The New York City Taxi & Limousine Commission’s open data of taxi trip records is rightly a go-to test piece for analytical methods for largish data. Check out, for example:

* The benchmark of interesting analysis is set by [Todd W Schneider’s well-written and rightly famous blog post ‘Analyzing 1.1 Billion NYC Taxi and Uber Trips, with a Vengeance’](https://toddwschneider.com/posts/analyzing-1-1-billion-nyc-taxi-and-uber-trips-with-a-vengeance/), for which he used PostgreSQL, PostGIS and R
* [Mark Litwintschik provides benchmarks](https://tech.marksblogg.com/benchmarks.html) of four different queries of the data on different software and hardware setups, mostly clusters of computers operating big data software such as BrytlytDB, MapD, Redshift, BigQuery and Spark ([source code](https://tech.marksblogg.com/billion-nyc-taxi-rides-redshift.html))
* Various providers use the data to show off the impressive speed of their ingest and analysis tools eg [MemSQL](https://www.memsql.com/blog/nyc-taxi-data-ingested-into-memsql/) or [Ocean9](https://medium.com/@gopalaj61/analyzing-1-2-billion-nyc-taxi-rides-83ea8012827e)

A week or so ago there was a flurry in my corner of the twitterverse with people responding to a great blog post by Jovan Veljanoski [How to analyse 100 GB of data on your laptop with Python](https://towardsdatascience.com/how-to-analyse-100s-of-gbs-of-data-on-your-laptop-with-python-f83363dda94). Veljanoski is one of the co-founders of [vaex.io](http://vaex.io/). Vaex is a powerful Python library that empowers a data scientist with the convenient R-like DataFrame-based feel of Pandas for data that is too large to reside in memory.

The trick used by Vaex is that the data is stored on disk in the efficient HDF5 format. Vaex only uses the minimal scan of the dataset it needs for any particular action. So once Veljanoski has connected to the data, operations like simple summaries of the 1+ billion taxi ride observations he demonstrates on can be done lightning fast. The most impressive example given is an interactive heatmap of New York, which can go to the right part of the data so fast that the user gets the sort of performance with this 100+ GB dataset that normally is only possible when all the data is in RAM.

That’s genuinely impressive, but how necessary is Vaex to analyse this sort of data on commodity hardware? Veljanoski argues in his post that “There are three strategies commonly employed when working with such datasets” :

* sub-sample the data
* distributed computing
* a single strong cloud instance with as much memory as required

As he points out, each of these comes with its disadvantages. But there’s a glaring omission from the list – use a database! A good old-fashioned relational database management system is the tool invented for this sort of data, and today’s database software is the highly optimised beneficiary of untold billions of dollars of investment from firms like IBM, Oracle and Microsoft (for example, Oracle alone spends about [$6 billion per year on research and development](https://www.statista.com/statistics/236990/research-and-development-spending-at-oracle/)). Schneider’s post showed that open source relational database technology (PostgreSQL) is more than up to the job. In fact, a number of the features of Vaex showcased in Veljanoski’s blog post sound almost like a re-invention of some aspects of the database approach:

* data held on disk can be inspected a few rows at once without loading it all into memory;
* the management system holds key metadata or indexes that let certain kinds of simpler analysis be performed via radical shortcuts;
* virtual views of the data can be created instantly as filtered subsets or with additional calculated columns, without copying the whole dataset.

So out of curiousity I set out to see how the data can be handled by my unremarkable personal laptop and the software toolkit I most commonly use at the moment – R in combination with Microsoft SQL Server (which I chose because it’s my most common database in work contexts, has a free developer edition, and has powerful columnstore indexes which I think will help handle this sized data).

What did I find?

1. Even relatively tidy data is serious work to prepare for analysis – whether the prep is an extract-transform-load to a database, or conversion to HDF5.
2. The combination of a standard laptop, SQL Server (or other powerful database tool) and R is fine for dealing with data of several hundred gigabytes in size, but I would want a bigger computer and a large fast solid state drive for much bigger than this.
3. Most analysis on this sort of data has aggregation and summary operations as its first step, so by the time you hit a statistical model or a graphic you are likely working with a much smaller dataset than the original.
4. Most of the interesting things to say about the NYC taxi data have already been said.

Let’s see how that works out.

**Data preparation**

“80% of data scientists’ work is managing, tidying and cleaning data. And most of the other 20% is complaining about it.”

*Source : I wouldn’t know where to start to find who said this first.*

The original taxicab data can be downloaded, one CSV file per month, from the [New York City Taxi & Limousine Commission’s website](https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page). The data are available to 2019, but the data model changed several times, and by the second half of 2016 latitude and longitude were not being reported as per the earlier detail, probably for belated privacy reasons.You could also see [Todd W Schneider’s PostgreSQL and R version](https://github.com/toddwschneider/nyc-taxi-data) which I suspect is better managed.

Unfortunately Veljanoski’s post doesn’t link to the code he used to process the original CSVs into HDF5 format. His Jupyter notebook starts on the basis the data is available in HDF5 format on your laptop. He links to example code for converting CSVs to HDF5 but a) it’s for a different dataset and b) it’s clear the task is non-trivial.

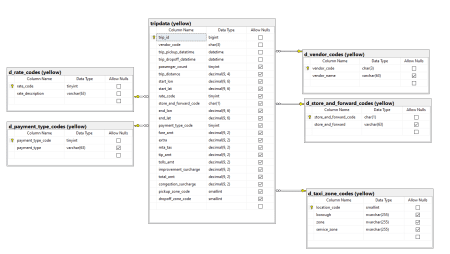
So I can’t directly compare his process to mine, but I can say that the load process with my toolkit was a significant effort. The contributing factors to this were:

* irrespective of software choices, the data are large for the hardware I’m using them for, which leads to sub-optimal workarounds. For example, my hard drive isn’t large enough to store at the same time all the original CSVs, a staging copy of the data in the database, and the final analysis-ready copy of the data. This meant I had to delete the CSVs as my transformation process was going; and if I found a problem (as happened several times) I had to repeat some or all of the original downloads (which takes 6+ hours even on my fast-by-Australian-standards 100+ MBPS interent connection).
* the data model changes several times. For example, there are half a dozen different sets of column names, sometimes varying by just capitalisation, sometimes by name changes such as trip\_pickup\_datetime becoming tpep\_pickup\_datetime. There is quite a [good user guide available](https://www1.nyc.gov/assets/tlc/downloads/pdf/trip_record_user_guide.pdf) but I foolishly didn’t look for it early enough. The [data dictionary](https://www1.nyc.gov/assets/tlc/downloads/pdf/data_dictionary_trip_records_yellow.pdf) is for the latest data model; there might be historical versions but I didn’t look hard enough to find them.
* some of the data is corrupt (eg some of the monthly CSVs in 2010 have several thousand rows with more columns in them than the other rows) – a relatively small amount, but enough to cause problems in combination with the other factors.

In the second half of 2016 something happened to the data I couldn’t understand, and this in combination with the fact that latitude and longitude are no longer recorded led me to give up at that point and confine my analysis to the first 7.5 years. At well over 200GB of CSVs this still seems an adequate test.

I would say I spent at least eight hours of effort over two weeks of calendar time on the extract-transform-load of the CSVs into their eventual form in the database, and I am far from satisfied with the result; I think another person day or three is needed to tidy it up, better document it, and introduce some efficiencies and performance enhancements. The two weeks of calendar time is significant too – because a number of the operations took multiple hours to run it was common to have to let it run and then go do something else (eg sleep, walk around the park, go do my real job, etc) – resulting in task switching costs when I came back to this project several days later, and meaning that even if I had devoted three full time days to the project I wouldn’t have been able to use them efficiently. The physical bottlenecks, ordered from most to least, were writing and reading to my slow physical disk (the only drive I had with space for this data); downloading large data over the internet; and processing.

But ultimately I got a working version of the data I’m fairly happy with. In the database it looks like this:



I’m satisfied its more analysis-ready than some of the alternative loads of this data I’ve seen out there. For example, I have all the codes built in to the database; no need to look up elsewhere that payment type 1 is cash, 2 is credit, etc. Plus my loading process standardises the different versions of vendor coding that are used – sometimes “1” and “2”, sometimes “CMT”, “VTS” and “DDS” (which seemed to disappear early in the series and isn’t listed in the data dictionary). To give an idea of what this means in terms of code development, here is one step in the overall process. This chunk of SQL code takes data from a staging table – dbo.tripdata\_0914, with the raw data from the first six years when the column sequencing (but not names or coding) was consistent – and loads it into the evental target table tripdata in the yellow schema:

-- First six years, with no surcharge data:

INSERT INTO yellow.tripdata(

vendor\_code,

trip\_pickup\_datetime,

trip\_dropoff\_datetime,

passenger\_count,

trip\_distance,

start\_lon,

start\_lat,

rate\_code,

store\_and\_forward\_code,

end\_lon,

end\_lat,

payment\_type\_code,

fare\_amt,

extra,

mta\_tax,

tip\_amt,

tolls\_amt,

total\_amt

)

SELECT

CASE

WHEN vendor\_name = '1' THEN 'CMT'

WHEN vendor\_name = '2' THEN 'VTS'

ELSE vendor\_name

END,

trip\_pickup\_datetime,

trip\_dropoff\_datetime,

passenger\_count,

TRY\_CAST(trip\_distance AS DECIMAL(9, 4)),

TRY\_CAST(start\_lon AS DECIMAL(9, 6)),

TRY\_CAST(start\_lat AS DECIMAL(9, 6)),

CASE

WHEN rate\_code IN (1,2,3,4,5,6) THEN rate\_code

ELSE NULL

END,

CASE

WHEN store\_and\_forward IN ('0', 'FALSE', 'N') THEN 'N'

WHEN store\_and\_forward IN ('1', 'TRUE', 'Y') THEN 'Y'

END,

TRY\_CAST(end\_lon AS DECIMAL(9, 6)),

TRY\_CAST(end\_lat AS DECIMAL(9, 6)),

CASE

WHEN payment\_type IN ('credit', 'cre', '1') THEN 1

WHEN payment\_type IN ('cash', 'csh', 'cas', '2') THEN 2

WHEN payment\_type IN ('no', 'no charge', 'noc', '3') THEN 3

WHEN payment\_type IN ('disput', 'dis', '4') THEN 4

WHEN payment\_type IN ('unknown', 'unk', '5') THEN 5

WHEN payment\_type In ('voided trip', 'voi', 'voided', '6') THEN 6

ELSE NULL

END,

TRY\_CAST(fare\_amt AS DECIMAL(9, 2)),

TRY\_CAST(surcharge AS DECIMAL(9, 2)),

TRY\_CAST(mta\_tax AS DECIMAL(9, 2)),

TRY\_CAST(tip\_amt AS DECIMAL(9, 2)),

TRY\_CAST(tolls\_amt AS DECIMAL(9, 2)),

TRY\_CAST(total\_amt AS DECIMAL(9, 2))

FROM dbo.tripdata\_0914

You can see that I also handle issues such as the inconsistent coding of “payment type”, and cut down the storage size of various numeric variables. For example, trip\_distance was loaded into my staging table as 15 digit precision floating point data which takes up 8 bytes of space per observation; reducing this to DECIMAL(9, 4) saves three bytes per row of the data while still allowing plenty of range and precision – numbers of the format 99999.1234. Three bytes per row adds up materially over 1+ billion rows. The TRY\_CAST() functions are needed because many of these numeric variables have physically impossible values (eg longitudes with four or more digits left of the decimal point) which cause numeric overflow errors otherwise.

It’s easy to see where that eight hours went… In contrast, I would say I have spent about four hours on analysis of any sort (mostly descriptive) and four hours on writing up this post. So the “80%” of time spent on data cleaning isn’t quite right here – more like 50%. Noting that in this case, the data is *exceptionally* clean and tidy already – other than having direct access to a well-maintained warehouse, one will rarely get data as nicely released by the NYC Taxi & Limousine Commission.

On the other hand, there’s big economies of scale here. If I did another couple of blog posts on this data, that 50% or so of effort that was on data preparation would pretty quickly go down. Worth noting.

With one of SQL Server’s powerful, fast clustered columnstore indexes on the main table, the total data size on disk is about 45 GB for main data and another 33 GB for an unclustered primary key identifying each row which I think is probably not needed and could be dropped. That 45GB is a lot of compression from the original size of more than 200GB, but more importantly the columnstore approach makes analysis pretty fast for dealing with 1.2+ billion rows even on my poor underpowered laptop and its old-school mechanical hard disk drive.

**Repeating previous analysis – maps and variable distribution**

**Passenger counts distribution**

So, let’s see what I can do now that the data’s ready. I repeated the first few bits of exploratory analysis from Veljanoski’s article. For example, here is R code to set up an analytical session and explore the distribution of “number of passengers” on each trip:

library(tidyverse)

library(scales)

library(odbc)

library(DBI)

library(clipr)

library(knitr)

library(kableExtra)

library(ggmap)

library(Cairo)

# assumes you have an ODBC data source named "nyc\_taxi" to the database created by <https://github.com/ellisp/nyc-taxis>:

nyc\_taxi <- dbConnect(odbc(), "localhost", database = "nyc\_taxi")

#' Theme for dark background maps to be used later

#'

#' @author minimally adapted from a Todd Schneider original

theme\_dark\_map <- function(base\_size = 12, font\_family = "Sans"){

theme\_bw(base\_size) +

theme(text = element\_text(family = font\_family, color = "#ffffff"),

rect = element\_rect(fill = "#000000", color = "#000000"),

plot.background = element\_rect(fill = "#000000", color = "#000000"),

panel.background = element\_rect(fill = "#000000", color = "#000000"),

plot.title = element\_text(family = font\_family),

panel.grid = element\_blank(),

panel.border = element\_blank(),

axis.text = element\_blank(),

axis.title = element\_blank(),

axis.ticks = element\_blank())

}

theme\_white\_map <- function(base\_size = 12, font\_family = main\_font){

theme\_bw(base\_size) +

theme(text = element\_text(family = font\_family, color = "#000000"),

rect = element\_rect(fill = "#ffffff", color = "#ffffff"),

plot.background = element\_rect(fill = "#ffffff", color = "#ffffff"),

panel.background = element\_rect(fill = "#ffffff", color = "#ffffff"),

plot.title = element\_text(family = font\_family),

panel.grid = element\_blank(),

panel.border = element\_blank(),

axis.text = element\_blank(),

axis.title = element\_blank(),

axis.ticks = element\_blank(),

plot.caption = element\_text(color = "grey50"))

}

the\_caption <- "Analysis by <http://freerangestats.info> with data from NYC Taxi & Limousine Commission"

#-----------------------passenger count frequencies------

sql <- "

SELECT

COUNT(1) AS freq,

passenger\_count

FROM yellow.tripdata

GROUP BY passenger\_count

"

# takes a couple of seconds:

pc\_freq <- dbGetQuery(nyc\_taxi, sql)

pc\_freq %>%

ggplot(aes(x = str\_pad(passenger\_count, 3, pad = "0"), y = freq)) +

geom\_col() +

scale\_y\_log10(label = comma) +

scale\_x\_discrete(labels = sort(unique(pc\_freq$passenger\_count))) +

labs(x = "Number of passengers (discrete scale - only values with at least one trip shown)",

y = "Number of trips (log scale)",

title = "Number of passengers per trip",

subtitle = "New York City yellow cabs January 2009 to mid 2016",

caption = the\_caption)

This produces the following image for us, very similar to Veljanoski original. It’s not identical because I’ve got six months more data and I’ve built more data-cleaning into my ETL – most notably where the number of trip passengers is zero I have replaced this with NULL in my database.

My code took less than five seconds to run in contrast to his 23 seconds with Vaex/Python, so I’m pretty pleased by performance so far.

**Trip distance distribution**

A very similar bit of SQL and R code gives us a similar plot for distribution of trip distance in miles, truncated at just those less than 250 miles. Because the trip distances are continuous variables, to show their distribution I’m going to need to do some kind of summary. With smaller data I would put all the observations into R and use geom\_density() to do the work, but as this would mean transferring a billion observations from the database to R’s environment for little gain, I opt to round the distances to the nearest tenth of a mile and aggregate them. In effect, I’m building my own fixed-bin-width histogram. Here’s my first go at that:

…produced with:

#-----------------------distance in miles--------------

sql <- "

SELECT

COUNT(1) AS freq,

ROUND(trip\_distance, 1) AS trip\_distance

FROM yellow.tripdata

GROUP BY ROUND(trip\_distance, 1)

"

# takes about 20 seconds

td\_freq <- dbGetQuery(nyc\_taxi, sql)

td\_freq %>%

filter(trip\_distance <= 250) %>%

ggplot(aes(x = trip\_distance, y = freq)) +

geom\_line() +

scale\_y\_log10(label = comma) +

labs(x = "Trip distance in miles - rounded to 0.1 of a mile",

y = "Number of trips (log scale)",

title = "Trip distances",

subtitle = "New York City yellow cabs January 2009 to mid 2016",

caption = the\_caption)

This looked quite markedly different from Veljanoski’s original so I tried rounding to the nearest mile rather than tenth of a mile and got this result (code not shown):

Either of these charts gives a good indication of what is happening, but the second is probably better. We see a suspiciously marked drop-off in distances at exactly 100 miles, which is almost certainly measurement error rather than a physical reality.

This time, my database was much slower than Valjanoski’s Vaex/Python combination – 20 seconds versus one second.

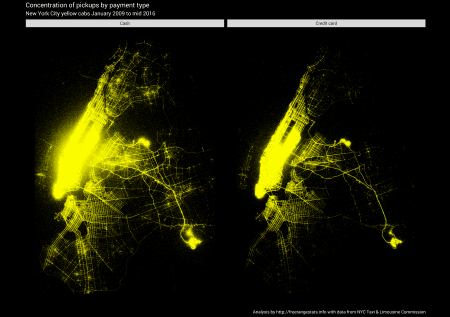
**Maps**

[Todd Schneider’s ground-breaking post with this data](https://toddwschneider.com/posts/analyzing-1-1-billion-nyc-taxi-and-uber-trips-with-a-vengeance/) was conspicuous because of the beauty of his clean high resolution images of pickups and drop off points. He wrote “These maps show every taxi pickup and drop off, respectively, in New York City from 2009-2015. The maps are made up of tiny dots, where brighter regions indicate more taxi activity.”

In fact, while the maps do indeed “show” every pickup and drop off, a moments reflection would show that it isn’t physically possible to each individual trip to get its own blob of light. His 10MB high resolution images are 2,880 by 4,068 pixels in size, about 12 million pixels each. There are nearly 100 times as many trips as pixels, so some summary is required. Effectively he creates a fine grid across the New York City area and returns the average number of trips on each point of his grid.

This approach is fine (in fact I had done it myself even before looking at his source code) – I’m just mentioning it to highlight that when dealing with large datasets of this sort, to visualise them meaningfully you need to somehow reduce the size of the data in a meaningful way first before rendering them on the screen.

Let’s have a go at something similar. Here’s a map showing the pickup points of taxis from 2009 to mid 2016, split into payment by cash or by credit card



It’s not as polished as Schneider’s original, which I think was enhanced by using GIS methods to remove some of the noise from locations that were in the water. But it’s certainly a nice start.

Performance for producing this charts was acceptable but not that pleasant: about 10 minutes to summarise the data and 4 or 5 to render the graphic. In contrast Veljanoski with Vaex did something of similar complexity at the mid point of his post with about 60 seconds of elapsed time. That’s extremely impressive.

**The time element**

Let’s look at ways of looking at our data that takes into account the element of time – particularly periodicity. I might come back to this later as the data is a nice example of multiple seasonalities in a time series. If we aggregate it by hour we have at least three seasonal patterns – hours in the day, days in the week, and the annual cycle.

Again, we need some kind of aggregation before the billion taxi rides can be assimilated by the human eye. If we aggregate it up by hour it is still too dense to take in:

That’s the level of temporal granularity I’d like to analyse this when I’ve got more time but it’s too much for the eye.

If we aggregate up to months we have lost most of the interest, but it’s still a useful starting point. Note how the rides per hour declines substantially over time as more competition came in:

There’s a notable dip in taxi rides each winter.

Most other blogs draw some day-of-week by hour-of-day heatmaps at this point, and there’s good reason to – they’re a great summary of this sort of data. Here’s the most obvious one to draw – pure frequency of the start of taxi trips:

All that yellow in the top right shows (unsuprisingly) more taxi activity in evenings, and on Wednesday, Thursday, Friday and Saturday rather than earlier in the week.

Here’s a replication of one of Veljanoski’s plots, showing the differing average fare collected per mile travelled at different times

Shorter taxi rides have a higher fare per mile, and I suspect we get more short taxi rides in during working hours on working days, which would explain the yellow highlight in the middle of that chart.

This analysis was easy to write with my database, but the performance was slower than Veljanoski’s (while still being totally acceptable). Whether this is due to my slow disk drive or software differences I’m not sure. Here’s the code for that data extract and analysis:

#--------------hour, week, year------------

# aggregate some summary data at hourly level

sql <- "

WITH a AS

(SELECT

trip\_distance,

fare\_amt,

YEAR(trip\_pickup\_datetime) AS yy,

DATEPART(hh, trip\_pickup\_datetime) AS hd,

DATEPART(dy, trip\_pickup\_datetime) AS dy,

DATEPART(dw, trip\_pickup\_datetime) AS dw,

DATEPART(m, trip\_pickup\_datetime) AS my

FROM yellow.tripdata

WHERE trip\_distance > 0 AND total\_amt > 0)

SELECT

COUNT(1) AS freq,

AVG(fare\_amt / trip\_distance) AS fare\_per\_mile,

yy, hd, dy, dw, my

FROM a

GROUP BY yy, hd, dy, dw, my

ORDER BY yy, dy, hd"

pickup\_ts <- dbGetQuery(nyc\_taxi, sql) %>% as\_tibble()

# Note that weekday = 1 means sunday

# Hourly time series

pickup\_ts %>%

mutate(meta\_hour = 1:n()) %>%

ggplot(aes(x = meta\_hour, y = freq, colour = as.ordered(my))) +

facet\_wrap(~yy, scales = "free\_x") +

geom\_line() +

scale\_colour\_viridis\_d(breaks = 1:12, labels = month.abb,

guide = guide\_legend(ncol = 2)) +

scale\_y\_continuous(label = comma) +

labs(x = "Day of the year",

y = "Number of taxi pickups",

colour = "",

caption = the\_caption,

title = "Taxi pickups time series",

subtitle = "Hourly observations 2009 to mid 2016") +

theme(legend.position = c(0.85, 0.15))

# Monthly time series

pickup\_ts %>%

group\_by(yy, my) %>%

summarise(freq = mean(freq)) %>%

ggplot(aes(x = my, y = freq)) +

facet\_wrap(~yy) +

geom\_line() +

scale\_y\_continuous(label = comma) +

scale\_x\_continuous(breaks = 1:12, labels = month.abb) +

labs(x = "",

y = "Average hourly taxi pickups\n",

caption = the\_caption,

title = "Taxi pickups time series",

subtitle = "Monthly averages based on hourly observations 2009 to mid 2016") +

theme(axis.text.x = element\_text(angle = 45, hjust = 1),

panel.grid.minor = element\_blank())

# Heatmap of frequency of trips

# add a labelled day of the week variable, still keeping the ordering

pickup\_ts2 <- pickup\_ts %>%

mutate(dwl = factor(dw, labels = c("Sun", "Mon", "Tue", "Wed", "Thur", "Fri", "Sat")))

pickup\_ts2 %>%

group\_by(dwl, hd) %>%

summarise(freq = mean(freq)) %>%

ggplot(aes(x = as.factor(hd), y = dwl, fill = freq)) +

geom\_tile() +

scale\_fill\_viridis\_c(label = comma) +

labs(caption = the\_caption,

x = "Hour of the day",

y = "Day of the week",

title = "Taxi frequency by time of day and day of week",

subtitle = "Hourly observations from 2009 to mid 2016",

fill = "Pickups\nper hour")+

theme(legend.position = "right")

# Heatmap of average fare per distance

pickup\_ts2 %>%

group\_by(dwl, hd) %>%

summarise(fare\_per\_mile = sum(fare\_per\_mile \* freq) / sum(freq)) %>%

ggplot(aes(x = as.factor(hd), y = dwl, fill = fare\_per\_mile)) +

geom\_tile() +

scale\_fill\_viridis\_c(label = dollar) +

labs(caption = the\_caption,

x = "Hour of the day",

y = "Day of the week",

title = "Taxi fare per distance by time of day and day of week",

subtitle = "Hourly observations from 2009 to mid 2016",

fill = "fare\nper mile") +

theme(legend.position = "right")

**Modelling average fare**

Finally, I wanted to go beyond simple exploratory analysis to at least touch on some statistical modelling. I’ve only time for a taster here (both in terms of my writing time, and wondering if any readers will get this far…). I thought I’d start with a simple statistical model layer over some of the spatial charts I’d been playing with. So here’s a contour map (in the style of a weather forecast) showing average fare to be expected by location. While it’s not as beautiful as some of the high res point maps we’ve seen with this data, it’s probably more useful in terms of helping understand a fairly complex phenomenon:

It’s a nice illustration of how, even in primarily exploratory analysis, a statistical model can be powerful for helping shape thinking. In this case, the model is a generalized additive model, fit to the data by the mgcv::bam function which is optimised for working with larger data. I commonly use a generalized additive model with a smoother over latitude and longitude (think a rubbery sheet in three dimensional space) as a way of dealing with spatially correlated data.

In this case, rather than fit the model to the original individual trip data I’ve taken average fares over a grid of about 100,000 observations and used that data, weighted by the number of observations at each point, to train my simple model. Here’s this final bit of code for this analysis:

#-------------Modelling average fare by space-------------------

# extract data

sql <- "

SELECT

AVG(fare\_amt) AS fare\_amt,

COUNT(1) AS freq,

ROUND(start\_lon, 3) AS start\_lon,

ROUND(start\_lat, 3) AS start\_lat

FROM yellow.tripdata

WHERE fare\_amt IS NOT NULL AND

start\_lon > -74.05 AND start\_lon < -73.75 AND

start\_lat > 40.58 AND start\_lat < 40.90 AND

end\_lon > -74.05 AND end\_lon < -73.75 AND

end\_lat > 40.58 AND end\_lat < 40.90 AND

passenger\_count > 0 AND passenger\_count < 7 AND

trip\_distance > 0 AND trip\_distance < 100

GROUP BY

ROUND(start\_lon, 3),

ROUND(start\_lat, 3)"

fare\_rounded <- dbGetQuery(nyc\_taxi, sql) # takes a while - 10 minutes or so

# about 95,000 observations

# Fit model

model <- bam(fare\_amt ~ s(start\_lon, start\_lat), weights = freq, data = fare\_rounded)

# predict values over a regular grid

all\_lons <-seq(from = -74.05, to = -73.75, length.out = 200)

all\_lats <- seq(from = 40.58, to = 40.90, length.out = 200)

the\_grid <- expand.grid(

start\_lon = all\_lons,

start\_lat = all\_lats

)

predicted <- predict.bam(model, newdata = the\_grid)

d <- the\_grid %>%

as\_tibble() %>%

mutate(predicted\_fare = predicted) %>%

filter(start\_lat < 40.8)

# get a background map

nyc\_map <- ggmap::get\_stamenmap(bbox = c(-74.05, 40.58,

-73.75, 40.80),

maptype = "toner-background")

# draw a contour map over that background:

ggmap(nyc\_map) +

geom\_raster(data = d,aes(x = start\_lon,

y = start\_lat,

fill = predicted\_fare), alpha = 0.6) +

geom\_contour(data = d, aes(x = start\_lon,

y = start\_lat,

z = predicted\_fare)) +

geom\_text\_contour(data = d, aes(x = start\_lon,

y = start\_lat,

z = predicted\_fare),

colour = "white") +

theme\_white\_map() +

scale\_fill\_viridis\_c(option = "A", label = dollar) +

# we need cartesian coordinates because of raster. Probably better ways to fix this:

coord\_cartesian() +

labs(title = "Expected taxi fare by pick-up point in New York",

subtitle = "New York City yellow cabs January 2009 to mid 2016",

caption = paste(the\_caption,

"Map tiles by Stamen Design, under CC BY 3.0. Data by OpenStreetMap, under ODbL.",

sep = "\n"),

fill = "Predicted\naverage fare")

It would be easy to extend this by looking for different spatial patterns (for example) by vendor, payment type, etc., but I’m lacking any particular motivation for that one. For now, I’m happy to conclude that my hardware and software is totally adequate for analysing this size data and I feel I’ve answered the implicit challenge of the “how to analyse 100s of GBs of data on your laptop with Python” blog post from a few week’s back. But I’ll need a bigger hard drive (or, more sensibly, to move to the cloud) if I do anything much bigger than this.