarian")        # alternative method to autoload packages

librarian::shelf(ggplot2, DesiQuintans / desiderata, pander)  # / options
   # apply to specify package for common functions <package>::<function>

df <- readRDS("df.RDS")                          # read R df data frame
load('df.RData')                                 # load df into R workspace
attach(df)                                       # set df as default df

df <- read_csv("C:/mydata/my_csv.csv")           # read csv, forward '/' paths

df <- read_excel("C:/mydata/my_excel.xlsx")  # read excel, all missing as 'NA'
                                                 # confirm date values and format

sdtm <- "c:/my_sdtm"                             # create full path reference name
df<-read_sas(file.path(sdtm,"adsl.sas7bdat")) #read dataset,missing ., '' or NA
mydata <- sasxport.get("c:/mydata.xpt") # read xpt, requires Hmisc package
```

Create Output Files: Data Frames, Text, Excel and SAS Datasets

```
setwd("C:/mydataframes")          # change default working folder
getwd()                           # confirm working folder

sink(file='r-program.txt')        # start writing results to text file
str(df)                           # data frame metadata
summary(df)                       # run one or more R function
sink(file=NULL)                   # stop writing results to text file

saveRDS(df, file = "df.RDS")      # save as permanent data frame

write.table(df,"C:/myoutput/mydm.txt", sep="\t")  # save df as text file

write.csv(df, "C:/myoutput/mydm.csv", row.names = FALSE) # save csv file

write_sas(df, "df.sas7bdat")      # save as SAS dataset, haven package

write_xpt(df, "C:/myfile/adef.xpt") # save as xpt file, haven package
                  # best practices to save as xpt instead of SAS dataset
```

Create Data Frames by Combining Variables
Variable Types: Numeric, Integer, Character, Factors, Logic, Dates, Complex

df <- data.frame(vr1 = c()) # collect items

```
# Create id, visit and aval variables using rep() options
id <- rep(c("1", "2", "3"), each = 2) # each = 2 option creates id = '1', '2', '3', '4',
'5', '6'

visit <- rep(c("baseline", "visit 1", "visit 2"), times = 2) # times = 2 creates visit
= 'baseline', 'visit 1', 'visit 2', 'baseline', 'visit 1', 'visit 2'

aval <- c(10, 15, 20, 8, 12, 18) # create aval with 6 records

df <- data.frame(vr1=c(50, 60), vr2=c('male', 'female'), vr3=c(TRUE, FALSE),
vr4=c("2021-01-25", "2022-02-10"))  # date entry of vr1 (numeric var), vr2
(character var), v3 (logic var), vr4 (date var) of 2 obs
                                                  # yyyy-mm-dd

df <- data.frame(vr1=c(50, NA), vr2=c('male', NA), vr3=c(TRUE, NA),
vr4=c("2021-01-25", NA)) # create data frame with missing values

df <- data.frame(vr1, vr2, vr3, stringsAsFactors=FALSE)  # vars are in order, do
not create factors for string vars (default)

df <-tibble( vr1=c(50, 60), vr2=c('male', 'female')) # create tibbles

# factors are limited unique character, numeric or dates that control order
# as.factor() function is used to convert to factors

x = 3L       # Create an integer variable, whole numbers
y = TRUE   # Create a logic variable
z = 1 + 2i   # Create a complex number or expression
```

Add Derived Numeric and Character Constants to Data Frame

```
df2 <- cbind(df1, vr1=1, vr2='Drug A')     # to df1, add vr1 and vr2
```

Base R Data Frame Options: Direct Row & Column Reference to Select, Filter, Order and Recode, Single [] and Double [[ ]] Brackets

df2 <- df1[*row* conditions / #, *column* conditions / #]

df2$vr2 <- R(df1$vr1 condition) # vr2 assignment
df2$vr2[df1$vr1 condition] <- constant # vr2 assignment

df[[*column* conditions / #]] # returns a vector

```
# in general, the use of quotes are required for base R functions and non-quotes around variables for Tidyverse R functions
# data frame options are best practices for selecting, filter and ordering

df[['vr']]      # double brackets display all values for vr variable

df[[1]]         # apply with caution, alternative to use index 1, first variable in df

print (data_frame[2]) # display as df index 2 variable, col2
print (data_frame[[2]]) # display as variable index 2 variable, col2

df2 <- df1[c('vr1', 'vr2')]          # select vr1 and vr2 by var name
df2 <- df1[c(vr1, vr2)]              # alternative method, in general, most R functions accept both methods, 'vr1' and vr1 but some may not

print( df[ c(1,3), ] ) # display index record number 1 and 3 in df

df2 <- df1[df1$vr1 == 'male', ]   # df [ row ref., column ref.] df options
                                  # rows: filter (==, <, >), order()
                                  # columns: keep all is default, c() to select vars

print(df1[df1$vr1 == 'male', c('vr1', 'vr2')])   # print vr1 & vr2 for males

data$vr2 <- ifelse(data$vr1 >= 4, 1, 0)  # derive vr2 by condition, True, False

class[class$Age>=mean(class$Age),] # filter age >= mean age, all class vars

data$agecat[num_range(40 – data$age – 60)] # derive agecat as 40 - 60

df2 <- df1[order(df1$vr1, df1$vr2, -df1$vr3),] # order by vr1, vr2 and -vr3
```

variable_format <- c('source' = 'target')
df$vr2 <- variable_format[df$vr1]

```
vfmt <- c("M"="Male", "MALE"="Male", "F"="Female", "FEMALE"= "Female")
# best practices to create variable format object as 'source' = 'target'

df$sex_full <- vfmt[df$sex] # apply variable format object as an R function to recode sex to sex_full to get only Male and Female values

# recode using df row and column references
# df["Column Name"][df["Column Name"] == "Old Value"] <- "New Value"
df["vr1"][df["vr1"] == value1] <- value2  # in vr1 recode value1 to value2

                                  # assign agecat var in dm data frame
dm$agecat[data$age < 20] = 1      # assign 1 for age < 20
dm$agecat[data$age >= 20 & data$age < 40] = 2 # assign 2 for 20<= age < 40
dm$agecat[data$age >= 40 & data$age < 60] = 3 # assign 2 for 40<= age < 60
```

SQL (DPLYR) to Select, Filter, Mutate & Order
Piping: Output of 1st function is input of 2nd function

filter(), select(), mutate(), arrange(), summarise()

Dplyr Verb	What it Does
`filter()`	Retrieve rows (based on values)
`select()`	Retrieve columns (based on names)
`mutate()`	Add new variables
`arrange()`	Change the order of the rows
`summarise()`	Compute a summary statistic

Order

```
6 adsl <- dm %>%    # separate lines per R command help for reading
1 select(studyid, subjid, age, sex, height, weight, race, scrfl) %>%
2 mutate(bmi = (weight*703)/height^2 ) %>%
3 filter(scrfl == "Y") %>%
4 select(-scrfl) %>%
5 arrange(studyid, subjid)
```

With %>%, several R commands execute together which is similar to SAS Procedures.

Order Alternative without %>% are One Line R commands

```
1 dm1 <- select(dm, studyid, subjid, age, sex, height, weight, race, scrfl)
2 dm2 <- mutate(dm1, bmi = (weight*703)/height^2 )
3 dm3 <- filter(dm2, scrfl == "Y")
4 dm4 <- select(dm3, -scrfl)
5 dm5 <- arrange(dm4, studyid, subjid)
6 adsl <- dm5
  objects()   # displays all objects created
```

Without %>% requires creating 5 intermediate objects that each contain updated results. Get to confirm each R function.

```
# mutate() is best practice for deriving new variables
# piping is best practice for concatenating R functions

myquery <- ToothGrowth %>%      # 1st is to access source data frame
  select(len, supp, dose) %>%   # 2nd is to select variables, - vr1 to drop
 filter(tolower(supp) == 'vc') %>%  # 3rd is to filter records, &(and),|(or), !(not)
  mutate(dose2 = (dose*2)) %>%  # 4th is to derive vars w/ simple expressions
  arrange(supp, dose)           # 5th is order records, desc() for descending

  # filter(year %in% c(2010, 2011)) %in% is used to filter multiple values
  # filter(sex == 'm', age > 21)      # and operator

df2 <- subset(df1, vr1 == 'A' & vr2 < 20)    # select with two conditions w/ and
df2 <- subset(df1, vr1 == 'A' | vr2 < 20)    # select with two conditions w/ or

df2 <- subset(df1, select=c(vr1, vr2) )      # select variables vr1 and vr2
df2 <- subset(df1, select=-vr1 )             # drop vr1 variable
df2 <- subset(df1, select=-c(vr1, vr2) )     # drop variables vr1 and vr2
```

Multiple Conditional Processing to Derive Variables

Function	Meaning	Example
	For numerical comparisons	
<	less than	`age < 21`
>	greater than	`age > 65`
==	matches exactly	`age == 39`
%in%	is one of	`age %in% c(18, 19, 20)`
	For character strings	
<	alphabetically before	`name < "ad"`
>	alphabetically after	`name > "ac"`
==	matches exactly	`name == "Jon"`
%in%	is one of	`name %in% c("Abby", "Abe")`

Another important operation, `ifelse()` allows you to translate each value in a variable to one of two values, depending on the result of a comparison. For instance

```
ifelse(age >= 18, "voter", "non-voter")
```

```
# recode 'val1' to 'val1a' and 'val2' to 'val2a'
df2 <- df1 %>% mutate(vr2=recode(vr1, 'val1'='val1a', 'val2'='val2a'))

df2 <- df1 %>%                          # copy df1 data frame to df2
  mutate(vr2 = case_when(               # derive vr2 by mutually exclusive  condition
  . < vr1 <= 25 ~ "25 or Below",        # if vr1 between 0 and 25 then label
  25 < vr1 < 50 ~"26 - 50",             # else if vr1 between 26 and 49 then label
  TRUE ~ ">= 50" ),                     # for best practices, otherwise >= 50
  vr4 = vr3 == "Y",                     # derive vr4 logical var (TRUE, FALSE) from vr3
  base = aval[visitnum==1],             # derive base from aval where visitnum = 1
  ablfl = ifelse(visitnum == 1,"Y",NA),                    # derive abfl flag variable
  chg = ifelse(!is.na(aval) & !is.na(base), aval-base, NA),  # derive chg variable
  pchg = ifelse(!is.na(aval) & !is.na(base), ((aval- base)/base)*100, NA) )
```

Character String Operations to Combine, Remove, Filter and Substring

vr3 <- paste(vr1, 'text', vr2, 'text', sep='')

```
df2 <- df1 %>% mutate(vr1 = paste(vr2,"(",      # derive vr1 by combining
format(round(vr3, digits = 2), nsmall = 2),")"  # vr2, round vr3 and format
  ) )                                           # end paste() and mutate() functions
```

```
# paste() can combine multiple variables and text with or without sep=’’
delimiter default is space

vr3 <- str_c(vr1, sep="", ‘N=’, vr2, ‘)’)  # concatenate two vars as vr1 (N=vr2)
vr3 <- str_c(vr1, sep="-", vr2)            # concatenate two vars as vr1-vr2

vr2 <- str_replace_all(vr1 , "Street" , "St" )  # replace all ‘Street’ with ‘St’

vr2 <- str_remove_all(vr1, "NA")   # remove all NA in vr1

vr2 <- str_trim(vr1, side='both')    # remove blanks from ‘left’ and ‘right’ sides

vr2 <- str_extract(vr1 , "\\d*" )     # char vr1, extract only digits
                                      # same as "\\d+"

vr2 <- str_extract(vr1 , " [a-z]+" )    # char vr1, extract only chars
                                    # see reference to extract special characters

filter(str_detect( vr1 , "Health"))    # filter records by searching text

mutate(df, vr2 = word(vr1, 1 ))          # scan and extract first word into vr2
mutate(df, vr2 = case_when(grep("Yes", vr1) ~ ‘Yes’) ,
          # best practice to use case_when to create vr2=’Yes’ when vr1=’Yes’
   TRUE ~ "")             # best practice to include otherwise TRUE option

vr2 <- str_sub(vr1 , 3 , 6 )       # substring vr1 text from 3rd to 6th position
vr2 <- substr(vr1 , 3 , 6 )       # alternative method to substring vr1 text
```

Variable Type Conversion Between Numeric & Character Variables

Test	Convert
is.numeric()	as.numeric()
is.character()	as.character()
is.vector()	as.vector()
is.matrix()	as.matrix()
is.data.frame()	as.data.frame()
is.factor()	as.factor()
is.logical()	as.logical()

```
vr2 <- as.character(vr1)   # number vr1 num var to vr2 char variable, 8 length
vr2 <- as.numeric(vr1)     # number in vr1 char var to vr2 num variable
vr2 <- as.integer(vr1)     # number in vr1 num var to vr2 whole num value
vr2 <- as.factors(vr1)     # vr1 num or character var to vr2 factors value
```

Date Management Packages (lubridate, parsedate, anytime)

vr1 <- as.Date("2021-01-25")

Import > Date (# of days since 1/1/1970) > Format
ymd(), mdy(), myd() year(), month(), yday()

Source / Target Full / Partial / Dates / DateTimes / Times
Split / Combine / Imports / Formats / ISO 8601
Character / Numeric / Date Variables
Short / Full Text / Time Zones / AM / PM / Versions
Units: Years / Months / Weeks / Days / Quarters / Hours / Seconds
Assignments / Calculations / Comparisons / Periods / Durations / Intervals

R stores dates as the number of days since 01JAN1070. Above are many factors that need to be considered when working with dates. Processing dates can be tricky since dates can be imported from character, numeric, date or date time variables. Once character or numeric variables are converted to date variables, then date operations can be performed. In addition, dates can be in most any format with various delimiters. Correctly importing and

formatting dates can be challenging without the appropriate date or datetime function and format.

Symbol	Definition	Example
%d	Day as a number	19
%a	Abbreviated weekday	Sun
%A	Unabbreviated weekday	Sunday
%m	Month as a number	04
%b	Abbreviated month	Feb
%B	Unabbreviated month	February
%y	2-digit year	14
%Y	4-digit year	2014

```
dt1 <- today()                  # return today's date, R stores # days since 1970
dt2 <- anydate()                # convert to date from any format

dt3 <- as.Date("1990-04-25"))   # convert string to date var
dt4 <- ymd("1990-04-25"))       # alternative to convert string to date var

y <- year(dt3)                  # extracts year from dt3, ex. 1990

m1 <- month(dt3)                # extracts month from dt3, ex. 04
m2 <- month(dt3, label=TRUE)    # extracts month from dt3, ex. Apr

d1 <- wday(dt3, label=TRUE)     # extracts work day name from dt3, ex. Mon
d2 <- yday(dt3)                 # extracts # of day from the year, ex. 284

dt5 <- makedate(1990, 04, 25)   # create date var from numbers

filter(dob > as.Date("1990-04-25"))   # import char value into date var

parse_date("04/15/99")          # import date in any format

dates <- c(30829, 38540)  # convert number date value from excel into dates

betterDates <- as.Date(dates, origin = "1899-12-30") # origin is Dec 30, 1899

format(betterDates, "%a %b %d")       # format dates as Mon Jan 1

parse_iso_8601("2013-02-08 09:30")  # Import ISO8601 dates into dates

datetime <- ISOdatetime(yr, mon, day, hr, min, sec) # format into ISO dates

format_iso_8601(parse_iso_8601("2013-02-08 09:34:00")) # to yymmdd10
```

Partial Dates, ex. '2014', '2014-12-31'

```
library(tidyverse)
library(lubridate)

my_df <- tribble( ~mychardate,       # assign vr name
  "2014-12-31", "2014-12", "2014" ) # create three dates

# break dates into yr, mon and day to assign first day of mon and dec month

my_df2 <- mutate(rowwise(my_df),
  raw_ parts = str_split(mychardate, pattern = "-"),  # break into components
  parts = length(raw_parts),                          # count components
  my_date = case_when( parts == 3 ~ ymd(mychardate),   # non-missing
parts == 2 ~ ceiling_date(ymd(paste0(mychardate, "-01")), unit = "month") -1,
         parts == 1 ~ ymd(paste0(mychardate, "-12-31"))    )  )  # yr non-missing
```

Study Day Calculations

```
aedm1 <- aedm %>%
  filter(!is.na(AESTDTC) & !is.na(RFSTDTC)) %>%      # non-missing dates
  mutate(
    AESTDY = case_when( AESTDTC >= RFSTDTC ~ (AESTDTC - RFSTDTC) + 1,
      AESTDTC < RFSTDTC ~ AESTDTC - RFSTDTC,
      TRUE ~ NA_real_ ))        # otherwise assign to '.', best practices
View(aedm1)
```

Date Operations: Assignment, Periods, Durations and Intervals

Durations are in seconds, Periods are in days, weeks, months and years, and Intervals are duration or difference between start and end date times

df[df$dt >= "2020-12-25" & df$dt <= "2020-12-28",] # subset between dates

df1$ndt=as.Date(df$cdt, format="%d%b%Y") # assign date in date9 format

vrdt = strptime(vrdtm, format = "%B %d, %Y %H:%M:%S") # import date time using format

vrdtc = format(vrdt, format = "%Y-%m-%dT%H:%M:%S") # final date time variable using format

format(date, format="%m/%d/%y") # format as mm/dd/yy
+ ddays(1), + dweeks(1), + dmonths(1), + dyears(1)
# four options to add 1 day, week, month or year

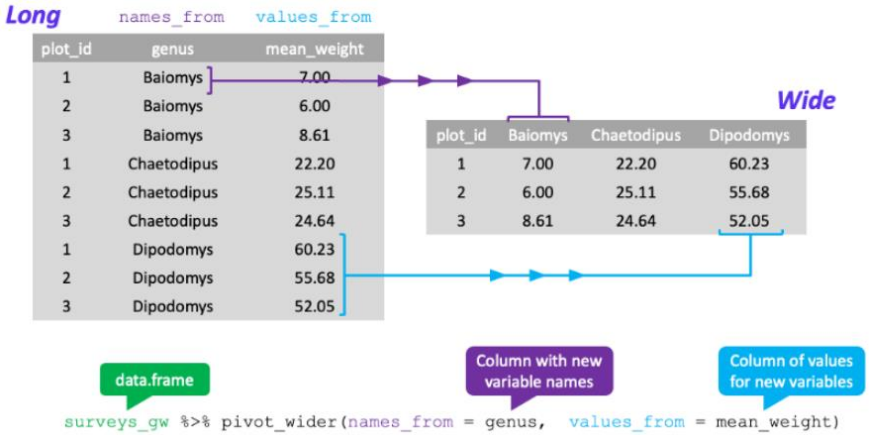
difftime("2020-5-16", "2020-1-15", units = "days") # duration between two dates in days

difftime(dtvr1, dtvr2, units = "weeks") # duration between two dates in weeks

interval(dtvr1, dtvr2) %/% months(1) # of months between dates

Transpose Data Frames Between Long (Records) and Wide (Variables)

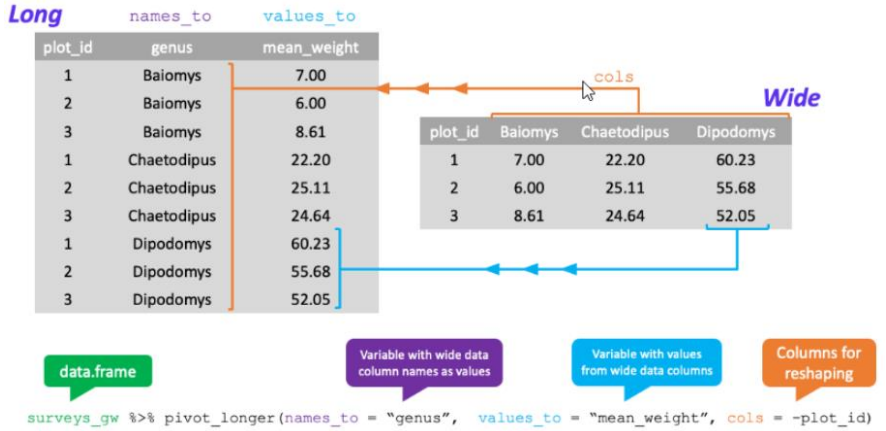
Long (Records) to Wide (Variables) Structure



df2 <- df1 %>% select(by_var, vr1, vr2) %>%
# vr1 contains new valid char variable name values
pivot_wider(names_from = vr1, values_from = vr2) # vr2 contains values
use prefix='col:' for numbers in vr1

spread(., vr1, vr2) # alternative method, vr1 has var names, vr2 has values

Wide (Variables) to Long (Records) Structure



df2 <- df1 %>% # all other variables in df1 are group by variables unless vars are selected
pivot_longer(c("vr1", "vr2")) # transpose vr1 to name and vr2 to value, ideal for limited number of variables to transpose

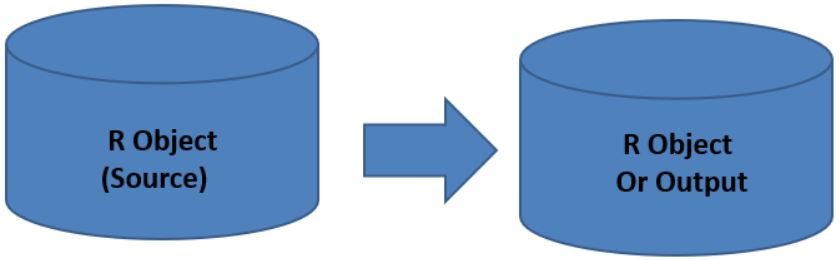
```
df2 <- df1 %>% pivot_longer(cols = c("hrvalue", "prvalue", "rrvalue"),
                            # alternative method to transpose three vars
   names_to = "testcd", # new variable with transposed variable names
   values_to = "value")  # new variable with transposed variable values

df2 <- df1 %>%      # alternative method for selecting wild card vars
  pivot_longer( cols = starts_with("wk"),    # select all wk* variables
    names_to = "week",        # create new week variable
    names_prefix = "wk",      #  select wk# variables to transpose
    values_to = "rank",       # create rank variable with wk variable values
    values_drop_na = TRUE)  # do not create records with NA for byvar obs
```

Group Processing to Derive Summary Variables

- ***Identify Unique and Duplicate Records***
- ***First and Last Group By Variables***
- ***Highest Value, Group by Count Variables***
- ***Descriptive Statistics of Numeric Variables***
- ***Frequency Counts of Character Variables***

Summary functions are by overall, unless group_by() is specified.
Summarize() function summarizes the original data frame while mutate() with summary functions keeps all original data frame variables and adds summary variables.

Summarize Data



One record per Patient
One record per Group
One summary record
Remove Duplicate Records
Transpose Records to Variables
Transpose Variables to Records

Identify Unique and Duplicate Records

```
distinct(df, vr1, .KEEP_ALL=TRUE)  # keep only unique records, all vars

df[unique(df$vr1)]              # identify unique vr1 values
unique( df[ , c('vr1','vr2') ] )  # identify unique vr1 and vr2 values
df2 <- unique(df1)      # identify unique values across all vars, also distinct(df1)

df[duplicated(df$vr1)]        # identify duplicate vr1 values

df1[!duplicated(df1$vr1), ]  # remove duplicate vr1 values

df_dups <- df[ c("vr1", "vr2") ]  # combine vr1 and vr2 into one object
df[!duplicated(df_dups),]    # remove duplicates from vr1 and vr2
```

Derive First (min) and Last (max) Group By Variables

First. (Dot) or Last. (Dot) Group By Var
group_by() %>%
slice() %>%
df3 <- df1 %>% mutate(vr1=, v2=) %>%
left_join(df2, by=)

```
first_mpg <- mtcars %>%   # mtcars input and first_mpg is final df
  group_by( mpg, cyl ) %>%      # group by mpg and cyl
     # apply one of the options below to select by group records: a – d
    slice(1)         # a. flag first group by records, similar to first. in SAS
    slice_min(order_by=var1) # b. flag min var1 records group by mpg
    slice_max(order_by=var1) # c. flag max var1 records group by mpg
    slice(n() )      # d. flag last group by records, similar to last. in SAS
  %>% mutate(flag="Y")         # create first or last flag variable for that record
  %>% right_join(mtcars, by=c('mpg', 'cyl'))  # right join to keep all records
and vars

first_mpg <- mtcars %>%   # mtcars input and first_mpg is final df
 group_by( mpg ) %>%      # group by mpg
 mutate(vr2=lead(vr1), vr3=lag(vr1) )  # store next or previous record values
  %>% upgroup()           # best practice to reset without group
```

```
df2a <- nafill(df1, "locf") # derive last observation carried forward for all df1
                            num vars, requires data.table package

df2b <- group_by( by_var ) %>%      # group by_var
  nafill(df1[, c('n') ], "locf")    # derive locf for n var
```

Derive highest value by vr1 and Group By Count Variables

```
df2 <- df %>% group_by(vr1) %>% select(vr2) %>%
 top_n(3) %>% arrange(vr1 , desc(vr2))
                        # group_by(vr1, vr2) for multiple group by variables
                        # top 3 vr2 values, sort by vr1 and descending vr2

df1 %>% group_by(vr1, vr2) %>%mutate(count(vr3 ='Total')) # Create
sequence number variable by group variables and count()

df1 %>% group_by(vr1, vr2) %>%mutate(count= row_number()) #alternative
method to create sequence number based on vr1 and vr2 sort order.
```

Derive Descriptive Statistics Variable, Left Join to Add to Data Frame One of Four Methods

In general summarizing variables is across records using summarize() and mutate(), but can also be done across multiple variables using rowMins() and rowwise(). In addition, best practices to include ungroup() after each group_by() to prevent subsequent group processing, best used with mutate() to keep all variables.

- Across Records using summarize(), mutate(), min(), mean(), percent()

df2 <- df1 %>% group_by(vr1) %>%
summarize(vr3 = summary_function(vr2)) %>%
ungroup()

```
mtcars_cyl_summary <- mtcars %>%  # final and source data frames
  summarize(mean_mpg = mean(mpg, na.rm = TRUE)) # best to ignore NA
                                  # derive overall mean_mpg variable, min(), max(), n(), count()

mtcars_cyl_summary <- mtcars %>%  # final and source data frames
  group_by(cyl) %>%               # group by cyl, without group by is overall
  summarize(mean_mpg = mean(mpg, na.rm = TRUE)) # best to ignore NA
 %>% ungroup()                    # best practices for ungroup subsequent processing
                                  # derive mean_mpg variable, min(), max(), n(), count()

mtcars_cyl_summary <- mtcars %>%  # alternative method
  group_by(cyl) %>%               # without group by is overall
  summarise_at( vars(mpg, wt),    # list vars to create x_m, x_sd vars
          list(m=mean, sd=sd), na.rm=TRUE)  # list statistics, mean & sd
 %>% ungroup()                    # best practices for ungroup subsequent processing
```

df2 <- df1 %>% group_by(vr1) %>%
mutate(vr2 = summary_function(vr1)) %>%
ungroup()

```
df2 <- df1 %>%                      # 1. source df
  filter(!is.na(vr1), vr1 > 5) %>%  # 2. filter non-missing 'NA' & > 5
  group_by(vr1) %>%                 # 3. group by vr1, else by overall
  mutate(vr3 = mean(vr2, .1)) %>%   # 4. summary function mean vr2, round
  ungroup()                         # 5. Best practices for ungroup subsequent processing
```

- Across one variable

```
vr1=c(2, 4, 6)      # combine 3 values into one variable
vr2 <- min(vr1)     # max(), sum(), count(), mean(), percent(), n() - obs
```

- Across Variables using rowMins(), rowMax(), rowMeans()

```
df3$vr1 <- rowMeans(df2, na.rm=TRUE )   # derive mean of all df2 vars
```

- Across Variables using rowwise() with min(), max()

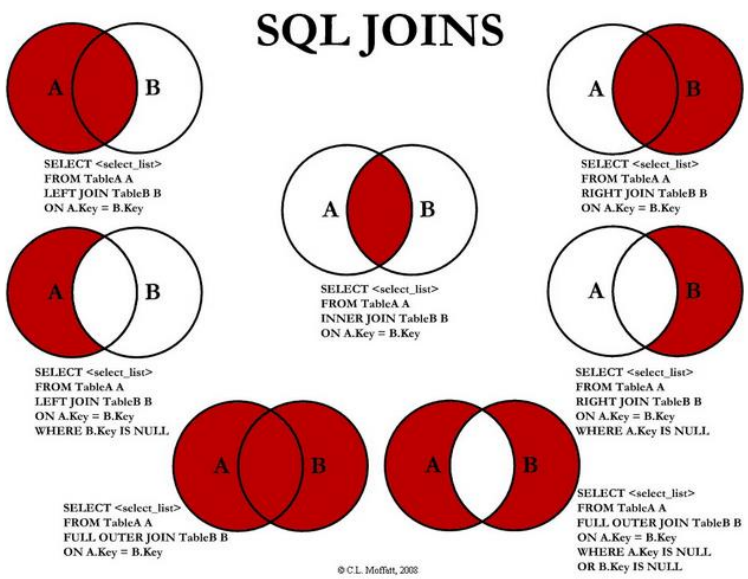
```
df2 <- df1 %>%          # derive min, max variables from vr1 and vr2
 rowwise() %>% mutate(vr3= min(vr1, vr2), vr4= max(vr1, vr2)) # multiple vars
```

Frequency Counts of Character Variables

```
table(df$vr)                                  # frequency counts of vr
table(df$vr1, df$vr2)                         # frequency counts of vr1*vr2
table('Column A'=df$vr1, 'Column B'=df$vr2)   # with column labels
```

Join Data Frames to Concatenate Variables

While join() from tidyverse and merge() functions from base R are similar, there are slight differences. Join() preserve the original order of data frame rows while merge() sorts the rows alphabetically based on the by column.



df3 <- left_join(df1, df2, by='vr1')

```
df3 <- df1 %>% left_join(df2, by='vr1')          # join by the same by variables
df3 <- left_join(df1, df2, by='vr1')             # alternative method

df3 <- right_join(df1, df2, by= c('vr1', 'vr2'))     # join by two by variables
df3 <- inner_join(df1, df2, by= c('vr1' = 'vr2'))    # join by different by variables
df3 <- full_join(df1, df2, by= c('vr1' = 'vr2'))     # join by different by variables

df3 <- setdiff(df1, df2) # unique df1 values by variables and rows
df3 <- setdiff(df2, df1) # unique df2 values by variables and rows

df3 <- semi_join(df1, df2) # returns only df1 records not in df2
df3 <- anti_join(df1, df2, by=c('vr1')) # list all non-matching vr1 by values
df3 <- intersect(df1, df2) # list all values common in df1 and df2
df3 <- union(df1, df2) # appends df2 to the end of df1, similar to bind_rows()

df3 = merge(x=df1, y=df2, by = 'vr1', all.x=TRUE) # left join by vr1
df3 = merge(x=df1, y=df2, by = 'vr1', all.y=TRUE) # right join by vr1
df3 = merge(x=df1, y=df2, by = 'vr1')             # inner join by vr1
df3 = merge(x=df1, y=df2, by = 'vr1', all=TRUE)   # full outer join by vr1

df_list <- list(df1, df2, df3)          # unite three data frames into a list of dfs
df4 <- df_list %>% reduce(full_join, by='id')  # full join three data frames

df3 <- bind_cols(df1[,c('vr1')], df2)   # record join w option & w/o by variables
df3 <- crossing(df1, df2)               # many-to-many join without by variables
```

Append Data Frame Records to End of First Data Frame Records

```
df3 <- bind_rows(df1, df2, df3)    # append data frames with uneven variables

df4 <- rbind(df1, df2, df3)    # append two or more even variables data frames
```

Subquery Condition and Piping in DF1 to Filter DF2 Records

```
                                  # Execution Priority Order
df3 <- df2 %>%                    # 5th to create final df3 data frame
  filter(vr1 %in% (               # 4th to filter vr1 values in df2 data frame
    df1 %>%                       # 1st to lookup df1 data frame
    filter(vr2 == 'male') %>%     # 2nd to filter vr2=males in df1
    pull(vr1) %>% unique))        # 3rd to select unique vr1 values for df2 filter
```

Custom Functions – Input: DF, Vr, Last Contact Date

```
last_date <- function(tbl, varname=REF){ # tbl parameter is df, varname is vr
 tbl %>%                                 # input df
  select(USUBJID, {{varname}}) %>%       # select usubjid and vr
  filter(!is.na({{varname}})) %>%        # not missing vr
  group_by(USUBJID) %>%                  # group by usubjid
  summarise(max_date = max({{varname}})) # max vr by usubjid as max_date
```

```
}

eg %>% last_date(EGDTC) -> eg_last  # call last_date function
# eg$EGDTC input, eg_last output
```

Compare Data frames – Requires arsenal package

```
summary(comparedf(prod, qc, by = "id")) # compare by id vars
                                        # See also setdiff(), semi_join() and anti_join()
```

Graphs: Scatter Plots, Lines, Box Plots, Bars and Histograms

```
ggplot(df                          # data frame name
, aes(x = vr1, y = vr2             # vr1 for x and vr2 for y axis variables
, fill= , color= , col=, size=)    # valid options with valid values
# one or more required options below, defaults unless options are specified
+ geom_point()       # scatter plot two quantitative variables
+ geom_line          # trend lines over time
+ geom_boxplot()     # box plot one continuous and one categorical variable
+ geom_bar()         # bar of variables, options: stat=
+ geom_histogram() # histogram of x-axis for counts
+ geom_smooth(method='lm', formula=y~x, se=F) # smooth option
# one or more format options below, defaults unless options are specified
+ theme(), + ggtitle(), + xlab(), + ylab()
```

Listings, Summary Tables and RTF Files

```
library(gt)                # use gt() to convert data frames to listing rtf files
gt(df, rowname_col = "df") %>% gtsave(df, "t1.rtf", path="C:/t1")

library(gtsummary)                 # create gtsummary() object table
df <- asl1[, c('TRT01A', 'SEX', 'RACE', 'AGE')] %>%  # mean/sd and n/%
  tbl_summary(
    by = TRT01A,                                # by variable
    statistic = list(                           # list method is applied
      all_continuous() ~ "{mean} ({sd})",       # all variables keywords
      all_categorical() ~ "{n} / {N} ({p}%)" ), # alternative method
    digits = all_continuous() ~ 2,              # c(SEX, RACE) ~ "{n} / {N} ({p}%)"
    label = AGE ~ "My Age",                     # column label
    missing_text = "(Missing)" )                # missing row label

as_gt(df) %>% gtsave("t1.rtf")   # convert gtsummary() to gt() object to rtf file
```

Debugging R: Syntax, Logic, Data

Preventive is the best defensive to debugging, apply best practice methods. Common Issues are mis-spelling, typos, function and data capitalization, invalid parameters or nested functions and missing values. Confirm and view intermediate data frames.

R Packages, Functions & Parameters
Data Frame, Variable Name and Value Case-Sensitivity
R symbols: <-, (), [], [[]], '', ==, $, |, !, NA, %>%
help('R_function_name')

ERROR TYPE	SOLUTIONS
Invalid or Missing Packages, Path names, Libraries not Loaded	Load and confirm packages, path names and libraries
Invalid or Missing Data Frames, Objects or Variables	Confirm correct and existing data frames (instead of matrix), objects and vars, lower case all names since case-sensitive, correct order of tasks (select, filter, etc.) within DPLYR (SQL) functions, apply group_by() before summary functions to prevent overall summaries
Invalid or Missing Functions or Operations	Confirm functions exist and correctly applied, confirm variable and function types are consistent
Invalid or Missing Parameters and Options	Confirm correct function usage, case-sensitive, cut/paste working example
Invalid or Missing Data or Format	Confirm data import is correct, lower case data since case-sensitive, remove extra spaces before and after data values, confirm correct date format, apply factors to assign invalid data as NA, data by descriptive stats, freq counts, min, max, etc.

Invalid Logic	Confirm process logic flow, test and view inputs and outputs of each function
Subscript out of Bounds	Confirm correct starting, ending and incrementing values, apply dim() to check # of rows and columns
NA or Missing Values	Confirm NA and missing values, convert NA to missing values, confirm joins and appends, apply ignore missing values parameter options
Incorrect Variable Types	Apply conversion functions to convert to character, numeric or dates before using in functions, confirm correct date format
Incorrect Symbols	Apply correct symbols: Assignment (<-), Direct Row and Column Reference ([]), Operators (==, &, |, !), Piping (output of first function %>% is input to second function)
Unbalanced Quotes, Parentheses	Confirm balanced quotes (‘) and parentheses , [], ()
Invalid Custom Functions	Confirm correct function structure syntax, parameters and collection of R functions, test using simple data frames, variables and values

SDTMs: Metatools and SDTMChecks

Metatools – Required & Compliance Vars, Labels, Order, Sort, CT

```
install.packages("metatools")    # install metatools package
library(metatools)               # load metatools library

qcdm <- dm %>%                          # expects SDTM spec
 drop_unspec_vars(dm_spec) %>%  # keep need vars
 set_variable_labels(dm_spec) %>% # set var labels
 order_cols(dm_spec) %>%             # order vars per spec
 check_variables(dm_spec) %>%          # confirm all compliance vars
 check_ct_col(dm_spec, DOMAIN) %>% # check control terminology
 check_ct_col(dm_spec, AGEU)           # repeat for each var
```

SDTM Checks

```
install.packages("devtools")
# note that r-package-namespace-rlang-1.1.0 is required
devtools::install_github("pharmaverse/sdtmchecks", ref="main")
library(sdtmchecks)   # load library

ae = read_sas("path/to/ae.sas7bdat") # load ae dataset
ds = read_sas("path/to/ds.sas7bdat") # load ds dataset

# save results to myreport data frame
myreport=run_all_checks(metads = sdtmchecksmeta,
                # parameters with values
        priority = c("High", "Medium", "Low"),
                # subset checks based on priority
        type = c("ALL", "ONC", "COVID", "PRO", "OPHTH"),
                # subset checks based category
          verbose = TRUE)
class(myreport)                  # results in a list object
names(myreport)                  # each check result is saved in a slot of the list
myreport[["check_ae_aedecod"]] # investigate the results of a check
```

ADaMs: Admiral, Xportr?