These are the points which then have to be processed into a cheat sheet.

1. table(tit\_train$Title)#group by and count/length

select Title, count(\*) as cnt

from tit\_train

group by Title

1. table(tit\_train$Sex, tit\_train$Title) #equivalent to groupby with implicit count/length using a case statement

select Title

,sum (case when Sex='female' then 1 else 0 end) as 'female'

,count (case when Sex='female' then 1 else Null end) as 'female'--same but using count

,count (case when Sex='female' then 1 end) as 'female'--same but using count

,sum(case when Sex='male' then 1 else 0 end) as 'male'

from tit\_train

group by Title

1. tapply(tit\_train$Sex, tit\_train$Title, length) # table(Title) calculates count(\*) by applying ‘Title’ grouping to the dataframe. tapply calculates count(Sex) by applying ‘Title’ grouping to the dataframe (so it excludes null Sex column values). Tapply applies an aggregation to column(vector) in a dataframe.

select Title, count(Sex) as cnt

from tit\_train

group by Title

1. Regarding aggregation, the format of the results returned by above is not the same as by SQL. So, use aggregate for dataframe and the either merge or cbind. Here it is using merge:
   1. merge(x = aggregate(x = list(Age\_av = df$Age,

Wages\_av = df$Wages,

Productivity\_av = df$Productivity),

by = list(Company = df$Company),

FUN = mean),

y = aggregate(x = list(Education.University\_sum = df$Education.University),

by = list(Company = df$Company),

FUN = sum),

by = "Company")

* 1. OR you can convert the dataframe to data table and then do:

library(data.table)

setDT(df)[, .(average.Age = mean(Age),

average.Wages = mean(Wages),

sum.Education.University = sum(Education.University),

average.Productivity = mean(Productivity)),

by = Company]

1. update a column using a where clause

tit\_train$Title[tit\_train$Title == 'Mlle'] <- 'Miss'

tit\_train$Title[tit\_train$Title %in% rare\_title] <- 'Rare Title'

1. subset(tit\_train,subset = !is.na(tit\_train$Age)) # equivalent to filtering a table using where clause.
2. Projection or selecting some column (using column names) from a dataset:

subset(auto, select = -c(name))

if we were dropping just by using indexes, then this would have worked:

df[,-3]

1. Filtering a matrix (like on returned by cor) can be done in various ways:

auto = ISLR::Auto

mat = cor(auto\_sb)

mat>.4 # returns a matrix with True/False values in the matrix

mat[mat>.4] # this returns a vector of values satisfying the condition

subset(mat,mat>.4) # this returns a vector of values satisfying the condition

which(mat>.4, arr.ind = T) #This give a matrix with values being indexes satisfying the condition

mat[mat>.4] = 1 #i could also set the values like this which is equivalent to true/false.

mat[mat<=.4] = 0 #i could also set the values like this which is equivalent to true/false.

1. is.element(FALSE, is.na(Auto$year)) checks whether a value belongs to a set. In SQL one can use IN keyword to form a similar condition. Actually, in R as well there you have in:

FALSE %in% is.na(Auto$year)

1. Ordering and Top:

my.data.frame$ProductID[order(my.data.frame$"SUM(Units)",decreasing=TRUE)][1:5]

my.data.frame[order(my.data.frame$"SUM(Revenue)",decreasing=TRUE),]

1. Merge is SQL Join equivalent

Inner Join: merge(df1, df2) # joins the frames by common variable names

Inner Join: merge(df1, df2, by = "CustomerId")) # joins the frames by supplied variable names

Inner Join: merge(x=df1,y=df2, by.x=c("x\_col1","x\_col2"),by.y=c("y\_col1","y\_col2"))#join on multiple cols

Outer join: merge(x = df1, y = df2, by = "CustomerId", all = TRUE)

Left outer: merge(x = df1, y = df2, by = "CustomerId", all.x = TRUE)

Right outer: merge(x = df1, y = df2, by = "CustomerId", all.y = TRUE)

Cross join: merge(x = df1, y = df2, by = NULL)

1. T-SQL Lag window function has a somewhat equivalent in R called diff. Another way to do lag/lead is to use tidyverse (Can we do equivalent of window funcs in base R?):

;with CTE as

(

select \*,

cast(LowerDepth as float)- cast(LEAD(UpperDepth) over(partition by siteID order by siteID, LayerNo) as float) as diff

from CarbonSiteSampleMethod20210108\_2

)

select \*

from CTE

where diff>=0

Equivalent in R using tiyverse:

library(tidyverse)

site\_layers <- read.csv("C:/Users/sin17h/Downloads/CarbonSiteSampleMethod20210108\_2.csv")

site\_layers\_tb <- as\_tibble(site\_layers)

site\_layers\_tb\_proc <- site\_layers\_tb %>% group\_by(siteID) %>%

mutate(leadUpperDepth = with\_order(order\_by = LayerNo, fun = lead, x = UpperDepth)) %>%

arrange(siteID, leadUpperDepth)

site\_layers\_tb\_proc$diff = site\_layers\_tb\_proc$LowerDepth - site\_layers\_tb\_proc$leadUpperDepth

site\_layers\_tb\_proc\_filtered <- subset(site\_layers\_tb\_proc, site\_layers\_tb\_proc$diff>=0)

Someone suggested this in Base R but did not work:

site\_layers\_dt <- as.data.table(site\_layers)

site\_layers\_dt[,prev:=site\_layers\_dt[J(siteID,LayerNo-1), LowerDepth, mult='last']]

Generate random data to test joins:

library(data.table)

set.seed(1234)

n = 1e6

data\_frame\_1 = data.frame(id=paste("id\_", 1:n, sep=""),

factor1=sample(c("A", "B", "C"), n, replace=TRUE))

data\_frame\_2 = data.frame(id=sample(data\_frame\_1$id),

value1=rnorm(n))

data\_table\_1 = data.table(data\_frame\_1, key="id")

data\_table\_2 = data.table(data\_frame\_2, key="id")

system.time(df.merged <- merge(data\_frame\_1, data\_frame\_2))

# user system elapsed

# 17.983 0.189 18.063

system.time(dt.merged <- merge(data\_table\_1, data\_table\_2))

# user system elapsed

# 0.729 0.099 0.821