11: Maps

Maps:

- Map Projection Transitions (https://www.jasondavies.com/maps/transition/)
- True size (http://thetruesize.com/)
- Mercator puzzle (https://bramus.github.io/mercator-puzzle-redux/)
- Choosing a projection (http://www.geo.hunter.cuny.edu/~jochen/gtech201/lectures/lec6concepts/map%20coordinate%20systems/how%20to%20choose%20a%20projection.htm)

A dotmap with Altair

Let's start with altair. When your dataset is large, it is nice to enable a json data transformer. What it does is, instead of generating and holding the whole dataset in the memory, transform the dataset and save into a temporary file. This makes the whole plotting process much more efficient. For more information, check out: https://altair-viz.github.io/user_guide/data_transformers.html (https://altair-viz.github.io/user_guide/data_transformers.html)

In [3]:
```
import altair as alt

# saving data into a file rather than embedding into the chart
alt.data_transformers.enable("json")

# alt.renderers.enable('notebook')
# alt.renderers.enable('jupyterlab')
alt.renderers.enable("default")
```

Out[3]:
```
RendererRegistry.enable('default')
```

Maybe we need a dataset with geographical coordinates. This `zipcodes` dataset contains the location and zipcode of each zip code area.

In [6]:
```
from vega_datasets import data

zipcodes_url = data.zipcodes.url
zipcodes = data.zipcodes()
zipcodes.head()
```

Out[6]:

	zip_code	latitude	longitude	city	state	county
0	00501	40.922326	-72.637078	Holtsville	NY	Suffolk
1	00544	40.922326	-72.637078	Holtsville	NY	Suffolk
2	00601	18.165273	-66.722583	Adjuntas	PR	Adjuntas
3	00602	18.393103	-67.180953	Aguada	PR	Aguada
4	00603	18.455913	-67.145780	Aguadilla	PR	Aguadilla

In [7]:
```
zipcodes = data.zipcodes(dtype={"zip_code": "category"})
zipcodes.head()
```

Out[7]:

	zip_code	latitude	longitude	city	state	county
0	00501	40.922326	-72.637078	Holtsville	NY	Suffolk
1	00544	40.922326	-72.637078	Holtsville	NY	Suffolk
2	00601	18.165273	-66.722583	Adjuntas	PR	Adjuntas
3	00602	18.393103	-67.180953	Aguada	PR	Aguada
4	00603	18.455913	-67.145780	Aguadilla	PR	Aguadilla

In [9]:
```
zipcodes.zip_code.dtype
```

Out[9]:
```
CategoricalDtype(categories=['00501', '00544', '00601', '00602', '00603', '00604',
                  '00605', '00606', '00610', '00611',
                  ...
                  '99919', '99921', '99922', '99923', '99925', '99926',
                  '99927', '99928', '99929', '99950'],
, ordered=False, categories_dtype=object)
```

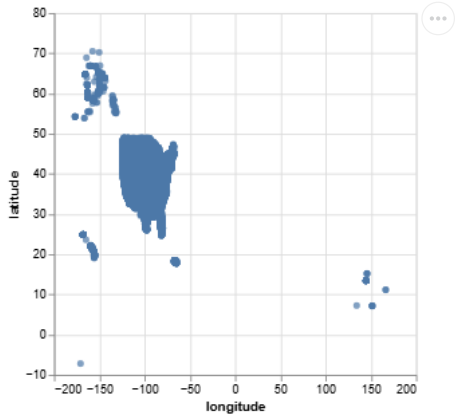
You will have fewer issues if you pass URL instead of a dataframe to `alt.Chart` .

Drawing maps

Now that we have the dataset loaded let's start drawing some plots. Say you don't know anything about map projections. What would you try with geographical data? Probably the simplest way is considering (longitude, latitude) as a Cartesian coordinate and directly plot them.

In [10]:
```
alt.Chart(zipcodes_url).mark_circle().encode(
    x="longitude:Q",
    y="latitude:Q",
)
```

Out[10]:



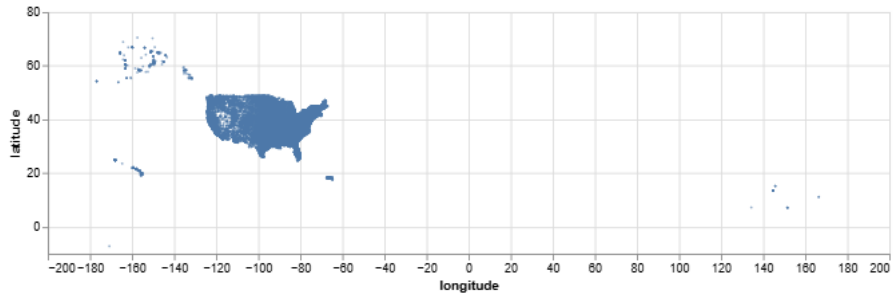
Actually this itself is a map projection called Equirectangular projection (https://en.wikipedia.org/wiki/Equirectangular_projection). This projection (or almost a *non-projection*) is super straight-forward and doesn't require any processing of the data. So, often it is used to just quickly explore geographical data. As you dig deeper, you still want to think about which map projection fits your need best. Don't just use equirectangular projection without any thoughts!

Make it look slighly better by reducing the size of the circles and adjusting the aspect ratio.

Q: Can you adjust the circle size, width and height of the chart?

In [15]:
```
alt.Chart(zipcodes_url).mark_circle(size=3, opacity=0.6).encode(
    x="longitude:Q",
    y="latitude:Q"
).properties(
    width=700,
    height=200,

)
```

Out[15]:



But, a much better way to do this is explicitly specifying that they are lat, lng coordinates by using `longitude=` and `latitude=`, rather than `x=` and `y=`. If you do that, altair automatically adjust the aspect ratio.

Q: Try it?

In [23]:
```
alt.Chart(zipcodes_url).mark_circle(size=3, opacity=0.6).encode(
    longitude="longitude:Q",
    latitude="latitude:Q"
).properties(
    width=900,
    height=300,
)
```

Out[23]:

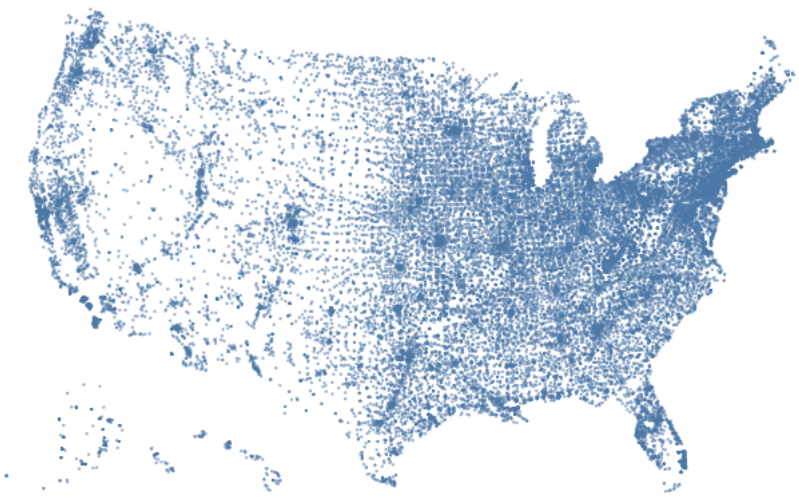
Because the American empire is far-reaching and complicated (ie. (https://www.youtube.com/watch?v=ASSOQDQvVLU)), the information density of this map is very low (although interesting). A common projection for visualizing US data is AlbersUSA (https://bl.ocks.org/mbostock/5545680), which uses Albers (equal-area) projection (https://en.wikipedia.org/wiki/Albers_projection). This is a standard projection used in United States Geological Survey and the United States Census Bureau. Albers USA contains a composition of US main land, Alaska, and Hawaii.

To use it, we call `project` method and specify which variables are `longitude` and `latitude`.

Q: use the `project` method to draw the map in the AlbersUsa projection.

In [26]:
```
alt.Chart(zipcodes_url).mark_circle(size=5, opacity=0.6).encode(
    longitude='longitude:Q',
    latitude='latitude:Q'
).project(
    type='albersUsa'
).properties(
    width=700,
    height=400
)
```

Out[26]:

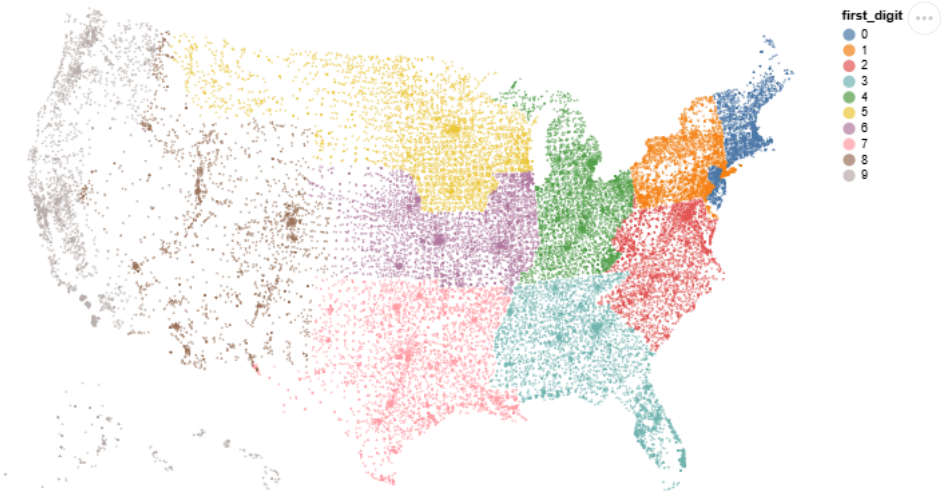


Let's visualize the large-scale zipcode patterns. We can use the fact that the zipcodes are hierarchically organized. That is, the first digit captures the largest area divisions and the other digits are about smaller geographical divisions.

Altair provides some data transformation functionalities. One of them is extracting a substring from a variable.

In [28]:
```
# from altair.expr import datum, substring
# Obsolete ^^^^^^^^^^

alt.Chart(zipcodes_url).mark_circle(size=2).transform_calculate(
    first_digit="substring(datum.zip_code, 0, 1)" # use in string instead
).encode(
    longitude="longitude:Q",
    latitude="latitude:Q",
    color="first_digit:N"
).project(
    type="albersUsa"
).properties(
    width=700,
    height=400
)
```

Out[28]:

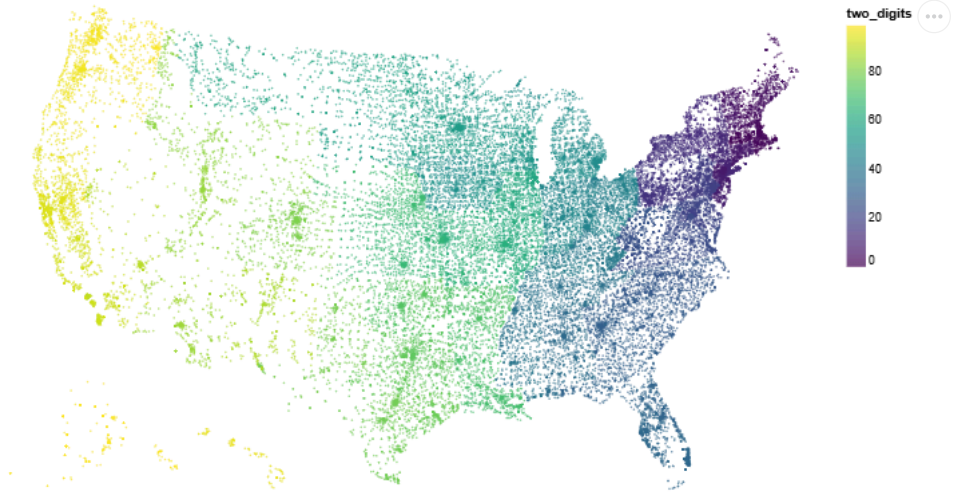


For each row (`datum`), you obtain the `zip_code` variable and get the substring (imagine Python list slicing), and then you call the result `first_digit` . Now, you can use this `first_digit` variable to color the circles. Also note that we specify `first_digit` as a *nominal* variable, not quantitative, to obtain a categorical colormap. But we can also play with it too.

Q: Extract the first two digits, name it as `two_digits` , and declare that as a quantitative variable? Any interesting patterns? (It tells us something about the history of US.)

In [ ]:
```
alt.Chart(zipcodes_url).mark_circle(size=2).transform_calculate(
    two_digits="parseInt(substring(datum.zip_code, 0, 2))"
).encode(
    longitude="longitude:Q",
    latitude="latitude:Q",
    color=alt.Color("two_digits:Q", scale=alt.Scale(scheme="viridis"))
).project(
    type="albersUsa"
).properties(
    width=700,
    height=400
)

# Notice a gradient moving from east to west
# Dense clusters in areas like the Northeast
# The ZIP code system started counting from the most important and dense establishments and postal hubs
```

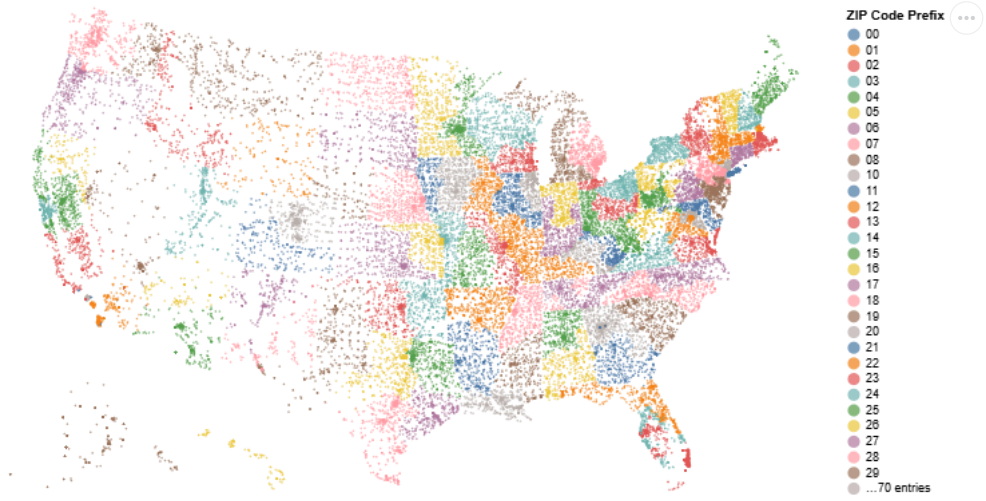
Out[ ]:



Q: Also try it with declaring the first two digits as a categorical variable

In [ ]:
```
alt.Chart(zipcodes_url).mark_circle(size=2).transform_calculate(
    two_digits="substring(datum.zip_code, 0, 2)"
).encode(
    longitude="longitude:Q",
    latitude="latitude:Q",
    color=alt.Color("two_digits:N", title="ZIP Code Prefix")
).project(
    type="albersUsa"
).properties(
    width=700,
    height=400
)
```

Out[ ]:

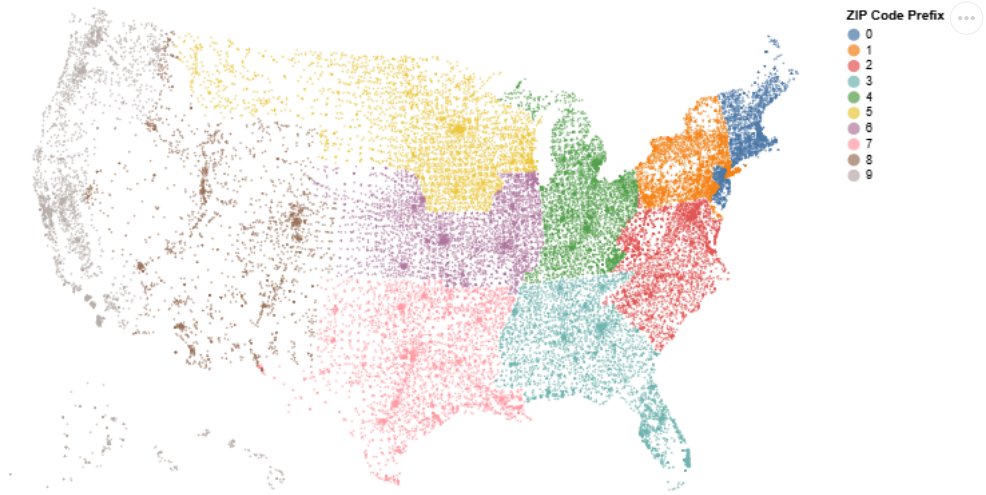


You can always click "view source" or "open in Vega Editor" to look at the json object that **defines** this visualization. You can embed this json object on your webpage and easily put up an interactive visualization.

Q: Can you put a tooltip that displays the zipcode when you mouse-over? Example [https://altair-viz.github.io/gallery/scatter_tooltips.html (https://altair-viz.github.io/gallery/scatter_tooltips.html)](https://altair-viz.github.io/gallery/scatter_tooltips.html)

In [32]:
```
alt.Chart(zipcodes_url).mark_circle(size=2).transform_calculate(
    two_digits="substring(datum.zip_code, 0, 1)"
).encode(
    longitude="longitude:Q",
    latitude="latitude:Q",
    color=alt.Color("two_digits:N", title="ZIP Code Prefix"),
    tooltip=["zip_code:N"]
).project(
    type="albersUsa"
).properties(
    width=700,
    height=400
)
```

Out[32]:



Choropleth

Let's try some choropleth now. Vega datasets have US county / state boundary data (`us_10m`) and world country boundary data (`world-110m`). You can take a look at the boundaries on GitHub (they renders topoJSON files):

- https://github.com/vega/vega/blob/main/docs/data/us-10m.json (https://github.com/vega/vega/blob/main/docs/data/us-10m.json)
- https://github.com/vega/vega/blob/main/docs/data/world-110m.json (https://github.com/vega/vega/blob/main/docs/data/world-110m.json)

If you click "Raw" then you can take a look at the actual file, which is hard to read.

Essentially, each file is a large dictionary with the following keys.

In [33]:
```
usmap = data.us_10m()
usmap.keys()
```

Out[33]:
```
dict_keys(['type', 'transform', 'objects', 'arcs'])
```

In [34]:
```
usmap["type"]
```

Out[34]:
```
'Topology'
```

In [35]:
```
usmap["transform"]
```

Out[35]:
```
{'scale': [0.003589294092944858, 0.0005371535195261037],
 'translate': [-179.1473400003406, 17.67439566600018]}
```

This `transformation` is used to *quantize* the data and store the coordinates in integer (easier to store than float type numbers).

https://github.com/topojson/topojson-specification#212-transforms (https://github.com/topojson/topojson-specification#212-transforms)

In [36]:
```
usmap["objects"].keys()
```

Out[36]:
```
dict_keys(['counties', 'states', 'land'])
```

This data contains not only county-level boundaries (objects) but also states and land boundaries.

In [37]:
```
usmap["objects"]["land"]["type"], usmap["objects"]["states"]["type"], usmap["objects"]["counties"]["type"]
```

Out[37]:
```
('MultiPolygon', 'GeometryCollection', 'GeometryCollection')
```

`land` is a multipolygon (one object) and `states` and `counties` contains many geometrics (multipolygons) because there are many states (counties). We can look at a state as a set of arcs that define it. It's `id` captures the identity of the state and is the key to link to other datasets.

In [38]:
```
state1 = usmap["objects"]["states"]["geometries"][1]
state1
```

Out[38]:
```
{'type': 'MultiPolygon',
 'arcs': [[[10337]],
  [[10342]],
  [[10341]],
  [[10343]],
  [[10834, 10340]],
  [[10344]],
  [[10345]],
  [[10338]]],
 'id': 15}
```

The `arcs` referred here is defined in `usmap['arcs']` .

In [39]:
```
usmap["arcs"][:10]
```

Out[39]:
```
[[[15739, 57220], [0, 0]],
 [[15739, 57220], [29, 62], [47, -273]],
 [[15815, 57009], [-6, -86]],
 [[15809, 56923], [0, 0]],
 [[15809, 56923], [-36, -8], [6, -210], [32, 178]],
 [[15811, 56883], [9, -194], [44, -176], [-29, -151], [-24, -319]],
 [[15811, 56043], [-12, -216], [26, -171]],
 [[15825, 55656], [-2, 1]],
 [[15823, 55657], [-19, 10], [26, -424], [-26, -52]],
 [[15804, 55191], [-30, -72], [-47, -344]]]
```

It seems pretty daunting to work with this dataset, but fortunately people have already built tools to handle such data.

In [40]:
```
# states
states = alt.topo_feature(data.us_10m.url, "states")

# us counties
us_counties = alt.topo_feature(data.us_10m.url, "counties")
```

In [41]:
```
states
```

Out[41]:
```
UrlData({
  format: TopoDataFormat({
    feature: 'states',
    type: 'topojson'
  }),
  url: 'https://cdn.jsdelivr.net/npm/vega-datasets@v1.29.0/data/us-10m.json'
})
```

Q. Can you find a mark for geographical shapes from here [https://altair-viz.github.io/user_guide/marks/index.html#marks (https://altair-viz.github.io/user_guide/marks/index.html#marks)](https://altair-viz.github.io/user_guide/marks/index.html#marks) and draw the states?

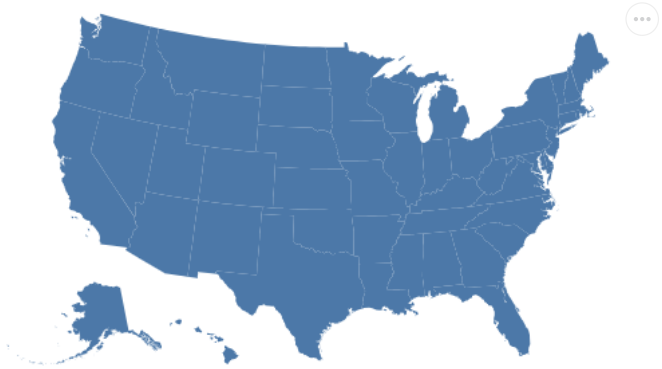
In [45]:
```
alt.Chart(states).mark_geoshape(
    stroke="white"
).encode(
    longitude="longitude:Q",
    latitude="latitude:Q",
).properties(
    width=700,
    height=300
)
```

Out[45]:

And then project it using the `albersUsa` ?

In [ ]:
```
alt.Chart(states).mark_geoshape(
    stroke="white"
).project(
    type="albersUsa"
).properties(
    width=700,
    height=400
)
```

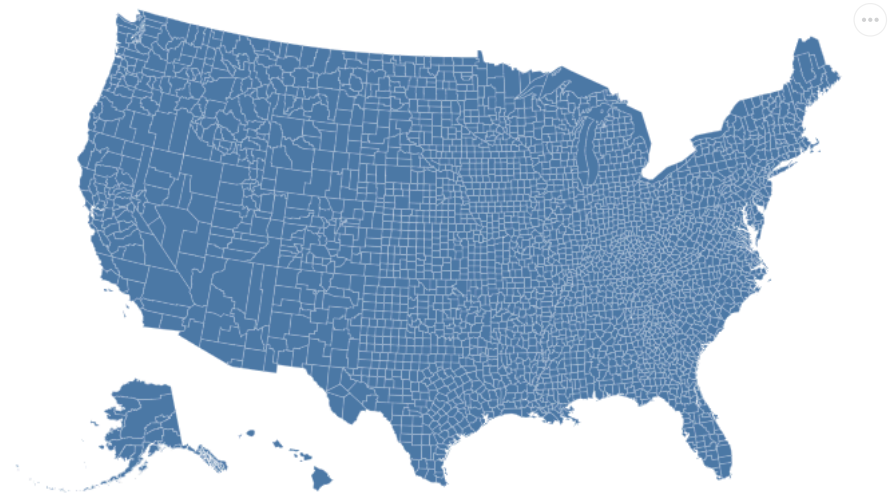
Out[ ]:



Can you do the same thing with counties and draw county boundaries? (hint: you have to use `alt.topo_feature()`)

In [46]:
```
counties = alt.topo_feature(data.us_10m.url, "counties")

alt.Chart(counties).mark_geoshape(
    stroke="white",
    strokeWidth=0.2
).project(
    type="albersUsa"
).properties(
    width=700,
    height=400
)
```

Out[46]:



Let's load some county-level unemployment data.

In [47]:
```
unemp_data = data.unemployment(sep="\t")
unemp_data.head()
```

Out[47]:

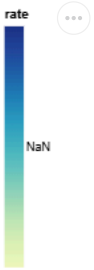
	id	rate
0	1001	0.097
1	1003	0.091
2	1005	0.134
3	1007	0.121
4	1009	0.099

This dataset has unemployment rate. (When? We don't know.) We don't care about data provenance here because the goal is quickly try out choropleth. But if you're working with a real dataset, you should be very sensitive about the provenance of your dataset. Make sure you understand where the data came from and how it was processed.

Anyway, for each county specified with `id` . To combine two datasets, we use "Lookup transform" - https://vega.github.io/vega/docs/transforms/lookup/ (https://vega.github.io/vega/docs/transforms/lookup/). Essentially, we use the `id` in the map data to look up (again) `id` field in the `unemp_data` and then bring in the `rate` variable. Then, we can use that `rate` variable to encode the color of the `geoshape` mark.

In [48]:
```
alt.Chart(us_counties).mark_geoshape().project(type="albersUsa").transform_lookup(
    lookup="id", from_=alt.LookupData(unemp_data, "id", ["rate"])
).encode(color="rate:Q").properties(width=700, height=400)
```

Out[48]:



There you have it, a nice choropleth map.

Raster visualization with datashader

Although many geovisualizations use vector graphics, raster visualization is still useful especially when you deal with images and lots of datapoints. Datashader is a package that aggregates and visualizes a large amount of data very quickly. Given a *scene* (visualization boundary, resolution, etc.), it quickly aggregate the data and produce **pixels** and send them to you.

To appreciate its power, we need a fairly large dataset. Let's use NYC taxi trip dataset on Kaggle: https://www.kaggle.com/kentonnlp/2014-new-york-city-taxi-trips (https://www.kaggle.com/kentonnlp/2014-new-york-city-taxi-trips) You can download even bigger trip data from NYC open data website: https://opendata.cityofnewyork.us/data/ (https://opendata.cityofnewyork.us/data/)

And you want to install the datashader, bokeh, and holoviews first if you don't have them yet.

```
pip install -U datashader bokeh holoviews
```

or

```
conda install datashader bokeh holoviews
```

In [1]:
```
%matplotlib inline

import pandas as pd
import datashader as ds
from datashader import transfer_functions as tf
from colorcet import fire
```

Because the dataset is pretty big, let's use a small sample first. For this visualization, we only keep the dropoff location.

In [2]:
```
nyctaxi_small = pd.read_csv(
    "./nyc_taxi_data_2014.csv",
    nrows=10000,
    usecols=["dropoff_longitude", "dropoff_latitude"],
)
nyctaxi_small.head()
```

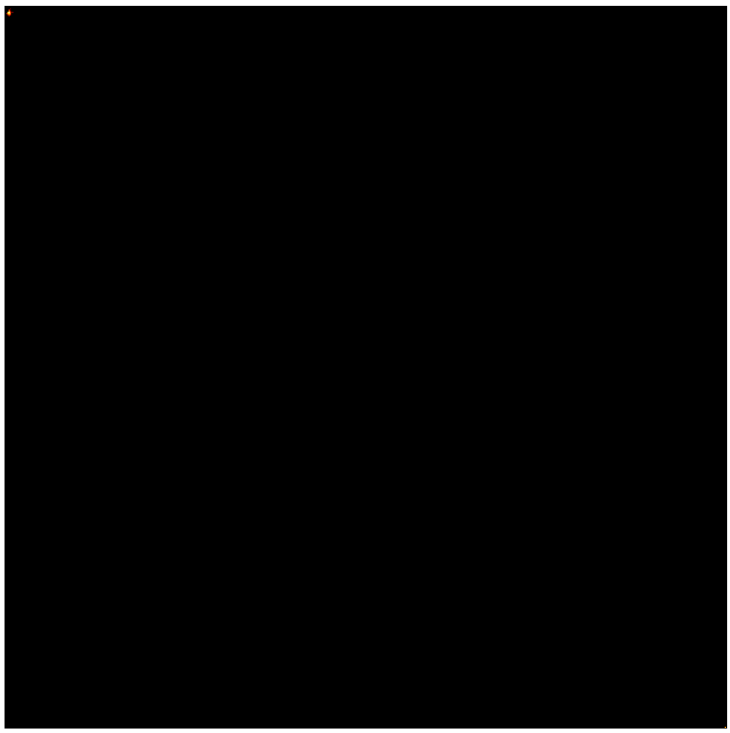
Out[2]:

	dropoff_longitude	dropoff_latitude
0	-73.982227	40.731790
1	-73.960449	40.763995
2	-73.986626	40.765217
3	-73.979863	40.777050
4	-73.984367	40.720524

Although the dataset is different, we can still follow the example here: https://datashader.org/getting_started/Introduction.html (https://datashader.org/getting_started/Introduction.html)

In [3]:
```
agg = ds.Canvas().points(nyctaxi_small, "dropoff_longitude", "dropoff_latitude")
tf.set_background(tf.shade(agg, cmap=fire), "black")
```

Out[3]:



Why can't we see anything? Do you see the small dots on the left top? Can that be New York City? Maybe we don't see anything because some people travel very far? or because the dataset has some missing data?

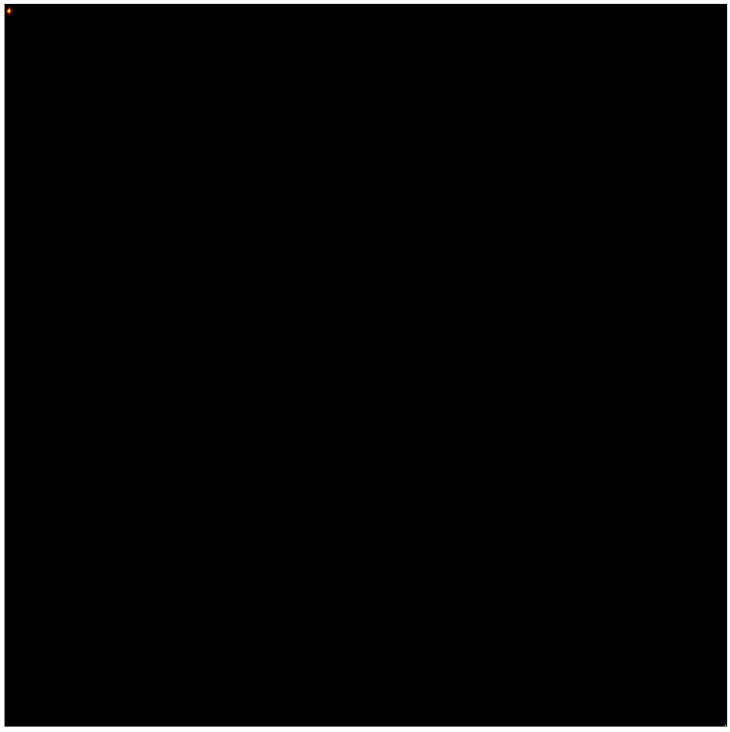
Q: Can you first check whether there are NaNs? Then drop them and draw the map again?

In [4]:
```
print(nyctaxi_small[['dropoff_longitude', 'dropoff_latitude']].isna().sum())
```

```
dropoff_longitude    1
dropoff_latitude     1
dtype: int64
```

In [5]:
```
nyctaxi_clean = nyctaxi_small.dropna(subset=['dropoff_longitude', 'dropoff_latitude'])
agg = ds.Canvas().points(nyctaxi_clean, "dropoff_longitude", "dropoff_latitude")
tf.set_background(tf.shade(agg, cmap=fire), "black")
```

Out[5]:

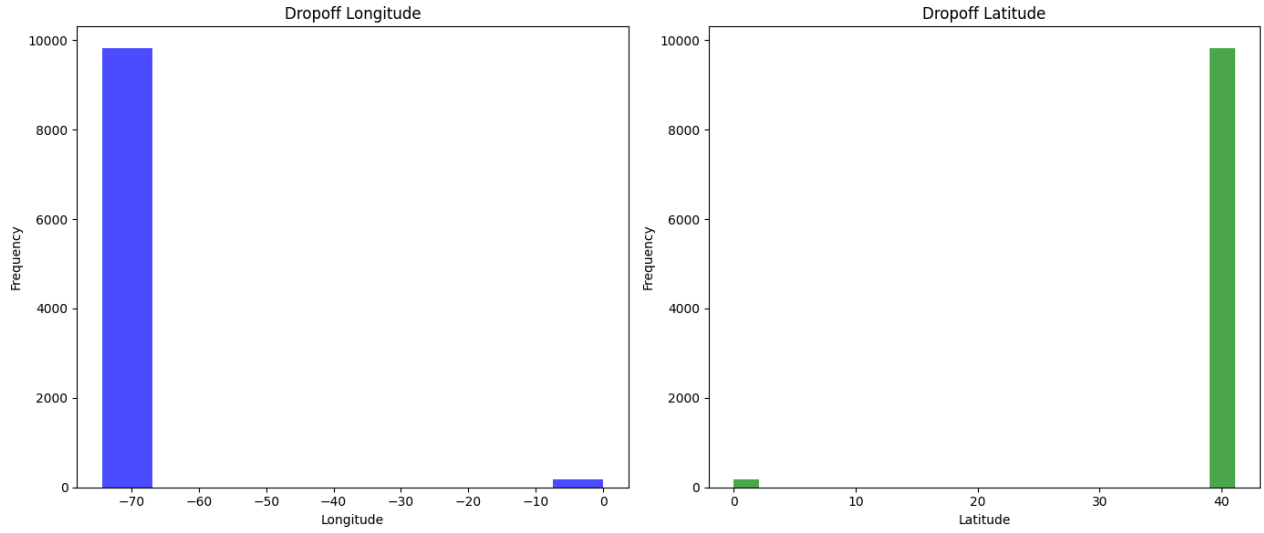


So it's not about the missing data.

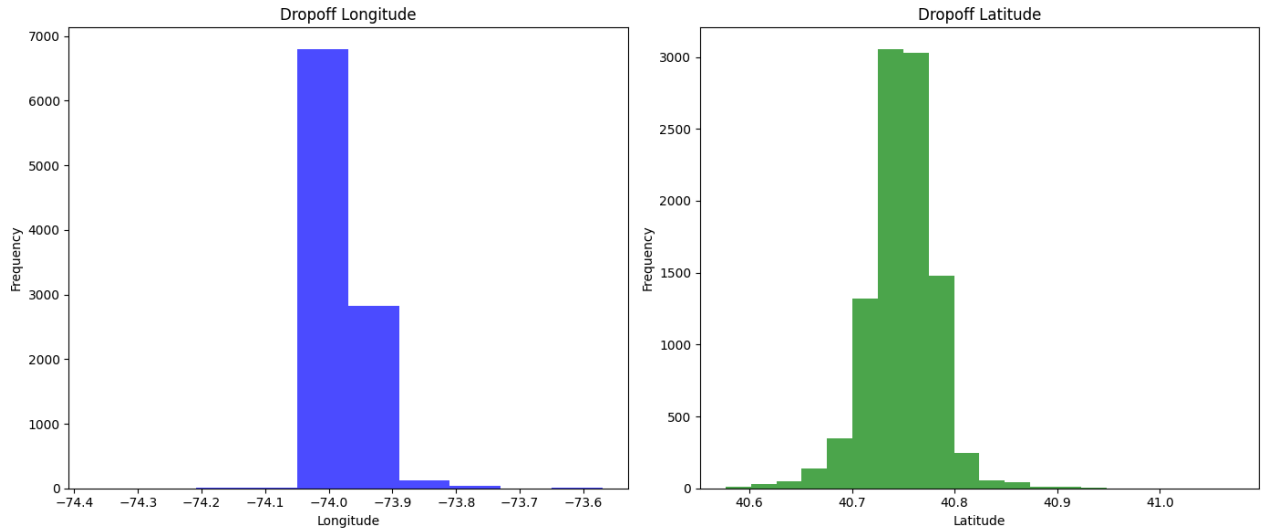
Q: Can you identify the issue and draw the map like the following?

hint: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.between.html (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.between.html) and histograms may be helpful.

In [6]:
```
import matplotlib.pyplot as plt

plt.figure(figsize=(14, 6))

plt.subplot(1, 2, 1)
plt.hist(nyctaxi_small['dropoff_longitude'], bins=10, color='blue', alpha=0.7)
plt.title("Dropoff Longitude")
plt.xlabel("Longitude")
plt.ylabel("Frequency")

plt.subplot(1, 2, 2)
plt.hist(nyctaxi_small['dropoff_latitude'], bins=20, color='green', alpha=0.7)
plt.title("Dropoff Latitude")
plt.xlabel("Latitude")
plt.ylabel("Frequency")

# Show the plots
plt.tight_layout()
plt.show()
```

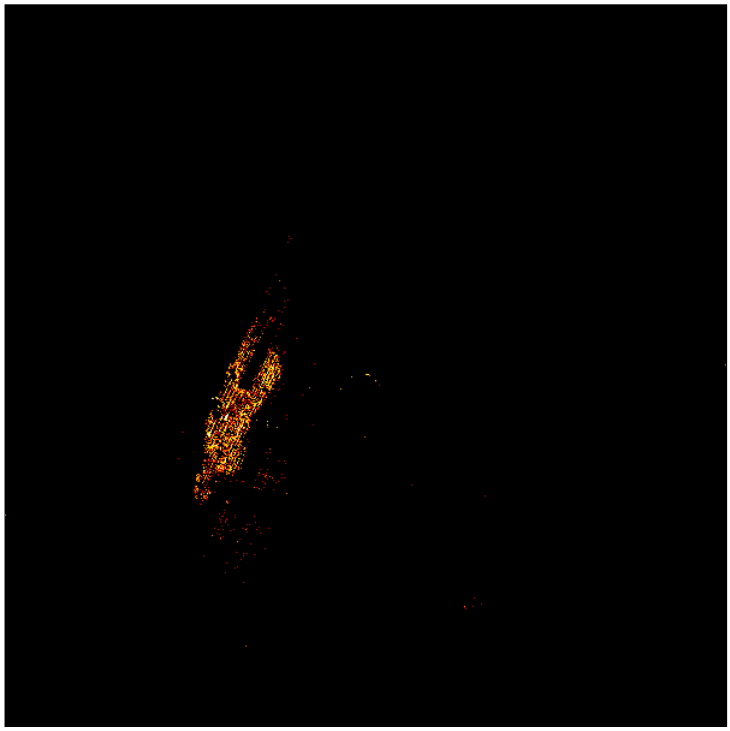


In [7]:
```
import matplotlib.pyplot as plt

tmp = nyctaxi_small[nyctaxi_small['dropoff_longitude'].between(-80, -60)
                    & nyctaxi_small['dropoff_latitude'].between(30, 50)]

plt.figure(figsize=(14, 6))

plt.subplot(1, 2, 1)
plt.hist(tmp['dropoff_longitude'], bins=10, color='blue', alpha=0.7)
plt.title("Dropoff Longitude")
plt.xlabel("Longitude")
plt.ylabel("Frequency")

plt.subplot(1, 2, 2)
plt.hist(tmp['dropoff_latitude'], bins=20, color='green', alpha=0.7)
plt.title("Dropoff Latitude")
plt.xlabel("Latitude")
plt.ylabel("Frequency")

# Show the plots
plt.tight_layout()
plt.show()
```



In [8]:
```
valid_lon = nyctaxi_small['dropoff_longitude'].between(-74.3, -63)
valid_lat = nyctaxi_small['dropoff_latitude'].between(39.5, 41)

valid_data = nyctaxi_small[valid_lon & valid_lat]

canvas = ds.Canvas()
agg = canvas.points(valid_data, 'dropoff_longitude', 'dropoff_latitude')
tf.set_background(tf.shade(agg, cmap=fire), 'black')
```

Out[8]:



Do you see the black empty space at the center? That looks like the Central Park. Maybe we can explore the data interactively.

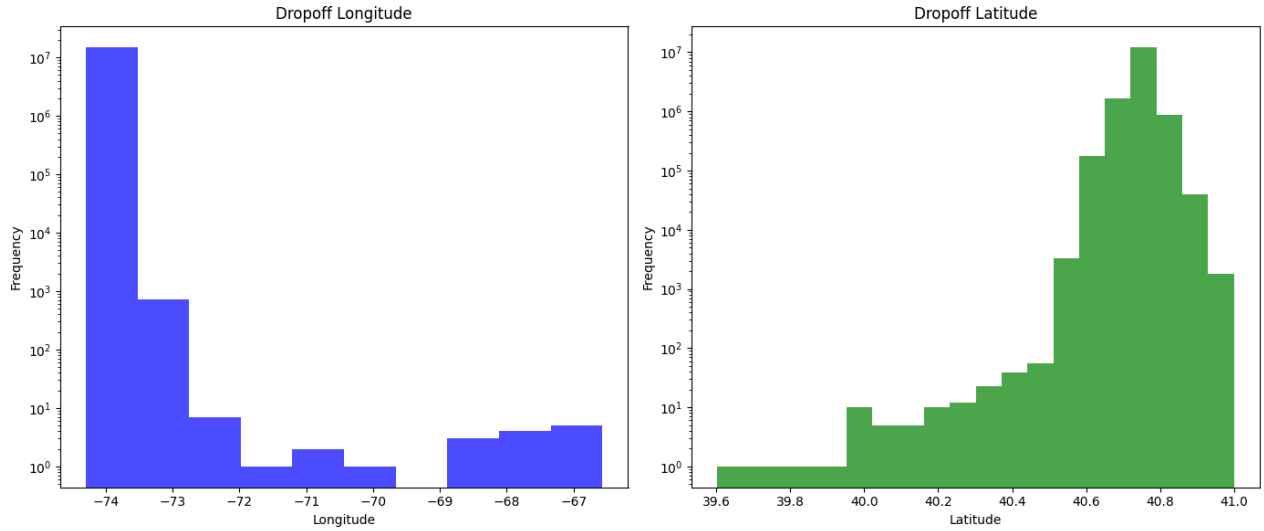
Q. Let's load the **whole** dataset. It may take some time. **Apply the same data cleaning procedure.**

In [9]:
```
import gc
gc.collect()

nyctaxi = pd.read_csv(
    "./nyc_taxi_data_2014.csv",
    usecols=["dropoff_longitude", "dropoff_latitude"],
)

valid_lon = nyctaxi['dropoff_longitude'].between(-74.3, -63)
valid_lat = nyctaxi['dropoff_latitude'].between(39.5, 41)

nyctaxi_clean = nyctaxi[valid_lon & valid_lat]

nyctaxi_clean.head()
```

Out[9]:

	dropoff_longitude	dropoff_latitude
0	-73.982227	40.731790
1	-73.960449	40.763995
2	-73.986626	40.765217
3	-73.979863	40.777050
4	-73.984367	40.720524

In [10]:
```
plt.figure(figsize=(14, 6))

plt.subplot(1, 2, 1)
plt.hist(nyctaxi_clean['dropoff_longitude'], bins=10, color='blue', alpha=0.7)
plt.title("Dropoff Longitude")
plt.xlabel("Longitude")
plt.ylabel("Frequency")
plt.yscale('log')

plt.subplot(1, 2, 2)
plt.hist(nyctaxi_clean['dropoff_latitude'], bins=20, color='green', alpha=0.7)
plt.title("Dropoff Latitude")
plt.xlabel("Latitude")
plt.ylabel("Frequency")
plt.yscale('log')

# Show the plots
plt.tight_layout()
plt.show()
```

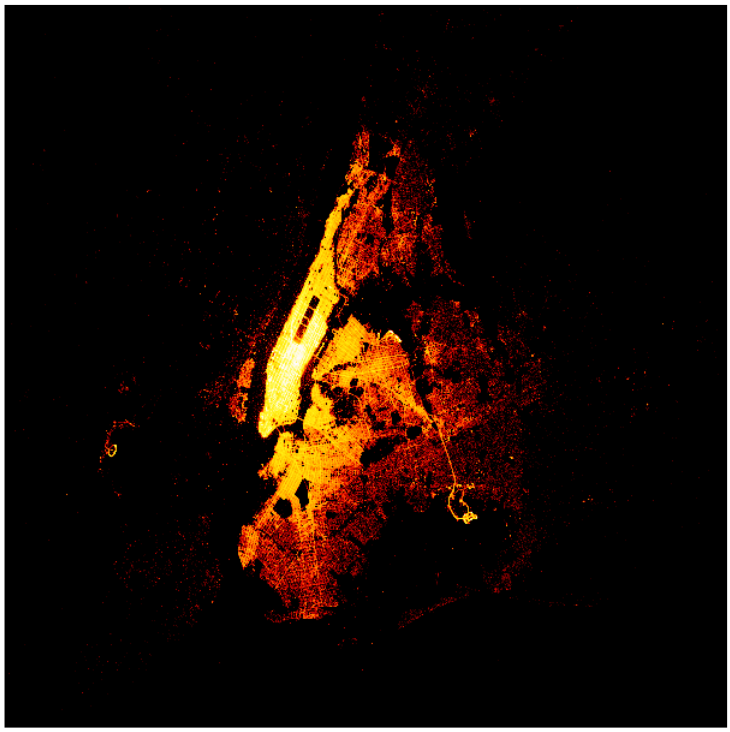


In [11]:
```
valid_lon = nyctaxi['dropoff_longitude'].between(-74.3, -73.5)
valid_lat = nyctaxi['dropoff_latitude'].between(40.5, 41)

nyctaxi_clean2 = nyctaxi[valid_lon & valid_lat]
```

Can you feed the data directly to datashader to reproduce the static plot, this time with the full data?

In [12]:
```
canvas = ds.Canvas()
agg = canvas.points(nyctaxi_clean2, 'dropoff_longitude', 'dropoff_latitude')
nyctaxi_filtered = nyctaxi_clean2
tf.set_background(tf.shade(agg, cmap=fire), 'black')
```

Out[12]:



Let's try the interactive version from here: https://datashader.org/getting_started/Introduction.html (https://datashader.org/getting_started/Introduction.html)

In [13]:
```
import gc
gc.collect()
```

Out[13]:
```
34968
```

In [14]:
```
import holoviews as hv
from holoviews.element.tiles import EsriImagery
from holoviews.operation.datashader import datashade

hv.extension("bokeh")

map_tiles = EsriImagery().opts(alpha=0.5, width=900, height=480, bgcolor="black")
points = hv.Points(nyctaxi_filtered, ["dropoff_longitude", "dropoff_latitude"])
taxi_trips = datashade(
    points, x_sampling=1, y_sampling=1, cmap=fire, width=900, height=480
)

map_tiles * taxi_trips
```

Out[14]:

Why does it say "map data not yet available"? The reason is the difference between two coordinate systems. If you google this error message, you can find https://stackoverflow.com/questions/44487898/map-background-with-datashader-map-data-not-yet-available (https://stackoverflow.com/questions/44487898/map-background-with-datashader-map-data-not-yet-available).

You can use `datashader.utils.lnglat_to_meters` to convert your latitudes and longitudes to a format that holoviews understands. More on this here: https://datashader.org/user_guide/Geography.html (https://datashader.org/user_guide/Geography.html)

Q: Can you draw an interactive map by converting the lnglat data to x, y coordinate explained above?

In [16]:
```
from datashader.utils import lnglat_to_meters
x, y = lnglat_to_meters(
    nyctaxi_filtered['dropoff_longitude'].values,
    nyctaxi_filtered['dropoff_latitude'].values
)
nyctaxi_filtered['x'] = x
nyctaxi_filtered['y'] = y
map_tiles = EsriImagery().opts(alpha=0.5, width=900, height=480, bgcolor="black")
points = hv.Points(nyctaxi_filtered, ["x", "y"])
taxi_trips = datashade(points, x_sampling=1, y_sampling=1, cmap=fire, width=900, height=480)
map_tiles * taxi_trips
```

```
C:\Users\Stefan\AppData\Local\Temp\ipykernel_11228\4247500848.py:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-cop
y
  nyctaxi_filtered['x'] = x
C:\Users\Stefan\AppData\Local\Temp\ipykernel_11228\4247500848.py:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-cop
y
  nyctaxi_filtered['y'] = y
```

Out[16]:

It's interactive! Actually, if you are running a bokeh server and there is a live python process, the map quickly refreshes and show more details as you zoom.

Q: how many rows (data points) are we visualizing right now?

In [17]:
```
len(nyctaxi_filtered)
```

Out[17]:
```
14779488
```

That's a lot of data points. If we are using a vector format, it is probably hopeless to expect any interactivity because you need to move that many points! Yet, datashader + holoviews + bokeh renders everything almost in real time!

Leaflet

Another useful tool is Leaflet. It allows you to use various map tile data (Google maps, Open streetmap, ...) with many types of marks (points, heatmap, etc.). Leaflet.js (https://leafletjs.com) is one of the easiest options to do that on the web, and there is a Python bridge of it: https://github.com/jupyter-widgets/ipyleaflet (https://github.com/jupyter-widgets/ipyleaflet). Although we will not go into details, it's certainly something that's worth checking out if you're using geographical data.

In [20]:
```
from ipyleaflet import Map, Marker, MarkerCluster, basemaps, basemap_to_tiles
sample_df = nyctaxi_filtered[['dropoff_latitude', 'dropoff_longitude']].dropna().sample(10000)
markers = [
    Marker(location=(row['dropoff_latitude'], row['dropoff_longitude']))
    for _, row in sample_df.iterrows()
]

marker_cluster = MarkerCluster(markers=markers)

m = Map(center=(40.7128, -74.0060), zoom=11, basemap=basemaps.Esri.WorldImagery)
m.add_layer(marker_cluster)
m
```

Out[20]:

In [ ]: