Week 4 more SQL (Ch7) and Kernel Density plots

Stat 240
Dr Dave Campbell

Data Science in the news

FEB 16, 2012 @ 11:02 AM 3,104,129 VIEWS

How Target Figured Out A Teen Girl Was Pregnant Before Her Father Did

Kashmir Hill, FORBES STAFF
Welcome to The Not-So Private Parts where technology & privacy collide FULL BIO

Every time you go shopping, you share intimate details about your consumption patterns with retailers. And many of those retailers are studying those details to figure out what you like, what you need, and which coupons are most likely to make you happy. Target TGT +0.24%, for example, has figured out how to data-mine its way into your womb, to figure out whether you have a baby on the way long before you need to start buying diapers.

Charles Duhigg outlines in the New York Times how Target tries to hook

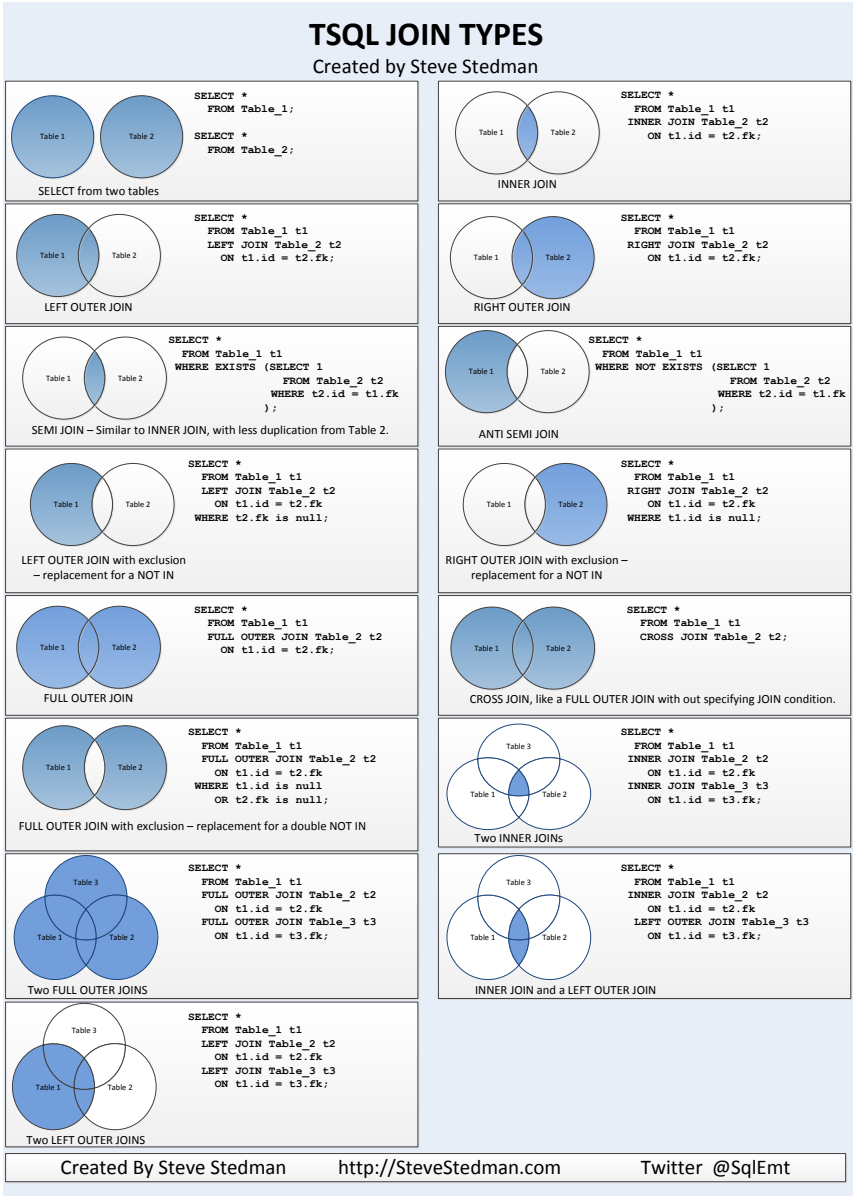
http://www.forbes.com/sites/kashmirhill/2012/02/16/how-target-figured-out-a-teen-girl-was-pregnant-before-her-father-did/

or using the link they want me to use:

http://www.forbes.com/sites/kashmirhill/2012/02/16/how-target-figured-out-a-teen-girl-was-pregnant-before-her-father-did/#b690b3634c62

SQL Join types

http://stevestedman.com/wp-content/uploads/VennDiagram1.pdf

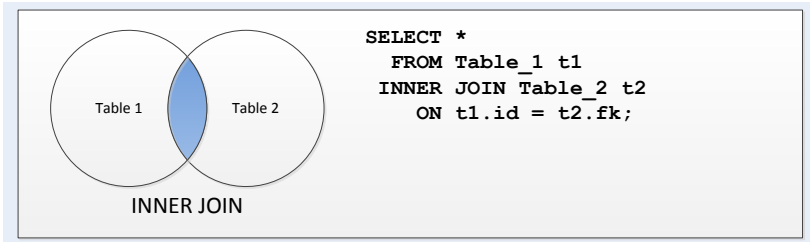


```
db_CA = dbConnect(RSQLite::SQLite(),
dbname="stat240Week4lab2019.sqlite")
```

The CA table is just postal codes. <— Table_1

The POP2006 has a ‘Canada’ entry <— Table_2

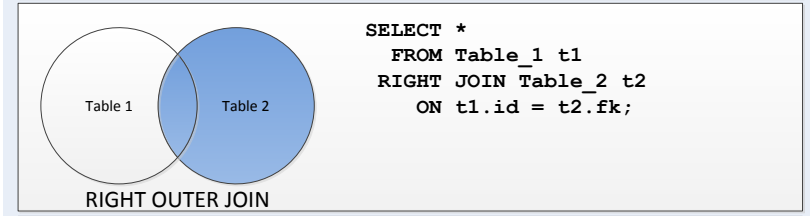
No Canada->




```
SELECT *
  FROM Table_1 t1
 INNER JOIN Table_2 t2
    ON t1.id = t2.fk;
```

INNER JOIN

Canada—>




```
SELECT *
  FROM Table_1 t1
 RIGHT JOIN Table_2 t2
    ON t1.id = t2.fk;
```

~~RIGHT JOIN~~

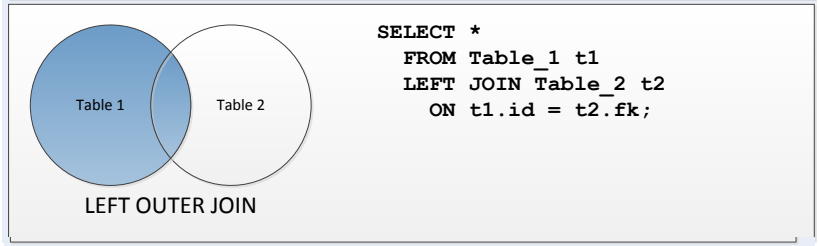
Without Specifying JOIN condition

Canada—>


```
SELECT *
  FROM Table_1 t1
  CROSS JOIN Table_2 t2;
```

CROSS JOIN

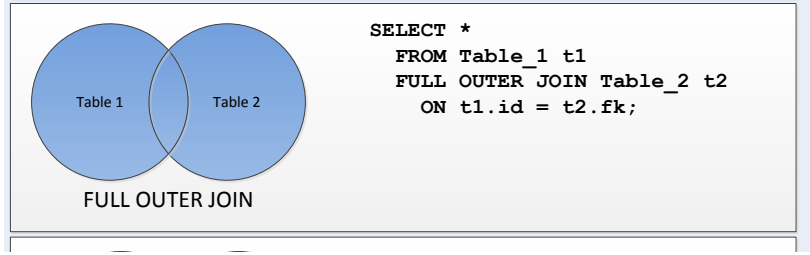
No Canada->




```
SELECT *
  FROM Table_1 t1
  LEFT JOIN Table_2 t2
    ON t1.id = t2.fk;
```

LEFT JOIN

Canada->




```
SELECT *
  FROM Table_1 t1
  FULL OUTER JOIN Table_2 t2
    ON t1.id = t2.fk;
```

~~FULL OUTER JOIN~~

Google the error messages

make a sheet with code

dbsendquery and dbgetquerry

SQL in real life

Previous week we loaded a table into R then used library(DBI) to produce the database. Then we accessed it using library(RSQLite)

We also loaded the entire database then used R commands like **head** and **aggregate**

Typically you can't load the database into memory.

Table info without loading the entire table into memory

```
dbcon = dbConnect(SQLite(),
dbname="stat240Week3labClass.sqlite")

dbListTables(dbcon)

query_table_info = "PRAGMA table_info('Pokem')"

dbGetQuery(dbcon,query_table_info)
```

(Note no “.” in variable names)

PRAGMA table_info

	cid	name	type	notnull	dflt_value	pk
Actual column location	index starting from 0	variable name	number, text,…	can it be NULL?	Default value	relates to primary key - i.e. beyond STAT 240

Run Query don’t return the results: dbSendQuery
Run Query do return the results: dbGetQuery

Expecting a lot of output? Don’t load it into memory:

dbGetQuery

sql_poke = "SELECT Name,Type_1,Type_2,Generation,isLegendary FROM Pokem"

(QuerryOut = dbGetQuery(dbcon, sql_poke))

Run Query don't return the results: dbSendQuery
Run Query do return the results: dbGetQuery

Expecting a lot of output? Don't load it into memory:

dbGetQuery

sql_poke = "SELECT Name,Type_1,Type_2,Generation,isLegendary
FROM Pokem"

(QuerryOut = **dbSendQuery**(dbcon, sql_poke))

dbFetch(QuerryOut, 5)

dbFetch(QuerryOut, -1)

dbClearResult(QuerryOut)

ORDER BY Results

Expecting a lot of output? Don't load it into memory:

dbGetQuery

sql_poke = "SELECT Name,Type_1,Type_2,Generation,isLegendary, Attack FROM Pokem **ORDER BY Attack**"

QueryOut = dbSendQuery(dbcon, sql_poke)

dbFetch(QueryOut, 5)

dbClearResult(QueryOut)

ORDER BY Results

Expecting a lot of output? Don’t load it into memory:

dbGetQuery

sql_poke = "SELECT Name,Type_1,Type_2,Generation,isLegendary, Attack FROM Pokem **ORDER BY Attack DESC**"

QueryOut = dbSendQuery(dbcon, sql_poke)

dbFetch(QueryOut, 5)

dbClearResult(QueryOut)

Limit output…(broken)

Expecting a lot of output? Don't load it into memory:

dbGetQuery

sql_poke = "SELECT Name,Type_1,Type_2,Generation,isLegendary, Attack FROM Pokem **ORDER BY Attack DESC WHERE Type_1 IN ('Flying','Ground')**"

QueryOut = dbSendQuery(dbcon, sql_poke)

dbFetch(QueryOut, 5)

Limit output..(Fixed!)

Expecting a lot of output? Don’t load it into memory:

dbGetQuery

sql_poke = "SELECT Name,Type_1,Type_2,Generation,isLegendary,
Attack FROM Pokem **WHERE Type_1 IN ('Flying','Ground')**
ORDER BY Attack DESC"

dbFetch(dbSendQuery(dbcon, sql_poke),5)

QueryOut = dbSendQuery(dbcon, sql_poke)

dbFetch(QueryOut, 5)

INSERT new row using SQL

Lab week 2 within R workflow: Bring entire table into memory, duplicate a row of the data frame, fill in values, bind it to the data frame, return entire table to database

Within SQL: fill in values of new row in the table

Check the default values. Then make a new row filling in what you know

```
dbGetQuery(dbcon,query_table_info)

sql_ins ="INSERT INTO Pokem  ( Name, Type_1, Type_2, isLegendary, Attack)

                                VALUES ( 'Statasaur', 'Normal','Flying', 'True', '314')"

dbSendQuery(dbcon, sql_ins)
```

Delete a row.
WARNING: GENERALLY A BAD IDEA

```
sql_del ="DELETE FROM Pokem WHERE Name
=='Statasaur'"

dbSendQuery(dbcon, sql_del)

sql_poke = "SELECT
Name,Type_1,Type_2,Generation,isLegendary, Attack FROM
Pokem WHERE Attack>300 ORDER BY Total DESC"

dbFetch(dbSendQuery(dbcon, sql_poke),5)
```

Data Science in the news

https://ca-political.com

http://www.spectator.co.uk/2016/12/the-british-data-crunchers-who-say-they-helped-donald-trump-to-win/

personality tests completed by Americans — online, by phone, or because someone with a clipboard approached them while they were shopping. You rate yourself against 120 statements — I dislike myself; I'm the life of the party; I make rash decisions — and are graded on what psychologists call the 'big five' personality traits: openness, conscientiousness, agree-ableness, extroversion, neuroticism.

'The impressive bit,' says Nix, is to expand the findings from those who took the personality tests to the entire American electorate of 230 million. They can do this because Cambridge Analytica also has '4,000–5,000 data points' — pieces of information — on every single adult in the US. This can be anything from age, gender and ethnicity to what magazines they buy, which TV programmes they watch, the food they eat, the cars they drive, even the golf clubs they belong to. This is indeed impressive — and a little bit creepy. Regardless, the data is for sale; Cambridge Analytica take it and (they have persuaded their clients) spin it into gold. There are two assumptions: first that people who buy the same things and have the same habits — the same 'data points' — have similar personalities; secondly that your personality will help predict, say, whether you go for Coke or Pepsi, Clinton or Trump. 'Behaviour is driven by personality,' Nix said.

CA Political

CA political has been provided the stronger relationship between data and marketing. Basically, it combines the predictive data analytics, behavioural sciences, and innovative ad tech into one award winning approach. With the use of CA political, you can engage the customers more effectively and efficiently. Recently, CA-political.com online service has received awards like Global Periodical Publisher and Employee Benefits Provider. CA-political is a data-driven marketing solution and it works together with the companies like SCL Group Limited, Cambridge Analytica Limited, SCL Elections Limited, SCL political Limited, and SCL Social Limited in order to deliver best solutions to its clients.

Essentially, CA political is providing an advantage of improving the brand's marketing effectiveness by influencing the customer behaviour with the use of behavioural sciences, predictive analytics, data driven solutions, and other technological benefits.

Achievements

CA-political has up to 5000 data points on over 230 million individual American consumers. By embedding with these data instruments, you can add to your own customer data and gain the benefit of establishing custom target audiences which lets to engage and motivate the individuals.

Efficiency and Transparency

Recent Posts

- The data we used on the 2016 US presidential campaign
- Cambridge Analytica responds to ICO comments
- Cambridge Analytica responds to false allegations in the media
- CA Congratulates Donald Trump and Mike Pence
- Message from acting CEO Dr. Alexander Tayler
- Cambridge Analytica responds to Facebook announcement
- Cambridge Analytica responds to use of entrapment and mischaracterization by Channel 4 News
- Data-driven services
- Cambridge Analytica responds to committee hearing
- CA responds to announcement that GSR dataset potentially contained 87 million records

Descriptive Statistics

5 number summary

Mean and Variance

Boxplots

Histograms

Density Estimation

Kernel Density Estimator

cheap hack version == histogram

Kernel Density Estimator

Histogram:

Count elements within a small interval

Apply to many distinct neighbouring intervals

Mathematically:

We need an indicator function

Kernel Density Estimator

Indicator Function:

$$I_{(a,b]}(x) = \begin{cases} 0 & x \leq a \\ 1 & a < x \leq b \\ 0 & b < x \end{cases}$$

x<a & x<=b

Kernel Density Estimator

Indicator Function:

$$I_{(a,b]}(x) = \begin{cases} 0 & x \leq a \\ 1 & a < x \leq b \\ 0 & b < x \end{cases}$$

x = c(1,2,3,4,5), a = 1, b=3

$$I_{(3,5]}(x) = ?$$

Kernel Density Estimator

Histogram bin height in the interval (a,b) =

$$\sum_x I_{(a,b]}(x)$$

sum(x<a & x<=b)

x = c(1,2,3,4,5), a = 1, b=3

$$\sum_x I_{(2,4]}(x)$$

```
IndicatorSum = function(x,a,b){
      #lower bound a
      #upper bound b
      #Count the number of x values
      #which are inside the (a,b] bounds
      out = rep(0,length(a))
      for(index in 1:length(a)){
        out[index]=sum(x>a[index] & x<=b[index])
      }
      return(out)
}
```

Kernel Density Estimator

Histogram bin height in the interval (a,b) =

$$\sum_x I_{(a,b]}(x)$$

x = rnorm(1000,0,1)

lowvalue = floor(min(x))

highvalue = ceiling(max(x))

width2use = 1

BinEdges = seq(from = lowvalue, to = highvalue, by = width2use)

Kernel Density Estimator

Histogram bin height in the interval (a,b) = $\sum_x I_{(a,b]}(x)$

a = BinEdges[-length(BinEdges)]

b = BinEdges[-1]

height = IndicatorSum(x,a,b)

plot((a+b)/2,height)

```
height = IndicatorSum(x,a,b)

hist(x,breaks = BinEdges)

points(apply(cbind(a,b),1,mean),height,
            pch = '*',cex = 10)
```

Special Case

table(floor(x))

Kernel Density, edge shift-.2

lowvalue = floor(min(x))

highvalue = ceiling(max(x))

width2use = 1

BinEdges = seq(from = lowvalue-.2, to = 1+highvalue-.2, by = width2use)

a = BinEdges[-length(BinEdges)]

b = BinEdges[-1]

height = IndicatorSum(x,a,b)

points((a+b)/2,height,pch='@',cex = 2)

Kernel Density, edge shift+.2

lowvalue = floor(min(x))

highvalue = ceiling(max(x))

width2use = 1

BinEdges = seq(from = lowvalue-1+.2, to = 1+highvalue+.2, by = width2use)

a = BinEdges[-length(BinEdges)]

b = BinEdges[-1]

height = IndicatorSum(x,a,b)

points((a+b)/2,height,pch='^',cex = 2)

Kernel Density Estimates:

Improvements:

Replace indicator with a **weight** function (generally Normal pdf)

```
y=seq(-4,4,length=1000)
plot(y,dnorm(y),type='l',xlab = 'y',ylab='normal pdf',main = 'Weight Function',ylim=c(0,1))
lines(y, y>0&y<=1)
```

Smoothly shift bin edges across entire interval and use weight function to find the density at the interval midpoint.

Kernel Density Estimates: density(output$Attack)

density(output$Attack)

?density

plot(density(output$Attack,bw,…))

About that Weight Function

$$I_{(a,b]}(x) = \begin{cases} 0 & x \leq a \\ 1 & a < x \leq b \\ 0 & b < x \end{cases}$$

Width of interval (bandwidth ‘bw’) makes a huge difference

How do we plot Attack Distribution within each Generation?

Tell me about the distribution of Attacks:

```
sql_attacks = "SELECT Name,Generation,ATTACK FROM Pokem"

output = dbGetQuery(dbcon, sql_attacks)

boxplot(output$Attack~output$Generation,las=2,xlab="Generation",main= "Attacks by Generation")

par(mfrow=c(2,3)); for( lp in 1:6){ hist(output$Attack[output$Generation==lp])}

par(mfrow=c(2,3)); for( lp in 1:6)
{ plot(density(output$Attack[output$Generation==lp]),main= paste("Generation",lp))}

par(mfrow=c(1,1)); plot(density(output$Attack),type='n',main= "Attacks by Generation");     for( lp in 1:6)
{lines(density(output$Attack[output$Generation==lp]),col=lp,lwd=2)};
legend("topright",paste("Generation",1:6),col=1:6,lwd=2)
```

Using a common bandwidth

```
par(mfrow=c(1,1));

plot(density(output$Attack),type='n',main= "Attacks by
Generation",xlab="Attacks");

bandw = density(output$Attack)

for( gen in 1:6){

        lines(density(output$Attack[output$Generation==gen],
        bw= bandw$bw),  col=gen, lwd=2)

};

legend("topright",paste("Generation",1:6),col=1:6,lwd=2)
```

Unknown number of Generations

```
par(mfrow=c(1,1));

plot(density(output$Attack),type='n',main= "Attacks by Generation",xlab="Attacks");

bandw = density(output$Attack)

UniqueGens = unique(output$Generation)

for( gen in 1:length(UniqueGens)){

                lines(density(

                output$Attack[output$Generation== UniqueGens [gen]],

                bw= bandw$bw),  col=gen, lwd=2)

};

legend("topright",paste("Generation", UniqueGens),col=1:6,lwd=2)#<—ensures order
```

Aggregating results without bringing everything into R

Expecting a lot of output? Don’t load it into memory:

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

sql_poke = "SELECT Generation, count(Generation) AS NumberPerG
FROM Pokem”

- QuerryOut = dbSendQuery(dbcon, sql_poke)
- dbFetch(QuerryOut, 5)

or

- dbGetQuery(dbcon, sql_poke)

Aggregating results without bringing everything into R

Expecting a lot of output? Don’t load it into memory:

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

sql_poke = "SELECT Generation, count(Generation) AS NumberPerG
FROM Pokem **GROUP BY Generation**”

- dbGetQuery(dbcon, sql_poke)

Aggregating results without bringing everything into R

Expecting a lot of output? Don’t load it into memory:

Count and come up with more summaries of Pokemon per generation:

sql_poke = "SELECT Generation, **SUM**(Attack) AS SumAttack, **COUNT**(Generation) AS NumberPerG, **AVG**(Attack) AS AvgAttacks FROM Pokem GROUP BY Generation”

dbGetQuery(dbcon, sql_poke)

Aggregating results without bringing everything into R

Expecting a lot of output? Don't load it into memory:

Count and come up with more summaries of Pokemon per generation:

```
sql_poke = "SELECT Generation, SUM(Attack) AS SumAttack,
COUNT(Generation) AS NumberPerG, AVG(Attack) AS AvgAttacks,
MIN(Attack) AS MinAttacks, MAX(Attack) AS MaxAttacks FROM Pokem
GROUP BY Generation"

dbGetQuery(dbcon, sql_poke)
```

Collecting DISTINCT pokemon Types

```
sql_poke = "SELECT DISTINCT Type_1 FROM Pokem"

dbGetQuery(dbcon, sql_poke)
```

```
sql_poke = "SELECT DISTINCT Type_1, Type_2 FROM Pokem"

dbGetQuery(dbcon, sql_poke)
```