Big Data Analytics

Lecture 7:

Visualization II | Data Storage, Databases Interaction with R

Prof. Dr. Ulrich Matter
22/04/2021

Updates

Status

1. Introduction: Big Data, Data Economy. Walkowiak (2016): Chapter 1.
2. Computation and Memory in Applied Econometrics.
3. Computation and Memory in Applied Econometrics II.*
4. Advanced R Programming. Wickham (2019): Chapters 2, 3, 17,23, 24.
5. Import, Cleaning and Transformation of Big Data. Walkowiak (2016): Chapter 3: p. 74-118.
6. Aggregation and Visualization. Walkowiak (2016): Chapter 3: p. 118-127; Wickham et al.(2015); Schwabish (2014).
7. **Data Visualization Part II & Data Storage, Databases Interaction with R. Walkowiak (2016): Chapter 5.**
8. Cloud Computing: Introduction/Overview, Distributed Systems, Walkowiak (2016): Chapter 4.
9. Applied Econometrics with Spark; Machine Learning and GPUs.
10. Q&A (7 May, 2020).
11. Q&A, Feedback. (14 May, 2020; Hand-in voice-over-slides presentations)

Recap Week 6

Setting

- Data source: NYC Taxi & Limousine Commission (TLC)
- Data on all trip records including pick-up and drop-off times/locations.
  - (2009-2018)
  - Trip-level observations
  - Amount of fare paid
  - Amount of tip paid, etc.
- All raw data: over 200GB
  - **Here: First 1 million observations (in January 2009)**

Data aggregation: The ‘split-apply-combine’ strategy

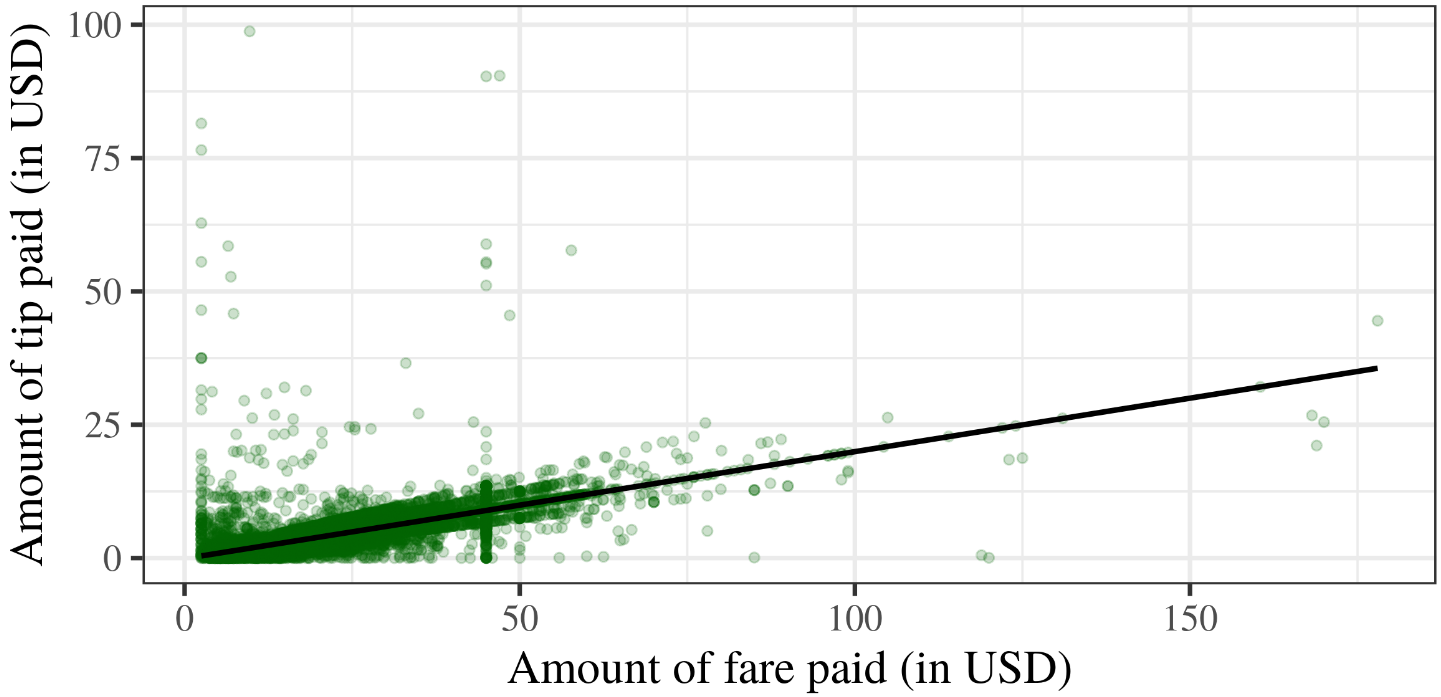
- Background: Compute a statistic for specific groups (e.g. women vs men, etc.)

1. Split the data into subsamples (e.g. one for women, one for men)
2. Compute the statistic for each of the subsamples.
3. Combine all results in one table.

Tow approaches discussed

- Data aggregation with chunked data files (`ff`)
- High-speed in-memory data aggregation with `data.table`

Visualization: Grammar of Graphics/`ggplot2`



Data Visualization Part II

Visualization of spatial data with `ggplot2`

- Data source: NYC Taxi & Limousine Commission (TLC).
- Data on all trip records including **pick-up and drop-off times/locations**.

Preparations

- Load packages for GIS data/operations

```
# load GIS packages
library(rgdal)
library(rgeos)
```

Download map data

```r
# download the zipped shapefile to a temporary file, unzip
URL <- "https://www1.nyc.gov/assets/planning/download/zip/data-maps/open-data/nycd_19a.zip"
tmp_file <- tempfile()
download.file(URL, tmp_file)
file_path <- unzip(tmp_file, exdir= "../data")
# delete the temporary file
unlink(tmp_file)
```

Import map data

```r
# read GIS data
nyc_map <- readOGR(file_path[1], verbose = FALSE)

# have a look at the polygons that constitute the map
summary(nyc_map)
```

```
## Object of class SpatialPolygonsDataFrame
## Coordinates:
##        min       max
## x 913175.1 1067382.5
## y 120121.9  272844.3
## Is projected: TRUE
## proj4string :
## [+proj=lcc +lat_0=40.1666666666667 +lon_0=-74 +lat_1=41.0333333333333
## +lat_2=40.6666666666667 +x_0=300000 +y_0=0 +datum=NAD83 +units=us-ft +no_defs]
## Data attributes:
##      BoroCD        Shape_Leng       Shape_Area
##  Min.   :101.0   Min.   : 23963   Min.   : 24293239
##  1st Qu.:205.5   1st Qu.: 36611   1st Qu.: 48407357
##  Median :308.0   Median : 52246   Median : 82702417
##  Mean   :297.2   Mean   : 74890   Mean   :118724012
##  3rd Qu.:405.5   3rd Qu.: 85711   3rd Qu.:136615357
##  Max.   :595.0   Max.   :270660   Max.   :599062130
```

Change map projection

```r
# transform the projection
nyc_map <- spTransform(nyc_map,
                       CRS("+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0"))
# check result
summary(nyc_map)
```

```
## Object of class SpatialPolygonsDataFrame
## Coordinates:
##         min       max
## x -74.25559 -73.70001
## y  40.49612  40.91553
## Is projected: FALSE
## proj4string : [+proj=longlat +datum=WGS84 +no_defs]
## Data attributes:
##      BoroCD        Shape_Leng       Shape_Area
##  Min.   :101.0   Min.   : 23963   Min.   : 24293239
##  1st Qu.:205.5   1st Qu.: 36611   1st Qu.: 48407357
##  Median :308.0   Median : 52246   Median : 82702417
##  Mean   :297.2   Mean   : 74890   Mean   :118724012
##  3rd Qu.:405.5   3rd Qu.: 85711   3rd Qu.:136615357
##  Max.   :595.0   Max.   :270660   Max.   :599062130
```

Prepare map for plotting with `ggplot2`

```
nyc_map <- fortify(nyc_map)
```

Prepare pick-up and drop-off data

```
# taxi trips plot data
taxi_trips <- taxi[start_long <= max(nyc_map$long) &
                        start_long >= min(nyc_map$long) &
                        dest_long <= max(nyc_map$long) &
                        dest_long >= min(nyc_map$long) &
                        start_lat <= max(nyc_map$lat) &
                        start_lat >= min(nyc_map$lat) &
                        dest_lat <= max(nyc_map$lat) &
                        dest_lat >= min(nyc_map$lat)
                        ]
taxi_trips <- taxi_trips[sample(nrow(taxi_trips), 50000)]
```

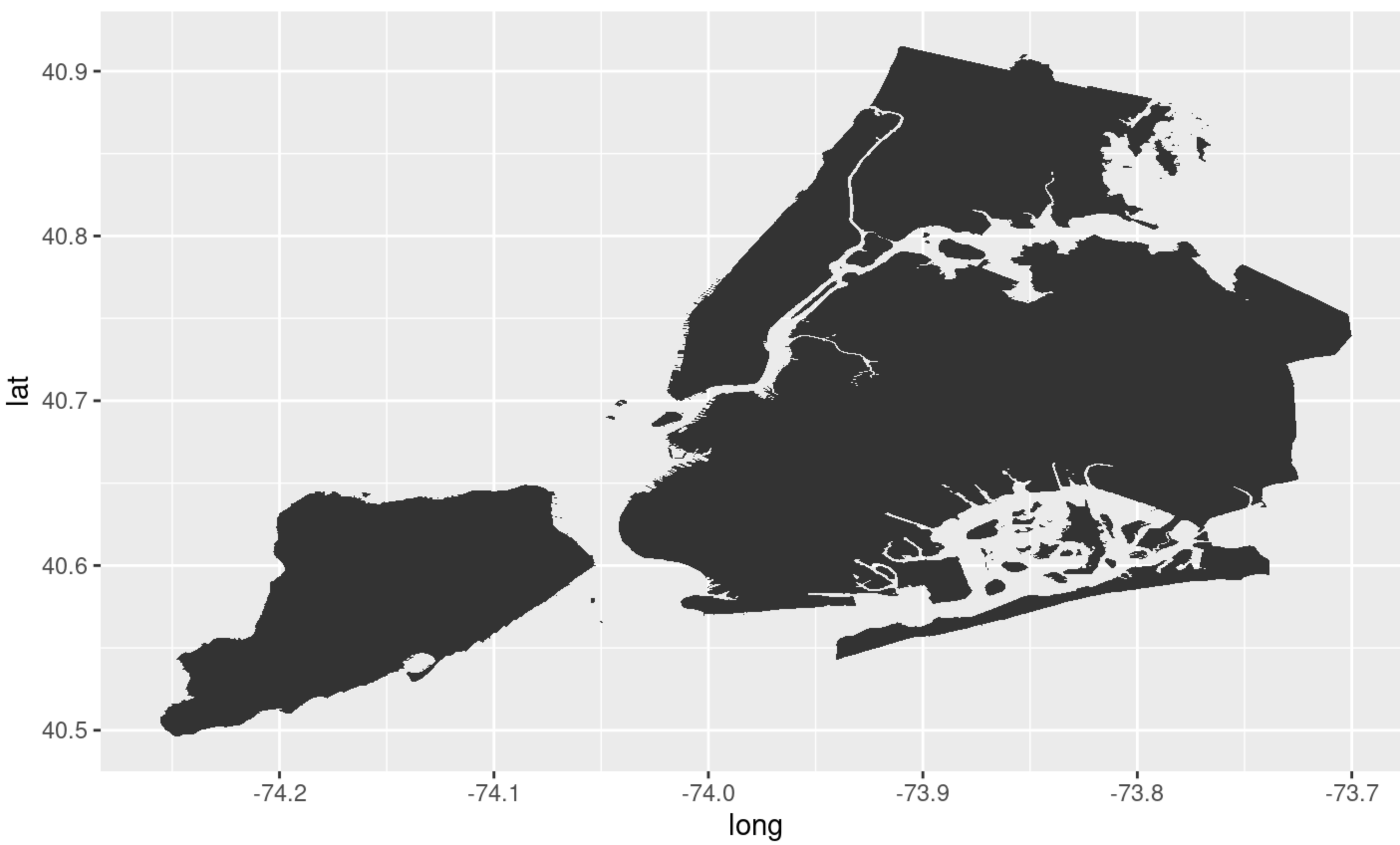
Code time dimension(s)

```
taxi_trips$start_time <- hour(taxi_trips$pickup_time)


# define new variable for facets
taxi_trips$time_of_day <- "Morning"
taxi_trips[start_time > 12 & start_time < 17]$time_of_day <- "Afternoon"
taxi_trips[start_time %in% c(17:24, 0:5)]$time_of_day <- "Evening/Night"
taxi_trips$time_of_day  <- factor(taxi_trips$time_of_day, levels = c("Morning", "Afternoon", "Evenir
```

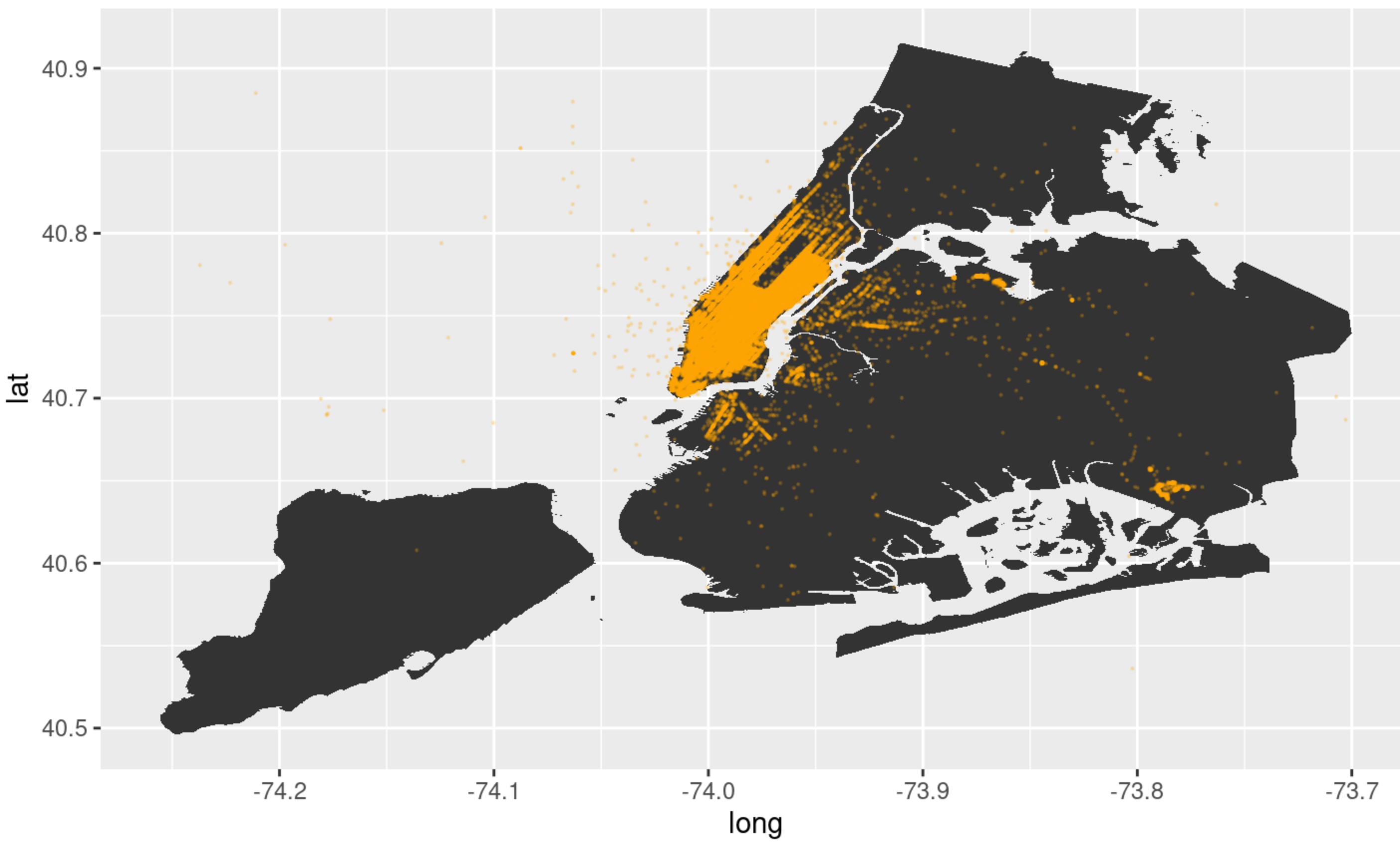
Base plot: Map of NYC

```
# set up the canvas
locations <- ggplot(taxi_trips, aes(x=long, y=lat))
# add the map geometry
locations <- locations + geom_map(data = nyc_map,
                                  map = nyc_map,
                                  aes(map_id = id))
locations
```



Add pick-up locations

```
# add pick-up locations to plot
locations +
     geom_point(aes(x=start_long, y=start_lat),
                color="orange",
                size = 0.1,
                alpha = 0.2)
```

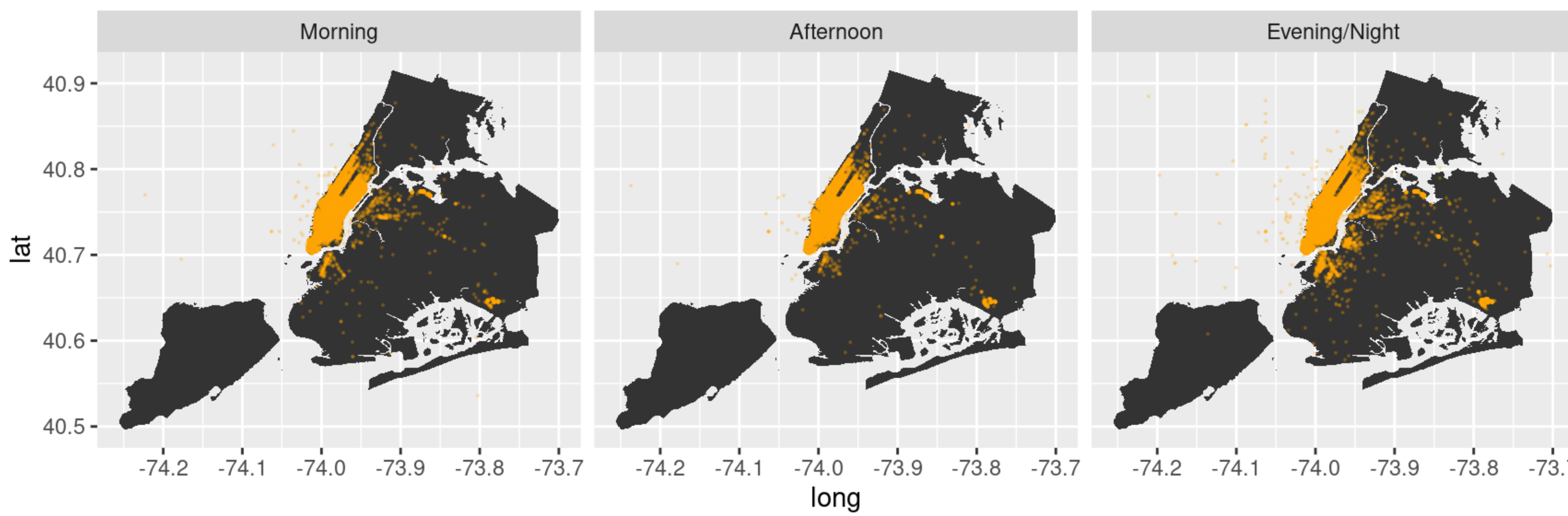


Add drop-off locations

```
# add pick-up locations to plot
locations +
     geom_point(aes(x=dest_long, y=dest_lat),
                color="steelblue",
                size = 0.1,
                alpha = 0.2) +
     geom_point(aes(x=start_long, y=start_lat),
                color="orange",
                size = 0.1,
                alpha = 0.2)
```

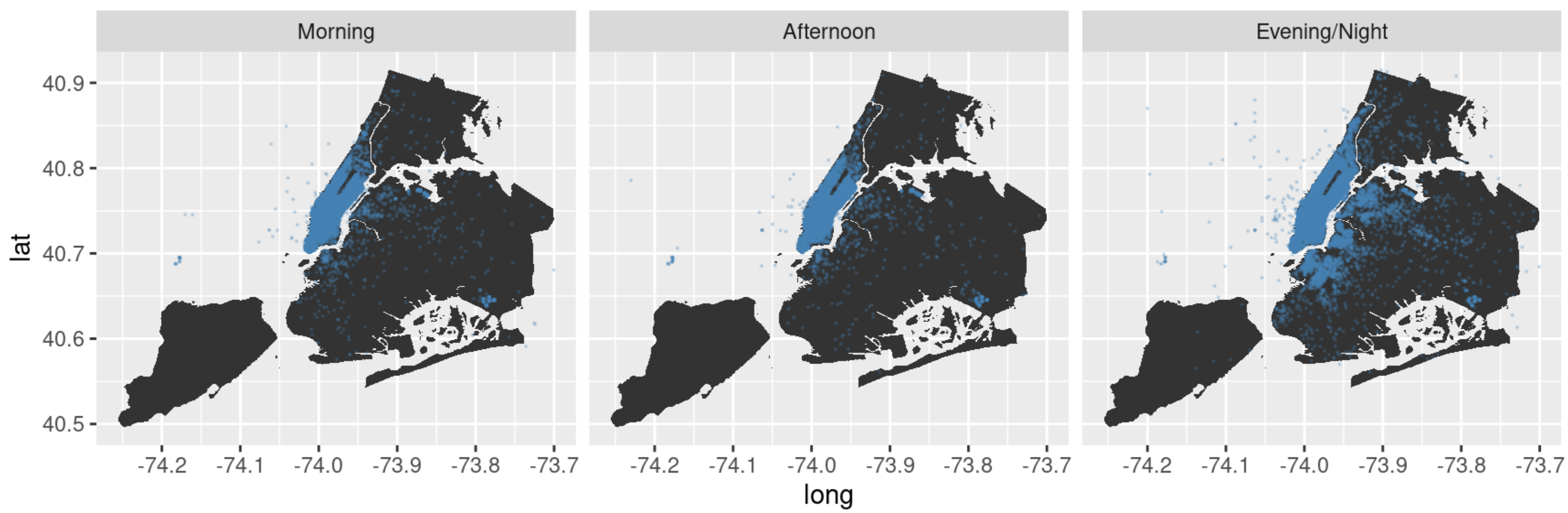
Taxi traffic over the course of a day

```r
# pick-up locations
locations +
     geom_point(aes(x=start_long, y=start_lat),
                color="orange",
                size = 0.1,
                alpha = 0.2) +
     facet_wrap(vars(time_of_day))
```



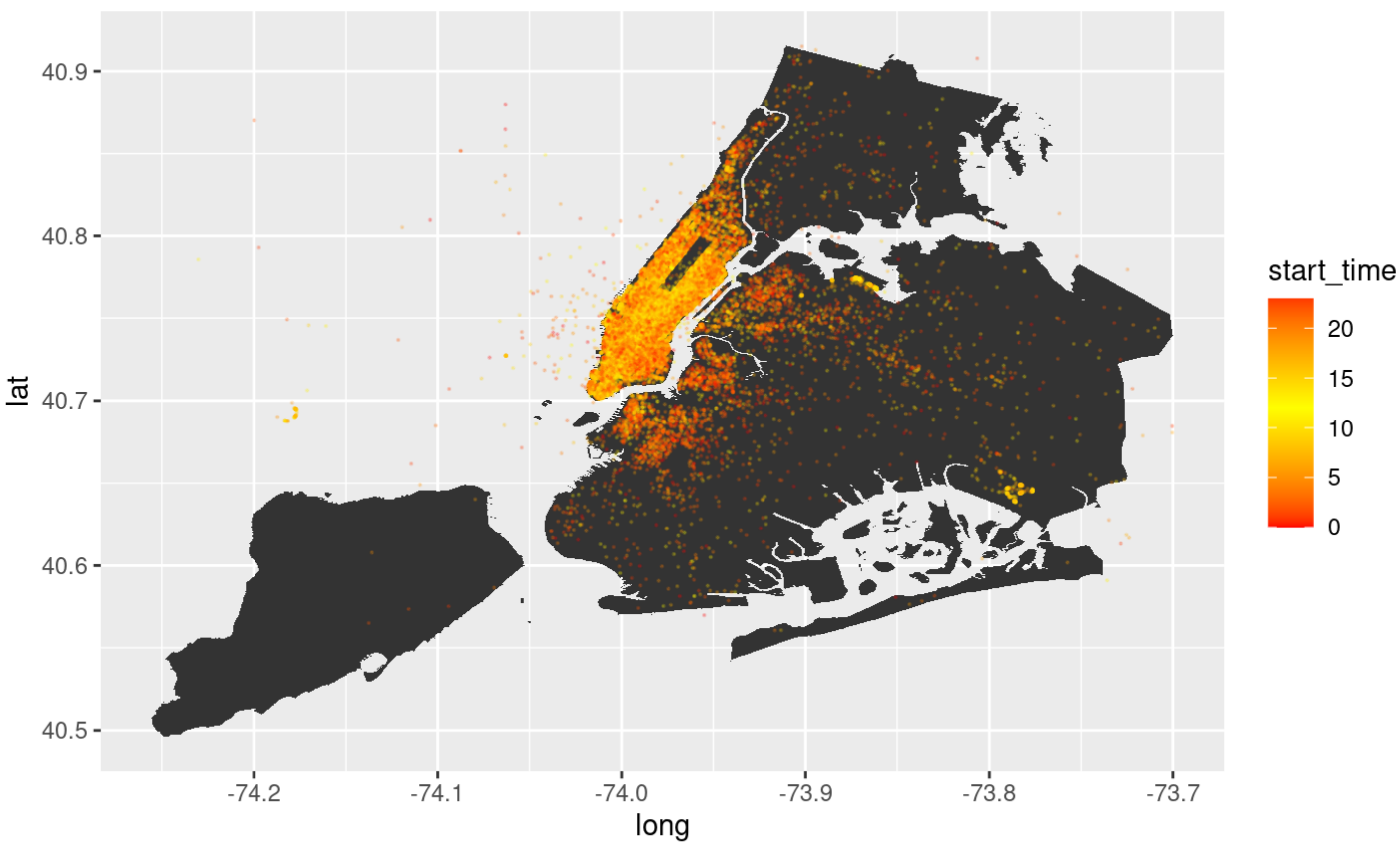
Taxi traffic over the course of a day

```r
# drop-off locations
locations +
     geom_point(aes(x=dest_long, y=dest_lat),
                color="steelblue",
                size = 0.1,
                alpha = 0.2) +
     facet_wrap(vars(time_of_day))
```



Taxi traffic over the course of a day

```
# drop-off locations
locations +
     geom_point(aes(x=dest_long, y=dest_lat, color = start_time ),
                size = 0.1,
                alpha = 0.2) +
     scale_colour_gradient2( low = "red", mid = "yellow", high = "red",
                             midpoint = 12)
```



Data Storage and Databases

(Big) Data Storage

- (I) How can we store large data sets permanently on a mass storage device in an efficient way (here, efficient can be understood as 'not taking up too much space')?
- (II) How can we load (parts of) this data set in an efficient way (here, efficient~fast) for analysis?

We look at this problem in two situations:

- The data need to be stored locally (e.g., on the hard disk of our laptop).
- The data can be stored on a server ‘in the cloud’ (next lecture).

Many new database types for Big Data

NoSQL/NewSQL systems. Source: https://img.deusm.com/informationweek/2014/06/1269559/NoSQL-&-NewSQL.jpg

Simple distinction

- **SQL/Relational Database Systems (RDBMS)**: Relational data model, tabular relations.
  - In use for a long time, very mature, very accurate/stable.
- **NoSQL ('non-SQL', sometimes 'Not only SQL')**: Different data models, column, document, key-value, graph.
  - Horizontal scaling.
  - Non-tabular data.
  - Typically used to handle very large amounts of data.

RDBMS basics

- **Relational data model**
  - Data split into several tables (avoid redundancies).
  - Tables are linked via key-variables/columns.
  - Save storage space.
- **Indexing**
  - Table columns (particularly keys) are indexed.
  - Reduces number of disk accesses required to query data.
  - Makes querying/loading of data more efficient/faster.

Getting started with (R)SQLite

- SQLite
    - Free, full-featured SQL database engine.
    - Widely used across platforms.
    - Typically pre-installed on Windows/MacOSX.
- RSQLite
    - Embeds SQLite in R.
    - Use SQLite from within an R session.

Exercise 1: First steps in SQLite (Terminal)

- Set up a new database called `mydb.sqlite`.

```
cd materials/data
```

```
sqlite3 mydb.sqlite
```

```
.tables
```

Import data from CSV files

```
CREATE TABLE econ(
"date" DATE,
"pce" REAL,
"pop" INTEGER,
"psavert" REAL,
"uempmed" REAL,
"unemploy" INTEGER
);

.mode csv
.import economics.csv econ
```

Inspect the database

```
.tables
```

```
# econ
```

```
.schema econ
```

```
# CREATE TABLE econ(
# "date" DATE,
# "pce" REAL,
# "pop" INTEGER,
# "psavert" REAL,
# "uempmed" REAL,
# "unemploy" INTEGER
# );
```

Set options for output

```
.header on

.mode columns
```

Issue queries: Example 1

In our first query, we select all (*) variable values of the observation of January 1968.

```
select * from econ where date = '1968-01-01'
```

1 records

date	pce	pop	psavert	uempmed	unemploy
1968-01-01	531.5	199808	11.7	5.1	2878

Issue queries: Example 2

Now let's select all year/months in which there were more than 15 million unemployed, ordered by date.

```
select date from econ
where unemploy > 15000
order by date;
```

Displaying records 1 - 10

date
2009-09-01
2009-10-01
2009-11-01
2009-12-01
2010-01-01

Close SQLite

When done working with the database, we can exit SQLite with the `.quit` command.

Exercise 2: Indices and joins

- Import several related tables.
- Add indices to tables.

Initiate DB, import data

We set up a new database called `air.sqlite` and import the csv-file `flights.csv` (used in previous lectures) as a first table.

```
# create database and run sqlite
sqlite3 air.sqlite
```

Import data from CSVs

```
.mode csv
.import flights.csv flights
```

Inspect the `flights` table

Again, we can check if everything worked out well with `.tables` and `.schema`.

```
.tables
.schema flights
```

Related tables

- `airports.csv`: Describes the locations of US Airports (relates to `origin` and `dest`).
- `carriers.csv`: A listing of carrier codes with full names (relates to the `carrier`-column in `flights`.

Import related tables

Import from csv-file

```
.mode csv
.import airports.csv airports
.import carriers.csv carriers
```

Inspect the result

```
.tables
.schema airports
.schema carriers
```

Issue queries with joins

- Goal: A table containing flights data for all `United Air Lines Inc.`-flights departing from `Newark Intl` airport, ordered by flight number.
- For the sake of the exercise, we only show the first 10 results of this query (`LIMIT 10`).

Issue queries with joins

```
SELECT
year,
month,
day,
dep_delay,
flight
FROM (flights INNER JOIN airports ON flights.origin=airports.iata)
INNER JOIN carriers ON flights.carrier = carriers.Code
WHERE carriers.Description = 'United Air Lines Inc.'
AND airports.airport = 'Newark Intl'
ORDER BY flight
LIMIT 10;
```

Displaying records 1 - 10

year	month	day	dep_delay	flight
2013	1	4	0	1
2013	1	5	-2	1
2013	3	6	1	1

Add indices

```
CREATE INDEX iata_airports ON airports (iata);
CREATE INDEX origin_flights ON flights (origin);
CREATE INDEX carrier_flights ON flights (carrier);
CREATE INDEX code_carriers ON carriers (code);
```

Re-run the query (with indices)

```
SELECT
year,
month,
day,
dep_delay,
flight
FROM (flights INNER JOIN airports ON flights.origin=airports.iata)
INNER JOIN carriers ON flights.carrier = carriers.Code
WHERE carriers.Description = 'United Air Lines Inc.'
AND airports.airport = 'Newark Intl'
ORDER BY flight
LIMIT 10;
```

Displaying records 1 - 10

year	month	day	dep_delay	flight
2013	1	4	0	1
2013	1	5	-2	1
2013	3	6	1	1

SQLite from within R

- Use `RSQLite` to set up and query `air.sqlite` as shown above.
- All done from within an R session.

Creating a new database with RSQLite

```
# load packages
library(RSQLite)

# initiate the database
con_air <- dbConnect(SQLite(), "../data/air.sqlite")
```

Importing data

```r
# import data into current R sesssion
flights <- fread("../data/flights.csv")
airports <- fread("../data/airports.csv")
carriers <- fread("../data/carriers.csv")

# add tables to database
dbWriteTable(con_air, "flights", flights)
dbWriteTable(con_air, "airports", airports)
dbWriteTable(con_air, "carriers", carriers)
```

Issue queries with `RSQLite`

```
# define query
delay_query <-
"SELECT
year,
month,
day,
dep_delay,
flight
FROM (flights INNER JOIN airports ON flights.origin=airports.iata)
INNER JOIN carriers ON flights.carrier = carriers.Code
WHERE carriers.Description = 'United Air Lines Inc.'
AND airports.airport = 'Newark Intl'
ORDER BY flight
LIMIT 10;
"
```

Issue queries with RSQLite

```
# issue query
delays_df <- dbGetQuery(con_air, delay_query)
delays_df
```

```
##    year month day dep_delay flight
## 1  2013     1   4         0      1
## 2  2013     1   5        -2      1
## 3  2013     3   6         1      1
## 4  2013     2  13        -2      3
## 5  2013     2  16        -9      3
## 6  2013     2  20         3      3
## 7  2013     2  23        -5      3
## 8  2013     2  26        24      3
## 9  2013     2  27        10      3
## 10 2013     1   5         3     10
```

Close the connection to SQLite

```
dbDisconnect(con_air)
```

References