OSM in Location Science

Jaak Laineste
@jaakl
What is CARTO
CARTO

PIONEERS IN LOCATION INTELLIGENCE

1,200 Customers
300K End-users
120+ Team members
CARTO IS THE PLATFORM FOR TURNING LOCATION DATA INTO BUSINESS OUTCOMES
A self-service business user application for spatial analysis and visualization.

- Built in drag and drop analytics or custom functions
- Both in the cloud and on premise
- Auto-styling and Publishing
- Rapid application deployment
- Publish interactive dashboards that update analysis and filter live

BRINGING LOCATION INTELLIGENCE TO THE MASSES
ENGINE

POWER YOUR APPS WITH LOCATION INTELLIGENCE

The one-stop shop for developers to power location applications in their organization.

- Easy-to-use, open source APIs & SDKs
- Location Data Services
- Built for developers and designers
- Native and custom analysis libraries
ENGINE APIs

Auth API
Create and manage credentials that grant specific permissions to data and access to APIs for different projects and apps.

SQL API
Interact with your tables and data inside CARTO, as if you were running SQL statements on your own database.

Maps API
Generate maps based on data hosted in your account and customize the SQL, CartoCSS, and other parameters.

Import API
Import files with different formats and manipulate them by using a set of HTTP commands.

Data Services API
Geocode your data and perform other Location Intelligence analysis operations.
DATA OBSERVATORY

DON’T LET YOUR DATA LIMIT YOUR ANALYSIS

Augment your own data and broaden your analysis with thousands of datasets and measurements.

- Demographic segments
- Income, employment, and family datasets
- Real estate and financial data
- Many more...
- But: no OSM (yet)!
Location Science tools
Why Python?

- Most popular language for data scientists
- Extremely flexible
- Huge + innovative community

Image: xkcd.com/353
Jupyter

“open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text”
Given a specific $\lambda$, the expected value of an exponential random variable is equal to the inverse of $\lambda$, that is:

$$E[Z | \lambda] = \frac{1}{\lambda}$$

```python
import numpy as np
import matplotlib.pyplot as plt

# Example code

a = np.linspace(0, 4, 100)
expo = stats.expon
lambda_ = [0.5, 1]

for l, col in zip(lambda_, ['red', 'blue', 'green']):
    plt.plot(a, expo.pdf(a, scale=1./l), lw=3, color=col, label=f'\$\lambda = {l:.2f}$ & $l$')
    plt.fill_between(a, expo.pdf(a, scale=1./l), color=col, alpha=0.33)

plt.legend()
plt.xlabel('PDF at $\propto$')
plt.ylabel('$\propto$')
plt.title('Probability density function of an Exponential random variable; differing $\lambda$ values')
```

But what is $\lambda$? This question is what motivates statistics. In the real world, $\lambda$ is hidden from us. We see only $Z$, and must go backwards to try and determine $\lambda$. The problem is difficult because there is no one-to-one mapping from $Z$ to $\lambda$. Many different methods have been created to solve the problem of estimating $\lambda$, but since $\lambda$ is never actually observed, no one can say for certain which method is best!

---

**Jupyter notebooks**

- The de facto standard for communicating work
- Discovery environment of choice for many data scientists
- Clearly shows reproducible workflows
Geo python goodies
from mpl_toolkits.basemap import Basemap
import matplotlib.pyplot as plt
import numpy as np

map = Basemap(projection='ortho',
              lat_0 = 50,
              lon_0 = -100,
              resolution = 'l',
              area_thresh = 1000.)
map.drawcoastlines()
map.drawcountries()
map.fillcontinents(color = 'coral')
map.drawmapboundary()
map.drawmeridians(np.arange(0, 360, 30))
map.drawparallels(np.arange(-90, 90, 30))
plt.show()
```
import geopandas as gpd

boroughs = gpd.datasets.get_path('nybb')
df = gpd.read_file(boroughs)
df.plot(column='Shape_Area',
        figsize=(10, 10),
        alpha=0.5)
```
Folium

Leaflet and Python integrated

```python
import folium
import pandas as pd

state_data = pd.read_csv('data.csv')
state_geo = 'us_states.geojson'
m = folium.Map(location=[48, -102])
m.choropleth(
    geo_data=state_geo,
    name='choropleth',
    data=state_data,
    columns=[['State', 'Unemployment']],
    key_on='feature.id',
    fill_color='YlGn',
    fill_opacity=0.7,
    line_opacity=0.2,
    legend_name='Unemployment Rate (%)'
)
```
```python
import cartoframes
from cartoframes import Layer, styling, BaseMap

cc = cartoframes.CartoContext()
cc.map(layers=Layer('all_month_3'))
```
OSM in CartoFrames
Case study

What are the most popular names in different countries?

1. Extract data from OSM global database
2. Reduce data size, preprocess, filter, geocode
3. Do analysis with Python
4. Make a map!
5. Rinse, repeat
Data extract - the hard part

1. Planet → Imposm3 → PostGIS → SQL → result

\[
\text{UPDATE osm\_roads AS } r \text{ SET admin2 = (SELECT a.name\_iso FROM adm0 AS a WHERE } r.\text{geometry } \&\& a.\text{geom LIMIT 1);}
\]

\[
\text{SELECT admin2, name, count(*) FROM osm\_roads GROUP BY name, admin2;}
\]

2. Overpass API. Can get names, even countries, but output format is tricky
   Hard to write queries, too big result

3. “Big Data” as a service providers.
   AWS has OSM Planet, weekly updated, queryable via Athena SQL. No polygon query or custom functions.
   Google BigQuery does not have OSM (yet). But it has user functions, can do point-in-polygon
Winning method

1. Download per-country packages from Geofabrik
2. Osmconvert to o5m
3. Osmfilter for key stats to CSV
4. Sort and head
5. Py: transpose data
6. Py: make a CartoFrames map

Fast! Convert: few seconds for small country, Italy (1.3G) ~1 minute. Filter: also ~1 minute
Notebook:

https://github.com/jaakla/osm-name-stats

Final map in CARTO:

https://cartomobile-team.carto.com/u/jaakl/builder/166ca721-7c3a-47d6-80d8-7d340e22b0ab/embed

The most popular placename

1. Preprocessing

The aim of preprocessing is to do heavy processing using
million named highways get reasonable data sizes to be

- **wget** to download per-country osm.pbf files from
goecode' the 35M roads
- **osmconvert** to get faster to be processed o5m form
- **osmfilter** to get top 'name' tag values for every country

Preprocessing steps
Key learnings

- Reduce big data to small
- Use preprocessed data
- Preprocess with proper tools
- There are no universal tools for big datasets
- Most tools are not ok for the Planet queries. E.g. PostGIS
- Use optimized formats
- Test with samples
- Some steps remain manual
Read about CARTOframes

Silas Toms, Eric van Rees, and Paul Crickard

Includes a full chapter on CARTOframes
Try yourself

Visit https://github.com/CartoDB/cartoframes

and click launch binder
Thanks!

Jaak Laineste
jaak@carto.com / jaak@nutiteq.com
@jaakl

Credits: Michelle Ho @ CARTO