dplyr

Jeff Allen

Dallas R Users Group
7/11/15

@trestleJeff

My Background

- Computer Scientist
- First encountered R as a programming language (2007)
- Only later used it for data analysis
- Now a Software Engineer at RStudio (2013)

Your Background

- New to R?
- Intermediate-Advanced R user?
- Used dplyr before?

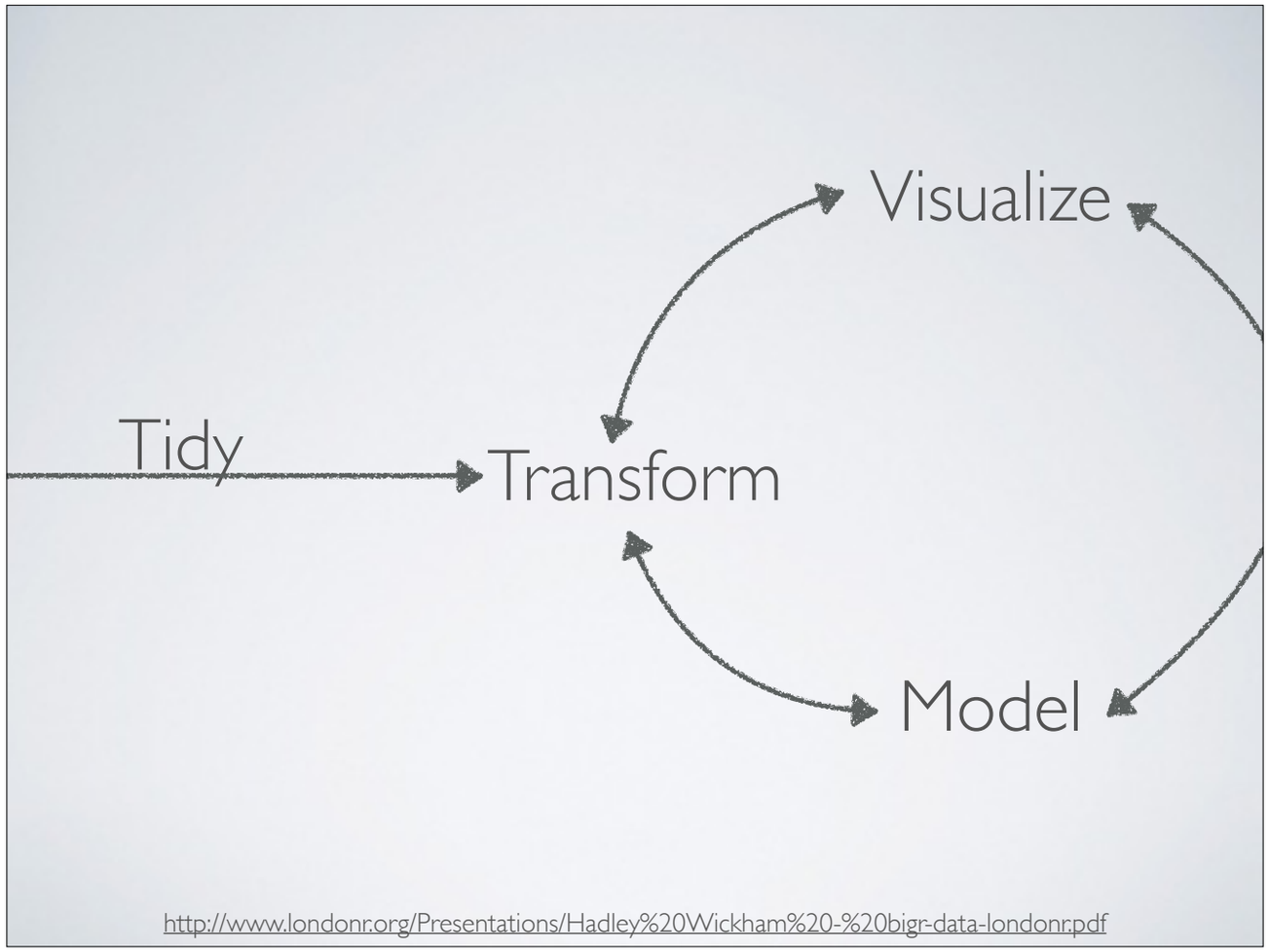
R Consortium

- Support R Core with development and finances
- Organized by Linux Foundation
- New R-forge, documentation, etc.
- https://www.r-consortium.org/

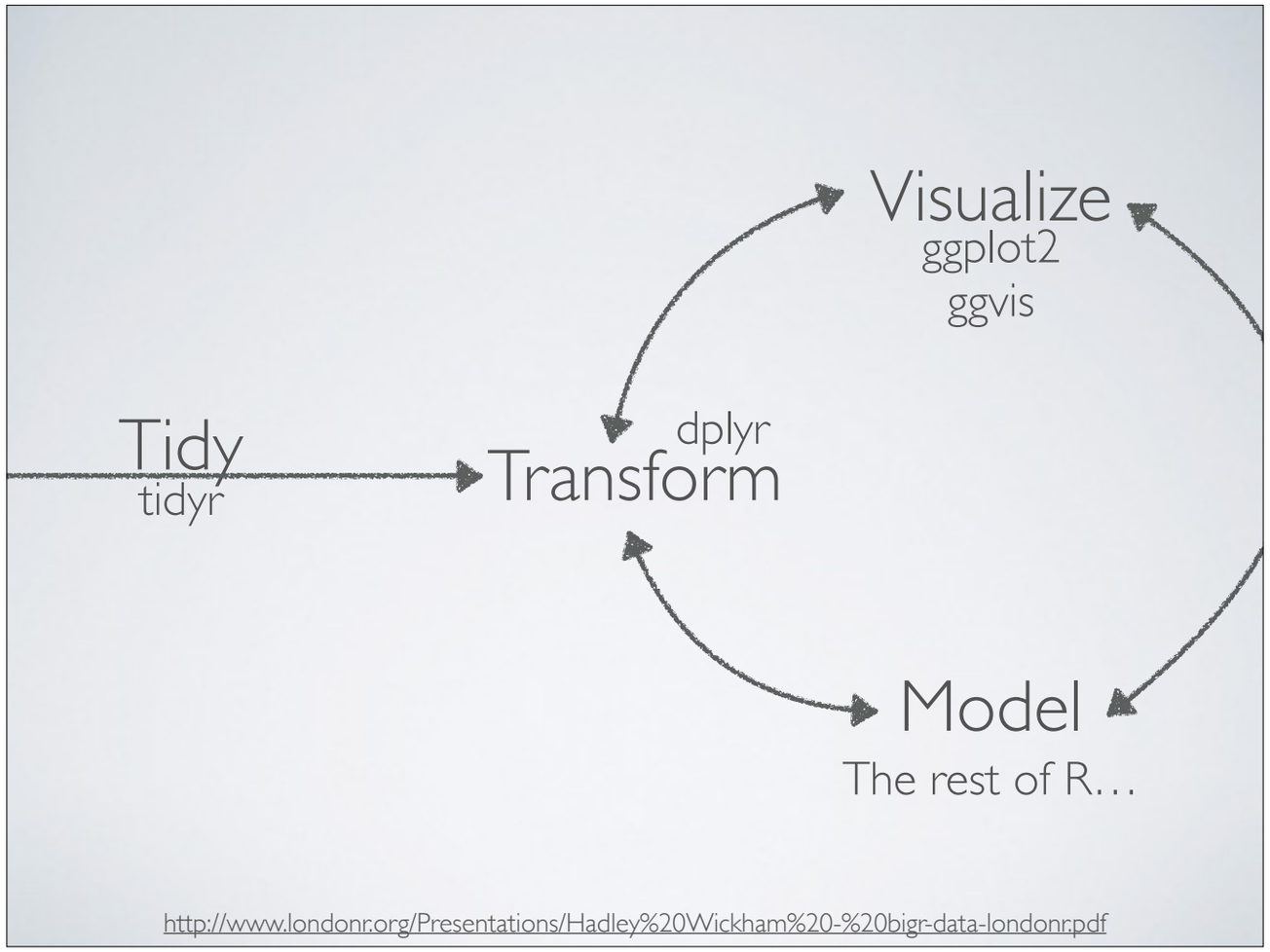
dplyr

- Open-source R package
- From Hadley Wickham (ggplot2, plyr, devtools, …)
- Grammar of data manipulation
  - Operates on data.frames

```
library(dplyr)
set.seed(1234)
```



http://www.londonr.org/Presentations/Hadley%20Wickham%20-%20bigr-data-londonr.pdf


Visualize
ggplot2
ggvis
Tidy
tidyr
dplyr
Transform
Model
The rest of R…
http://www.londonr.org/Presentations/Hadley%20Wickham%20-%20bigr-data-londonr.pdf

Motivation

- Unified syntax, captures 90% of data transformation tasks
- Consistent interface (great for "piping")
- **Performance** (up to100x in certain cases)
- More to come…

Data Intake

- At simplest: A special data.frame
- All the same properties of a data.frame
- `tbl_df(myDataFrame)`

```
# Create an example data.frame
data <- data.frame(
  let=LETTERS[1:26],
  num=sample(1:26, 26, replace=FALSE),
  dat=seq(as.Date("2015-01-01"),
          to = as.Date("2015-07-11"),
          length.out = 26),
  stringsAsFactors = FALSE
)
data

# Convert to a dplyr table
dp <- tbl_df(data)
dp
dp[1,1]
mean(dp$num)
```

Fundamental verbs

- **select** - subset columns
- **filter** - subset rows
- **mutate** - add new columns
- **arrange** - re-order rows
- **summarize** - reduce to single row
- **group_by** - "bin" data

Fundamental verbs

- **select** - subset columns
- **filter** - subset rows
- **mutate** - add new columns
- **arrange** - re-order rows
- **summarize** - reduce to single row
- **group_by** - "bin" data

select

- Take a subset of columns
- Use column names without quotes
- "-" to exclude a variable
- `starts_with(), ends_with(), matches(), …`

```
## SELECT
select(dp, let, dat)
select(dp, -let)
select(dp, ends_with("t"))
```

Fundamental verbs

- **select** - subset columns
- **filter** - subset rows
- **mutate** - add new columns
- **arrange** - re-order rows
- **summarize** - reduce to single row
- **group_by** - "bin" data

filter

- Take a subset of rows
- Use regular R Boolean vector logic

```
# FILTER
filter(dp, let=="H")
filter(dp, num < 5)
filter(dp, num < 5 & let > "J")
```

Fundamental verbs

- **select** - subset columns
- **filter** - subset rows
- **mutate** - add new columns
- **arrange** - re-order rows
- **summarize** - reduce to single row
- **group_by** - "bin" data

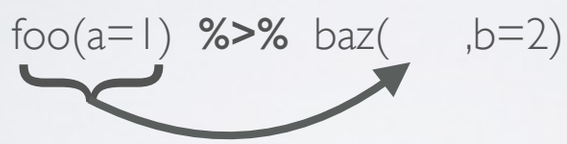
mutate

- Add new columns
- Potentially based on existing columns

```
## MUTATE
mutate(dp, one=1)
mutate(dp, metric = num * 2.54)
mutate(dp, prefixed = paste0("letter", let))
```

A Brief Interruption

- Pipes offer an alternative syntax to nest functions




```
baz(
   foo(a=1),
   b=2
)
```

==

```
foo(a=1) %>%
  baz(b=2)
```

- Comes from the magrittr package

```
## COMBINING & PIPES

# Store disposable variables
datNum <- select(dp, dat, num)
filteredDatNum <- filter(datNum, num < 5)
mutatedFilteredDatNum <- mutate(filteredDatNum, square = num^2)
mutatedFilteredDatNum
# ...

# Functional
mutate(
  filter(
    select(dp, dat, num),
    num < 5
  ), square = num^2
)


# Pipes
```

```
tumble_after(
  broke(
    fell_down(
      fetch(
        went_up(jack_jill, "hill"),
        "water"),
      jack
    ),
    "crown"),
  "jill"
)
```

```
jack_jill %>%
  went_up("hill") %>%
  fetch("water") %>%
  fell_down("jack") %>%
  broke("crown") %>%
  tumble_after("jill")
```

Fundamental verbs

- **select** - subset columns
- **filter** - subset rows
- **mutate** - add new columns
- **arrange** - re-order rows
- **summarize** - reduce to single row
- **group_by** - "bin" data

arrange

- Sort rows
- Use `desc()` to sort in decrementing order

```
## ARRANGE
dp
arrange(dp, num)
dp %>% arrange(num)


sorted <- arrange(dp, desc(dat))
dp %>% arrange(desc(dat))
```

Fundamental verbs

- **select** - subset columns
- **filter** - subset rows
- **mutate** - add new columns
- **arrange** - re-order rows
- **summarize** - reduce to single row
- **group_by** - "bin" data

summarize

- Aggregate data into a single row
- Provide a summarization function for each column you want to keep
- Special functions like `n()` to get the count

```
## SUMMARIZE
summarize(dp, total=sum(num))
dp %>% summarize(total=sum(num))

summarize(dp,
          first=min(dat),
          last=max(dat),
          avg=mean(num))
dp %>%
  summarize(first=min(dat), last=max(dat), avg=mean(num))
```

Fundamental verbs

- **select** - subset columns
- **filter** - subset rows
- **mutate** - add new columns
- **arrange** - re-order rows
- **summarize** - reduce to single row
- **group_by** - "bin" data

group_by

- Bin data into independent sets
- By itself, doesn't change the data
- Perform further actions — such as `summarize()` — independently on each group

```
split <- mutate(dp,firstHalf = num < median(num))
split

grouped <- group_by(split, firstHalf)
grouped # No different!
summarize(grouped, avg=mean(num))

dp %>%
  mutate(firstHalf = num < median(num)) %>%
  group_by(firstHalf) %>%
  summarize(avg=mean(num))
```

nycflights13

NYC Flights

```
# All outgoing flights from NYC (EWR, JFK, and LGA) in 2013
library(nycflights13)
# Includes 5 tables:
flights
weather
planes
airports
airlines

flights
# Group exercise: Show me the planes with the highest
#    median departure delay.
flights %>%
  group_by(tailnum) %>%
  summarize(avg_delay = median(dep_delay, na.rm=TRUE), count=n()) %>%
  filter(count > 10) %>%
  arrange(desc(avg_delay)) %>%
```

Joins

- Bind data from two tables together
- left_join(), right_join(), inner_join(), full_join(), …
- Concatenates columns together for rows that have corresponding keys

```
airlines
flights
flights %>%
  left_join(airlines) %>%
  select(name, flight, origin, dest)

# Naive joining
flights %>%
  filter(month==1 & day==1) %>%
  left_join(planes)

flights %>%
  filter(month==1 & day==1) %>%
  left_join(planes) %>%
  select(tailnum, manufacturer)

# Specify keys
flights %>%
  filter(month==1 & day==1) %>%
```

Joins

User	Age	Dept
joe	41	QA
kim	39	IT
steve	32	IT

Dept	Room#
IT	307
QA	410

Joins

User	Age	Dept
joe	41	QA
kim	39	IT
steve	32	IT

Dept	Room#
IT	307
QA	410

Joins

User	Age	Dept
joe	41	QA
kim	39	IT
steve	32	IT

Dept	Room#
IT	307
QA	410

User	Age	Dept	Room#
joe	41	QA	410
kim	39	IT	307
steve	32	IT	307

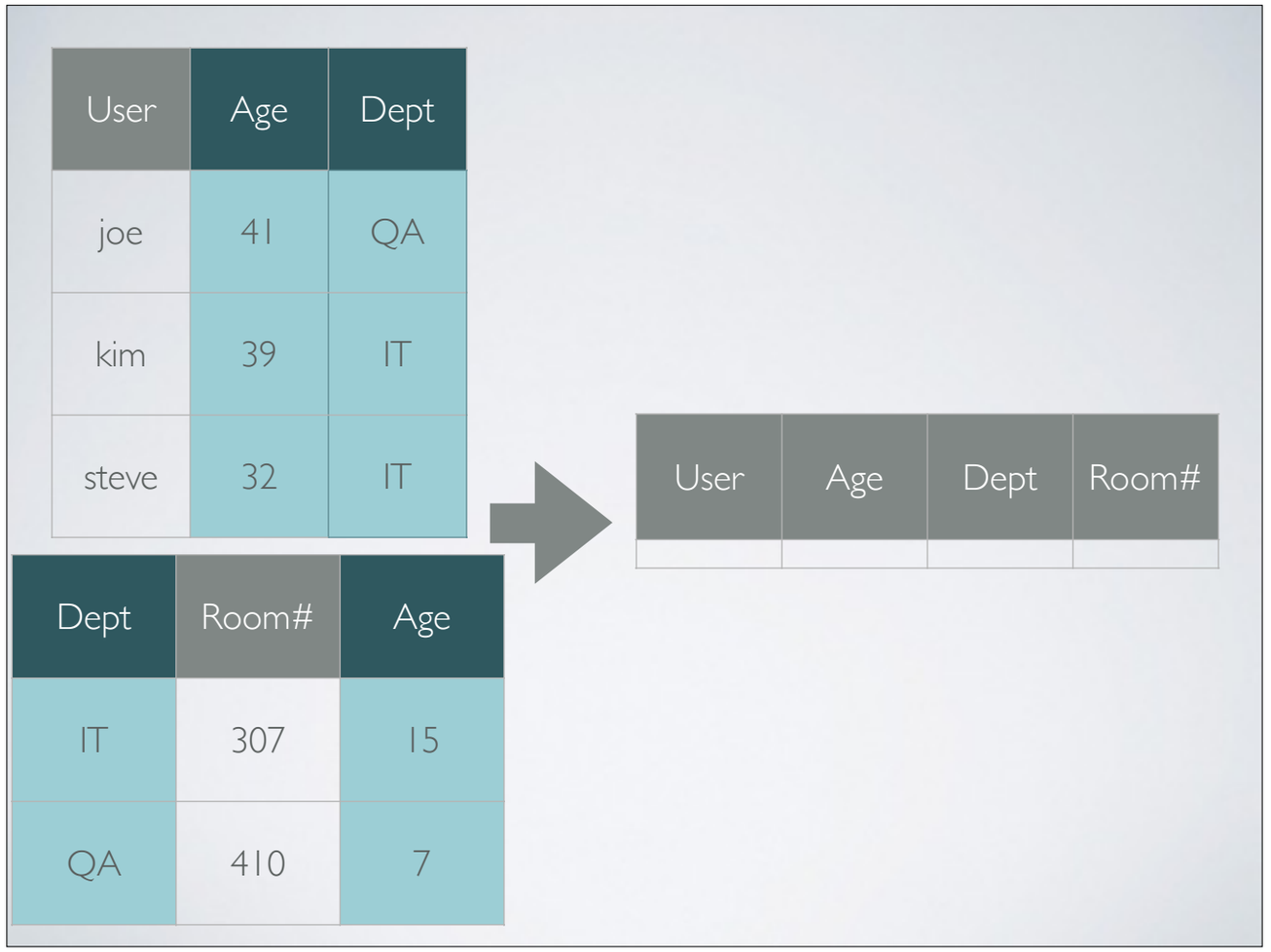
Join Key Collisions

User	Age	Dept
joe	41	QA
kim	39	IT
steve	32	IT

Dept	Room#
IT	307
QA	410

User	Age	Dept
joe	41	QA
kim	39	IT
steve	32	IT

Dept	Room#	Age
IT	307	15
QA	410	7


User
Age
Dept
joe
41
QA
kim
39
IT
steve
32
IT
Dept
Room#
Age
IT
307
15
QA
410
7
User
Age
Dept
Room#

User	Age	Dept
joe	41	QA
kim	39	IT
steve	32	IT

Dept	Room#	Age
IT	307	15
QA	410	7

by="Dept"

User	Age	Dept	Room#
joe	41	QA	410
kim	39	IT	307
steve	32	IT	307

Data Sources

- ✔ Local data.frame or data.table
- ☐ Local SQLite database
- ☐ Remote MySQL/PostgreSQL database
- ☐ Google BigQuery, Amazon RedShift, MonetDB

dplyr + MySQL

- dplyr views MySQL as just another data source
- `translate_sql()` does the behind-the-scenes magic
  - Converts what it can to a SQL query
  - Runs everything else locally in R

```
## REMOTE DATABASES

# To copy the nycflights13 data into MySQL...
# dplyr::copy_nycflights13(src_mysql("nycflights", host="192.168.42.11", password="dallasrug"))



mysql <- src_mysql("nycflights",
                   "192.168.42.11",
                   password="dallasrug")
msFlights <- tbl(mysql, "flights")
msPlanes <- tbl(mysql, "planes")
msAirlines <- tbl(mysql, "airlines")

# Works just like a data.frame-backed table
msFlights
msFlights %>% filter(tailnum=="N14228")

# Laziness
```

Lazy Evaluation

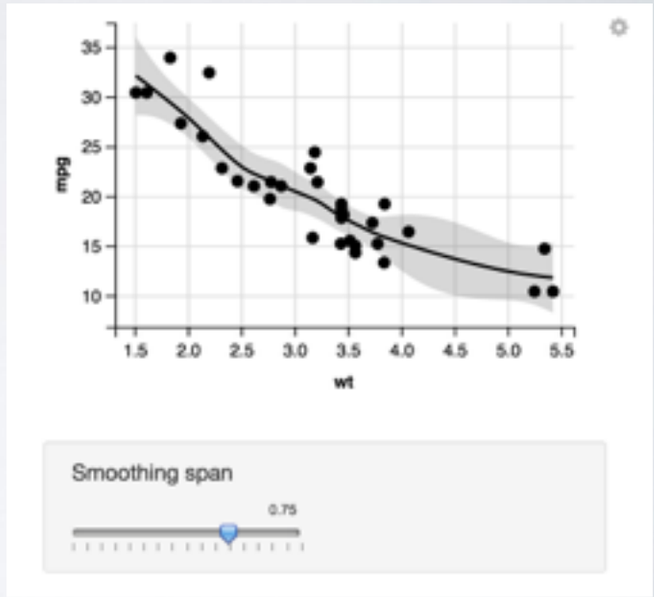
- dplyr avoids executing queries until it absolutely has to
- Use `explain()` to ask the RDBMS about the execution plan for this query.
- Use `collect()` to force evaluation

What's Next?

ggvis

ggvis

- Successor to ggplot2
- Same "grammar of graphics." Updated syntax
- *Of* the Web — runs in a browser
- Built-in reactivity
- Pipeable, like dplyr
- http://ggvis.rstudio.com/

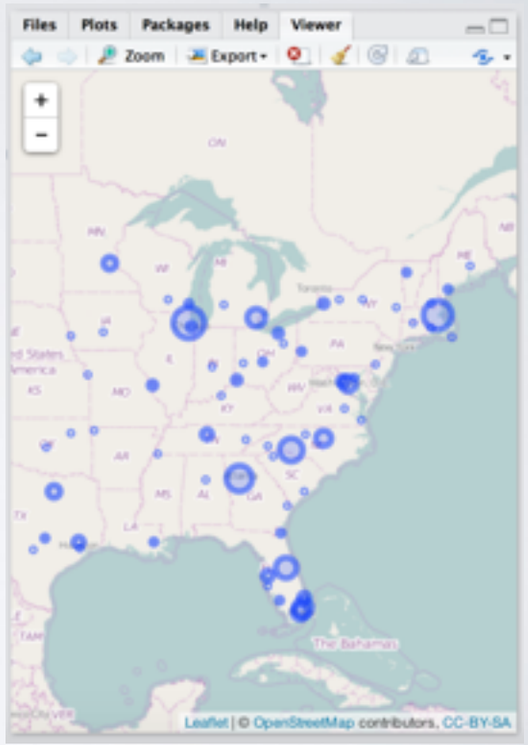



```
## GGVIS

## ggvis
library(ggvis)

flights %>%
  group_by(carrier) %>%
  summarize(avg_delay = mean(dep_delay, na.rm=TRUE)) %>%
  ggvis(~carrier, ~avg_delay) %>%
  layer_bars()

#:= for raw, unscaled value

flights %>%
  filter(month==1, day==1, !is.na(arr_delay), !is.na(dep_delay)) %>%
  ggvis(~arr_delay, ~dep_delay) %>%
  layer_points(opacity := 0.2, size := 20) %>%
  layer_smooths(stroke:="red")
```

leaflet

leaflet

- R package for creating interactive maps
- A new major release recently
- Trivial to use




```
## leaflet

library(leaflet)
airports %>%
  leaflet() %>%
  addTiles() %>%
  addMarkers(~lon, ~lat)

majorAirports <- flights %>%
  group_by(dest) %>%
  summarize(count = n()) %>%
  filter(count > 250) %>%
  inner_join(airports, by=c("dest" = "faa"))

majorAirports %>%
  leaflet() %>%
  addTiles() %>%
  addMarkers(~lon, ~lat, popup=~name)
```