Module2

February 25, 2023

[2]:
```
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

py.init_notebook_mode()
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
```

```
/Users/t0pth4t/Downloads/CUNY/DATA608/CUNY_DATA_608/module2/Module2.ipynb Cell
 ↪in 1
----> <a href='vscode-notebook-cell:/Users/t0pth4t/Downloads/CUNY/DATA608/
 ↪CUNY_DATA_608/module2/Module2.ipynb#W0sZmlsZQ%3D%3D?line=0'>1</a> import
 ↪datashader as ds
      <a href='vscode-notebook-cell:/Users/t0pth4t/Downloads/CUNY/DATA608/
 ↪CUNY_DATA_608/module2/Module2.ipynb#W0sZmlsZQ%3D%3D?line=1'>2</a> import
 ↪datashader.transfer_functions as tf
      <a href='vscode-notebook-cell:/Users/t0pth4t/Downloads/CUNY/DATA608/
 ↪CUNY_DATA_608/module2/Module2.ipynb#W0sZmlsZQ%3D%3D?line=2'>3</a> import
 ↪datashader.glyphs

ModuleNotFoundError: No module named 'datashader'
```

For module 2 we'll be looking at techniques for dealing with big data. In particular binning strategies and the datashader library (which possibly proves we'll never need to bin large data for visualization ever again.)

To demonstrate these concepts we'll be looking at the PLUTO dataset put out by New York City's department of city planning. PLUTO contains data about every tax lot in New York City.

PLUTO data can be downloaded from here. Unzip them to the same directory as this notebook, and you should be able to read them in using this (or very similar) code. Also take note of the data dictionary, it'll come in handy for this assignment.

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

# bk = pd.read_csv('PLUTO17v1.1/BK2017V11.csv')
# bx = pd.read_csv('PLUTO17v1.1/BX2017V11.csv')
# mn = pd.read_csv('PLUTO17v1.1/MN2017V11.csv')
# qn = pd.read_csv('PLUTO17v1.1/QN2017V11.csv')
# si = pd.read_csv('PLUTO17v1.1/SI2017V11.csv')

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

ny = pd.read_csv('pluto_22v2.csv')


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

```
/var/folders/f4/v17stl257nn9w40qtyt496380000gn/T/ipykernel_65343/3065575406.py:1
1: DtypeWarning:

Columns (21,22,24,26,28) have mixed types. Specify dtype option on import or set
low_memory=False.
```

I'll also do some prep for the geographic component of this data, which we'll be relying on for datashader.

You're not required to know how I'm retrieving the lattitude and longitude here, but for those interested: this dataset uses a flat x-y projection (assuming for a small enough area that the world is flat for easier calculations), and this needs to be projected back to traditional lattitude and longitude.

[ ]:
```
# 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"))
```

0.1 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:

[51]:
```
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.FigureWidget(data = [trace], layout = layout)
```

```
fig.show()
```

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.

0.1.1 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?)

0.1.2 Answer

First we can check what the range of values for numfloors is:

[30]:
```
print(ny['numfloors'].describe()[['min', 'max']])
```

```
min      1.0
max    104.0
Name: numfloors, dtype: float64
```

Then we can visualize the distribution of numfloors using a histogram:

[52]:
```
hist_trace = go.Histogram(x=ny['numfloors'], nbinsx=50)

# set the layout options
layout = go.Layout(title='Distribution of Number of Floors in NY buildings',
                   xaxis=dict(title='Number of Floors'),␣
 ↪yaxis=dict(title='Count'))

# create the Figure object and show the plot
fig = go.Figure(data=[hist_trace], layout=layout)

fig.show()
```

We can see the vast majority of buildings have 10 or fewer floors, but there are a few outliers. We can use a logarithmic scale to better visualize the distribution of the data:

[34]:
```
hist_trace = go.Histogram(x=ny['numfloors'], nbinsx=50)

# set the layout options with log scaling for the y-axis
layout = go.Layout(title='Distribution of Number of Floors in NY buildings (log␣
 ↪scale)',
                   xaxis=dict(title='Number of Floors'),␣
 ↪yaxis=dict(title='Count', type='log'))

# create the Figure object and show the plot
fig = go.Figure(data=[hist_trace], layout=layout)
```

```python
fig.show()
```

From the early plot of buildings built by year it seems that certain years had few or no buildings built. Let's check the number of buildings built in each year using a subset of the data:

[36]:
```python
# get the number of buildings built in each year between 1900 and 1940
year_counts = ny[(ny['yearbuilt'] >= 1900) & (ny['yearbuilt'] <=
 ↪1940)]['yearbuilt'].value_counts().sort_index()
print(year_counts)
```

```
1900.0     6440
1901.0    22230
1902.0      476
1903.0      460
1904.0      522
1905.0     6914
1906.0     1026
1907.0      996
1908.0      839
1909.0     1540
1910.0    41983
1911.0     1083
1912.0      883
1913.0      732
1914.0      668
1915.0    15363
1916.0      687
1917.0      534
1918.0      291
1919.0      369
1920.0    87703
1921.0     1118
1922.0     1246
1923.0     1641
1924.0     2246
1925.0    69677
1926.0     3256
1927.0     3598
1928.0     4346
1929.0     2189
1930.0    74907
1931.0    31000
1932.0     1224
1933.0      946
1934.0      374
1935.0    25085
1936.0      643
1937.0      701
```

```
1938.0      764
1939.0      929
1940.0    37904
Name: yearbuilt, dtype: int64
```

We can see while there are no years with 0 buildings built in this range there is a lot of variance in the data. For example 1933 only has 946 buildings built, while 1931 has 31,000. This gives us good reason to aggregate the data by decade.

Now we can create a chart that shows the number of buildings built in each decade by number of floors. We will bin using the min and max floor numbers in increments of 10. We will also make sure to use the log scale for the y-axis to better visualize the data.

[61]:
```python
# Create new column for decade using floor division
ny['decade'] = (ny['yearbuilt'] // 10) * 10

# create bins from min to max numfloors in increments of 10
floor_stats = ny['numfloors'].describe()[['min', 'max']]
floor_min = floor_stats['min']
floor_max = floor_stats['max']
bins = np.arange(floor_min, floor_max, 10)

# use pd.cut to bin the data
df_binned = ny.groupby([pd.cut(ny['numfloors'], bins), 'decade']).size().
 ↪reset_index(name='count')

# Create trace for each bin
floor_bin_lines = list(map(lambda floor_bin:
                           go.Scatter(
                               x=df_binned[df_binned['numfloors'] ==␣
 ↪floor_bin]['decade'],
                               y=df_binned[df_binned['numfloors'] ==␣
 ↪floor_bin]['count'],
                               name=f'{floor_bin.left}-{floor_bin.
 ↪right}-stories',
                               mode='lines+markers',
                               line=dict(width=2),
                               marker=dict(size=5),
                           ), df_binned['numfloors'].unique()))

# Create layout with log y-axis
layout = go.Layout(
    title='Number of Buildings by Decade and Number of Floors',
    xaxis=dict(title='Decade Built'),
    yaxis=dict(title='Number of Buildings', type='log')
)

# Create figure and plot
```

```
fig = go.Figure(data=floor_bin_lines, layout=layout)
fig.show()
```

It is clear from the plot above that even using a logarithmic scale, the 1-20 floor buildings are dominating the plot. Also, the line scatter chart format gives us a lot of noise in the form of data resolution that we do not need in order to visualize the data. Let's filter out buildings with less than 20 floors and replot using stacked bar charts:

[60]:
```
floor_stats = ny['numfloors'].describe()[['min', 'max']]
floor_max = floor_stats['max']
bins = np.arange(floor_min, floor_max, 10)

#filter out buildings with less than 20 floors
ny_more_than_twenty = ny[ny['numfloors'] >= 20]

# Bin buildings by decade and number of floors
df_binned = ny_more_than_twenty.groupby([pd.
 ↪cut(ny_more_than_twenty['numfloors'], bins), 'decade']).size().
 ↪reset_index(name='count')

# Group by decade and floor range and sum counts
df_grouped = df_binned.groupby(['decade', 'numfloors']).agg({'count': 'sum'}).
 ↪reset_index()

# Pivot to wide format for stacked bar chart
df_pivot = df_grouped.pivot(index='decade', columns='numfloors',␣
 ↪values='count').fillna(0)
df_pivot.columns = [str(col) for col in df_pivot.columns]  # Convert Interval␣
 ↪objects to strings

# Create stacked bar chart
data = [
    go.Bar(x=df_pivot.index, y=df_pivot[col], name=col) for col in df_pivot.
 ↪columns
]

# Update layout with log y-axis
layout = go.Layout(
    title='Number of Buildings by Decade and Number of Floors',
    xaxis=dict(title='Decade Built'),
    yaxis=dict(title='Number of Buildings', type='log'),
    barmode='stack'
)

fig = go.Figure(data=data, layout=layout)
fig.show()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [60], in <cell line: 19>()
     15 df_pivot = df_grouped.pivot(index='decade', columns='numfloors', values='count').fillna(0)
     16 #df_pivot.columns = [str(col) for col in df_pivot.columns]  # Convert Interval objects to strings
     17
     18 # Create stacked bar chart
---> 19 data = [
     20     go.Bar(x=df_pivot.index, y=df_pivot[col], name=col) for col in df_pivot.columns
     21 ]
     23 # Update layout with log y-axis
     24 layout = go.Layout(
     25     title='Number of Buildings by Decade and Number of Floors',
     26     xaxis=dict(title='Decade Built'),
     27     yaxis=dict(title='Number of Buildings', type='log'),
     28     barmode='stack'
     29 )

Input In [60], in <listcomp>(.0)
     15 df_pivot = df_grouped.pivot(index='decade', columns='numfloors', values='count').fillna(0)
     16 #df_pivot.columns = [str(col) for col in df_pivot.columns]  # Convert Interval objects to strings
     17
     18 # Create stacked bar chart
     19 data = [
---> 20     go.Bar(x=df_pivot.index, y=df_pivot[col], name=col) for col in df_pivot.columns
     21 ]
     23 # Update layout with log y-axis
     24 layout = go.Layout(
     25     title='Number of Buildings by Decade and Number of Floors',
     26     xaxis=dict(title='Decade Built'),
     27     yaxis=dict(title='Number of Buildings', type='log'),
     28     barmode='stack'
     29 )
```

```
File ~/opt/anaconda3/lib/python3.9/site-packages/plotly/graph_objs/_bar.py:3074
↪in Bar.__init__(self, arg, alignmentgroup, base, basesrc, cliponaxis,
↪constraintext, customdata, customdatasrc, dx, dy, error_x, error_y, hoverinfo
↪hoverinfosrc, hoverlabel, hovertemplate, hovertemplatesrc, hovertext,
↪hovertextsrc, ids, idssrc, insidetextanchor, insidetextfont, legendgroup,
↪legendgrouptitle, legendrank, marker, meta, metasrc, name, offset,
↪offsetgroup, offsetsrc, opacity, orientation, outsidetextfont, selected,
↪selectedpoints, showlegend, stream, text, textangle, textfont, textposition,
↪textpositionsrc, textsrc, texttemplate, texttemplatesrc, uid, uirevision,
↪unselected, visible, width, widthsrc, x, x0, xaxis, xcalendar, xhoverformat,
↪xperiod, xperiod0, xperiodalignment, xsrc, y, y0, yaxis, ycalendar,
↪yhoverformat, yperiod, yperiod0, yperiodalignment, ysrc, **kwargs)
   3072 _v = name if name is not None else _v
   3073 if _v is not None:
-> 3074     self["name"] = _v
   3075 _v = arg.pop("offset", None)
   3076 _v = offset if offset is not None else _v

File ~/opt/anaconda3/lib/python3.9/site-packages/plotly/basedatatypes.py:4827,
↪in BasePlotlyType.__setitem__(self, prop, value)
   4823         self._set_array_prop(prop, value)
   4825     # ### Handle simple property ###
   4826     else:
-> 4827         self._set_prop(prop, value)
   4828 else:
   4829     # Make sure properties dict is initialized
   4830     self._init_props()

File ~/opt/anaconda3/lib/python3.9/site-packages/plotly/basedatatypes.py:5171,
↪in BasePlotlyType._set_prop(self, prop, val)
   5169         return
   5170     else:
-> 5171         raise err
   5173 # val is None
   5174 # -----------
   5175 if val is None:
   5176     # Check if we should send null update

File ~/opt/anaconda3/lib/python3.9/site-packages/plotly/basedatatypes.py:5166,
↪in BasePlotlyType._set_prop(self, prop, val)
   5163 validator = self._get_validator(prop)
   5165 try:
-> 5166     val = validator.validate_coerce(val)
   5167 except ValueError as err:
   5168     if self._skip_invalid:

File ~/opt/anaconda3/lib/python3.9/site-packages/_plotly_utils/basevalidators.py:
↪1103, in StringValidator.validate_coerce(self, v)
   1101         v = str(v)
   1102     else:
-> 1103         self.raise_invalid_val(v)
```

```
   1105 if self.no_blank and len(v) == 0:
   1106     self.raise_invalid_val(v)

File ~/opt/anaconda3/lib/python3.9/site-packages/_plotly_utils/basevalidators.py:
↪289, in BaseValidator.raise_invalid_val(self, v, inds)
    286             for i in inds:
    287                 name += "[" + str(i) + "]"
--> 289         raise ValueError(
    290             """
    291     Invalid value of type {typ} received for the '{name}' property of␣
↪{pname}
    292         Received value: {v}
    293 
    294 {valid_clr_desc}""".format(
    295                 name=name,
    296                 pname=self.parent_name,
    297                 typ=type_str(v),
    298                 v=repr(v),
    299                 valid_clr_desc=self.description(),
    300             )
    301         )

ValueError:
    Invalid value of type 'pandas._libs.interval.Interval' received for the␣
↪'name' property of bar
        Received value: Interval(1.0, 11.0, closed='right')

    The 'name' property is a string and must be specified as:
      - A string
      - A number that will be converted to a string
```

0.2 Part 2: Datashader

Datashader is a library from Anaconda that does away with the need for binning data. It takes in all of your datapoints, and based on the canvas and range returns a pixel-by-pixel calculations to come up with the best representation of the data. In short, this completely eliminates the need for binning your data.

As an example, lets continue with our question above and look at a 2D histogram of YearBuilt vs NumFloors:

[26]:
```
fig = go.FigureWidget(
    data = [
        go.Histogram2d(x=ny['yearbuilt'], y=ny['numfloors'], autobiny=False,␣
↪ybins={'size': 1}, colorscale='Greens')
    ]
)
```

```
fig.show()
```

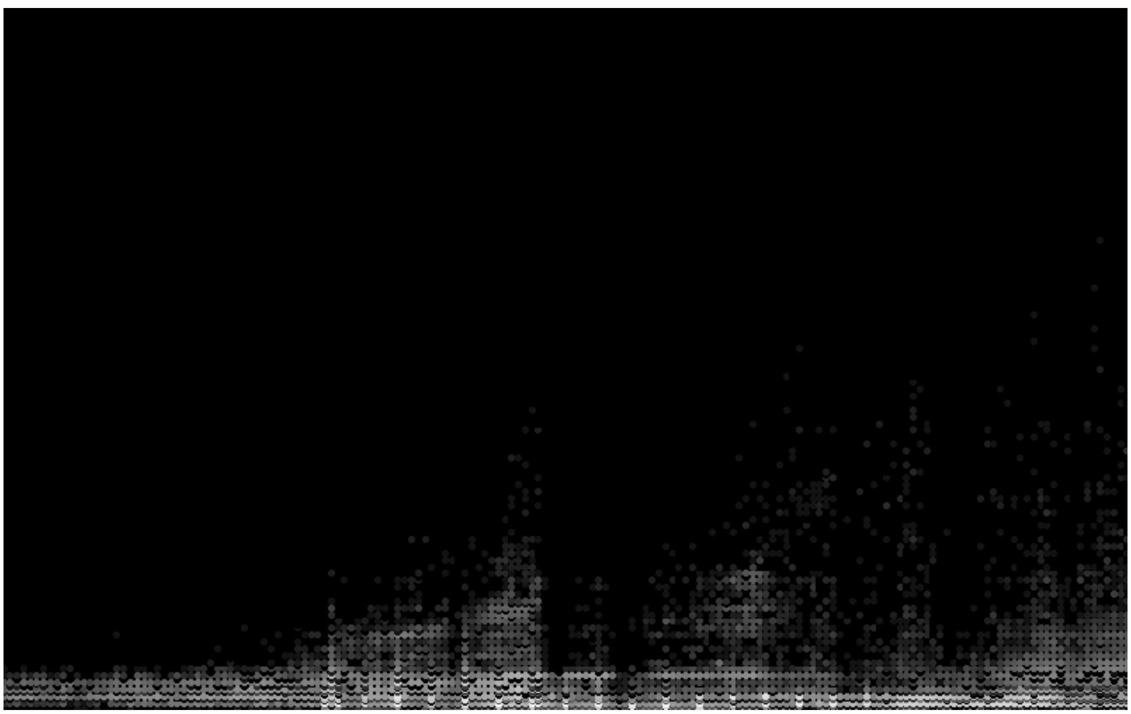
This shows us the distribution, but it's subject to some biases discussed in the Anaconda notebook Plotting Perils.

Here is what the same plot would look like in datashader:

[15]:
```
#Defining some helper functions for DataShader
background = "black"
export = partial(export_image, background = background, export_path="export")
cm = partial(colormap_select, reverse=(background!="black"))

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')
```

[15]:



That's technically just a scatterplot, but the points are smartly placed and colored to mimic what one gets in a heatmap. Based on the pixel size, it will either display individual points, or will color the points of denser regions.

Datashader really shines when looking at geographic information. Here are the latitudes and longitudes of our dataset plotted out, giving us a map of the city colored by density of structures:

[16]:
```
NewYorkCity   = (( 913164.0,  1067279.0), (120966.0, 272275.0))
cvs = ds.Canvas(700, 700, *NewYorkCity)
agg = cvs.points(ny, 'xcoord', 'ycoord')
view = tf.shade(agg, cmap = cm(inferno), how='log')
export(tf.spread(view, px=2), 'firery')
```

[16]:

Interestingly, since we're looking at structures, the large buildings of Manhattan show up as less dense on the map. The densest areas measured by number of lots would be single or multi family townhomes.

Unfortunately, Datashader doesn't have the best documentation. Browse through the examples from their github repo. I would focus on the visualization pipeline and the US Census Example for the question below. Feel free to use my samples as templates as well when you work on this problem.

0.2.1 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.

0.2.2 Answer

There is no separated value of the structural assessment in the dataset but we have the total and land assessment values so we can subtract the land value from the total and get what we assume to be the structural assessment value.

[17]:
```
ny['assessstruct'] = ny['assesstot'] - ny['assessland']
ny.head()
```

[17]:
```
  borough  block  lot     cd    bct2020     bctcb2020  ct2010  cb2010  \
0      SI   1597  125  502.0  5029104.0  5.029104e+10  291.04  3007.0
2      BK   4794    1  309.0  3080600.0  3.080600e+10  806.00  2000.0
3      BK   1488  105  303.0  3037500.0  3.037500e+10  375.00  1001.0
4      BK   4794   17  309.0  3080600.0  3.080600e+10  806.00  2000.0
5      BK   4794   78  309.0  3080600.0  3.080600e+10  806.00  2000.0

   schooldist  council  …  plutomapid firm07_flag  pfirm15_flag  version  \
0        31.0     50.0  …           1         NaN           NaN     22v2
2        17.0     41.0  …           1         NaN           NaN     22v2
3        16.0     41.0  …           1         NaN           NaN     22v2
4        17.0     41.0  …           1         NaN           NaN     22v2
5        17.0     41.0  …           1         NaN           NaN     22v2

   dcpedited   latitude  longitude notes floor_bin assessstruct
0        NaN  40.611140 -74.164376   NaN   (0, 10]      46020.0
2        NaN  40.661794 -73.942532   NaN   (0, 10]     308700.0
3        NaN  40.686484 -73.920169   NaN   (0, 10]      40740.0
4        NaN  40.661859 -73.941991   NaN   (0, 10]      46380.0
```

```
5          NaN  40.661517 -73.942539    NaN    (0, 10]       78480.0

[5 rows x 94 columns]
```

It wasn't clear to me how to generate the color palettes, so I used this tool https://colorbrewer2.org/#type=diverging&scheme=PiYG&n=9/

[18]:
```python
colors = {'1A':'#c51b7d','2A':'#de77ae','3A':'#f1b6da','1B':'#fde0ef','2B':
↪'#f7f7f7','3B':'#e6f5d0','1C':'#b8e186','2C':'#7fbc41','3C':'#4d9221'}
```

Going to use **Quantile** segmentation for the data which because we are segmenting into 3 distinct groups means simply dividing the data set into three parts and labeling it.

[19]:
```python
# Use array split to split into 3 groups after sorting the by assessland
first, second, third = np.array_split(ny['assessland'].sort_values(),3)
firstMinMax = [first.iloc[0],first.iloc[-1]]
secondMinMax = [second.iloc[0],second.iloc[-1]]
thirdMinMax = [third.iloc[0],third.iloc[-1]]

print(firstMinMax)
print(secondMinMax)
print(thirdMinMax)

#then add a column which represents the first lookup in our color dictionary
def conditions(ny):
    if firstMinMax[0] <= ny['assessland'] <= firstMinMax[1]:
        return '1'
    elif secondMinMax[0] <= ny['assessland'] <= secondMinMax[1]:
        return '2'
    else:
        return '3'

ny['colorOne'] = ny.apply(conditions,axis=1)
```

```
[0.0, 11580.0]
[11580.0, 18120.0]
[18120.0, 3205633833.0]
```

[20]:
```python
ny.head()
```

[20]:
```
  borough  block  lot     cd    bct2020     bctcb2020  ct2010  cb2010  \
0      SI   1597  125  502.0  5029104.0  5.029104e+10  291.04  3007.0
2      BK   4794    1  309.0  3080600.0  3.080600e+10  806.00  2000.0
3      BK   1488  105  303.0  3037500.0  3.037500e+10  375.00  1001.0
4      BK   4794   17  309.0  3080600.0  3.080600e+10  806.00  2000.0
5      BK   4794   78  309.0  3080600.0  3.080600e+10  806.00  2000.0

   schooldist  council  …  firm07_flag pfirm15_flag  version  dcpedited  \
0        31.0     50.0  …          NaN          NaN     22v2        NaN
```

```
2          17.0      41.0  …          NaN           NaN      22v2          NaN
3          16.0      41.0  …          NaN           NaN      22v2          NaN
4          17.0      41.0  …          NaN           NaN      22v2          NaN
5          17.0      41.0  …          NaN           NaN      22v2          NaN

    latitude  longitude  notes floor_bin assessstruct colorOne
0  40.611140 -74.164376    NaN   (0, 10]      46020.0        1
2  40.661794 -73.942532    NaN   (0, 10]     308700.0        1
3  40.686484 -73.920169    NaN   (0, 10]      40740.0        2
4  40.661859 -73.941991    NaN   (0, 10]      46380.0        3
5  40.661517 -73.942539    NaN   (0, 10]      78480.0        3

[5 rows x 95 columns]
```

[21]:
```python
# Repeat above except for assessstruct
a, b, c = np.array_split(ny['assessstruct'].sort_values(),3)
aMinMax = [a.iloc[0],a.iloc[-1]]
bMinMax = [b.iloc[0],b.iloc[-1]]
cMinMax = [c.iloc[0],c.iloc[-1]]

print(aMinMax)
print(bMinMax)
print(cMinMax)


def conditions(ny):
    if aMinMax[0] <= ny['assessstruct'] <= aMinMax[1]:
        return 'A'
    elif bMinMax[0] <= ny['assessstruct'] <= bMinMax[1]:
        return 'B'
    else:
        return 'C'

ny['colorTwo'] = ny.apply(conditions,axis=1)
```

```
[0.0, 33300.0]
[33300.0, 60000.0]
[60000.0, 4343286717.0]
```

[22]:
```python
ny.head()
```

[22]:
```
  borough  block  lot     cd    bct2020     bctcb2020  ct2010  cb2010  \
0      SI   1597  125  502.0  5029104.0  5.029104e+10  291.04  3007.0
2      BK   4794    1  309.0  3080600.0  3.080600e+10  806.00  2000.0
3      BK   1488  105  303.0  3037500.0  3.037500e+10  375.00  1001.0
4      BK   4794   17  309.0  3080600.0  3.080600e+10  806.00  2000.0
5      BK   4794   78  309.0  3080600.0  3.080600e+10  806.00  2000.0
```

```
   schooldist  council  …  pfirm15_flag version  dcpedited   latitude  \
0        31.0     50.0  …           NaN    22v2        NaN  40.611140
2        17.0     41.0  …           NaN    22v2        NaN  40.661794
3        16.0     41.0  …           NaN    22v2        NaN  40.686484
4        17.0     41.0  …           NaN    22v2        NaN  40.661859
5        17.0     41.0  …           NaN    22v2        NaN  40.661517

   longitude  notes  floor_bin  assessstruct  colorOne  colorTwo
0 -74.164376    NaN    (0, 10]       46020.0         1         B
2 -73.942532    NaN    (0, 10]      308700.0         1         C
3 -73.920169    NaN    (0, 10]       40740.0         2         B
4 -73.941991    NaN    (0, 10]       46380.0         3         B
5 -73.942539    NaN    (0, 10]       78480.0         3         C

[5 rows x 96 columns]
```

[23]:
```
# Combine the values so they can be used as a lookup. Must use this pd.
↪Categorical or you will get errors
ny['combined'] = pd.Categorical(ny['colorOne'] + ny['colorTwo'])
```

[24]:
```
ny.head()
```

[24]:
```
  borough  block  lot     cd    bct2020      bctcb2020  ct2010  cb2010  \
0      SI   1597  125  502.0  5029104.0  5.029104e+10  291.04  3007.0
2      BK   4794    1  309.0  3080600.0  3.080600e+10  806.00  2000.0
3      BK   1488  105  303.0  3037500.0  3.037500e+10  375.00  1001.0
4      BK   4794   17  309.0  3080600.0  3.080600e+10  806.00  2000.0
5      BK   4794   78  309.0  3080600.0  3.080600e+10  806.00  2000.0

   schooldist  council  …  version dcpedited   latitude  longitude  notes  \
0        31.0     50.0  …     22v2       NaN  40.611140 -74.164376    NaN
2        17.0     41.0  …     22v2       NaN  40.661794 -73.942532    NaN
3        16.0     41.0  …     22v2       NaN  40.686484 -73.920169    NaN
4        17.0     41.0  …     22v2       NaN  40.661859 -73.941991    NaN
5        17.0     41.0  …     22v2       NaN  40.661517 -73.942539    NaN

   floor_bin  assessstruct colorOne colorTwo combined
0    (0, 10]       46020.0        1        B       1B
2    (0, 10]      308700.0        1        C       1C
3    (0, 10]       40740.0        2        B       2B
4    (0, 10]       46380.0        3        B       3B
5    (0, 10]       78480.0        3        C       3C

[5 rows x 97 columns]
```

[25]:
```
# finally render the graph, copied the code from above
NewYorkCity   = (( 913164.0,  1067279.0), (120966.0, 272275.0))
```

```python
cvs = ds.Canvas(700, 700, *NewYorkCity)
agg = cvs.points(ny, 'xcoord', 'ycoord', ds.count_cat('combined'))
img = tf.shade(agg, color_key=colors)
export(img, 'answer')
```

[25]: