Data 608 Module 2

September 30, 2019

[5]:
```
import plotly
```

[6]:
```
import datashader as ds
import datashader.transfer_functions as tf
import datashader.glyphs
from datashader import reductions
from datashader.core import bypixel
from datashader.utils import lnglat_to_meters as webm, export_image
from datashader.colors import colormap_select, Greys9, viridis, inferno
import copy


from pyproj import Proj, transform
import numpy as np
import pandas as pd
import urllib
import json
import datetime
import colorlover as cl

import plotly.offline as py
import plotly.graph_objs as go
from plotly import tools

from shapely.geometry import Point, Polygon, shape
# In order to get shapley, you'll need to run [pip install shapely.geometry]␣
↪from your terminal

from functools import partial

from IPython.display import GeoJSON
```

[45]:
```
# Code to read in v17, column names have been updated (without upper case␣
↪letters) for v18

bk = pd.read_csv('/Users/christinakasman/Desktop/DATA 608 Pluto/BK2017V11.csv')
bx = pd.read_csv('/Users/christinakasman/Desktop/DATA 608 Pluto/BX2017V11.csv')
```

```
mn = pd.read_csv('/Users/christinakasman/Desktop/DATA 608 Pluto/MN2017V11.csv')
qn = pd.read_csv('/Users/christinakasman/Desktop/DATA 608 Pluto/QN2017V11.csv')
si = pd.read_csv('/Users/christinakasman/Desktop/DATA 608 Pluto/SI2017V11.csv')

ny = pd.concat([bk, bx, mn, qn, si], ignore_index=True)

#ny = pd.read_csv('nyc_pluto_18v2_csv/pluto_18v2.csv')


# Getting rid of some outliers
ny = ny[(ny['YearBuilt'] > 1850) & (ny['YearBuilt'] < 2020) & (ny['NumFloors'] !
 ↪= 0)]

ny.head()
```

```
/Users/christinakasman/anaconda3/lib/python3.6/site-
packages/ipykernel_launcher.py:9: FutureWarning:

Sorting because non-concatenation axis is not aligned. A future version
of pandas will change to not sort by default.

To accept the future behavior, pass 'sort=False'.

To retain the current behavior and silence the warning, pass 'sort=True'.
```

[45]:

```
     APPBBL APPDate              Address  AreaSource  AssessLand  AssessTot  \
1       0.0     NaN        10 JAY STREET           2      834300   10156950
17      0.0     NaN    1 PLYMOUTH STREET           2      129600     351000
20      0.0     NaN   65 PLYMOUTH STREET           2      178200     236700
22      0.0     NaN  135 PLYMOUTH STREET           2      514350    7785450
23      0.0     NaN        20 JAY STREET           2      972000   40980600

           BBL  BldgArea BldgClass  BldgDepth  …  YearAlter1  YearAlter2  \
1   3000010050    163894        O6      195.0  …        1994        2015
17  3000070021      9585        G1      113.0  …        1988           0
20  3000160005      5000        Q0      165.0  …           0           0
22  3000180001    211386        D5       48.0  …        2014           0
23  3000190001    500000        O6      207.0  …        1999           0

    YearBuilt ZMCode  ZipCode  ZoneDist1  ZoneDist2  ZoneDist3  ZoneDist4  \
1        1920    NaN  11201.0  M1-4/R8A       M3-1        NaN        NaN
17       1988    NaN  11201.0       M3-1       NaN        NaN        NaN
20       1920    NaN  11201.0       PARK       NaN        NaN        NaN
22       1900    NaN  11201.0  M1-4/R8A        NaN        NaN        NaN
23       1911    NaN  11201.0  M1-4/R8A        NaN        NaN        NaN
```

```
    ZoneMap
1       12d
17      12d
20      12d
22      12d
23      12d

[5 rows x 89 columns]
```

[13]:

```
wgs84 = Proj("+proj=longlat +ellps=GRS80 +datum=NAD83 +no_defs")
nyli = Proj("+proj=lcc +lat_1=40.66666666666666 +lat_2=41.03333333333333␣
 ↪+lat_0=40.16666666666666 +lon_0=-74 +x_0=300000 +y_0=0 +ellps=GRS80␣
 ↪+datum=NAD83 +to_meter=0.3048006096012192 +no_defs")
ny['XCoord'] = 0.3048*ny['XCoord']
ny['YCoord'] = 0.3048*ny['YCoord']
ny['lon'], ny['lat'] = transform(nyli, wgs84, ny['XCoord'].values, ny['YCoord'].
 ↪values)

ny = ny[(ny['lon'] < -60) & (ny['lon'] > -100) & (ny['lat'] < 60) & (ny['lat']␣
 ↪> 20)]

#Defining some helper functions for DataShader
background = "black"
export = partial(export_image, background = background, export_path="export")
cm = partial(colormap_select, reverse=(background!="black"))
```

Part 1: Binning and Aggregation Binning is a common strategy for visualizing large datasets. Binning is inherent to a few types of visualizations, such as histograms and 2D histograms (also check out their close relatives: 2D density plots and the more general form: heatmaps.

While these visualization types explicitly include binning, any type of visualization used with aggregated data can be looked at in the same way. For example, lets say we wanted to look at building construction over time. This would be best viewed as a line graph, but we can still think of our results as being binned by year:

[14]:

```
trace = go.Scatter(
    # I'm choosing BBL here because I know it's a unique key.
    x = ny.groupby('YearBuilt').count()['BBL'].index,
    y = ny.groupby('YearBuilt').count()['BBL']
)

layout = go.Layout(
    xaxis = dict(title = 'Year Built'),
    yaxis = dict(title = 'Number of Lots Built')
)

fig = go.Figure(data = [trace], layout = layout)
```

```
py.iplot(fig)
```

Something looks off… You're going to have to deal with this imperfect data to answer this first question.

But first: some notes on pandas. Pandas dataframes are a different beast than R dataframes, here are some tips to help you get up to speed:

Hello all, here are some pandas tips to help you guys through this homework:

Indexing and Selecting: .loc and .iloc are the analogs for base R subsetting, or filter() in dplyr

Group By: This is the pandas analog to group_by() and the appended function the analog to summarize(). Try out a few examples of this, and display the results in Jupyter. Take note of what's happening to the indexes, you'll notice that they'll become hierarchical. I personally find this more of a burden than a help, and this sort of hierarchical indexing leads to a fundamentally different experience compared to R dataframes. Once you perform an aggregation, try running the resulting hierarchical datafrome through a reset_index().

Reset_index: I personally find the hierarchical indexes more of a burden than a help, and this sort of hierarchical indexing leads to a fundamentally different experience compared to R dataframes. reset_index() is a way of restoring a dataframe to a flatter index style. Grouping is where you'll notice it the most, but it's also useful when you filter data, and in a few other split-apply-combine workflows. With pandas indexes are more meaningful, so use this if you start getting unexpected results.

Indexes are more important in Pandas than in R. If you delve deeper into the using python for data science, you'll begin to see the benefits in many places (despite the personal gripes I highlighted above.) One place these indexes come in handy is with time series data. The pandas docs have a huge section on datetime indexing. In particular, check out resample, which provides time series specific aggregation.

Merging, joining, and concatenation: There's some overlap between these different types of merges, so use this as your guide. Concat is a single function that replaces cbind and rbind in R, and the results are driven by the indexes. Read through these examples to get a feel on how these are performed, but you will have to manage your indexes when you're using these functions. Merges are fairly similar to merges in R, similarly mapping to SQL joins.

Apply: This is explained in the "group by" section linked above. These are your analogs to the plyr library in R. Take note of the lambda syntax used here, these are anonymous functions in python. Rather than predefining a custom function, you can just define it inline using lambda.

Browse through the other sections for some other specifics, in particular reshaping and categorical data (pandas' answer to factors.) Pandas can take a while to get used to, but it is a pretty strong framework that makes more advanced functions easier once you get used to it. Rolling functions for example follow logically from the apply workflow (and led to the best google results ever when I first tried to find this out and googled "pandas rolling")

Google Wes Mckinney's book "Python for Data Analysis," which is a cookbook style intro to pandas. It's an O'Reilly book that should be pretty available out there.

Question After a few building collapses, the City of New York is going to begin investigating older buildings for safety. The city is particularly worried about buildings that were unusually tall when they were built, since best-practices for safety hadn't yet been determined. Create a graph that shows how many buildings of a certain number of floors were built in each year (note: you may want to use a log scale for the number of buildings). Find a strategy to bin buildings (It should be clear 20-29-story buildings, 30-39-story buildings, and 40-49-story buildings were first built in large numbers, but does it make sense to continue in this way as you get taller?)

[113]:
```
ny['XCoord'].describe()
```

[113]:
```
count    8.117820e+05
mean     1.006355e+06
std      3.243865e+04
min      9.131230e+05
25%      9.895860e+05
50%      1.008949e+06
75%      1.029431e+06
max      1.067279e+06
Name: XCoord, dtype: float64
```

[114]:
```
ny['YCoord'].describe()
```

[114]:
```
count    811782.000000
mean     191449.447992
std       30465.111991
min      120308.000000
25%      168169.000000
50%      189198.000000
75%      210843.000000
max      272275.000000
Name: YCoord, dtype: float64
```

[50]:
```
grouped = ny.groupby(['YearBuilt', 'NumFloors']).count()
grouped.head()
```

[50]:
```
                     APPBBL  APPDate  Address  AreaSource  AssessLand  \
YearBuilt NumFloors
1851      3.0             3        0        3           3           3
          4.0             4        1        4           4           4
          5.0             1        1        1           1           1
1852      2.0             2        0        2           2           2
          3.0             7        0        7           7           7

                     AssessTot  BBL  BldgArea  BldgClass  BldgDepth  …  \
YearBuilt NumFloors                                                  …
1851      3.0                3    3         3          3          3  …
          4.0                4    4         4          4          4  …
```

```
             5.0                   1     1         1           1           1  …
1852         2.0                   2     2         2           2           2  …
             3.0                   7     7         7           7           7  …

                         YCoord  YearAlter1  YearAlter2  ZMCode  ZipCode  \
YearBuilt NumFloors
1851      3.0                 3           3           3       0        3
          4.0                 4           4           4       0        4
          5.0                 1           1           1       0        1
1852      2.0                 2           2           2       0        2
          3.0                 7           7           7       0        7

                         ZoneDist1  ZoneDist2  ZoneDist3  ZoneDist4  ZoneMap
YearBuilt NumFloors
1851      3.0                    3          0          0          0        3
          4.0                    4          0          0          0        4
          5.0                    1          0          0          0        1
1852      2.0                    2          0          0          0        2
          3.0                    7          0          0          0        7

[5 rows x 87 columns]
```

[170]:
```python
ny2 = ny[['YearBuilt', 'NumFloors','BBL']].copy()
```

[174]:
```python
ny2['decadebuilt'] = (np.floor(ny_floors['YearBuilt'] / 10.0).astype(int) * 10)
ny2.head()
```

[174]:

	YearBuilt	NumFloors	BBL	decadebuilt
1	1920	10.0	3000010050	1920
17	1988	1.0	3000070021	1980
20	1920	1.0	3000160005	1920
22	1900	7.0	3000180001	1900
23	1911	11.0	3000190001	1910

[191]:
```python
ny2['NumFloors'].describe()
```

[191]:
```
count    814035.000000
mean          2.442139
std           1.936177
min           0.500000
25%           2.000000
50%           2.000000
75%           2.500000
max         119.000000
Name: NumFloors, dtype: float64
```

[189]:
```
fig = px.box(ny2, x="decadebuilt", y="NumFloors")
fig.show()
```

[203]:
```
ny2.sort_values(by=['YearBuilt'], inplace = True)

#Bin levels
Levels = {
    1: "< 5 Floors",
    2: "5-10 Floors",
    3: "10-15 Floors",
    4: "15-20 Floors",
    5: "> 20 Floors"


}

def floorLevel(floor):
    if floor <=5:
        return Levels[1]
    elif floor > 5 and floor <= 10:
        return Levels[2]
    elif floor > 10 and floor <= 15:
        return Levels[3]
    elif floor > 15 and floor <= 20:
        return Levels[4]
    elif floor > 20:
        return Levels[5]
```

[213]:
```
ny2['level']= ""
ny2['level']= ny2['NumFloors'].apply(floorLevel)

ny2 = ny2[ny2.level != '< 5 Floors']
ny2 = ny2[ny2.level != '5-10 Floors']
ny2.head()
```

[213]:

	YearBuilt	NumFloors	BBL	decadebuilt	level
408723	1853	11.0	1002177501	1850	10-15 Floors
388488	1868	12.0	1013940023	1860	10-15 Floors
70164	1885	11.0	3024140025	1880	10-15 Floors
408695	1890	13.0	1002127503	1890	10-15 Floors
377755	1890	13.0	1008320066	1890	10-15 Floors

[214]:
```
fig = go.Figure(go.Histogram2d(
        x = ny2['decadebuilt'],
        y = ny2['level'],
    nbinsx = 20))
```

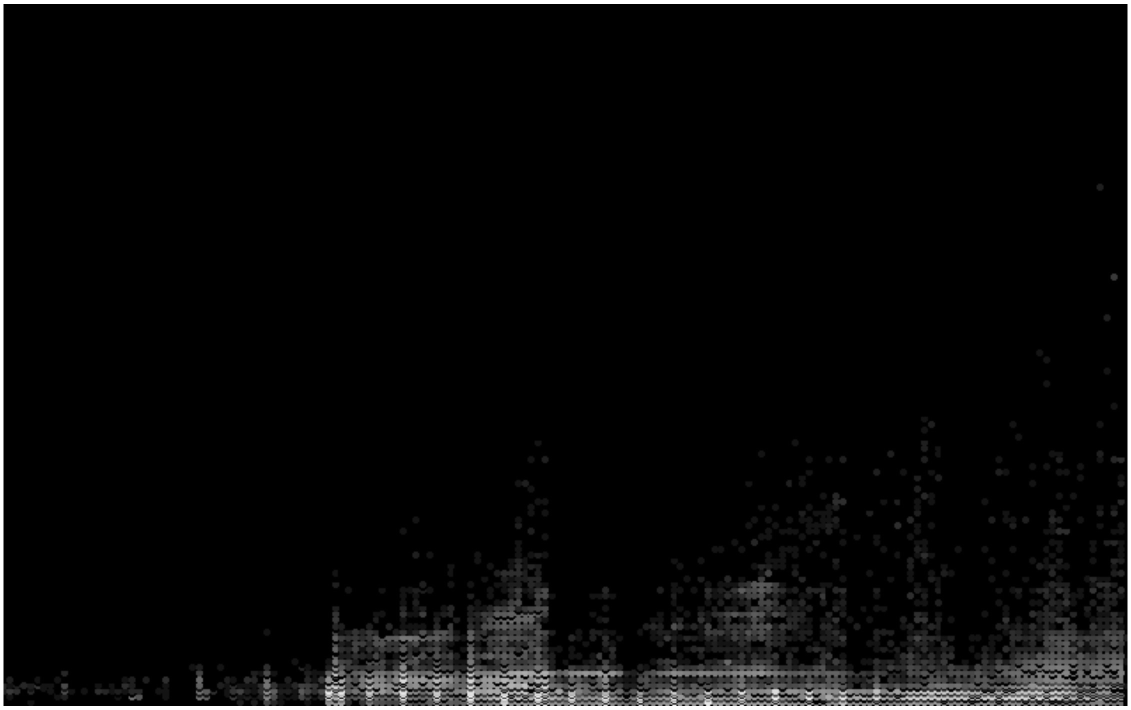
```
#fig.update_layout(yaxis_type="log")
fig.show()
```

Removed floors less than 10 to show unusually tall buildings

[91]:
```
yearbins = 200
floorbins = 200

yearBuiltCut = pd.cut(ny['YearBuilt'], np.linspace(ny['YearBuilt'].min(),␣
 ↪ny['YearBuilt'].max(), yearbins))
numFloorsCut = pd.cut(ny['NumFloors'], np.logspace(1, np.log(ny['NumFloors'].
 ↪max()), floorbins))

xlabels = np.floor(np.linspace(ny['YearBuilt'].min(), ny['YearBuilt'].max(),␣
 ↪yearbins))
ylabels = np.floor(np.logspace(1, np.log(ny['NumFloors'].max()), floorbins))

data = [
    go.Heatmap(z = ny.groupby([numFloorsCut, yearBuiltCut])['BBL'].count().
 ↪unstack().fillna(0).values,
              colorscale = 'Greens', x = xlabels, y = ylabels)
]

py.iplot(data)
```

[118]:
```
cvs = ds.Canvas(800, 500, x_range = (ny['YearBuilt'].min(), ny['YearBuilt'].
 ↪max()),
                                y_range = (ny['NumFloors'].min(),␣
 ↪ny['NumFloors'].max()))
agg = cvs.points(ny, 'YearBuilt', 'NumFloors')
view = tf.shade(agg, cmap = cm(Greys9), how='log')
export(tf.spread(view, px=2), 'yearvsnumfloors')
```

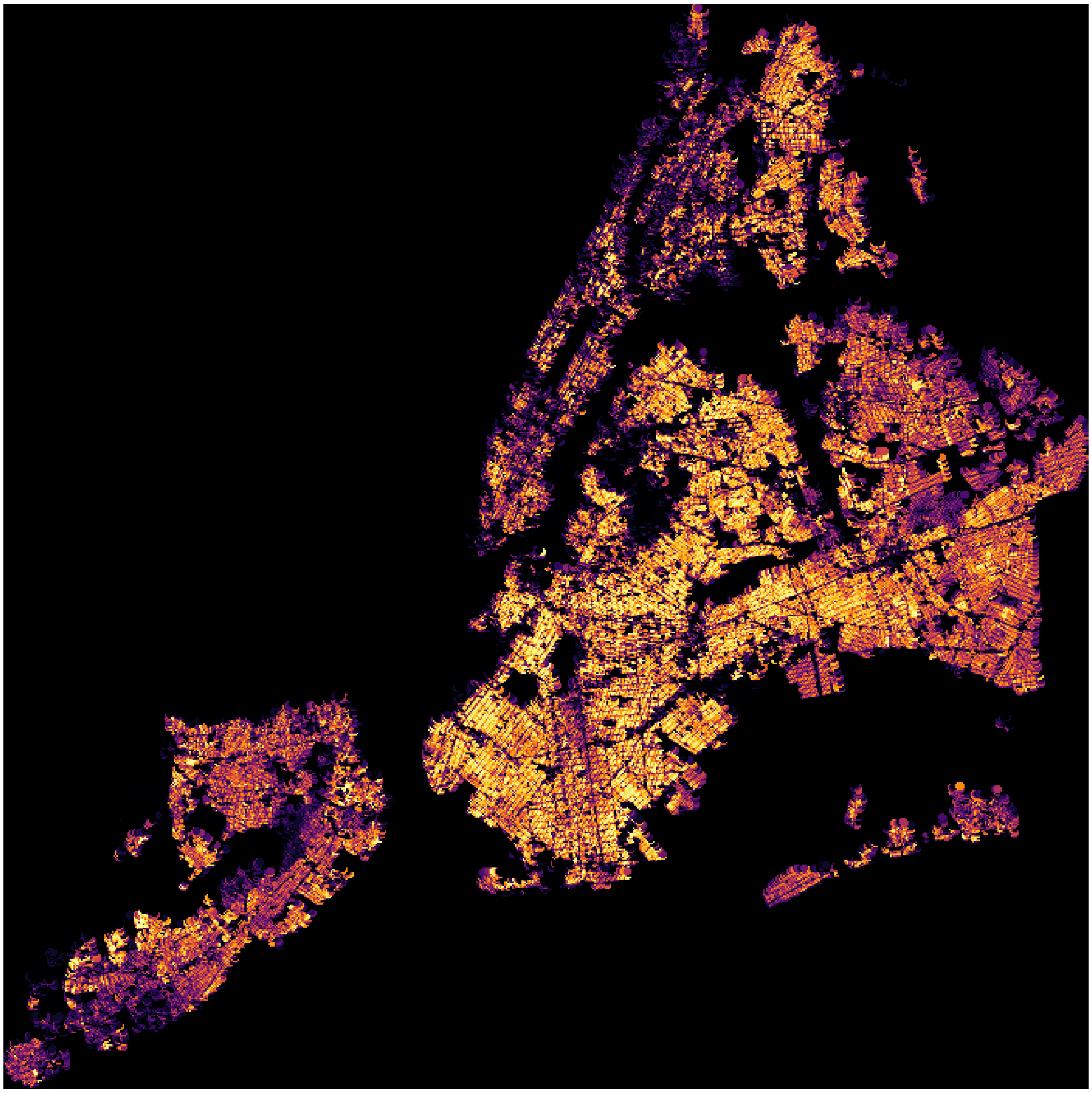
[118]:



[ ]:

[130]:
```
NewYorkCity   = (( -74.29,  -73.69), (40.49, 40.92))
cvs = ds.Canvas(700, 700)
agg = cvs.points(ny, 'XCoord', 'YCoord')
view = tf.shade(agg, cmap = cm(inferno))
export(tf.spread(view, px=2), 'firery')
```

[130]:



[133]:

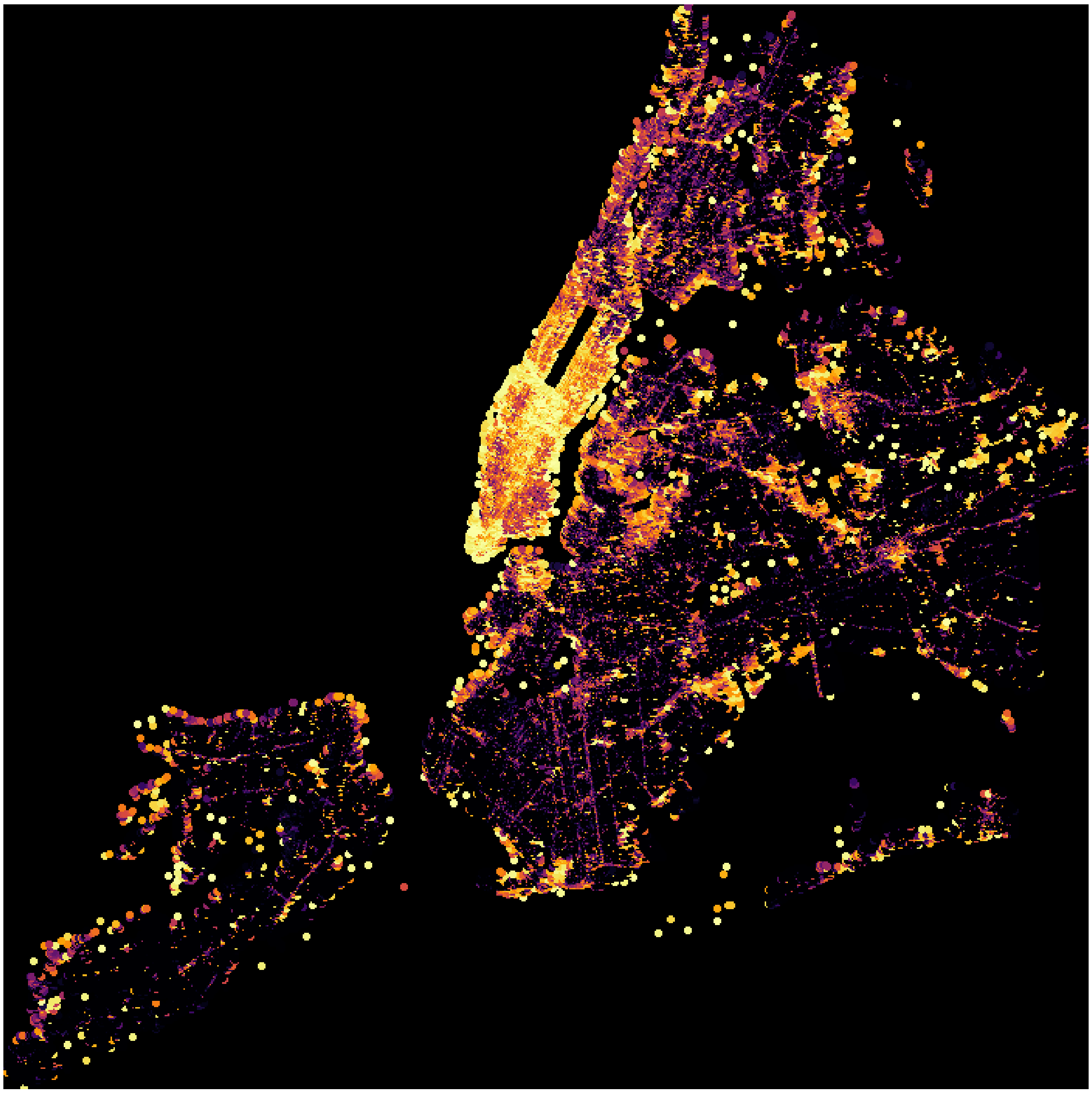
```

```

Question You work for a real estate developer and are researching underbuilt areas of the city. After looking in the Pluto data dictionary, you've discovered that all tax assessments consist of two parts: The assessment of the land and assessment of the structure. You reason that there should be a correlation between these two values: more valuable land will have more valuable structures on them (more valuable in this case refers not just to a mansion vs a bungalow, but an apartment tower vs a single family home). Deviations from the norm could represent underbuilt or overbuilt areas of the city. You also recently read a really cool blog post about bivariate choropleth maps, and think the technique could be used for this problem.

Datashader is really cool, but it's not that great at labeling your visualization. Don't worry about providing a legend, but provide a quick explanation as to which areas of the city are overbuilt, which areas are underbuilt, and which areas are built in a way that's properly correlated with their land value.

[139]:
```
NewYorkCity   = (( -74.29,  -73.69), (40.49, 40.92))
cvs = ds.Canvas(700, 700)
agg = cvs.points(ny, 'XCoord', 'YCoord',ds.mean('AssessLand'))
view = tf.shade(agg, cmap = cm(inferno))
export(tf.spread(view, px=2), 'firery')
```

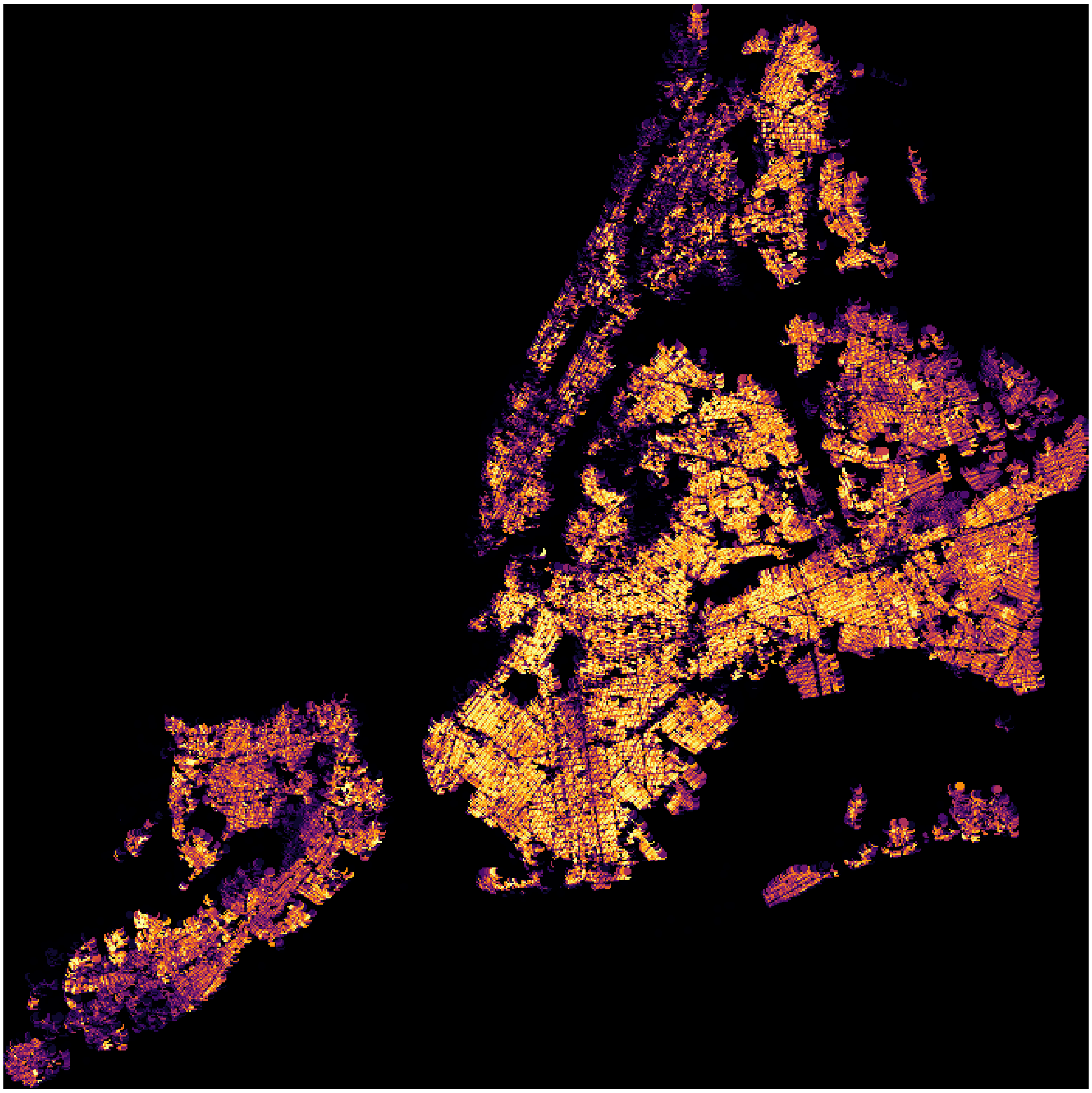
[139]:



[137]:
```
NewYorkCity   = (( -74.29,  -73.69), (40.49, 40.92))
cvs = ds.Canvas(700, 700)
```

```
agg = cvs.points(ny, 'XCoord', 'YCoord',ds.count('BBL'))
view = tf.shade(agg, cmap = cm(inferno))
export(tf.spread(view, px=2), 'firery')
```

[137]:



[218]:

```
ny['assessbldg'] = ny['AssessTot']-ny['AssessLand']
ny['assessRatio']=ny['assessbldg']/ny['AssessTot']
```

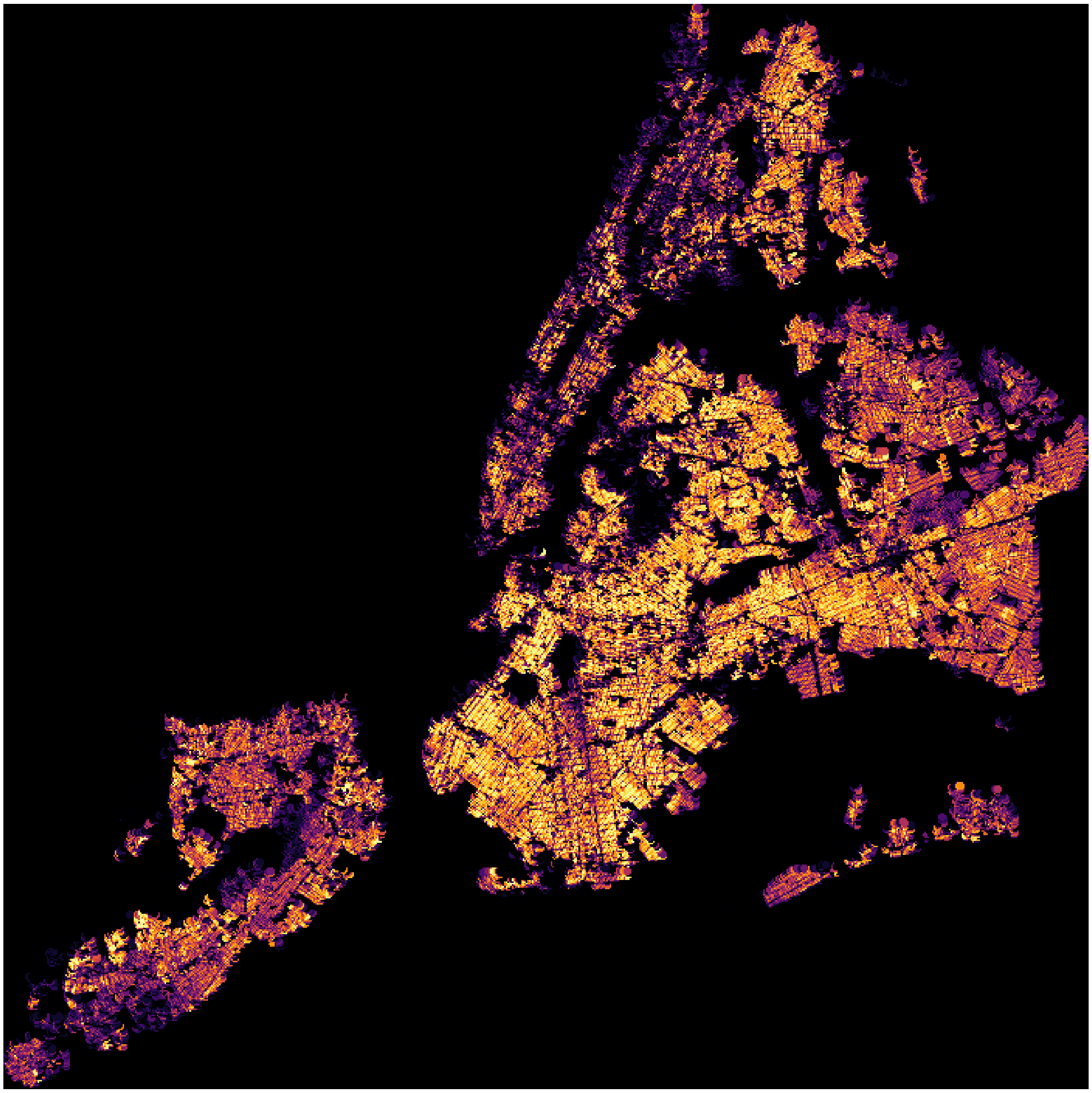
Answer Created a new column with ratio of building assessment to total assessment The brighter areas on the map have a higher ratio meaning that the buildings are overvalued compared to the land (and vice versa)

[222]:

```
NewYorkCity   = (( -74.29,  -73.69), (40.49, 40.92))
cvs = ds.Canvas(700, 700)
```

```
agg = cvs.points(ny, 'XCoord', 'YCoord',ds.count('assessRatio'))
view = tf.shade(agg, cmap = cm(inferno))
export(tf.spread(view, px=2), 'firery')
```

[222]:



[ ]:

[ ]: