WEEK2 LAB SUMMARY:

1. *Pch=* means PLOTS CHARACTER for points. By default, R plots points with open circles, but many other symbols are available. Ex. Use pch=16, stand for “closed circles”.
2. par(mfrow=c( , ))means how would you places your plots. Ex. 2,2 or 1,3.
3. legend(x = 3, y = 1000, col = 1:3, pch = c(NA, NA, 1), lty = c(1, 2, NA), c("Volume","Area", "Surface Area"))

Explain: x,y means the coordinate you want to put your legend. NA means omits points or lines. You could also use words for coordinate, like x= “topleft”.

PS: When you put the names for legend, use “ ”.

1. When you don’t put type for lines or points, the default type is 1.
2. Read csv.file, use : Ex. poke = read.csv(file = "pokemon\_2019.csv",

header = TRUE, sep = ",")

1. Get the variables names: names()
2. Rotated the plot using horiz=TRUE and rotated the labels using las=2 Ex:

plot(poke[, "Type\_1"], horiz = TRUE, las = 2)

1. Poke$HP >200 : give you T/F for every rows.

Poke[Poke$HP >200, “HP”] give you the all the HP values which are greater than 200

Poke[poke$HP >200, ] give you all the rows which poke’s hp is greater than 200

1. Which() : give you the data in exact # row number.
2. poke[poke$Height\_m>2 & poke$isLegendary=="True","Name"] : Select certain criterias to extract data from data frame
3. if you wanna create a new row with variable names: test = poke[1,]\*NA :pick the first row
4. attach this new row of data to the original poke data.frame by binding it as a new row using rbind. You could eventually bind together some other variable as a column using cbind. Ex: pokemonextra = rbind(pokenew, poke)
5. Convert from strings into factors: ex:

pokemonextra[,"Type\_1"] = factor(pokemonextra[,"Type\_1"])

1. Levels() function: gives back how many disctinct variables in the factor.
2. Legend ex: legend(x=25,y=225,col = 1:18,pch = 1:18, unique(poke$Type\_1),cex = 0.5)
3. poke[,"Attack"] ~ poke[,"Type\_1"] : tells R to split the variable "Attack" by "Type 1". EX: boxplot(poke[, "Attack"] ~ poke[, "Type\_1"], main = "Pokemon Attacks by Type\_1", las = 2)
4. How to sort a dataframe: use order(): ex: canadasort = canada[order(canada$TotalNumber),] default is ascending use – sign to make it descending.

Week3LabSQL:

INSTALL SQL PACKAGES:

install.packages("RSQLite")

library(RSQLite)

1. Connection to a database: dbcon = dbConnect(SQLite(), dbname = "stat240Week3lab.sqlite")
2. List tables from databases: Ex: dbListTables(dbcon)
3. Get at the column headings of the data table: names(dbReadTable(dbcon, "CA")) and head().
4. sql\_qry = "SELECT \* FROM zip WHERE Latitude>55"

dbGetQuery(dbcon, sql\_qry)

Explain: the \* means 'select everything', but this could be replaced with speci\_c column names. FROM tells you which table to use. WHERE says which speci\_c pieces of data you want.

1. Merging Tables: In SQL there are four types of merging tables. Here we will be using the

simplest method which is INNER JOIN. The SQL INNER JOIN returns all rows from multiple tables where the join condition is met.

Ex: "SELECT \* FROM A INNER JOIN B ON A.X=B.X"

Ps: If you want to extract specific information from the merged data table. Use: WHERE

1. Extract Text info in SQL:

Ex: "SELECT \* FROM POP2011 WHERE Geographic\_name LIKE V5% ".

Note that the % symbol acts like a wildcard in that it allows any other text to appear after V5. By not including it before the V5 means we are only interested in text that begins with "V5".

1. table(): to tally the number of occurrences of certain column.
2. Include a horizontal line: abline( h = sth.)
3. Mapping: Ex: how to get North America Map:

worldmap = getMap(resolution = “high”)

nrthAm = worldmap[worldmap$REGION == “North America”, ]

plot(NrthAm, col = “white”, bg= “lightblue”,xlim = c(-140,-55), ylim = c(55,60))

1. Making boxplot about density:

Ex: boxplot(join4$Pop\_2006 / join4$Total\_privateDwellings ~ join4$prov\_acr, las=2, xlab = “”,ylab=””, main=””)

Week4LabSummary:

1. connect to a database. First install the package and add libraries and then

dbcon = dbConnect(SQLite(), dbname = "stat240Week4lab2019.sqlite")

1. when the database is too large, we don’t want to load the entire table into R. We use PRAGMA to get table info.

query\_table\_info = "PRAGMA table\_info(TableNameGoesHere)"

dbGetQuery(dbcon, query\_table\_info)

1. SQL Query Distinct: The DISTINCT keyword can be used to return only distinct/unique values from a SQL database.

sql\_dstc = "SELECT DISTINCT year, place FROM WinterO"

1. dbGetQuery : send a SQL query, retrieves results. This function is for SELECT queries only.
2. Order By: To clarify we could use the command ORDER BY to sort the

result-set by one or more columns.(Without DESC, Order by ascending)

sql\_ord = "SELECT DISTINCT year,place FROM WinterO ORDER BY place DESC"

dbGetQuery(dbcon, sql\_ord)

1. dbSendQuery and dbFetch: this function sumits SQL query but it does not extract any records. For that you need to use *dbFetch* , then you must call dbClearResult to clear the results when you finish fetching the records you need.

sql\_pop = "SELECT Population\_\_2011, Region FROM CA INNER JOIN

POP2011 ON CA.Geographic\_name=POP2011.Geographic\_name WHERE

province == Saskatchewan"

QuerryOut = dbSendQuery(dbcon, sql\_pop)

dbFetch(QuerryOut, 5)

To get all remaining rows set the number to -1.

Remember to clean up the query: dbClearResult().

1. write a query to extract and count the number of times each pokemon appears:

"SELECT name, count(name) FROM Vanpoke GROUP BY name"

sql\_poke2 = "SELECT DISTINCT moveset FROM Vanpoke WHERE name IN ('Hitmoncha

n','Lickitung','Nidoqueen','Primeape','Sharpedo','Victreebel') ORDER BY move

set"

1. SQL Query INSERT INTO: INSERT INTO statement is used to insert new records in a table. For this statement, after write the statement, you also use dbSendQuery to work.

Ex: "INSERT INTO WinterO (place, year, Country, Medal, Event, Athlete,Record) VALUES ('Sochi','2014','Canada','Silver','ice dancing mixed','VIRTUE Tessa / MOIR Scott', ’’ ’’) "

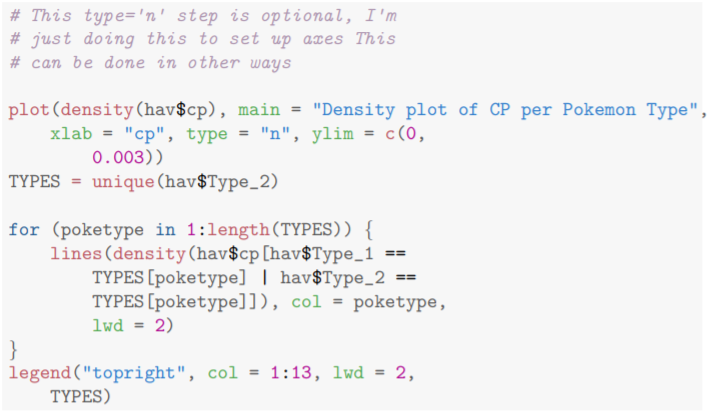
dbSendQuery(dbcon,sql\_insSecond)

1. DeleteRows in SQL: sql\_del = "DELETE FROM WinterO WHERE Athlete =='Zhifan Shen'"

dbsendQuery(dbcon, sql\_del)

1. Produce a new density plot for the cp of pokemon from all types:





1. *When you forget how to use the function, use ? to check it out.*
2. *Attach(file) let you to get aceess to colume title directly attach only apply on data frame*
3. *Tail( file, 2)*
4. *Summary() : for factor , it will count.*
5. table(Country) # Tally the number of occurrences after using the attach, so you could use country as the column name directly.
6. Detach() : Make sure which attach you want to work on.
7. ***Apply*** *will run any functions .ex:* **apply(exchanges[,4:6],1,sum)** Apply the function sum along dimension #1of the data.frame (or matrix)
8. *Aggregate funtion : putting things together.*
9. na.rm = ignore NA values. (na.rm = T)
10. *world map ! :* here = mapCountryData(spdf, nameColumnToPlot="Count of # Schools and Terms Availble for Exchange \n Apply by Jan 25 Application Deadline", catMethod="fixedWidth",numCats = 24,lwd=1)

*AddingTheLegend:*do.call(addMapLegend,c(here,sigFigs=2,legendLabels="all",legendIntervals="page"))

*Week 4*

(QuerryOut = dbSendQuery(dbcon, sql\_poke))

• dbFetch(QuerryOut, 5)

• dbFetch(QuerryOut, -1) -1 means pull up anything else.

• dbClearResult(QuerryOut) you should clear the command every time.

sql\_poke = "SELECT Name,Type\_1,Type\_2,Generation,isLegendary, Attack FROM Pokem **ORDER BY Attack"**

DESC： FOR DECENDING. YOU COULD USE IT AFTER THE ORDER BY. Ex. ORDER BY Attack DESC".

sql\_poke = "SELECT Name,Type\_1,Type\_2,Generation,isLegendary, Attack FROM Pokem WHERE Type\_1 IN ('Flying','Ground') ORDER BY Attack DESC" You should put where in front of ORDER.

Making a new row: sql\_ins ="**INSERT INTO** Pokem ( Name, Type\_1, Type\_2, isLegendary, Attack) VALUES ( 'Statasaur', 'Normal','Flying', 'True', '314')"

• dbSendQuery(dbcon, sql\_ins)

You can do it together: dbFetch(dbSendQuery(dbcon, sql\_poke),5)

Kernel Density Estimator:

cheap hack version == histogram

indicator function. Ex. x = c(1,2,3,4,5), a = 3, b=5, I (3,5] (x) = ( 0,0,0,1,1) and sum is 2

density(output$Attack)

• plot(density(output$Attack))

WEEK 5

* (Correct way to) count the number of Pokemon per generation:

• sql\_poke = "SELECT Generation, count(Generation) AS NumberPerG FROM Pokem **GROUP BY** Generation"

* If you want to get decimal number, 🡪 SUM(Attack)/(COUNT(Generation)\*1.0) COUNT() produces an integer so the result must be an integer. Multiplying by 1.0 converts it to a Double and allows the result to be a double.
* **initExtension**: Give u more functions to use in SQL.
* How, using SQL do we get the unique types without the type combinations?

• UNION stacks (and sorts) the values from two queries (duplicates are removed)

• sql\_poke = "SELECT Type\_1 from Pokem UNION SELECT isLegendary FROM Pokem"

UNION ALL : duplicates are kept.

* 2D Density:

points(poke$longitude,poke$latitude,pch='.') this is not 2D.

• est2 = kde2d(poke$longitude,poke$latitude,n = c(121,150))

• contour(est2, add=TRUE,col=2,lwd=3)

* VIEW: A VIEW is a virtual table in the database. Create a query as a VIEW and then call a new query on that VIEW.

Ex:sql\_poke="CREATE VIEW pokemean(the table name) AS SELECT Generation, SUM(Attack)/COUNT(Generation) AS myavg, AVG(Attack) AS AvgAttacks FROM Pokem GROUP BY Generation"

dbSendQuery(dbcon, sql\_poke) Send this, not get.(U will get a new table from your database)

dbSendQuery(dbcon, "drop view pokemean") (delete the table you create)

WITH : lets you define a temporary table. It disappears after the query is called. (Different from VIEW)

Ex: sql\_poke = "WITH pokestuff AS (SELECT Generation, SUM(Attack)/COUNT(Generation) AS myavg, AVG(Attack) AS AvgAttacks, STDEV(Attack) AS stdev FROM Pokem) SELECT Pokem.Attack, pokestuff.AvgAttacks , Pokem.Attackpokestuff.AvgAttacks AS Resids FROM pokestuff, Pokem" ( Here, u may use . instead of $)

Week6:

More SQL\_ish:

pokeLegGen = subset(poke, select = c(Legendary, Generation)) //Selecting a few column

#adding conditions:

pokeLegGenFire = subset(poke, Type.1=="Fire"| Type.2=="Fire",select = c(Legendary, Generation))

summary()

Unique() /unique combination

Tweeter! :

* SSC = getUser("SSC\_stat")

?user

SSC$description

SSC$followersCount

SSC$statusesCount

SSCTweets = userTimeline("SSC\_stat",n=3000)

* VancouverSnow = searchTwitter(searchString = 'VancouverSnow', n = 5000, lang = "en") //for #
* #Get our location in terms of a twitter location id:

location = closestTrendLocations(lat = 49.278603, -122.917419)

#Get local trends:

TrendsNow = getTrends(location$woeid)

* SnowMageddon2019 = searchTwitter(searchString = 'SnowMageddon2019', n = 5000, lang = "en")

• #Consider the date information:

• #Make a data frame

• SnowMageddon2019DF = twListToDF(SnowMageddon2019)

• SnowMageddon2019DF2 = subset(SnowMageddon2019DF, select = c(text, favoriteCount, created, screenName, isRetweet, retweetCount))

* Tweeter timing:

table(weekdays(SnowMageddon2019DF2[,"created"]))

* head(grammysDF2$text)

• #split into words by splitting a string: strsplit

• grammys\_words = strsplit(grammysDF2$text, " ")

grammys\_words\_nolist= unlist(grammys\_words) // unlist

* #Counting popular words

grammys\_words\_tab = table(grammys\_words\_nolist) ////Counting

////#Sorted

barplot(sort(grammys\_words\_tab,decreasing=TRUE))

barplot(sort(grammys\_words\_tab,decreasing=FALSE))

* # produce a word cloud

• library(wordcloud)

•wordcloud(names(grammys\_words\_tab),grammys\_words\_tab,min.freq=50,colors=rainbow(8))