Week 5
RSQLite, ~~the~~ our final frontier

https://www.indeed.com/q-Junior-Data-Scientist-jobs.html

https://neuvoo.ca/jobs/?k=data+scientist&l=Vancouver

https://www.glassdoor.ca/Job/vancouver-data-scientist-jobs-SRCH_IL.0,9_IC2278756_KO10,24.htm

https://www.wowjobs.ca/careers-data%20scientist-jobs-in-Vancouver

```
dbcon = dbConnect(RSQLite::SQLite(),
dbname="stat240Week5.sqlite")
```

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:

More Math on the Pokemon within 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:

More Math on the Pokemon within generation:

```
sql_poke = "SELECT Generation,
SUM(Attack)/COUNT(Generation)   AS myavg, AVG(Attack) AS
AvgAttacks FROM Pokem GROUP BY Generation"

dbGetQuery(dbcon, sql_poke)
```

What is wrong here?

Aggregating results without bringing everything into R

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

More Math on the Pokemon within generation:

```
sql_poke = "SELECT Generation,
SUM(Attack)/(COUNT(Generation)*1.0)   AS myavg, AVG(Attack) AS
AvgAttacks FROM Pokem GROUP BY Generation"

dbGetQuery(dbcon, sql_poke)
```

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.

Aggregating results without bringing everything into R

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

More basic 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)
```

Aggregating results without bringing everything into R

To do more general math you need to move some operations to the database:

```
initExtension(dbcon)

sql_poke = "SELECT Generation, STDEV(Attack) AS stdev, AVG(Attack) AS AvgAttacks FROM Pokem GROUP BY Generation"

dbGetQuery(dbcon, sql_poke)
```

Which functions:

```
?initExtension
```

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)
```

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"

dbGetQuery(dbcon, sql_poke)

How, using SQL do we get the unique types without the type combinations?

UNION ALL stacks the values from two queries (duplicates are kept)

sql_poke = "**SELECT Type_1 from Pokem UNION ALL SELECT isLegendary FROM Pokem**"

dbGetQuery(dbcon, sql_poke)

Counting distinct Combinations of Column 1 and Column 2

```
sqlcheck = "SELECT Type_1, isLegendary,
COUNT(*) AS NOccurences FROM (SELECT
Type_1, isLegendary  FROM Pokem) GROUP BY
Type_1, isLegendary"

dbGetQuery(dbcon, sqlcheck)
```

Average (in SQL)

Relating back to Density Estimation and extensions

Windowed Average, in Math

Moving Average

The Average (the mean)

For some vector of values, that we'll call

$X = c(x_1, x_2, x_3, \ldots, x_N)$

The average is calculated:

$$\bar{X} = \frac{\sum_{i=1}^{N} x_i}{N}$$

$$\bar{X} = \sum_{i=1}^{N} \frac{x_i}{N}$$

$$\bar{X} = \frac{x_1}{N} + \frac{x_2}{N} + \frac{x_3}{N} + \cdots + \frac{x_N}{N}$$

The Average (the mean)

For some vector of values, that we'll call

$X = c(x_1, x_2, x_3, \ldots, x_N)$

The average is calculated:

$$\bar{X} = \frac{\sum_{i=1}^{N} x_i}{N}$$

$$\bar{X} = \sum_{i=1}^{N} \frac{x_i}{N}$$

$$\bar{X} = \frac{x_1}{N} + \frac{x_2}{N} + \frac{x_3}{N} + \ldots + \frac{x_N}{N}$$

Or equivalently, for some weights $w_{i=1}$

$$\bar{X} = \frac{\sum_{i=1}^{N} x_i w_i}{\sum_{i=1}^{N} w_i}$$

Histograms

Histograms count values within a bin with edges $(b_1,b_2]$

Bin height = COUNT(X) WHERE $X>b_1$ AND $X<= b_2$

We can get this by adding up logical flags:

height(b1) = sum (X>b1 &X<=b2)

In other words:

$$\text{Height}(b_1) = \sum_{i=1}^{N} \mathbb{I}_{(b_1,b_2]}(x)$$

where:

$$\mathbb{I}_{(b_1,b_2]}(x) = \begin{cases} 0 & \text{if } x <= b_1 \\ 1 & \text{if } x \in (b_1,b_2] \\ 0 & \text{if } x > b_2 \end{cases}$$

```
x = c(rnorm(1000,2,1),rnorm(1000,7,1))
```

```
hist(x,100,xlab="x = 2 Normals",probability =
TRUE,main = "100 breaks",xlim=c(-4,13))
```

Density Plot

Density Estimates count values close to b_1

Close values are more similar —> count for more than far values.

Bin height(b_1) = COUNT(X) But Weight points based on (x-b_1) distance

We can get this by adding up **Weighted** logical flags:

For example at some point b_1:

$$\text{Height}(b_1) = \sum_{i=1}^{N} exp\left(-\frac{1}{2\sigma^2}(x_i - b_1)^2\right)$$

Density Plot

Density Estimates count values close to b_1

Close values are more similar —> count for more than far values.

Bin height(b_1) = COUNT(X) But Weight points based on (x-b_1) distance

We can get this by adding up **Weighted** logical flags:

For example at some point b_1:

$$\text{Height}(b_1) = \sum_{i=1}^{N} exp\left(-\frac{1}{2\sigma^2}(x_i - b_1)^2\right) = \sum_{i=1}^{N} w_i(b_1)$$

Calculating Densities

Bandwidth,σ = 0.54

Height(x=5):

```
Heightat5       = sum(exp(-1/(2*.54^2)*(x - 5)^2))

Heightat3       = sum(exp(-1/(2*.54^2)*(x - 3)^2))

Heightat0       = sum(exp(-1/(2*.54^2)*(x - 0)^2))

HeightatNeg1 = sum(exp(-1/(2*.54^2)*(x + 1)^2))

abline(v=c(-1,0,3,5))
```

$$\text{Height}(b_1) = \sum_{i=1}^{N} exp\left(-\frac{1}{2\sigma^2}(x_i - b_1)^2\right) = \sum_{i=1}^{N} w_i(b_1)$$

```
#x = c(rnorm(1000,2,1),rnorm(1000,7,1))


#hist(x,100,xlab=“x",probability = TRUE,main = "mixture of 2
Normals",xlim=c(-4,13))

lines(density(x),col=2,lwd=3)

grid = seq(-4,13,length=1000);

lines(grid,.5*dnorm(grid,2,1,)+.5*dnorm(grid,7,1,),lty=2,col=4,lwd=4)

legend("topleft",lty=c(0,1,2),col=c(1,3,4),c("100 bin histogram”,"Density","Truth"))
```

2D Density Plot
same but b_1 has 2D and x_i has 2D

Density Estimates count values close to b_1

Close values are more similar —> count for more than far values.

Bin height(b_1) = COUNT(X) But Weight points based on (x-b_1) distance

We can get this by adding up **Weighted** logical flags:

For example at some point b_1:

$$\text{Height}(b_1) = \sum_{i=1}^{N} exp\left(-\frac{1}{2\sigma^2}(x_i - b_1)^2\right) = \sum_{i=1}^{N} w_i(b_1)$$

2D Density Plot

```
library(MASS)

library(sp)

library(rworldmap)

library(rworldxtra)

worldmap = getMap(resolution = "high")

NrthAm = worldmap[which(worldmap$REGION =="North America"),]

plot(NrthAm,xlim=c(-124,-122.4),     ylim=c(48.7,49.6),main = "Vancouver-ish")

points(poke$longitude,poke$latitude,pch='.')
```

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

contour(est2, add=TRUE,col=2,lwd=3)

The Average (the mean)

For some vector of values, that we'll call

$X = c(x_1, x_2, x_3, \ldots, x_N)$

The average is calculated:

$$\bar{X} = \frac{\sum_{i=1}^{N} x_i}{N}$$

$$\bar{X} = \sum_{i=1}^{N} \frac{x_i}{N}$$

$$\bar{X} = \frac{x_1}{N} + \frac{x_2}{N} + \frac{x_3}{N} + \ldots + \frac{x_N}{N}$$

Or equivalently, for some weights $w_{i=1}$

$$\bar{X} = \frac{\sum_{i=1}^{N} x_i w_i}{\sum_{i=1}^{N} w_i}$$

The Moving Average

For some vector of values, that we'll call

$X = c(x_1, x_2, x_3, \ldots, x_N)$

The (moving) average is calculated:

$$\bar{X} = \frac{\sum_{i=1}^{N} x_i w_i}{\sum_{i=1}^{N} w_i}$$

But the weights are calculated as

$$w_i(b_1) = \mathbb{I}_{(b_1, b_2]}(x)$$

$$w_i(b_1) = exp\left(-\frac{1}{2\sigma^2}(x_i - b_1)^2\right)$$

Doing a query on a query

Sometimes a complex query might be easiest if done in steps.

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

Create, find, examine, and remove a VIEW

```
sql_poke = "CREATE VIEW pokemean AS SELECT Generation,
SUM(Attack)/COUNT(Generation)   AS myavg, AVG(Attack) AS AvgAttacks
FROM Pokem GROUP BY Generation"

dbSendQuery(dbcon, sql_poke)

dbListTables(dbcon)

query_table_info = "PRAGMA table_info('pokemean')"

dbGetQuery(dbcon,query_table_info)

dbSendQuery(dbcon, "drop view pokemean")

dbListTables(dbcon)
```

Doing a query on a query

Sometimes a complex query might be easiest if done in steps.

WITH lets you define a temporary table. It disappears after the query is called.

Using Temporary Tables

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.Attack-pokestuff.AvgAttacks AS Resids FROM pokestuff, Pokem"

(out = dbGetQuery(dbcon, sql_poke))

par(mfrow=c(2,1))

hist(out$Attack,50)

hist(out$Resids,50)

Using Temporary Tables

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.Attack-pokestuff.AvgAttacks AS Resids FROM pokestuff, Pokem"

(out = dbGetQuery(dbcon, sql_poke))

par(mfrow=c(2,1))

hist(out$Attack,50)

hist(out$Resids,50)

The Normal Distribution

Continuous values

Most of the values are close to the mean

Values are less and less likely as you move away

Standardized scores

Calculate the distance from the mean in units of Standard Deviations:

If you're above average, are you 'way above average' or 'within typical variation above average'?

Random samples from a normal distribution

hist(rnorm(1000),50,main="1000 random samples from the Normal Distribution",xlab = "Distance from the Mean in SD units")