building your first ML application

2020-11-17

> whoami

David Dai

MSc (Biostatistics)

Senior Data Scientist, St. Michael's Hospital

10% statistician/10% data scientist/80% master at Googling

> data science is easy

Most applied modeling tasks are simple

- Which clients are most likely to churn next month?
- How can we catalog these unlabeled documents?
- What are the types of patients in our hospital?

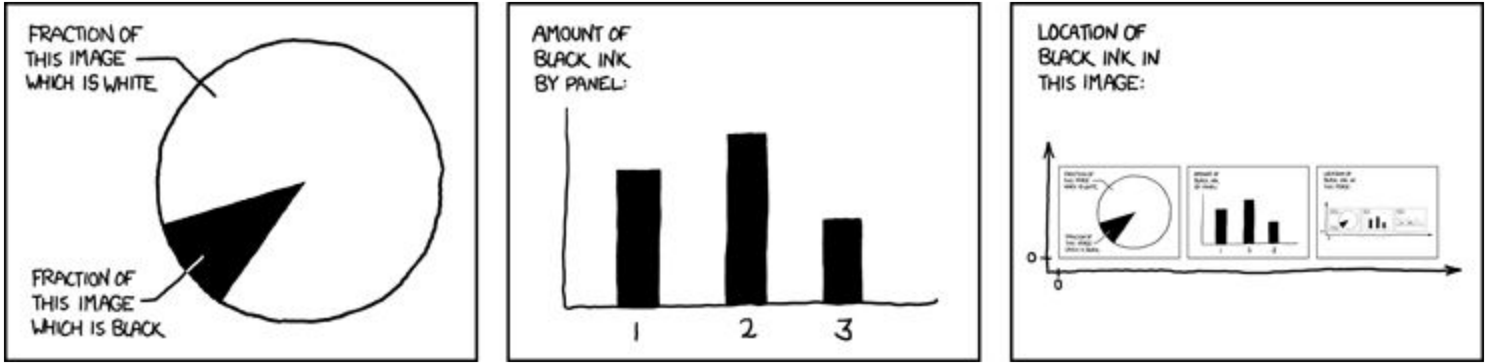
> data science is easy

Most applied modeling tasks are simple

- *Which clients are most likely to leave next month?*
  - **Logistic regression**
- *How can we catalog these unlabeled documents?*
  - **Topic modeling**
- *What are the types of patients in our hospital?*
  - **K-means clustering**

> data science is easy

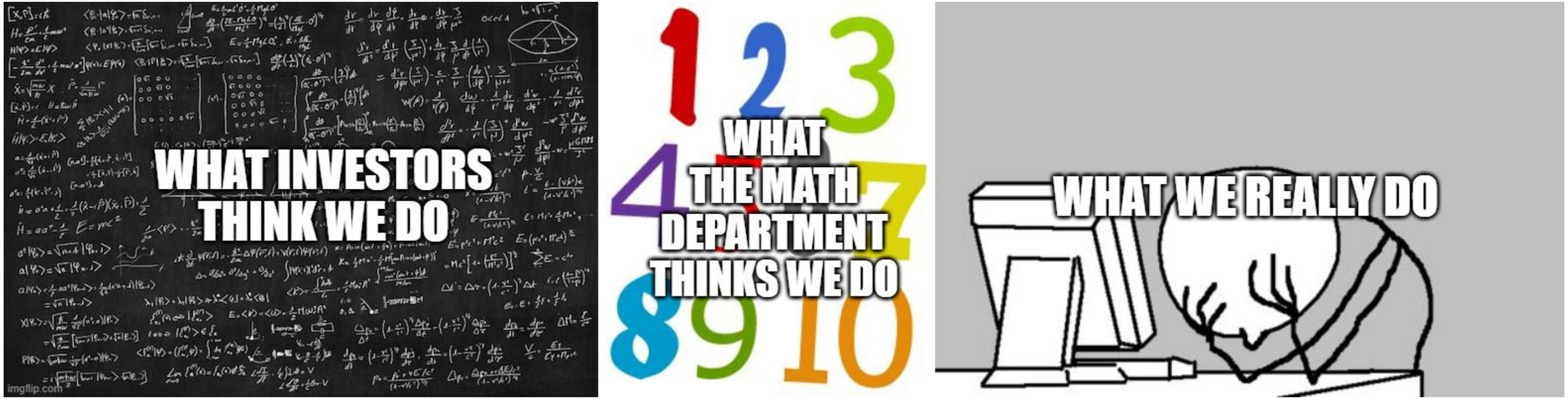
In fact, most of the time, you probably won't even need a model!



> data science is scary!

But what about learning BERT and all his cousins? Or using deep reinforcement learning to play Fortnite?

> data science is ???



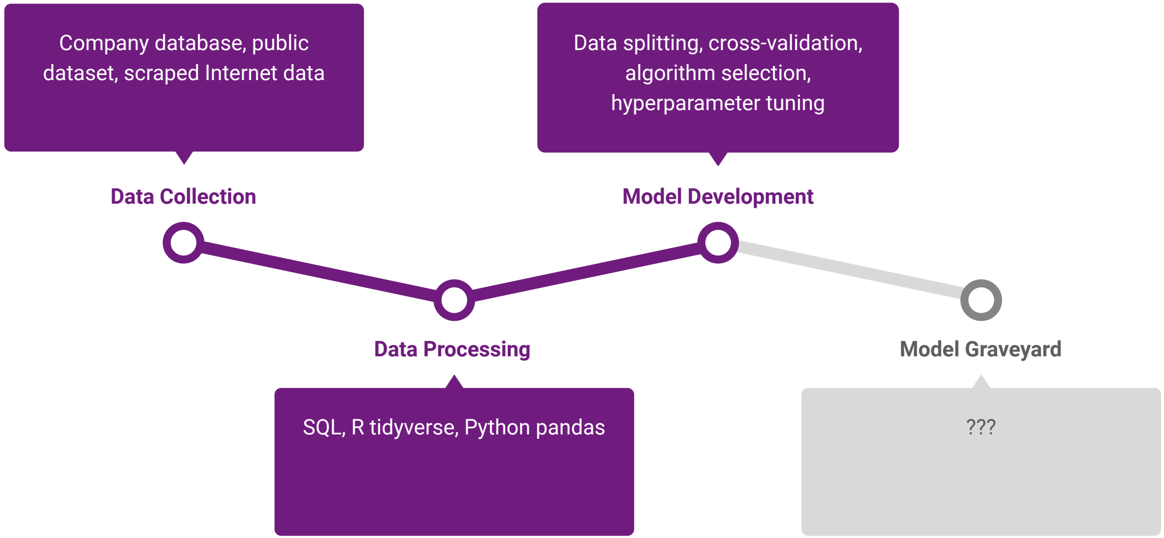
> data science is ???

Data Scientist: The Sexiest Job of the 21st Century

by Thomas H. Davenport and D.J. Patil

From the October 2012 Issue

> data science *can be hard*



> **data science** *in practice*

An important skill for any data scientist to have is the ability to ***deliver results***.

How?

1. Reports (written communication)
2. Presentations (oral communication)
3. Deploy your model

> today's objective

1. Through a case study, get exposure to various parts of a typical data science workflow
2. Learn about some ways to avoid the "model graveyard"
3. Deploy an ML application!

> case study: house pricing app

Suppose you want to build a model that predicts housing prices in Ontario

Of interest:

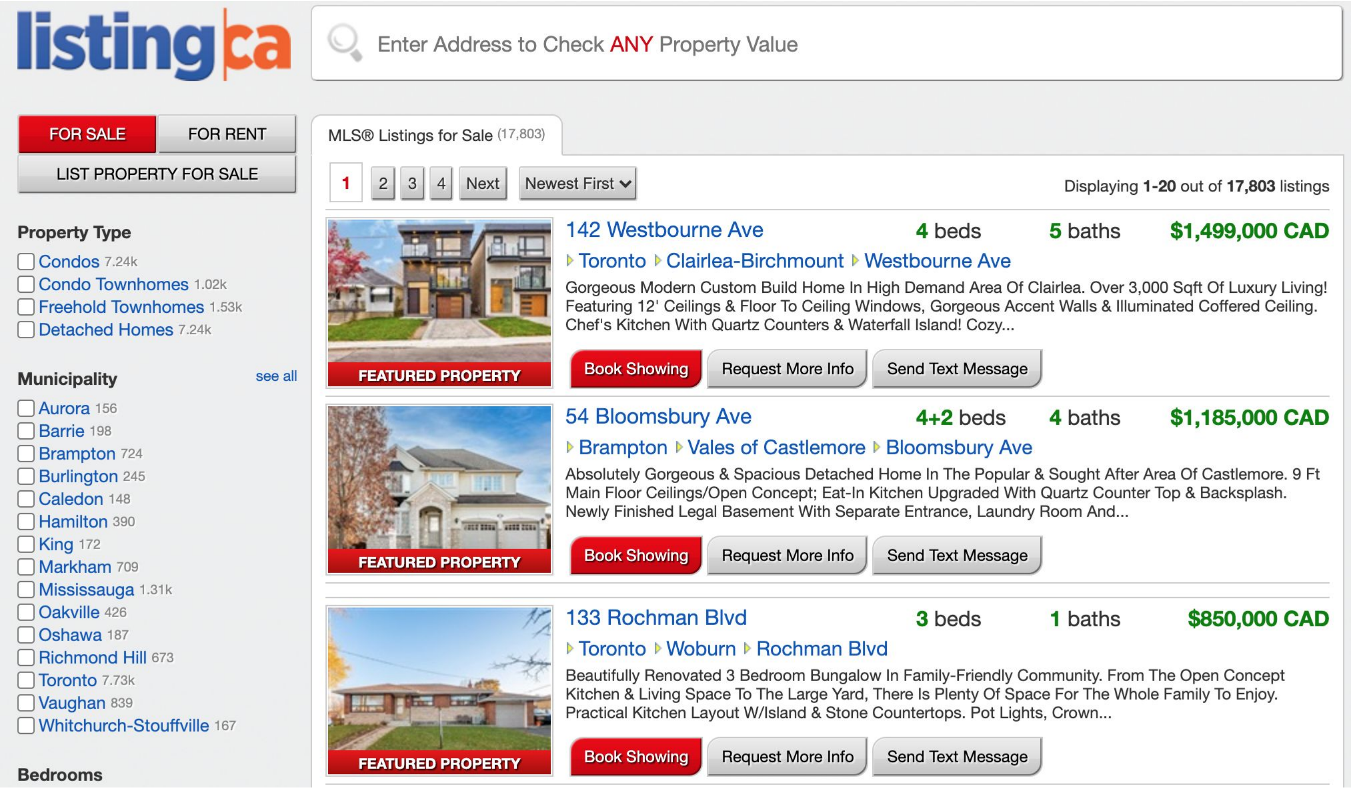
- What is the predicted house listing price for a 2 bedroom condo in Toronto?
- What about a 3 bedroom townhouse in Oakville?

> case study: steps

1. Data collection
2. Data processing
3. Model development
4. Model deployment

> data collection

> data source



listing ca
Enter Address to Check ANY Property Value
FOR SALE
FOR RENT
LIST PROPERTY FOR SALE
MLS® Listings for Sale (17,803)
1 2 3 4 Next
Newest First
Displaying 1-20 out of 17,803 listings
Property Type
Condos 7.24k
Condo Townhomes 1.02k
Freehold Townhomes 1.53k
Detached Homes 7.24k
Municipality
see all
Aurora 156
Barrie 198
Brampton 724
Burlington 245
Caledon 148
Hamilton 390
King 172
Markham 709
Mississauga 1.31k
Oakville 426
Oshawa 187
Richmond Hill 673
Toronto 7.73k
Vaughan 839
Whitchurch-Stouffville 167
Bedrooms
142 Westbourne Ave
4 beds
5 baths
$1,499,000 CAD
Toronto Clairlea-Birchmount Westbourne Ave
Gorgeous Modern Custom Build Home In High Demand Area Of Clairlea. Over 3,000 Sqft Of Luxury Living! Featuring 12' Ceilings & Floor To Ceiling Windows, Gorgeous Accent Walls & Illuminated Coffered Ceiling. Chef's Kitchen With Quartz Counters & Waterfall Island! Cozy...
FEATURED PROPERTY
Book Showing
Request More Info
Send Text Message
54 Bloomsbury Ave
4+2 beds
4 baths
$1,185,000 CAD
Brampton Vales of Castlemore Bloomsbury Ave
Absolutely Gorgeous & Spacious Detached Home In The Popular & Sought After Area Of Castlemore. 9 Ft Main Floor Ceilings/Open Concept; Eat-In Kitchen Upgraded With Quartz Counter Top & Backsplash. Newly Finished Legal Basement With Separate Entrance, Laundry Room And...
FEATURED PROPERTY
Book Showing
Request More Info
Send Text Message
133 Rochman Blvd
3 beds
1 baths
$850,000 CAD
Toronto Woburn Rochman Blvd
Beautifully Renovated 3 Bedroom Bungalow In Family-Friendly Community. From The Open Concept Kitchen & Living Space To The Large Yard, There Is Plenty Of Space For The Whole Family To Enjoy. Practical Kitchen Layout W/Island & Stone Countertops. Pot Lights, Crown...
FEATURED PROPERTY
Book Showing
Request More Info
Send Text Message

> data source

listing.ca contains real estate listings in Ontario

- Street address
- Number of bedrooms and bathrooms
- Listing price
- Listing type (condo, condo townhome, townhome, detached)
- Municipality

> data source

Prediction target of interest: listing price

Predictors of interest:

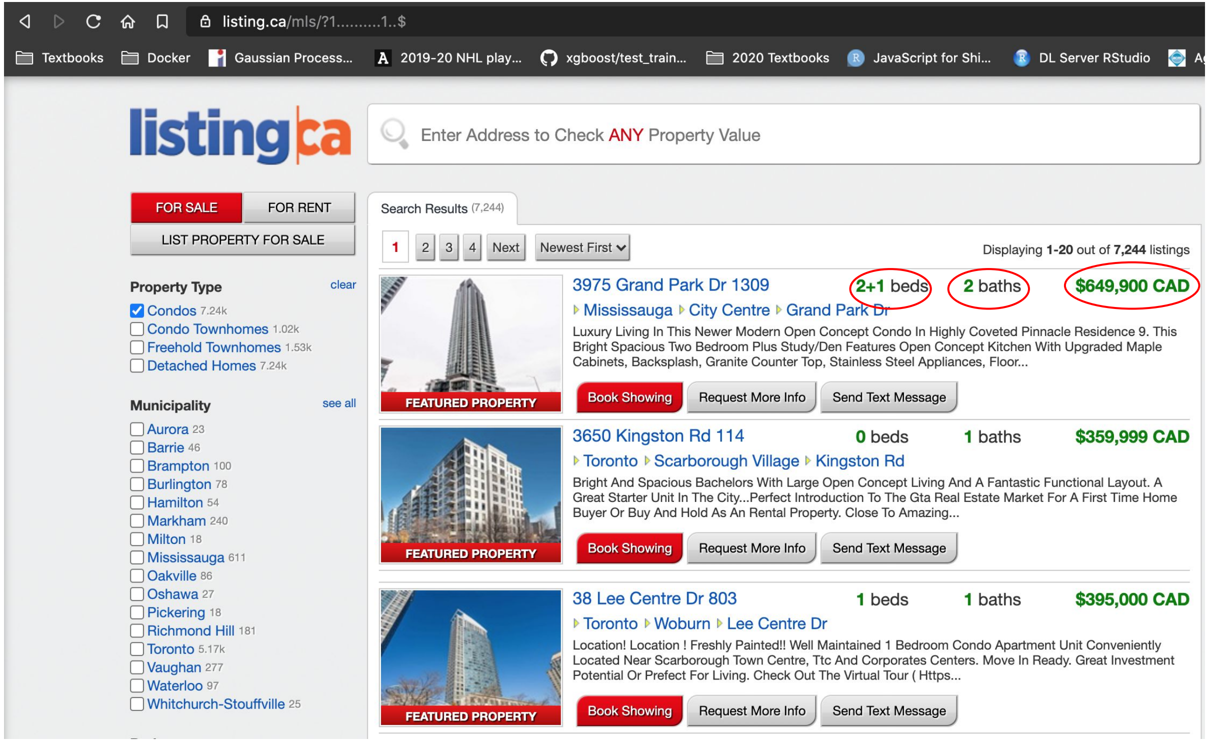
- Home type (eg. condo vs townhome)
- Number of bedrooms and bathrooms
- Location (eg. postal code, municipality, etc)

> scrape listings data

```
# General URL structure https://listing.ca/mls/?.cy.........{page_number}..$

# For specific types of listings:
# Condos listings https://listing.ca/mls/?1..........{page_number}..$
# Townhomes listings https://listing.ca/mls/?2..........{page_number}..$
```

> scrape listings data



> scrape listings data

```r
# Scrape basic data from listing.ca
scraper <- function(url) {
  require(rvest)
  require(magrittr)

  webpage <- read_html(url)

  data.frame(
    # Address
    address = html_nodes(webpage, ".slt_address a") %>%
      html_text(),

    # Number of bedrooms
    n_beds = html_nodes(webpage, ".slt_beds") %>%
      html_text(),

    # Number of bathrooms
    n_baths = html_nodes(webpage, ".slt_baths") %>%
      html_text(),

    # Listing prices
    prices = html_nodes(webpage, ".slt_price") %>%
      html_text()
  )
}
```

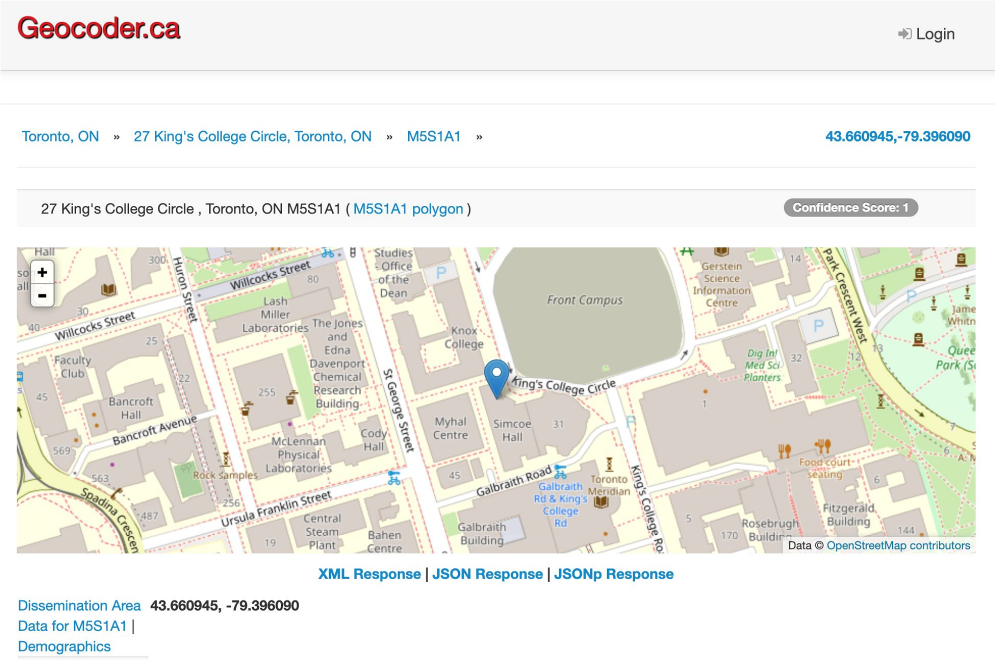
> listings data

	address	n_beds	n_baths	prices	listing_type
1	9973 Keele St 210	1+1 beds	1 baths	$549,000 CAD	condo
2	3975 Grand Park Dr 1309	2+1 beds	2 baths	$649,900 CAD	condo
3	3650 Kingston Rd 114	0 beds	1 baths	$359,999 CAD	condo
4	38 Lee Centre Dr 803	1 beds	1 baths	$395,000 CAD	condo
5	40 Homewood Ave 514	1 beds	1 baths	$499,900 CAD	condo
6	665 Kennedy Rd 704	3 beds	2 baths	$525,000 CAD	condo
7	880 Grandview Way 1001	3 beds	2 baths	$869,000 CAD	condo

> obtaining geocoding data

We can use public API's like geocoder.ca to obtain postal code, longitude, and latitude from the street address

- Supports HTTP request with JSON response
- Public API has a daily rate limiter



https://geocoder.ca/

> Google's Geocoding API

https://developers.google.com/maps/documentation/geocoding/overview

```
# API request format
https://maps.googleapis.com/maps/api/geocode/json?
address=1600+Amphitheatre+Parkway,+Mountain+View,+CA&key=YOUR_API_KEY

# API JSON response
{
   "results" : [
      {
         "address_components" : [
            ...,
            {
               "long_name" : "94043",
               "short_name" : "94043",
               "types" : [ "postal_code" ]
            }
         ],
         "formatted_address" : "1600 Amphitheatre Pkwy, Mountain View, CA 94043, USA",
         "geometry" : {
            "location" : {
               "lat" : 37.4267861,
               "lng" : -122.0806032
            },
            ...
         },
         ...
      }
   ],
   "status" : "OK"
}
```

> get postal code, long/latitude

```r
#' Get postal code, longitude, and latitude data using Google Geocoding API
get_geocode_data <- function(address){

  # Replace spaces with + to match API specification
  address <- str_replace_all(address, " ", "+")

  # Get personal Google Cloud API Key
  API_KEY <- Sys.getenv("GCLOUD_API_KEY")

  # Fetch response from API
  res <- httr::GET(url = glue::glue(
    "https://maps.googleapis.com/maps/api/geocode/json?address={address},+ON,+Canada&key=
{API_KEY}"
  ))

  content <- httr::content(res, as = "text") %>%
    jsonlite::fromJSON()

  return(content)
}
```

> data processing

> data processing overview

- Numeric variables
  - Strip all non-digit characters (eg. $ sign from price)
- Categorical variables
  - Collapse rare categories (eg. rural municipalities/postal codes)
  - One-hot encoding

> data processing recipe

- Data processing steps need to be reproduced for all future data!
    - Eg. if you normalized a numeric variable, you need to retain the mean and SD to normalize future data
    - Eg. for categorical variables, you need to recreate all one-hot encoded columns to match training data
- Need to create a “recipe” that calculates this metadata from the training set, and store it for future use

> data processing recipe

- In R, you can use the recipes package

```
library(recipes)
recipe <- recipe( ~ n_beds + n_baths + listing_type + locality + latitude +
                    longitude + postal_code_abb,
                  data = housing_data) %>%
  step_dummy(listing_type, locality, postal_code_abb)

saveRDS(recipe, "preprocessing_recipe.rds")
```

> model development

> basic regression model

Regression outcome: listing price

Regression predictors:

- n_beds, n_baths (integer)
- listing_type (listing_type_condo, listing_type_townhome, ...)
- locality (locality_Toronto, locality_Mississauga, locality_Oakville, …)
- postal_code (postal_code_M5S, postal_code_L4B, …)
- latitude, longitude (numeric)

> model **object**

Class of xgb.Booster, with predict method:

- Expects an input vector/matrix containing feature values to predict on
- Input data should be similarly processed as training data (reuse data processing recipe)
- Returns model predictions

> model deployment

> model deployment strategies

1. Generate batch predictions at regular intervals (eg. daily, weekly)
2. Build a graphical interface (eg. web app) that allows users to interact with the model
3. Build an application programming interface (ie. API) that allows users to use the model programmatically

> batch predictions

- Users to send batches of new data to generate predictions on
- Data elements must be similar to the scraped data from listing.ca (street address, n_beds, n_baths)
- Process data similarly to training pipeline (ie. use geocoding API to fetch postal code, longitude/latitude)
- Execute predict function in R

> batch predictions

Pros:

1. No additional code needs to be written
2. Simpler pipeline, less chance for software bugs

Cons:

1. Manual process
2. User is reliant on the data scientist to get results (ie. you will get lots of emails)

> batch predictions

In practice, this could look like:

- Manually re-running the prediction pipeline upon new requests
- Scheduling a prediction pipeline R script to run regularly, using CRON/Windows Task Scheduler

> graphical interface

- A graphical interface that allows users to interact directly with the model using familiar tools (eg. web browser)
- Web applications are a popular choice
- Model object can be bundled with application
- Users can enter data directly into input fields
- Application server can receive requests and generate real-time predictions

> graphical user interface

Pros:

1. Data scientist has control over the data format (eg. via use of dropdown menus)
2. Users have on-demand access to the model

Cons:

1. Requires more work to develop and maintain an application
2. Requires a way to share application to users
3. Not a flexible way of interacting with the model (eg. hard to scale up to thousands of predictions)

> graphical interface

In practice, this could look like:

- Use R + Shiny to build a web application
- Deploy the application to a server (eg. shinyapps.io)
- Share the application URL to user
- User accesses the application over the Internet/Intranet

> application programming interface

- An application programming interface (API) is a way to share your model for programmatic usage
- APIs allow developers to use your program (or model) in their programs
- An API accepts some input data (eg. an input vector of data) and returns some output data -> a model object is technically an API!
- Eg. geocoder.ca is supported by an API that allows users to enter a street address (ie. input data) and returns some geocoding data related to that address (ie. output data)

> application programming interface

Pros:

1. Users have on-demand access to the model
2. Users can flexibly interact with your model (eg. if they are Python users, they can still easily use your R model)

Cons:

1. Requires additional work to build an API
2. Users need more technical knowledge (eg. how to submit an API request) to use your model
3. Requires a way to deploy and expose your

> application programming interface

In practice, this could look like:

- Using R + Plumber to create a REST API that serves your model as an API endpoint
- Use Docker to wrap API into a containerized deployment
- Deploy Docker image to cloud service (eg. Google Cloud Run) or local server
- Provide users an API endpoint URL that they can use to send API requests, and documentation on how to format their queries

> deployment strategies showdown

	Batch predictions	Graphical interface	API
The Good	• Low entry barrier	• Good for non-technical end users	• Good for technical end users • Flexible and scalable
The Bad	• Dependent on data scientist • Potential data quality issues	• Can’t scale up for high volume predictions	• Requires technical knowledge

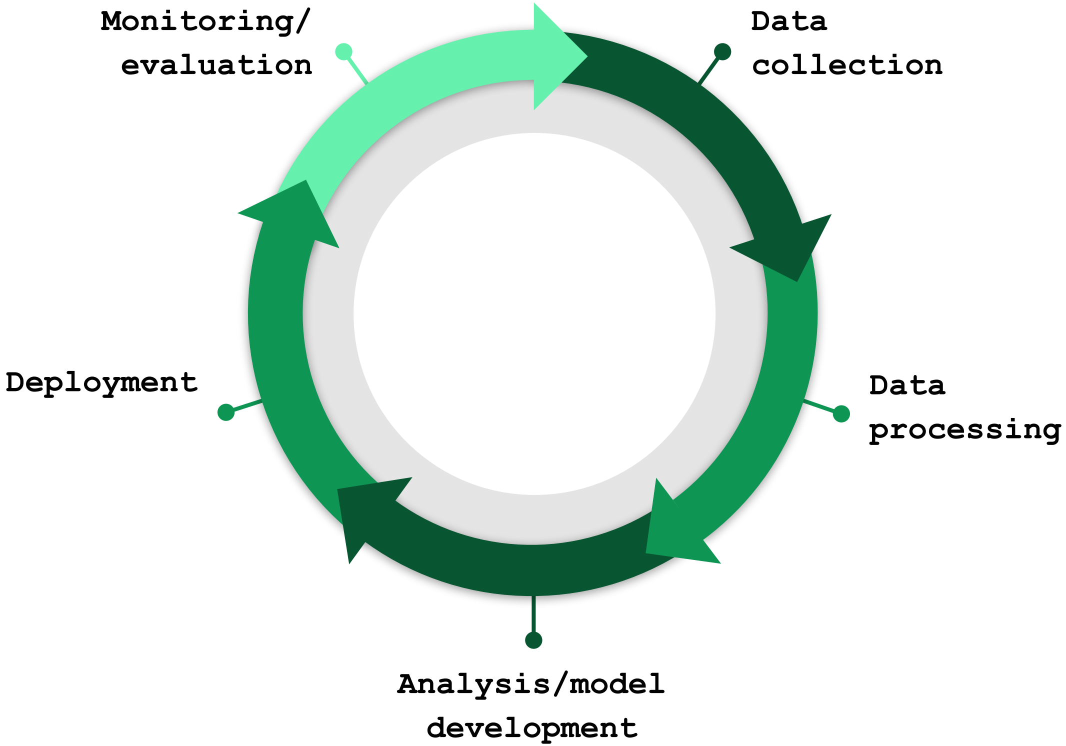
> post-deployment… what now?

If you have made it this far in your project, that’s farther than the vast majority of data science projects! 👏

Post-deployment tasks:

1. Monitoring
   a. Input data distributions (eg. are there a lot of requests for non-Ontario homes?)
   b. Model performance
2. Model retraining
   a. Model performance has degraded
   b. Additional data is available

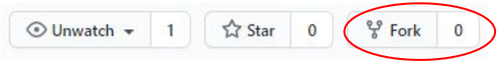
> data science loop



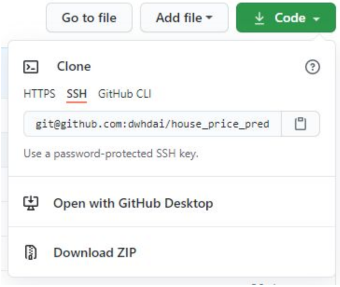
> let’s deploy an app!

> clone example app

1. Go to https://github.com/dwhdai/house_price_predictor. Create a fork of this repository



2. Go to the forked repository (should be under https://github.com/your_username/house_price_predictor)
3. Copy the remote URL for the repository



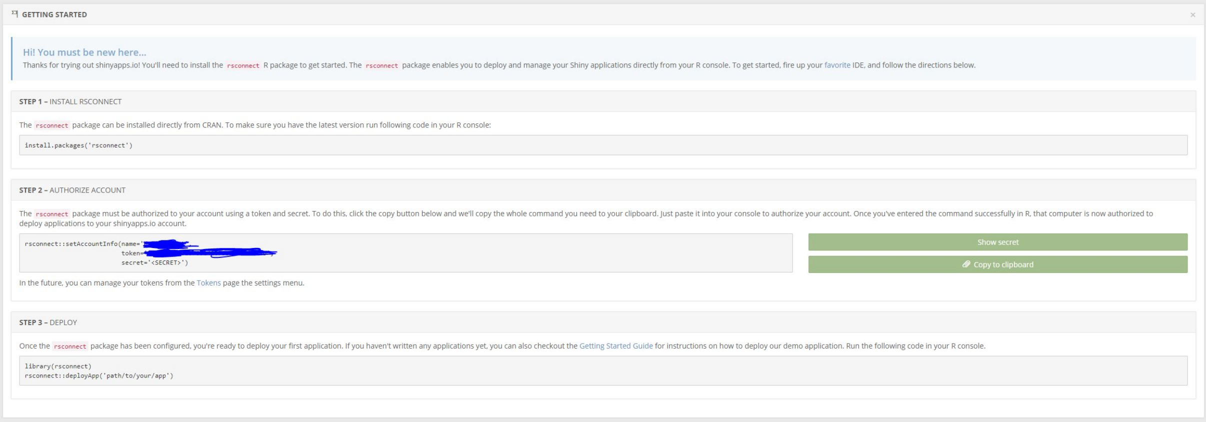
> clone example app

4. Clone the repository and open the project in RStudio.

5. Run renv::restore() in the R console to install package dependencies. This may take a while.

> set up shinyapps.io

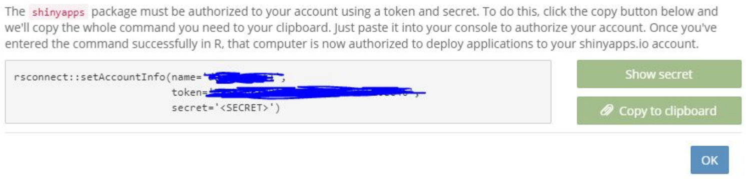
1. Go to shinyapps.io, and create an account (if you don’t already have one)
2. Follow the “Get Started” steps (should look like below)



> set up shinyapps.io

If you already have a shinyapps.io account and have not configured it with RStudio:

1. Log in to your shinyapps.io account.
2. Click the user icon in the top-right; click “Tokens”.
3. Create a new token, and copy the following command:



4. Execute the command in the RStudio R console.

> deploy the app!

1. Switch back to RStudio. Make sure you are on the `housing_prices` project.
2. Install the rsconnect package (`install.packages("rsconnect")`), if not already installed.
3. Run `rsconnect::deployApp(appDir = "03_app/", appName = "name_of_your_app")`
4. 🎉🎉🎉

Resources

	R	Python
Automation	• tasksheduleR	• apscheduler
Web application	• Shiny • shinyapps.io	• Streamlit • Streamlit Sharing (beta)
REST API	• Plumber	• Flask
HTTP requests	• httr	• requests
JSON parsing	• jsonlite	• json

> ?questions

David Dai

twitter: @dwhdai

davidwh.dai@gmail.com