What did the database say to R?

Let's talk: How to talk to databases in R using (almost) no SQL

Sam Albers
Knowledge Management Branch

BC Ministry of Environment and Climate Change Strategy

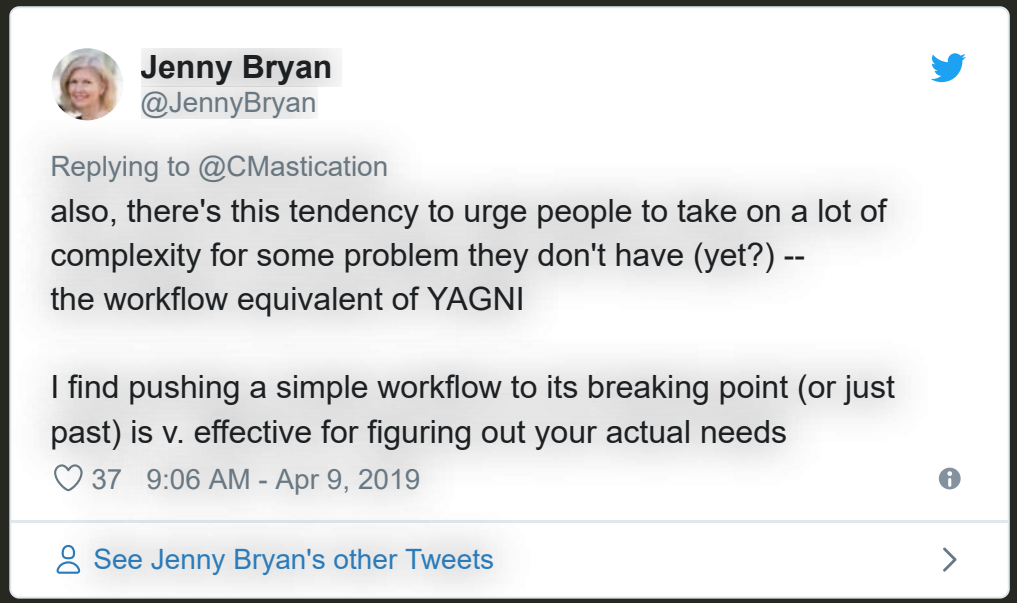
bcgov Data Science Demo Day

2019-04-17

Which data science tool?

Guiding Principles

> I find pushing a simple workflow to its breaking point (or just past) is v. effective for figuring out your actual needs



Guiding Principles

Disclaimers

- I think programmatic approaches are best
- Only make it as hard as it needs to be
- This is how I know how to solve this problem
- There is no single best way to accomplish this

For instance...

```
package_db <- as_tibble(tools::CRAN_package_db(), .name_repair = "unique")

filter(package_db, str_detect(Package, "xl")) %>%
  pull(Package)
```

```
##  [1] "dataframes2xls" "maxlike"        "mixlink"        "mixlm"          "openxlsx"
##  [6] "popprxl"        "readxl"         "tablaxlsx"      "table1xls"      "taxlist"
## [11] "tidyxl"         "writexl"        "xlsimple"       "xlsx"           "xlsxjars"
## [16] "xltabr"         "xlutils3"
```

Road map

The Problem

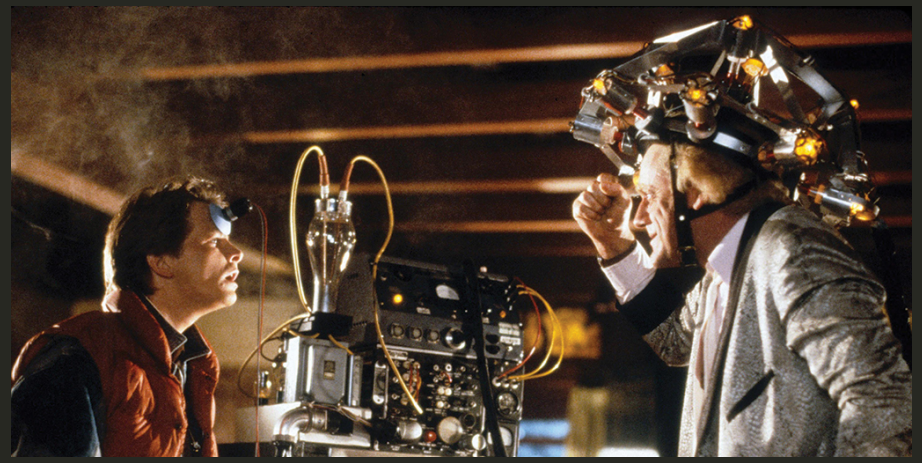
An in-memory example

A small database example

A very very large database example

The Problem

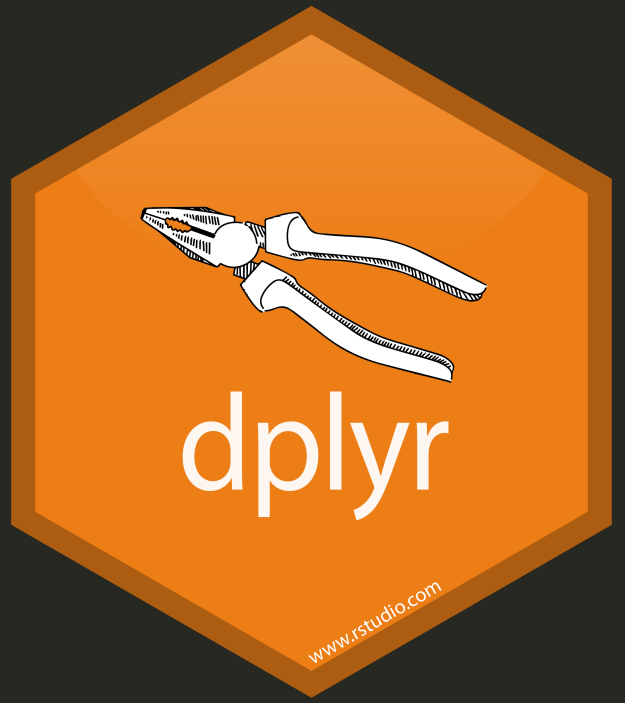
- Many data science tasks are repetitive yet interactive
- Helpful to abstract away unneeded complexity when possible
- A clean and easy to remember syntax reduces your cognitive load when doing data science



Enter `dplyr`

> a consistent set of verbs that help you solve the most common data manipulation challenges

- Independent of the data source
- Designed for data science



`dplyr` verbs

Functions with English meanings that map directly to the action being taken when that function is called

- `mutate()` adds new variables that are functions of existing variables
- `select()` picks variables based on their names.
- `filter()` picks cases based on their values.
- `summarise()` reduces multiple values down to a single summary.
- `arrange()` changes the ordering of the rows.

Artwork by @allison_horst

An in-memory Example

```r
library(dplyr)
```

```r
starwars
```

```
## # A tibble: 87 x 13
##    name  height  mass hair_color skin_color eye_color birth_year gender homeworld species
##    <chr>  <int> <dbl> <chr>      <chr>      <chr>           <dbl> <chr>  <chr>     <chr>
##  1 Luke~    172    77 blond      fair       blue             19   male   Tatooine  Human
##  2 C-3PO    167    75 <NA>       gold       yellow          112   <NA>   Tatooine  Droid
##  3 R2-D2     96    32 <NA>       white, bl~ red              33   <NA>   Naboo     Droid
##  4 Dart~    202   136 none       white      yellow           41.9 male   Tatooine  Human
##  5 Leia~    150    49 brown      light      brown            19   female Alderaan  Human
##  6 Owen~    178   120 brown, gr~ light      blue             52   male   Tatooine  Human
##  7 Beru~    165    75 brown      light      blue             47   female Tatooine  Human
##  8 R5-D4     97    32 <NA>       white, red red              NA   <NA>   Tatooine  Droid
##  9 Bigg~    183    84 black      light      brown            24   male   Tatooine  Human
## 10 Obi-~    182    77 auburn, w~ fair       blue-gray        57   male   Stewjon   Human
## # ... with 77 more rows, and 3 more variables: films <list>, vehicles <list>,
## #   starships <list>
```

An API that grew - base R

```
starwars_sub <- starwars[starwars$eye_color == "yellow",]

starwars_sub$hair_color <- factor(starwars_sub$hair_color, exclude = "")

setNames(aggregate(height ~ hair_color, starwars_sub, mean), c("hair_color", "mean_height"))
```

```
##   hair_color mean_height
## 1      black         137
## 2     blonde         168
## 3       grey         170
## 4       none         186
## 5      white         198
## 6       <NA>         167
```

An API that was designed - dplyr

```
starwars %>% tally()
```

```
## # A tibble: 1 x 1
##       n
##   <int>
## 1    87
```

```
starwars %>%
  filter(eye_color == "yellow") %>%
  group_by(hair_color) %>%
  summarise(mean_height = mean(height))
```

```
## # A tibble: 6 x 2
##   hair_color mean_height
##   <chr>            <dbl>
## 1 <NA>               167
## 2 black              137
## 3 blonde             168
## 4 grey               170
## 5 none               186
## 6 white              198
```

Why use a database?

- That's where the data is
- Bigger data than what you can handle locally
- Same common tasks can be handled by the database

Why not?

- Cost to learn SQL from scratch
- Diverse landscape

--

How?

- Normally you query a database with `SQL` code:

```sql
SELECT genus, species
FROM species
WHERE taxa = 'Bird'
ORDER BY species_id ASC;
```

- `dplyr` translate to `SQL` - Take advantage of readable code
- Transfer those skills to a database without learning any `SQL`
- Only ever pull the data when to explicitly ask for it
- Lower cognitive cost of switching languages: Stay in R!

Road map

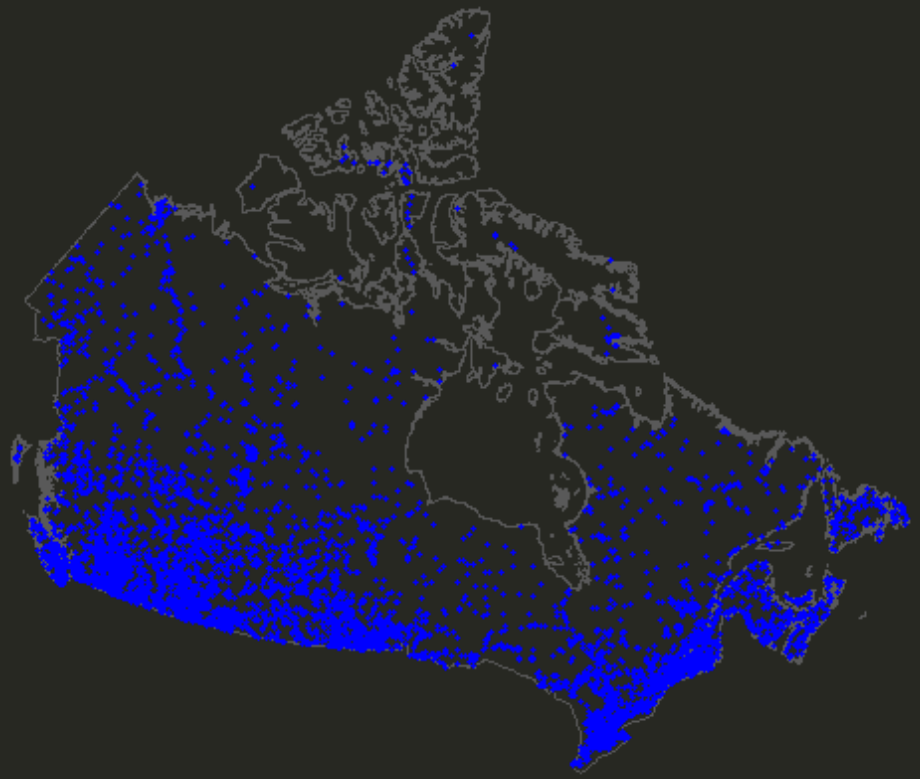
The Problem

An in-memory example

A small database example

A very very large database example

River database



```
fs::file_size("Hydat.sqlite3")
```

```
## 1.01G
```

7831 stations

SQLite database

Self contained

River database

```
library(DBI)
library(RSQLite)

con <- dbConnect(SQLite(), "Hydat.sqlite3")

dbListTables(con)
```

```
##  [1] "AGENCY_LIST"            "ANNUAL_INSTANT_PEAKS"   "ANNUAL_STATISTICS"
##  [4] "CONCENTRATION_SYMBOLS"  "DATA_SYMBOLS"           "DATA_TYPES"
##  [7] "DATUM_LIST"             "DLY_FLOWS"              "DLY_LEVELS"
## [10] "MEASUREMENT_CODES"      "OPERATION_CODES"        "PEAK_CODES"
## [13] "PRECISION_CODES"        "REGIONAL_OFFICE_LIST"   "SAMPLE_REMARK_CODES"
## [16] "SED_DATA_TYPES"         "SED_DLY_LOADS"          "SED_DLY_SUSCON"
## [19] "SED_SAMPLES"            "SED_SAMPLES_PSD"        "SED_VERTICAL_LOCATION"
## [22] "SED_VERTICAL_SYMBOLS"   "STATIONS"               "STN_DATA_COLLECTION"
## [25] "STN_DATA_RANGE"         "STN_DATUM_CONVERSION"   "STN_DATUM_UNRELATED"
## [28] "STN_OPERATION_SCHEDULE" "STN_REGULATION"         "STN_REMARKS"
## [31] "STN_REMARK_CODES"       "STN_STATUS_CODES"       "VERSION"
```

```
tbl(con, "DLY_FLOWS") %>% tally()
```

```
## # Source:   lazy query [?? x 1]
## # Database: sqlite 3.22.0 [C:\_dev\presentations\database_ds_cop\Hydat.sqlite3]
##         n
##     <int>
## 1 1584069
```

```
tbl(con, "DLY_FLOWS") %>%
  filter(YEAR >= 2000) %>%
  group_by(STATION_NUMBER, YEAR) %>%
  summarise(annual_mean = mean(MONTHLY_MEAN, na.rm = TRUE))
```

```
## # Source:   lazy query [?? x 3]
## # Database: sqlite 3.22.0 [C:\_dev\presentations\database_ds_cop\Hydat.sqlite3]
## # Groups:   STATION_NUMBER
##    STATION_NUMBER  YEAR annual_mean
##    <chr>          <int>       <dbl>
##  1 01AD002         2000        278.
##  2 01AD002         2001        200.
##  3 01AD002         2002        193.
##  4 01AD002         2003        304.
##  5 01AD002         2004        273.
##  6 01AD002         2005        393.
##  7 01AD002         2006        358.
##  8 01AD002         2007        269.
```

```r
tbl(con, "DLY_FLOWS") %>% tally() %>% show_query()
```

```
## <SQL>
## SELECT COUNT() AS `n`
## FROM `DLY_FLOWS`
```

```r
tbl(con, "DLY_FLOWS") %>%
  filter(YEAR >= 2000) %>%
  group_by(STATION_NUMBER, YEAR) %>%
  summarise(annual_mean = mean(MONTHLY_MEAN, na.rm = TRUE)) %>%
  show_query()
```

```
## <SQL>
## SELECT `STATION_NUMBER`, `YEAR`, AVG(`MONTHLY_MEAN`) AS `annual_mean`
## FROM `DLY_FLOWS`
## WHERE (`YEAR` >= 2000.0)
## GROUP BY `STATION_NUMBER`, `YEAR`
```

A comparison

```
DLY_FLOWS_memory <- read_csv("data/DLY_FLOWS.csv")

DLY_FLOWS_memory %>% tally()

DLY_FLOWS_memory %>%
  filter(YEAR >= 1950) %>%
  group_by(STATION_NUMBER, YEAR) %>%
  summarise(annual_mean = mean(MONTHLY_MEAN, na.rm = TRUE))
```

Road map

The Problem

An in-memory example

A small database example

A very very large database example


New York City Yellow Cab Rides

BigQuery

```
library(DBI)
library(bigrquery)
library(modeldb)
library(dbplot)
```

```
con <- dbConnect(
  bigquery(),
  project = "bigquery-public-data",
  dataset = "new_york_taxi_trips",
  billing = "bigrquery-237721",
  use_legacy_sql = FALSE
)

tbl(con, "tlc_yellow_trips_2018") %>%
  tally()
```

```
## # Source:   lazy query [?? x 1]
## # Database: BigQueryConnection
##           n
##       <int>
## 1 63356111
```

Talking to the database

```
tbl(con, "tlc_yellow_trips_2018") %>%
  select(tip_amount) %>%
  top_n(10)
```

```
## Selecting by tip_amount

## # Source:   lazy query [?? x 1]
## # Database: BigQueryConnection
##    tip_amount
##         <dbl>
##  1       496
##  2       496
##  3       442.
##  4       441.
##  5       422
##  6       422
##  7       415
##  8       412.
##  9       411
## 10       410
```

Some data cleaning

```
query <- tbl(con, "tlc_yellow_trips_2018") %>%
  filter(pickup_datetime > "2018-01-01 00:00:00", pickup_datetime < "2018-03-01 00:00:00") %>%
  filter(fare_amount >= 0, tip_amount >= 0, !is.na(fare_amount)) %>%
  filter(fare_amount < 500)
show_query(query)
```

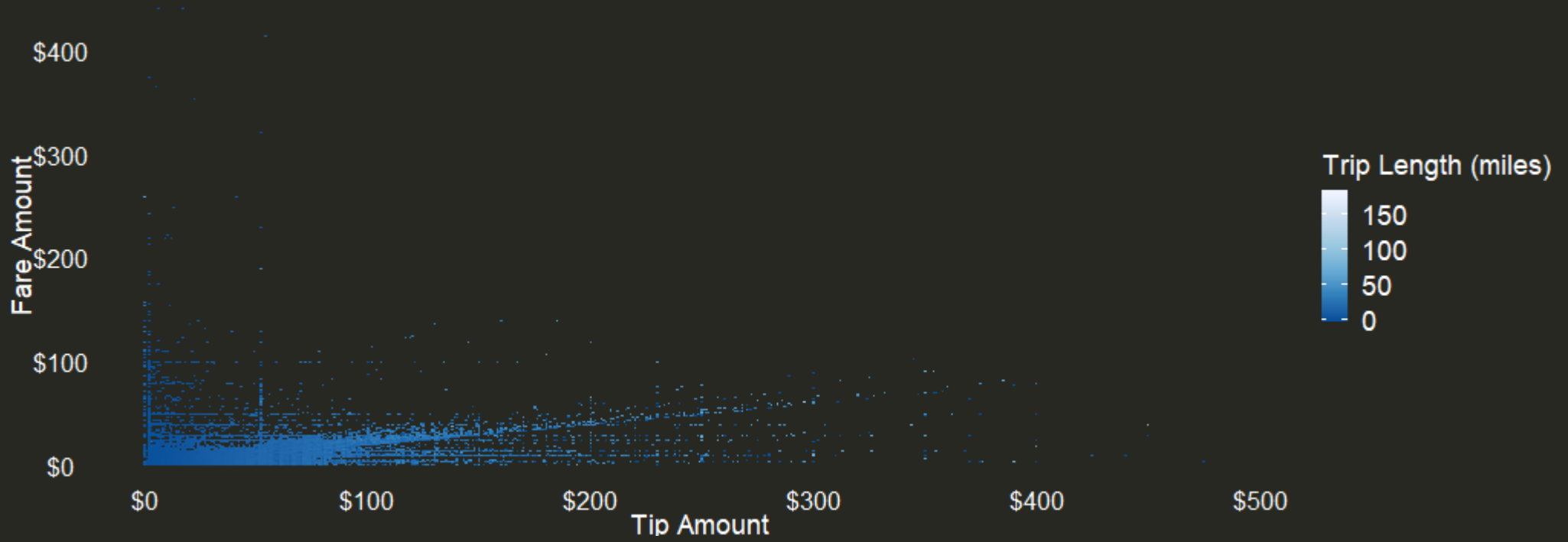
```
## <SQL>
## SELECT *
## FROM (SELECT *
## FROM (SELECT *
## FROM `tlc_yellow_trips_2018`
## WHERE ((`pickup_datetime` > '2018-01-01 00:00:00') AND (`pickup_datetime` < '2018-03-01 00:00:00')))
## WHERE ((`fare_amount` >= 0.0) AND (`tip_amount` >= 0.0) AND (NOT(((`fare_amount`) IS NULL)))))
## WHERE (`fare_amount` < 500.0)
```

```
query %>% tally()
```

```
## # Source:   lazy query [?? x 1]
## # Database: BigQueryConnection
##           n
##       <int>
## 1 17244829
```

New York City Yellow Cab Rides

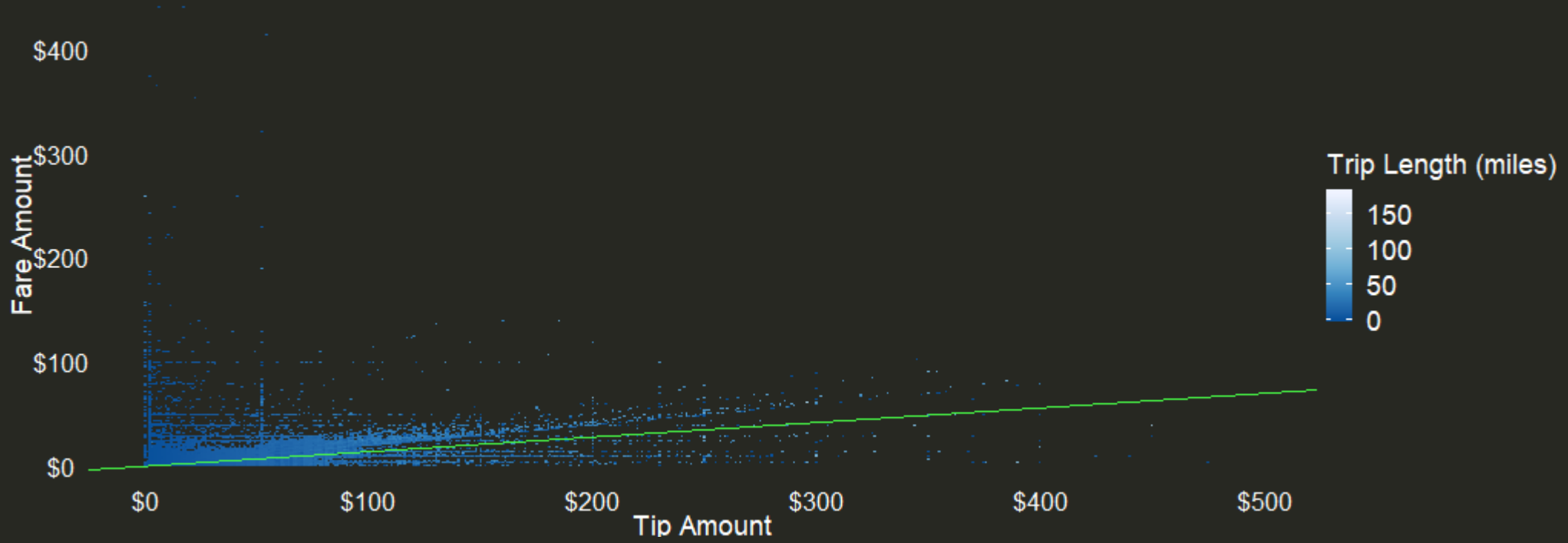
```
(p <- dbplot_raster(query, x = fare_amount, y = tip_amount, resolution = 500,
                    fill = mean(trip_distance, na.rm = TRUE)) +
  scale_fill_distiller(name = "Trip Length (miles)") +
  scale_y_continuous(labels = scales::dollar) +
  scale_x_continuous(labels = scales::dollar) +
  labs(x = "Tip Amount", y = "Fare Amount"))
```



Model 17 million records

```
reg <- query %>%
  select(fare_amount, tip_amount) %>%
  linear_regression_db(tip_amount)

p + geom_abline(intercept = reg$`(Intercept)`, slope = reg$fare_amount, colour = "#48fb47")
```



`sum(presentation, na.rm = TRUE)`

Activities in data sciences are often repetitive

`dplyr` lowers the barriers to accomplishing common data science tasks

Working with databases enhances the power of your computer

If possible let the database do the work for you

Resources

- Best Practices for working with databases
- Databases using R
- RStudio database page
- R for Data Science
- dplyr
- bigquery
- dbplot
- Allison Hill xaringan slides