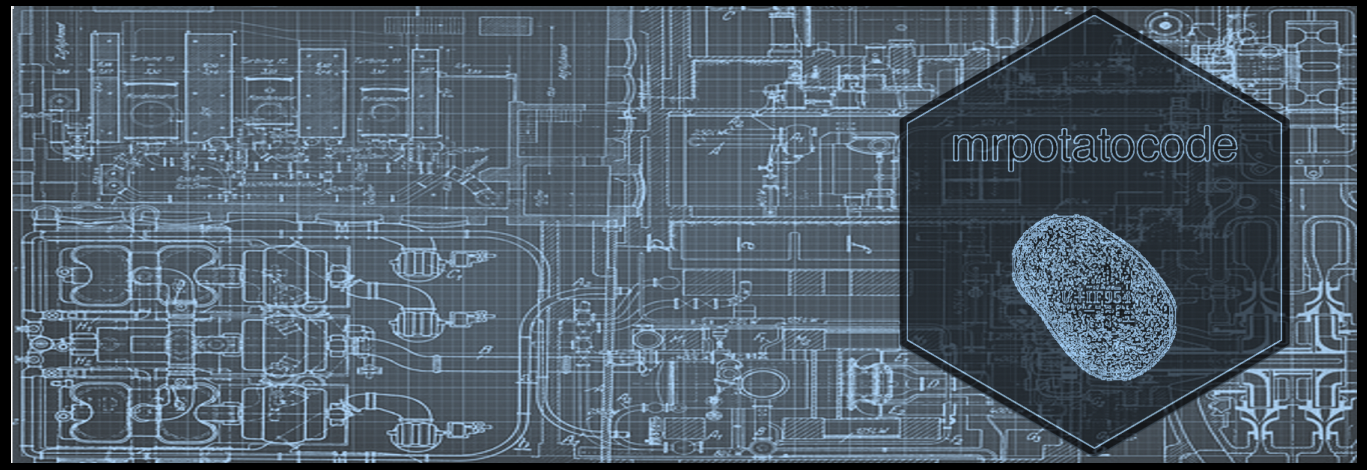



# Introduction to Data Access and Storage

Thomas Rosenthal – DSI @ UofT

Module 06

# SQL and Machine Learning:

What is Machine Learning?

SQL's Role

Run SQLite in python and/or R

Connecting the Dots

# SQL and Machine Learning:

What is Machine Learning?

SQL's Role

Run SQLite in python and/or R

Connecting the Dots

# What is Machine Learning?

Supervised Learning

Unsupervised Learning

Basic Machine Learning Process

# What is Machine Learning?

- Machine Learning (ML) uses *data* to discover *patterns* by the use of *statistics*
- ML is everywhere:
  - Predictive Models estimate the outcome based on the relationships of the model's inputs
    - Economic/Financial Studies
    - Health Studies
  - Classification Models highlight anomalies, determine how things can be sorted into predefined categories
    - Image classification ("select the pictures of stairs")
    - Fraudulent credit card transactions
  - Clustering Models create groupings based on similarities
    - Customer Groupings
    - News Topics
  - Recommender Models suggest what to watch, click, serve, display, etc
    - News Feeds (social media)
    - What to Watch (Netflix, etc)
    - Personalized Ads
  - Language Models analyze human speech
    - Chatbots on websites
    - ChatGPT / GPT3
    - Text Summarization tools
  - Generative Models can even create "art"?
    - DALL-E

# What is Machine Learning?

- We can broadly split ML into three types:
  - Supervised Learning
  - Unsupervised Learning
  - Reinforcement Learning (we'll skip this one)

- Supervised: we provide the algorithms with explicit values/labels to base future predictions off of
- Supervised can be split into two tasks:
  - Regression
  - Classification

- Unsupervised: we don't provide the algorithms with explicit values/labels, requiring the machine to create groupings based on other mathematical properties
- Unsupervised can be split into several tasks:
  - Clustering
  - Association
  - Decomposition
  - ...more

# What is Machine Learning?

Supervised Learning

Unsupervised Learning

Basic Machine Learning Process

# Supervised Learning

- Regression predicts numerical values
  - How much should this insurance policy cost?
    - Input: driver's experience, maximum financial damages possible, car characteristics, etc
    - Output: predicted premium value

- Classification predicts categories
  - Is this email spam or not?
    - Input: all of the emails from Enron (I'm not joking here), labels generated by humans
    - Output: "Buy Medication Cheap" --> SPAM!

# Supervised Learning

- Regression Algorithms:
  - Linear regression, predicting values along a linear line
  - Regularized (i.e. Ridge, LASSO) regression, predicting values by penalizing coefficients to reduce model complexity
  - Support Vector Machines (SVM) regression, predicting values within a hyperdimensional space
  - ...and many more

- Classification Algorithms:
  - Logistic regression (despite its name!), predicting labels along a sigmoid (binomial: either 0 or 1 )
  - Decision Tree classification, predicting labels based on information gained by splitting nodes
  - Support Vector Machines (SVM) classification, predicting labels within a hyperdimensional space
  - ...and many more

# What is Machine Learning?

Supervised Learning

Unsupervised Learning

Basic Machine Learning Process

# Unsupervised Learning

- Clustering objects/concepts (data points) that are most similar to one another
  - Using behavioural data, which customers are similar to one another?
    - Input: behavioural data, other features deemed relevant
    - Output: customer X belongs to group 1, customer Y belongs to group 2, etc

- Association
  - Using purchase history, what things are often bought together?
    - Input: purchases, other features deemed relevant
    - Output: "fathers, while you're buying diapers, do you also want to buy beer?"

- Decomposition
  - What combination of features describes the value of a real estate asset (e.g. house, condo, etc)?
    - Input: interior details, exterior details, geographic features, neighbourhood amenities, nearness to transit...many, many more
    - Output: nearness to transit and fewer bedrooms vs quality of schools and parking spaces available (etc, many more)

# Unsupervised Learning

- Clustering algorithms:
  - K-means, discovery of groupings based on a given point's nearness (by use of a distance metric, e.g. Euclidean) to a cluster's center
  - Hierarchical clustering, discovery of groupings based on hierarchy creation (either agglomerative or divisive)
  - ...and a few more

- Association algorithms:
  - *a priori* association, discovery of frequent combinations in an itemset (by use of Cartesian joins)
  - ...and a few more

- Decomposition algorithms:
  - Principal Component Analysis (PCA), discovery of groupings based on maximizing variance between groupings
  - ...and a few more

General ML Questions?

# What is Machine Learning?

Supervised Learning

Unsupervised Learning

Basic Machine Learning Process

# Basic Machine Learning Process

This process describes supervised learning. Unsupervised learning is similar, but skips comparisons against test datasets.

- Acquire data
- Clean data
- Preprocess data

- Divide data into training, testing, (and validation) datasets
- Using the training dataset, allow the selected algorithm to generate variable relationships (e.g. coefficients)
- Using these coefficients, produce predicted values on the test dataset

- Evaluate the model's performance by comparing predicted values in the test dataset to actual values in the test dataset
- If the model performs well, predict values on data that does not yet have actual values
- Separately, collect and store actual values ("ground truth", e.g. what actually happened), to compare the model's performance over time

ML Pipeline Questions?

# SQL and Machine Learning:

What is Machine Learning?

SQL's Role

SQLite Meets python and/or R

Connecting the Dots

# SQL's Role

- SQL isn't designed to run ML algorithms, but it can help serve as the foundation for good ML pipelines
  - Dataset reliability will lend itself to higher quality models over time
  - Observing changes in data requires good data architecture (one of SQL's strengths!)
    - We'll revisit this as "Model Monitoring"

- SQL DBs can be connected to python/R fairly easily
  - Preprocessing can be performed in SQL beforehand, in python/R after extraction, or both
  - Instantiating simple preprocessing actions in a permanent table can reduce the need to rerun calculations that would be stored in memory for python/R
    - e.g. if you write several window functions and save these to a table, these values are available for current and future ML pipelines without the need to run calculations again in a seperate python/R script (or if models were rerun)

- Some RMDBs can run python/R scripts within procedures, saving the output in a SQL table
  - SQLite isn't one of these, but there are other ways we'll create similar outcomes

Questions about SQL's Role in ML?

# SQL and Machine Learning:

What is Machine Learning?

SQL's Role

SQLite Meets python and/or R

Connecting the Dots

# SQLite Meets python and/or R

Why python and/or R?

Connect to a SQLite DB

Run SQLite Commands

# SQLite Meets python and/or R

Why python and/or R?

Connect to a SQLite DB

Run SQLite Commands

# Why python and/or R?

- Python/R are open source, free, and flexible programming languages
- ML is relatively mature in both:
  - Python: SKLearn serves as a great foundation for the most common algorithms
    - The cutting edge for ML is generally in python: PyTorch, TensorFlow, other Deep Learning algorithms
  - R: a bit more of a cobbled together history (caret, clustering packages, etc), but now well served by tidymodels
    - R is designed for statistics...building customized modelling approaches may be easier for statisticians than python (less programming overhead)

- Both python/R can connect to SQLite!
- Both python/R can run SQLite queries!

# Why python and/or R?

- Python/R can also be instrumental in data engineering
- Previously, it was more challenging to move data between SQL systems
  - We'd use ETL tools like SQL Server Integration Services (SSIS), establish our connections to each SQL system, perform whatever transformations needed to be done (e.g. normalization), and load the result sets into the appropriate corresponding tables
  - The overhead was challenging to maintain
  - The tools often weren't free

- Now, we can perform many of the same transformations using libraries like pandas (python) or tidyverse (R)
  - The overhead is still challenging to maintain, but modularization, reproducibility, and sensible pipeline design can make it easier
  - Automating these tasks can be easier
  - The tools for automating tasks can be free too

- We can also do much more complex transformations or incorporate more specific architectural details using the flexibility of libraries in python/R that may not be native to SQL
  - Geographies, shape files, distance calculations, etc
  - Image and file related processing
  - Greater mathematical needs, like ML algorithms, etc

# SQLite Meets python and/or R

Why python and/or R?

Connect to a SQLite DB

Run SQLite Commands

# Connect to a SQLite DB

## Python Example

We can easily connect SQLite to python:

```
import pandas as pd
import sqlite3


#set your location, slash direction will change for windows and mac
DB = '/Users/thomas/Documents/GitHub/02-intro_SQL/SQL/FarmersMarket.db'

#establish your connection
conn = sqlite3.connect(DB, isolation_level=None,
                       detect_types=sqlite3.PARSE_COLNAMES)
```

# Connect to a SQLite DB

## Python Example

Using pandas, we can use this connection to run queries:

```
#run your query, use "\" to allow line breaks
db_df = pd.read_sql_query("SELECT p.*,pc.product_category_name \
                          FROM product p \
                          JOIN product_category pc \
                             ON p.product_category_id = pc.product_category_id"
                          ,conn)
```

`db_df` is now available as a dataframe (a specialized type of array in python) for any pandas command

Like extracting a query result to csv:

```
#save
db_df.to_csv('database-py.csv', index=False)
```

# Connect to a SQLite DB

## R Example

We can do the same in R:

```
library(DBI)
library(RSQLite)

#set your location, slash direction will change for windows and mac
DB = '/Users/thomas/Documents/GitHub/02-intro_SQL/SQL/FarmersMarket.db'

#establish your connection
conn <- dbConnect(SQLite(), DB)
```

# Connect to a SQLite DB

## R Example

And similarly produce a csv, or any other R command we might want to perform on the dataframe:

```
#run your query
db_df <- dbGetQuery(conn, "SELECT p.*,pc.product_category_name
                           FROM product p
                           JOIN product_category pc
                             ON p.product_category_id = pc.product_category_id")

#save
write.csv(db_df, file = "database-R.csv")
```

# Connect to a SQLite DB

(live coding in R)

Questions about SQLite connections?

# SQLite Meets python and/or R

Why python and/or R?

Connect to a SQLite DB

Run SQLite Commands

# Run SQLite in python and/or R

The previous section showed how to connect a SQLite database to python and/or R and interact with tables

**...but you can also run SQLite queries on python and/or R dataframe objects!**

- Both languages offer good support
  - Often there is corresponding python and/or R syntax to achieve similar results
    - As always with data, there is no prescribed way of doing something!

# Run SQLite in python and/or R

## Python Example

```
import pandas as pd
import pandasql as sql #this allows us to run SQLite queries!

p = "https://raw.githubusercontent.com/allisonhorst/palmerpenguins/master/inst/extdata/penguins.csv"
penguins = pd.read_csv(p) #create a dataframe
yrly_penguins = sql.sqldf('''SELECT DISTINCT year, COUNT(*) AS count,
                          SUM(COUNT(*)) OVER (ORDER BY year) AS running_total
                          FROM penguins
                          GROUP BY year''') #run a SQLite query with sqldf()
```

| year | count | running_total |
|---|---|---|
| 2007 | 110 | 110 |
| 2008 | 114 | 224 |
| 2009 | 120 | 344 |

# Run SQLite in python and/or R

## R Example

```
library(palmerpenguins) #load some data
library(sqldf) #this allows us to run SQLite queries!

penguins <- penguins #create a dataframe
avg_mass <- sqldf('SELECT DISTINCT species, sex, AVG(body_mass_g) as avg_mass
      FROM penguins
      WHERE sex <> "NA"
      GROUP BY species, sex') #run a SQLite query with sqldf()
```

| species | sex | avg_mass |
|---|---|---|
| Adelie | female | 3368.836 |
| Adelie | male | 4043.493 |
| Chinstrap | female | 3527.206 |
| Chinstrap | male | 3938.971 |
| Gentoo | female | 4679.741 |
| Gentoo | male | 5484.836 |

# Run SQLite in python and/or R

(live coding in python)

Questions about Running SQLite commands?

# SQL and Machine Learning:

What is Machine Learning?

SQL's Role

SQLite Meets python and/or R

Connecting the Dots

# Connecting the Dots

End-to-End Machine Learning

Data Extraction

Data Manipulation

Simple Decision Tree (Classification)

Model Monitoring

# Connecting the Dots

End-to-End Machine Learning

Data Extraction

Data Manipulation

Simple Decision Tree (Classification)

Model Monitoring

# Connecting the Dots

## End-to-End Machine Learning

The purpose of this section is to introduce you to an ML example, starting in SQL and finishing with a live model we can test.

We'll use the Palmer Penguins Dataset, developed by Allison Horst. The full history of this dataset and its purpose are available here: link

The objective will be to predict which species of Palmer Penguin a penguin is, based on bill length, bill depth, flipper length, body mass, sex, and island (i.e. where the penguin was when the data was collected).

There are three species: Chinstrap, Gentoo, and Adelie.


Artwork by @allison_horst

# Connecting the Dots

## End-to-End Machine Learning

- Using SQL, load our dataset (penguins) as a csv
- Update the column types so that `bill_length_mm`, `bell_depth_mm`, `flipper_length_mm` are `NUMERIC`, `rowid`, `body_mass_g`, `year` are `INTEGER`, and `species`, `island`, `sex` are `TEXT`
- Connect to our SQLite DB in either python or R (we'll use R in our example)
    - Build a query to extract the data from SQL
- Build a Decision Tree classification model to predict `species`
    - Use tidymodels in R
    - Use SKLearn in python
- Test the model's performance
    - Confusion matrix
- Provide our model new data
    - In our example, we'll create a live API connection in R to our model and provide it with JSON formatted data exported from SQLite
        - This will be our "monitoring" concept

# Connecting the Dots

End-to-End Machine Learning

Data Extraction

Data Manipulation

Simple Decision Tree (Classification)

Model Monitoring

# Connecting the Dots

## Data Extraction

- Build your connection to SQLite
  - python:

```
import sqlite3
#set your location, slash direction will change for windows and mac
DB = '/Users/thomas/Documents/GitHub/02-intro_SQL/SQL/FarmersMarket.db'
#establish your connection
conn = sqlite3.connect(DB, isolation_level=None,
                       detect_types=sqlite3.PARSE_COLNAMES)
```

  - R:

```
library(DBI)
library(RSQLite)
#set your location, slash direction will change for windows and mac
DB = '/Users/thomas/Documents/GitHub/02-intro_SQL/SQL/FarmersMarket.db'
#establish your connection
conn <- dbConnect(SQLite(), DB)
```

# Connecting the Dots

End-to-End Machine Learning

Data Extraction

Data Manipulation

Simple Decision Tree (Classification)

Model Monitoring

# Connecting the Dots

## Data Manipulation

- Write a query:
  - python:

```
db_df = pd.read_sql_query("SELECT ... FROM penguins ...",conn)
```

  - R:

```
db_df <- dbGetQuery(conn,"SELECT... FROM penguins ...")
```

- For our example, only select `species`, `island`, `bill_length_mm`, `bill_depth_mm`, `flipper_length_mm`, `body_mass_g`, `sex`
- And filter out missing values for the variable `sex`
  - `WHERE sex IS NOT NULL`
  - What other options could we use to replace missing values?

# Connecting the Dots

End-to-End Machine Learning

Data Extraction

Data Manipulation

Simple Decision Tree (Classification)

Model Monitoring

# Connecting the Dots

## Simple Decision Tree (Classification)

- Decision Trees in Machine Learning are like Decision Trees in real life!
- Start with a set of rules to evaluate the results:
  - Does the animal breathe air?
    - Yes -> Does the animal lay eggs?
      - Yes -> Bird
      - No -> Mammal
    - No -> Fish
- Metal or Normal?

- So...we ask the machine to give us a set of rules based on the data
- Decision Trees are useful because they are:
  - Explainable: *humans can understand how the data yielded the result*
  - Flexible: *will ignore irrelevant features, can perform both regression and classification, allows missing data, allows categorical data*
  - Fast: *computationally cheap (usually), especially compared to other ML algorithms*

# Connecting the Dots

## Simple Decision Tree (Classification)

- Decision trees work on Information Gain
- Use a top-down, recursive approach to determine the most sensible *"root node"* (the top of the tree, from which we will split)
  - Mathematically, this is based on the ID3, "Iterative Dichotomiser 3" method

- After determining an appropriate root node (whichever has provided the highest information gain), recursively iterate on the remaining attributes to find the next highest information gain and split again
  - Decision Trees are greedy: this means that there is a good chance that better combinations of rules *can* exist, but ID3 did not test for them

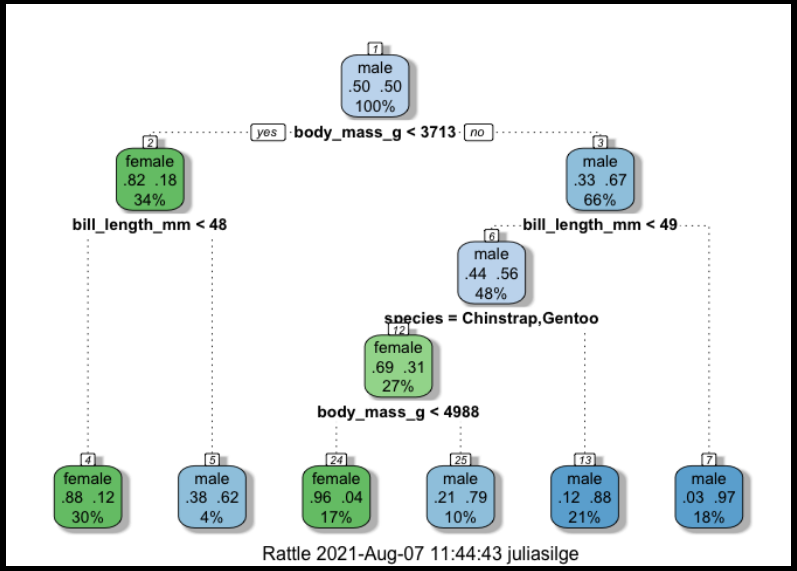
- Split the newly created nodes again (and again...and again...) until:
  - All values for a given node belong to the same class
  - No values exist without a classification (for all nodes)
  - There are no remaining attributes to partition on
  - The algorithm has been told to stop (e.g. with a parameter)

# Connecting the Dots

## Simple Decision Tree (Classification)

Let's consider an example:

Some rows in penguins having missing sex:




We can estimate sex based on the following ruleset:

- body_mass_g < 3717
  - Yes -> bill_length_mm < 48
    - Yes -> female (88% meeting this criteria were female)
    - No -> male (62% male)
  - No -> bill_length_mm < 49
    - Yes -> species either Chinstrap or Gentoo
      - Yes -> body_mass_g < 4988
        - Yes -> female (96% female)
        - No -> male (79% male)
      - No -> male (88% male)
    - No -> male (97% male)

# Connecting the Dots

(live end-to-end machine learning in R, part 1)

# Connecting the Dots

## Simple Decision Tree (Classification)

**How did our model do?**

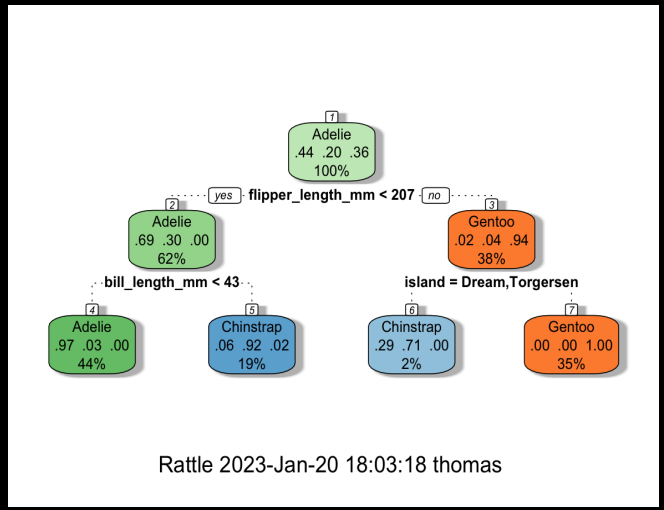
Overall accuracy was 93%!

84 penguins were divided as follows:

| | | Truth | | |
|---|---|---|---|---|
| | | Adelie | Chinstrap | Gentoo |
| Predicted | Adelie | 35 | 0 | 0 |
| | Chinstrap | 2 | 14 | 0 |
| | Gentoo | 0 | 3 | 30 |

(we call this a "confusion matrix")

And our estimated ruleset was:



# Connecting the Dots

End-to-End Machine Learning

Data Extraction

Data Manipulation

Simple Decision Tree (Classification)

Model Monitoring

# Connecting the Dots

## Model Monitoring

- Model predictions are based on historical data
  - There is no certainty that future data will appear the same as historical data
  - Models won't adjust to changes in the data

- Most algorithms are poor at explaining how they arrived at a prediction
  - Some models are "white-box" by nature (like decision trees), while others are "black-box" (like deep-learning neural nets)
  - The inherent explainability of white-box models makes them easier to evaluate over time

- When models change over time, we call this *model drift*
- There are two types of drift:
  - Concept Drift: the way the model 'understood' things to relate to each other has changed, in a way the model doesn't know, so the model is no longer helpful
  - Data Drift: information that the model has to deal with now is too different from the information the model has previously seen, so the model is no longer helpful

# Connecting the Dots

## Model Monitoring

- To observe model drift, we need to gather new data and compare how the model is performing
- If metrics (like accuracy, but there are others too) are changing over time, we are observing drift

- One type of new data we can collect is called "ground truth"
- Suppose we have an algorithm predicting whether or not a credit card transaction is fraudulent:
  - Each time the algorithm flags a transaction as fraudulent, a customer service representative calls the credit card holder
  - For instances where fraud occurred, the representative can mark the algorithm as correct
  - For instances where the transaction was legitimate, the representative can mark the algorithm as incorrect
  - What's missing from this?

- Ground truth, even in cases where it may be difficult to obtain, can greatly improve a model's usefulness and its longevity
- SQL is a perfect place to store ground truth data!
  - e.g. the customer service representative might have access to a frontend UI that allows them to mark fraudulent/non-fraudulent transactions, which are then stored in a SQL backend

# Connecting the Dots

## Model Monitoring

- We claimed our penguin model performed with 93% accuracy
- Because this class can't go and collect more penguin data, it'd be challenging to evaluate the model's accuracy over time 🥶
- However, we can emulate this process, and make the most of our SQL skills!

- Using the `vetiver` and `plumber` packages, and following fantastic instructions from Julia Silge's blog, we'll create a live connection to our model
- We'll then export a subset of penguins data (or make up our own? sure!) as JSON and watch our model predict!

# Connecting the Dots

## Model Monitoring

The API produces predictions as JSON, so we can input these predicted values and compare to actual values

```
DROP TABLE IF EXISTS temp.[predicted_penguins];

CREATE TABLE IF NOT EXISTS temp.[predicted_penguins]
(predictions BLOB);


INSERT INTO temp.[predicted_penguins](predictions)
VALUES('[{".pred_class": "Adelie"}]');


SELECT pen.*, JSON_EXTRACT(value, '$.pred_class') AS prediction

FROM (SELECT *
        FROM predicted_penguins,JSON_EACH(predicted_penguins.predictions,'$')) x

-- we need to generate a row number to keep track of which prediction was which
JOIN penguins_json_export pen ON key = row_number
```

We've now gone end-to-end: from SQL to ML and back to SQL again!

# Connecting the Dots

(live end-to-end machine learning in R, part 2)

Lingering Questions?