YOU CAN READ THE BELOW STUFF. !!! HOWEVER !!! THE TUTORIAL ON DPLYR FOUND HERE IS MUCH BETTER:

<https://github.com/beaunorgeot/dplyr-tutorial/blob/master/dplyr-tutorial.Rmd>

Best overall dplyr cheat sheet:<http://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf>

Cheatsheet for dplyr JOINS:<https://stat545-ubc.github.io/bit001_dplyr-cheatsheet.html>

Here’s another helpful one on Joins:<https://rstudio-pubs-static.s3.amazonaws.com/32935_c1ef8a2fe00746e1a3e1a838f2130e3a.html>

Hadley’s video tutorial:<http://datascience.la/hadley-wickhams-dplyr-tutorial-at-user-2014-part-1/>

part2:<http://datascience.la/hadley-wickhams-dplyr-tutorial-at-user-2014-part-2/>

Use mutate, ifelse to change values in a column. Replace

group\_by creates an “is a”. An entity with properties(a column variable or comibination of variables) you care about. All other remaining variables/columns in the data.frame become the “has a”. I could say a <Flight> “is a “ combination of Carrier and FlightNumber: group\_by(UniqueCarrier,FlightNum). Then a Flight “has a” Day, Destintation, ArrTime, etc

group\_by() doesn’t do anything by itself, it just modifies the verbs of future operations so that everything works on a per-group basis. (group\_by creates a list of indices containing the members of each group. You can see this in the -attr sections if you do str() on a data.frame that you have done a group\_by on)

group\_by() DOES NOT place all of the items in your group together in the data.frame, it doesn’t change the ordering at all, it just creates the indices. If you want to see all members of the groups together, use arrange(group\_name). However, arrange() is very computationally expensive, it requires copying the entire data.frame. If you’re working w/large data, avoid arrange() if possible.

Use the Window Function Lead/Lag() to look at changes over time

# Lead/Lag: change between observations over time (lag() gives the previous observation)

Compute What is the difference between the delay yesterday and today?

#Check was there a change? x != lag(x)

# What is % change? (x - lag(x))/x

# What was the fold-change? x/lag(x)

# Was the Previous observation false, and is the current observation now true? !lag(x) & x

order\_by= lets lag() know what the previous entry was. lag() will just go row-by-row and do these calculations if order\_by isn't specified. Here, we say the previous entry should be the previous DATE, not the previous row. Make sure DATE is properly formated as a date (posIX), mine isn't

remove na’s from a column w/o having to type na.rm =TRUE for each function

filter(!is.na(DepDelay))

#convert to factor

mutate\_each(funs(factor),ColumnName)

toy.df <- data.frame(originalVar = rep(c("A", "B", "C", "D", "E"), 10))

toy.df %>%mutate(newVar = factor(originalVar, labels = c("V", "W", "X", "Y", "Z")))

#Convert am variable to factor for better interpretations

mtcars$am <‐ as.factor(mtcars$am)

levels(mtcars$am) <‐ c("Automatic", "Manual")

factor(mtCarData$am, labels=c("automatic", "manual"))

rename columns/variables:

dataframe <- rename(dataframe, newName = oldName, otherNew = otherOld)

Filter using regex

filter(dataframe, grepl("pattern",columnNameToSearch))

filter(mtcars, grepl('Toyota|Mazda', type))

to do the other way around

filter(mtcars, !grepl('Toyota|Mazda', type))

#replace values in a column with mutate-replace

mutate(mpg=replace(mpg, cyl==4, NA) # here if Variable cyl==4, change the mpg value of that observation to be NA

#replace(columnName,oldValue,newValue)

filled <- activity %>%

group\_by(interval) %>%

mutate(steps = replace(steps, is.na(steps), mean(steps, na.rm =TRUE)))

#another replace method. See my ReproducibleResearch1.Rmd

#if weekday is Saturday or Sunday, replace with ‘weekend’, else replace with ‘weekday’

dayCategories <- dailyFilledSteps %>%

mutate(TypeOfDay=weekdays(as.Date(date))) %>%

mutate(TypeOfDay = ifelse(TypeOfDay == "Saturday" | TypeOfDay == "Sunday","Weekend","Weekday"))

**#Another example**

**vehics2 <- NEI %>%**

**filter(type == "ON-ROAD") %>%**

**filter(fips == "24510" | fips == "06037") %>%**

**mutate(fips = ifelse(fips == "24510", "Baltimore", "LA"))**

#NOTE, graphing pasted dates can be weird, but converting to POSIXct() seems to solve this problem.

mutate(DateTime = paste(as.Date(days2$Date,format = "%d/%m/%Y"), days2$Time)) %>%

mutate(posDT = as.POSIXct(DateTime))

#which flights were longest?

arrange(flights,desc(ArrTime-DepTime))

#create new variables based on other new variables

mutate(df, double = 2\*Value, quadruple = 2\*double)

#great way to view whole df as table in scrollable form to see all variables

View(df); OR View(arrange(df,desc(speed))

#Trick to transform 4 digit numbers into 2 sets of 2 digit numbers

mutate(flights, DepHours = DepTime %/% 100, DepMins = Dep %% 100) // turns 1423 into 14 & 23

%/% 100 takes all but the last 2 digits : it works left to right on a number

%% 100 takes the last 2 digits: it works right to left on a number

Special summarise() functions

n() # tells you how many observations were in a particular group (ie flights on day 1-2)

n\_distinct(x) # tells you how many different observations are in a variable.

sum(x >10) #How many values of x are >10

mean(x>10) # What proportion of x’s values are >10

#Often useful to do summaries of logical vectors b/c when you take a logical vector(boolean/T/F) and treat it like it’s a numeric(), all the Falses turn into 0’s and the Trues turn into 1’s.

-The sum(logical Vector) tells you how many T’s there were. mean(logical Vector) tells you the proportion of the total that were true.

It’s good to think in a series: group → filter → summarise

#Here’s a great command

f4 <- flights %>%

+ group\_by(Year,Month,DayofMonth) %>%

+ filter(!is.na(DepDelay)) %>%

+ summarise(med = median(DepDelay), mean =mean(DepDelay),max =max(DepDelay),q90= quantile(DepDelay, .9), proportionDelayed=mean(DepDelay > 0))

#####

reading through the above:

remove all na’s from the column of interest

group\_by date

q90 is 90th quantile. See mean(x>10) above for taking sums/means logicals

#####

What is the average hourly delay per day

hourly\_delay <- flights %>%

+ filter(!is.na(DepDelay)) %>%

+ group\_by(Year,Month,DayofMonth)%>%

+ summarise(delay=mean(DepDelay), number = n()) %>% #get avg delay & num flights for each day

+ filter(number > 10) #include only days that have more than 10 flights

#####

Which flights occur every day of the year, and where do they go

> daily <- flights %>%

+ group\_by(UniqueCarrier,FlightNum,Dest) %>%

+ mutate(DATE = paste(Year,Month,DayofMonth,sep = '\_')) %>%

+ summarise(num = n\_distinct(DATE)) %>%

+ arrange(desc(num))

#############

Which times of the day have the highest avg delays?

bob <- flights %>%

+ filter(Cancelled ==0)%>%

+ mutate(HOUR = DepTime %/% 100, MINS = DepTime %%100) %>% #see trick above to convert 24 hr time to Hours and Mins.

+ group\_by(HOUR,MINS) %>%

+ summarise(number = n(), AVGdepDelay = mean(DepDelay,na.rm=TRUE)) %>%

+ arrange(desc(AVGdepDelay))

#################

GROUP MUTATE/FILTER (Window Functions)

\*Creating new variables w/in a group is often useful

###

planes <- flights %>%

+ filter(!is.na(ArrDelay)) %>%

+ group\_by(TailNum) %>%

+ filter(n() > 30)

Give me the top 5, least delayed flight for each plane:

planes %>% filter(min\_rank(ArrDelay) <5)

####

The ranking functions basically just differ in how they handle ties

bob <- c(1,1,2,3)

min\_rank(bob) # produces positions 1,1,3,4 (tie for first, so ‘2’ gets 3rd place)

dense\_rank(bob) # no position lost. 1,1,2,3 (tie for first, but second place is still ranked 2nd)

row\_number(bob) #completely ignores ties. 1,2,3,4. Just gives you an ordering.

Good description of when to use which at 8:40:<http://datascience.la/hadley-wickhams-dplyr-tutorial-at-user-2014-part-2/>

dplyr: Manipulate and manage data in an easy and useful way. Can be used with base R, but also data.table() package and SQL interface for relational databases via the DBI package

library(dplyr)

6 Verbs:

1. select: return a subset of the columns
2. filter: extract a subset of rows based on logical conditions
3. arrange: re-order rows of a data frame based on the values of a column
4. mutate: create/add new variables/columns OR transform existing columns
5. rename: rename variables in a data.frame
6. groupby: split a data.frame according certain categorical variables
7. summarize/summarise: generate summary statistics of different variables, possibly within strata
   1. See example below for a standard 3 step process using mutate(), then groupby(), then summarize()

Arguments for all ‘verbs’:

1. The data.frame to operate on
2. All additional args are what to do to the df
3. returns a new df

In order for this to work df must be properly formated and annotated (correct annotation of factor levels for example, or making sure that variable names are all there)

chicago <- readRDS("chicago.rds")

dim(chicago)

str(chicago)

SELECT FUNCTION

#you can select a range of columns by name instead of number

head(select(chicago, city:dptp))

# select all columns EXCEPT the ones in this range

head(select(chicago, -(city:dptp)))

FILTER FUNCTION

#take all rows that value for pm25etc >30 & other conditions

chic.f <- filter(chicago, pm25tmean2 > 30 & tmpd > 80)

ARRANGE FUNCTION

#Arrange df by order of the date column

chicago <- arrange(chicago, date)

# Arrange in descending order

chicago <- arrange(chicago, desc(date))

RENAME

#rename(df, newName = oldName, youCanDoThis = forManyColAtAllTime)

chicago <- rename(chicago, dewpoint = dptp, pm25 = pm25tmean2)

MUTATE

# mutate(df, newVarName=someFunction/operationOnAnotherVar)

chicago <- mutate(chicago,pm25detrend=pm25-mean(pm25, na.rm=TRUE))

#look at the first few rows of last operation, but only look at relevant columns

**head(select(chicago, pm25, pm25detrend))**

#1.create tempature category variable to indicate whether a given day was hot or cold based on whether the temp was above 80 or not. The labeling is tricky, here’s what I think is happening: F, for False, comes before T, for true alphabetically. So all temps greater than 80 will be assigned the standard level of 1. The 1st level is then labeled cold. I don’t know why multiplication by one is necessary. NOT NECESSARY SEE BOTTOM

**chicago <- mutate(chicago,tempcat = factor(1 \* (tmpd > 80),labels = c("cold", "hot"))**

#use groupby to create new data structure based on the original data frame and the tempcat variable

**hotcold <- group\_by(chicago, tempcat)**

#what is mean pm25, for both hot and cold days. what is the max ozone for both hot & cold days. what is the median no2 for hot & cold days. Add na.rm=T if getting NAs for mean() or any other function.

**summarize(hotcold, pm25 = mean(pm25, na.rm = TRUE),o3 = max(o3tmean2),**

**no2 = median(no2tmean2))**

#create a summary for each year in the dataset

#1. create a year variable using POSIXlt()

**chicago <- mutate(chicago,year = as.POSIXlt(date)$year + 1900)**

#2. group the data set by year

**years <- group\_by(chicago, year)**

#3.use summary() to get info about the pollutant vars

**summarize(years, pm25 = mean(pm25, na.rm = TRUE), o3 = max(o3tmean2, na.rm = TRUE),no2 = median(no2tmean2, na.rm = TRUE))**

Using dplyr You can feed a dataset through a pipeline to create a new dataset, using %>% (which we can call the pipeline operator) to chain operations. When you do this, you provide the input dataset outfront, and you do not supply it to mutate,groupby and summarize.

#this does the same thing as above, except with a ‘month’ variable

**chicago %>% mutate(month = as.POSIXlt(date)$mon + 1)%>% group\_by(month)**

**%>% summarize(pm25 = mean(pm25, na.rm = TRUE),o3 = max(o3tmean2, na.rm = TRUE),no2 = median(no2tmean2, na.rm = TRUE))**

JOINS w/dplyr

\*Non dplyr method:

#dtFinal <- merge(table1,table2,by=c(“columnToMergeBy”)

dtF <- merge(GDPdt, EDdt, all = TRUE, by = c("CountryCode"))

You Can create binary factors without that stupid 1\*(stuff)

> str(df)

'data.frame': 3 obs. of 4 variables:

$ n: Factor w/ 2 levels "FALSE","TRUE": 1 2 2

$ s: Factor w/ 3 levels "aa","bb","cc": 1 2 3

$ b: Factor w/ 2 levels "FALSE","TRUE": 2 1 2

$ e: num 4 5 6

> df$e <- factor((df$e > 4), labels=c("low", "high") )

> str(df)

'data.frame': 3 obs. of 4 variables:

$ n: Factor w/ 2 levels "FALSE","TRUE": 1 2 2

$ s: Factor w/ 3 levels "aa","bb","cc": 1 2 3

$ b: Factor w/ 2 levels "FALSE","TRUE": 2 1 2

$ e: Factor w/ 2 levels "low","high": 1 2 2

EXERCISES:

Question1

Load the Gross Domestic Product data for the 190 ranked countries in this data set:

https://d396qusza40orc.cloudfront.net/getdata%2Fdata%2FGDP.csv

Load the educational data from this data set:

https://d396qusza40orc.cloudfront.net/getdata%2Fdata%2FEDSTATS\_Country.csv

Match the data based on the country shortcode. How many of the IDs match? Sort the data frame in descending order by GDP rank (so United States is last). What is the 13th country in the resulting data frame?

\*Download the files

dtGDP <- dtGDP[X != ""]

dtGDP <- dtGDP[, list(X, X.1, X.3, X.4)]

setnames(dtGDP, c("X", "X.1", "X.3", "X.4"), c("CountryCode", "rankingGDP",

"Long.Name", "gdp"))

dtEd <- data.table(read.csv(f))

#non-plyr method

#dt <- merge(dtGDP, dtEd, all = TRUE, by = c("CountryCode"))

#dplyr method

dt1 <- left\_join(dtGDP,dtED, by= c("CountryCode"))

sum(!is.na(unique(dt$rankingGDP)))

library(dplyr)

dt1 <- arrange(dt,desc(rankingGDP))

dt1[13]

Question 4

What is the average GDP ranking for the "High income: OECD" and "High income: nonOECD" group?

**dt1[,.(Mymean=mean(rankingGDP,na.rm=TRUE)),by=Income.Group]**

Question 5

Cut the GDP ranking into 5 separate quantile groups. Make a table versus Income.Group. How many countries are Lower middle income but among the 38 nations with highest GDP?

library(Hmisc)

**dt$GDPgroups = cut2(dt$rankingGDP,g=5)**

**table(dt$GDPgroups,dt$Income.Group)**

This solution came from the “Summarize,Explore,make New Variables” page in ‘Most Useful’