In [1]:
```
import sys
#!{sys.executable} -m pip install numpy
#import import_ipynb
!{sys.executable} -m pip install datashader
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
import plotly
import sys
!{sys.executable} -m pip install plotly
import sys
!{sys.executable} -m pip install shapely.geometry

import sys
!{sys.executable} -m pip install pyproj
import sys
!{sys.executable} -m pip install colorlover
import sys
!{sys.executable} -m pip install GeoJSON

from plotnine import *
from pyproj import Proj, transform
from plotly import tools
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 y
our terminal

from functools import partial

#from IPython.display import GeoJSON

py.init_notebook_mode()
```

```
Requirement already satisfied: datashader in c:\users\ajaya\anaconda2\lib\site-p
ackages (0.10.0)
Requirement already satisfied: param>=1.6.0 in c:\users\ajaya\anaconda2\lib\site
-packages (from datashader) (1.9.3)
Requirement already satisfied: bokeh in c:\users\ajaya\anaconda2\lib\site-packag
es (from datashader) (1.4.0)
Requirement already satisfied: numba>=0.37.0 in c:\users\ajaya\anaconda2\lib\sit
e-packages (from datashader) (0.45.1+0.ga9c107beb.dirty)
Requirement already satisfied: scipy in c:\users\ajaya\anaconda2\lib\site-packag
es (from datashader) (1.2.1)
Requirement already satisfied: pillow>=3.1.1 in c:\users\ajaya\anaconda2\lib\sit
e-packages (from datashader) (6.2.0)
Requirement already satisfied: scikit-image in c:\users\ajaya\anaconda2\lib\site
-packages (from datashader) (0.14.2)
Requirement already satisfied: datashape>=0.5.1 in c:\users\ajaya\anaconda2\lib\
site-packages (from datashader) (0.5.2)
Requirement already satisfied: pyct[cmd] in c:\users\ajaya\anaconda2\lib\site-pa
ckages (from datashader) (0.4.6)
Requirement already satisfied: pandas>=0.20.3 in c:\users\ajaya\anaconda2\lib\si
te-packages (from datashader) (0.24.2)
Requirement already satisfied: dask[complete]>=0.18.0 in c:\users\ajaya\anaconda
2\lib\site-packages (from datashader) (1.2.2)
Requirement already satisfied: toolz>=0.7.4 in c:\users\ajaya\anaconda2\lib\site
-packages (from datashader) (0.10.0)
Requirement already satisfied: numpy>=1.7 in c:\users\ajaya\anaconda2\lib\site-p
ackages (from datashader) (1.16.5)
Requirement already satisfied: xarray>=0.9.6 in c:\users\ajaya\anaconda2\lib\sit
e-packages (from datashader) (0.11.3)
Requirement already satisfied: colorcet>=0.9.0 in c:\users\ajaya\anaconda2\lib\s
ite-packages (from datashader) (2.0.2)
Requirement already satisfied: packaging>=16.8 in c:\users\ajaya\anaconda2\lib\s
ite-packages (from bokeh->datashader) (20.1)
Requirement already satisfied: python-dateutil>=2.1 in c:\users\ajaya\anaconda2\
lib\site-packages (from bokeh->datashader) (2.8.1)
Requirement already satisfied: tornado>=4.3 in c:\users\ajaya\anaconda2\lib\site
-packages (from bokeh->datashader) (5.1.1)
Requirement already satisfied: six>=1.5.2 in c:\users\ajaya\anaconda2\lib\site-p
ackages (from bokeh->datashader) (1.12.0)
Requirement already satisfied: PyYAML>=3.10 in c:\users\ajaya\anaconda2\lib\site
-packages (from bokeh->datashader) (5.1.2)
Requirement already satisfied: Jinja2>=2.7 in c:\users\ajaya\anaconda2\lib\site-
packages (from bokeh->datashader) (2.11.1)
Requirement already satisfied: futures>=3.0.3 in c:\users\ajaya\anaconda2\lib\si
te-packages (from bokeh->datashader) (3.3.0)
Requirement already satisfied: llvmlite>=0.29.0dev0 in c:\users\ajaya\anaconda2\
lib\site-packages (from numba>=0.37.0->datashader) (0.29.0)
Requirement already satisfied: enum34 in c:\users\ajaya\anaconda2\lib\site-packa
ges (from numba>=0.37.0->datashader) (1.1.6)
Requirement already satisfied: singledispatch in c:\users\ajaya\anaconda2\lib\si
te-packages (from numba>=0.37.0->datashader) (3.4.0.3)
Requirement already satisfied: funcsigs in c:\users\ajaya\anaconda2\lib\site-pac
kages (from numba>=0.37.0->datashader) (1.0.2)
Requirement already satisfied: matplotlib>=2.0.0 in c:\users\ajaya\anaconda2\li
b\site-packages (from scikit-image->datashader) (2.2.3)
Requirement already satisfied: PyWavelets>=0.4.0 in c:\users\ajaya\anaconda2\li
b\site-packages (from scikit-image->datashader) (1.0.3)
Requirement already satisfied: cloudpickle>=0.2.1 in c:\users\ajaya\anaconda2\li
b\site-packages (from scikit-image->datashader) (1.2.2)
Requirement already satisfied: networkx>=1.8 in c:\users\ajaya\anaconda2\lib\sit
e-packages (from scikit-image->datashader) (2.2)
Requirement already satisfied: multipledispatch>=0.4.7 in c:\users\ajaya\anacond
a2\lib\site-packages (from datashape>=0.5.1->datashader) (0.6.0)
Requirement already satisfied: requests; extra == "cmd" in c:\users\ajaya\anacon
da2\lib\site-packages (from pyct[cmd]->datashader) (2.22.0)
```

```
DEPRECATION: Python 2.7 will reach the end of its life on January 1st, 2020. Ple
ase upgrade your Python as Python 2.7 won't be maintained after that date. A fut
ure version of pip will drop support for Python 2.7. More details about Python 2
support in pip, can be found at https://pip.pypa.io/en/latest/development/releas
e-process/#python-2-support

Requirement already satisfied: plotly in c:\users\ajaya\anaconda2\lib\site-packa
ges (4.5.0)
Requirement already satisfied: six in c:\users\ajaya\anaconda2\lib\site-packages
(from plotly) (1.12.0)
Requirement already satisfied: retrying>=1.3.3 in c:\users\ajaya\anaconda2\lib\s
ite-packages (from plotly) (1.3.3)

DEPRECATION: Python 2.7 will reach the end of its life on January 1st, 2020. Ple
ase upgrade your Python as Python 2.7 won't be maintained after that date. A fut
ure version of pip will drop support for Python 2.7. More details about Python 2
support in pip, can be found at https://pip.pypa.io/en/latest/development/releas
e-process/#python-2-support

Collecting shapely.geometry

DEPRECATION: Python 2.7 will reach the end of its life on January 1st, 2020. Ple
ase upgrade your Python as Python 2.7 won't be maintained after that date. A fut
ure version of pip will drop support for Python 2.7. More details about Python 2
support in pip, can be found at https://pip.pypa.io/en/latest/development/releas
e-process/#python-2-support
  ERROR: Could not find a version that satisfies the requirement shapely.geometr
y (from versions: none)
ERROR: No matching distribution found for shapely.geometry

Requirement already satisfied: pyproj in c:\users\ajaya\anaconda2\lib\site-packa
ges (1.9.6)

DEPRECATION: Python 2.7 will reach the end of its life on January 1st, 2020. Ple
ase upgrade your Python as Python 2.7 won't be maintained after that date. A fut
ure version of pip will drop support for Python 2.7. More details about Python 2
support in pip, can be found at https://pip.pypa.io/en/latest/development/releas
e-process/#python-2-support

Requirement already satisfied: colorlover in c:\users\ajaya\anaconda2\lib\site-p
ackages (0.3.0)

DEPRECATION: Python 2.7 will reach the end of its life on January 1st, 2020. Ple
ase upgrade your Python as Python 2.7 won't be maintained after that date. A fut
ure version of pip will drop support for Python 2.7. More details about Python 2
support in pip, can be found at https://pip.pypa.io/en/latest/development/releas
e-process/#python-2-support

Requirement already satisfied: GeoJSON in c:\users\ajaya\anaconda2\lib\site-pack
ages (2.5.0)

DEPRECATION: Python 2.7 will reach the end of its life on January 1st, 2020. Ple
ase upgrade your Python as Python 2.7 won't be maintained after that date. A fut
ure version of pip will drop support for Python 2.7. More details about Python 2
support in pip, can be found at https://pip.pypa.io/en/latest/development/releas
e-process/#python-2-support
```

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 (https://www1.nyc.gov/assets/planning/download/zip/data-maps/open-data/nyc_pluto_17v1_1.zip). 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.

In [2]:
```
# Code to read in v17, column names have been updated (without upper case letters)
for v18
bk = pd.read_csv('BK2017V11.csv')
bx = pd.read_csv('BX2017V11.csv')
mn = pd.read_csv('MN2017V11.csv')
qn = pd.read_csv('QN2017V11.csv')
si = pd.read_csv('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)]
```

```
C:\Users\ajaya\Anaconda2\lib\site-packages\IPython\core\interactiveshell.py:271
4: DtypeWarning:

Columns (19,20,22,23,24,25,26,64,65,80) have mixed types. Specify dtype option o
n import or set low_memory=False.

C:\Users\ajaya\Anaconda2\lib\site-packages\IPython\core\interactiveshell.py:271
4: DtypeWarning:

Columns (19,20,22,23,64,65,80) have mixed types. Specify dtype option on import
or set low_memory=False.

C:\Users\ajaya\Anaconda2\lib\site-packages\IPython\core\interactiveshell.py:271
4: DtypeWarning:

Columns (19,20,22,24,26) have mixed types. Specify dtype option on import or set
low_memory=False.

C:\Users\ajaya\Anaconda2\lib\site-packages\IPython\core\interactiveshell.py:271
4: DtypeWarning:

Columns (19,20,22,23,64,65,77) have mixed types. Specify dtype option on import
or set low_memory=False.

C:\Users\ajaya\Anaconda2\lib\site-packages\IPython\core\interactiveshell.py:271
4: DtypeWarning:

Columns (19,20,22,24,64,65,80) have mixed types. Specify dtype option on import
or set low_memory=False.

C:\Users\ajaya\Anaconda2\lib\site-packages\ipykernel_launcher.py:8: FutureWarnin
g:

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'.
```

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.

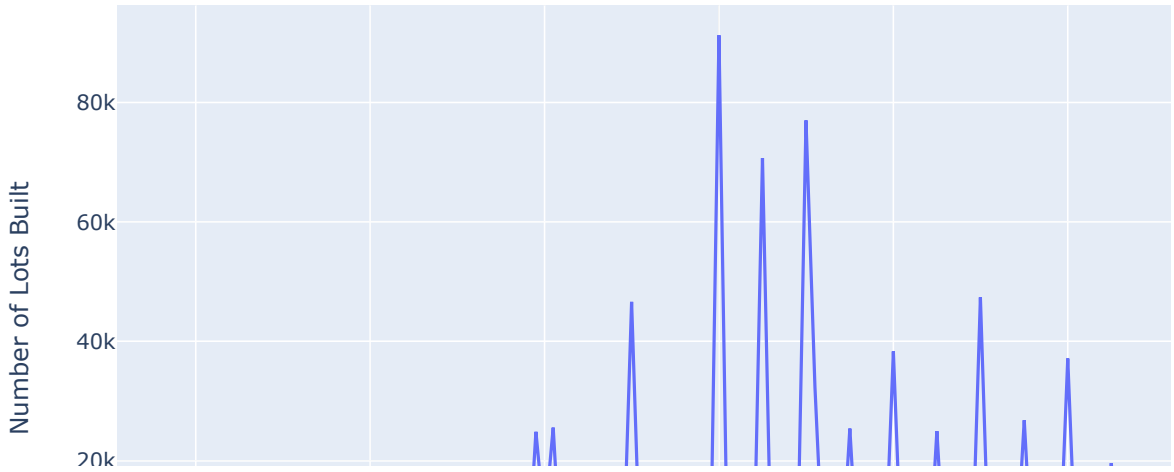
In [3]:
```
wgs84 = Proj("+proj=longlat +ellps=GRS80 +datum=NAD83 +no_defs")
nyli = Proj("+proj=lcc +lat_1=40.66666666666666 +lat_2=41.03333333333333 +lat_0=40.
16666666666666 +lon_0=-74 +x_0=300000 +y_0=0 +ellps=GRS80 +datum=NAD83 +to_meter=0.
3048006096012192 +no_defs")
ny['XCoord'] = 0.3048*ny['XCoord']
ny['YCoord'] = 0.3048*ny['YCoord']
ny['lon'], ny['lat'] = transform(nyli, wgs84, ny['XCoord'].values, ny['YCoord'].val
ues)

ny = ny[(ny['lon'] < -60) & (ny['lon'] > -100) & (ny['lat'] < 60) & (ny['lat'] > 2
0)]

#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 (https://plot.ly/python/2D-Histogram/) (also check out their close relatives: 2D density plots (https://plot.ly/python/2d-density-plots/) and the more general form: heatmaps (https://plot.ly/python/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:

In [4]:
```
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 (https://pandas.pydata.org/pandas-docs/stable/indexing.html): .loc and .iloc are the analogs for base R subsetting, or filter() in dplyr

Group By (https://pandas.pydata.org/pandas-docs/stable/groupby.html): 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() (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.reset_index.html).

Reset_index (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.reset_index.html): 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 (http://pandas.pydata.org/pandas-docs/stable/timeseries.html) on datetime indexing. In particular, check out resample (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.resample.html), which provides time series specific aggregation.

Merging, joining, and concatenation (https://pandas.pydata.org/pandas-docs/stable/merging.html): 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.

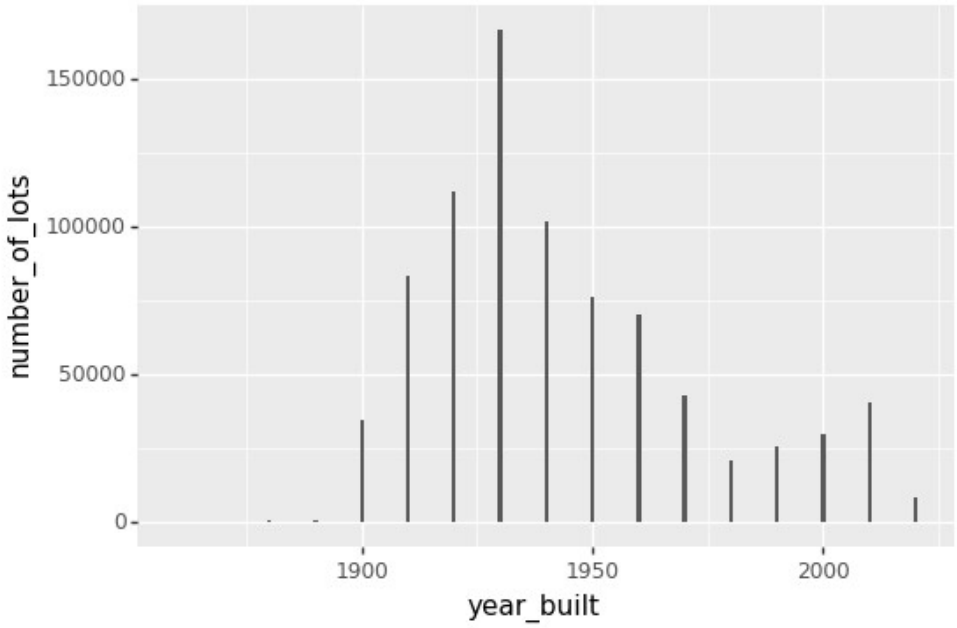
Browse through the other sections for some other specifics, in particular reshaping and categorical data (pandas' answer to factors.) Pandas can take a while to get used to, but it is a pretty strong framework that makes more advanced functions easier once you get used to it. Rolling functions for example 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

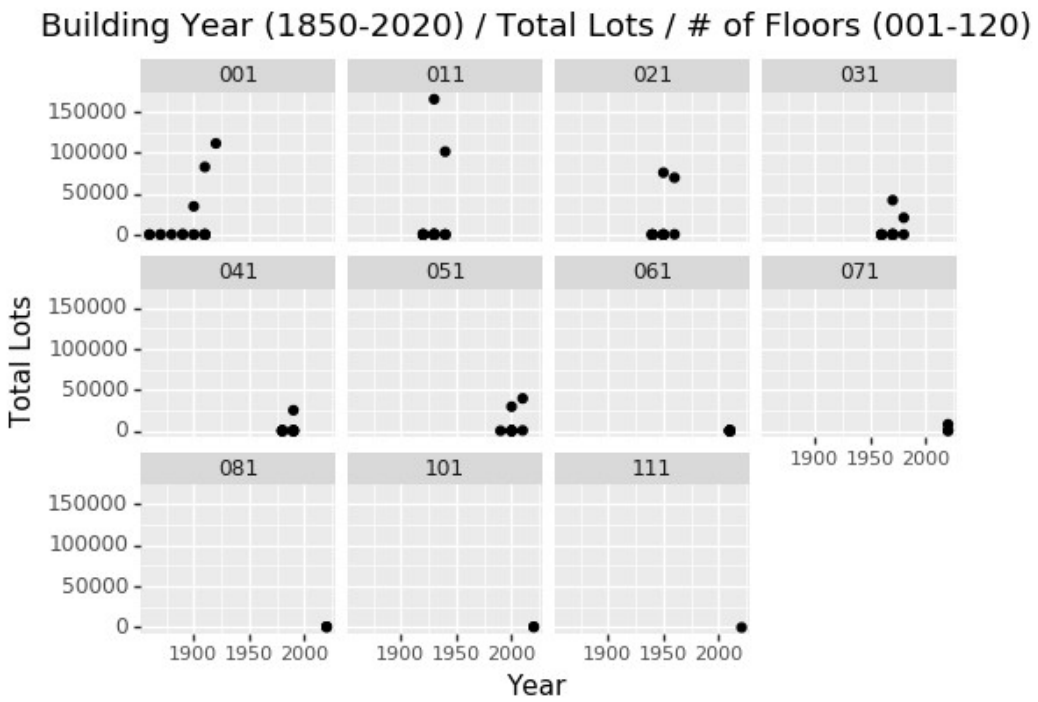
After a few building collapses, the City of New York is going to begin investigating older buildings for safety. The city is particularly worried about buildings that were unusually tall when they were built, since best-practices for safety hadn’t yet been determined. Create a graph that shows how many 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?)

In [15]:
```
#https://www1.nyc.gov/assets/planning/download/pdf/data-maps/open-data/pluto_datadictionary.pdf?v=17v1_1
#In general, YEAR BUILT is accurate for the decade, but not necessarily for the specific year.
#Between 1910 and 1985, the majority of YEAR BUILT values are in years ending in 5 or 0.
#A large number of structures built between 1800s and early 1900s have a YEAR BUILT between 1899 and 1901
#Make all the years round to next decade.
bydecades = ny[['YearBuilt', 'NumFloors', 'BBL']].copy()
bydecades['YearBuilt'] = (np.ceil(bydecades['YearBuilt'] / 10.0).astype(int) * 10)
floors = ((np.ceil(bydecades['NumFloors']) - 1) // 10 * 10 + 1).astype(int)
#print floors
year_built = bydecades.groupby('YearBuilt').count()['BBL'].index
number_of_lots = bydecades.groupby('YearBuilt').count()['BBL']

#data.groupby(['month', 'item'])['date'].count()
bydecades['Bins'] = ['{:03}'.format(x) for x in floors]
groups = bydecades.groupby(['YearBuilt', 'Bins'])
total_floors_df = pd.DataFrame()
total_floors_df['TotalFloors'] = groups['NumFloors'].count()

#print total_floors
#num_floors_pd['floorbuckets'] = groups['NumFloors']
#num_floors_pd['floorbuckets'] = groups.groupby(['YearBuilt','NumFloors']).sum()['NumFloors']
#total_floors = nybydecades.groupby('YearBuilt').count()['BBL']

#Which decade were most of the lots built?
result1 = pd.DataFrame({'x':year_built, 'y':number_of_lots})

ggplot(result1, aes(x='year_built', y = 'number_of_lots')) + \
    geom_bar(stat="identity")
```



Out[15]: <ggplot: (48803830)>

In [14]:
```
#num_floors_filtered = num_floors_pd[('floorcategory' >= 20) & ('floorcategory' <=
200)]
#print num_floors_filtered

#print groups['YearBuilt']

#g1['group'].apply(pd.DataFrame)
#total_floors['TotalFloors'].apply(pd.DataFrame())
#groups['YearBuilt'].apply(pd.DataFrame())
#groups = bydecades.groupby(['YearBuilt', 'Bins'])
year_built_list = bydecades.groupby(['YearBuilt', 'Bins']).size().index.get_level_v
alues('YearBuilt').tolist()
#print yb
#bins = bydecaes.groupby()
#groups.get_group(('1980','010'))

#year_built_df = pd.DataFrame()
#year_built_df = groups['YearBuilt'].values

total_floors_list = total_floors_df['TotalFloors'].tolist()
bins = bydecades.groupby(['YearBuilt', 'Bins']).size().index.get_level_values('Bins
').tolist()
#print bins
#bins_df = pd.DataFrame(bins)
#print bins
#print TotalFloorList
#YearBuiltList = groups['YearBuilt'].values


result2 = pd.DataFrame({'x':year_built_list, 'y':total_floors_list})
#print len(result2), len(total_floors), len(groups['Bins'])


ggplot(result2, aes(x='year_built_list', y='total_floors_list')) + \
    geom_point() + \
    theme(axis_text_x = element_text(size=8)) + \
    facet_wrap("bins") + \
    scale_x_continuous(breaks = [1850,1900,1950,2000]) + \
    xlab("Year") + \
    ylab("Total Lots") + \
    labs(title = 'Building Year (1850-2020) / Total Lots / # of Floors (001-120)')
```



Out[14]: <ggplot: (50860022)>

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:

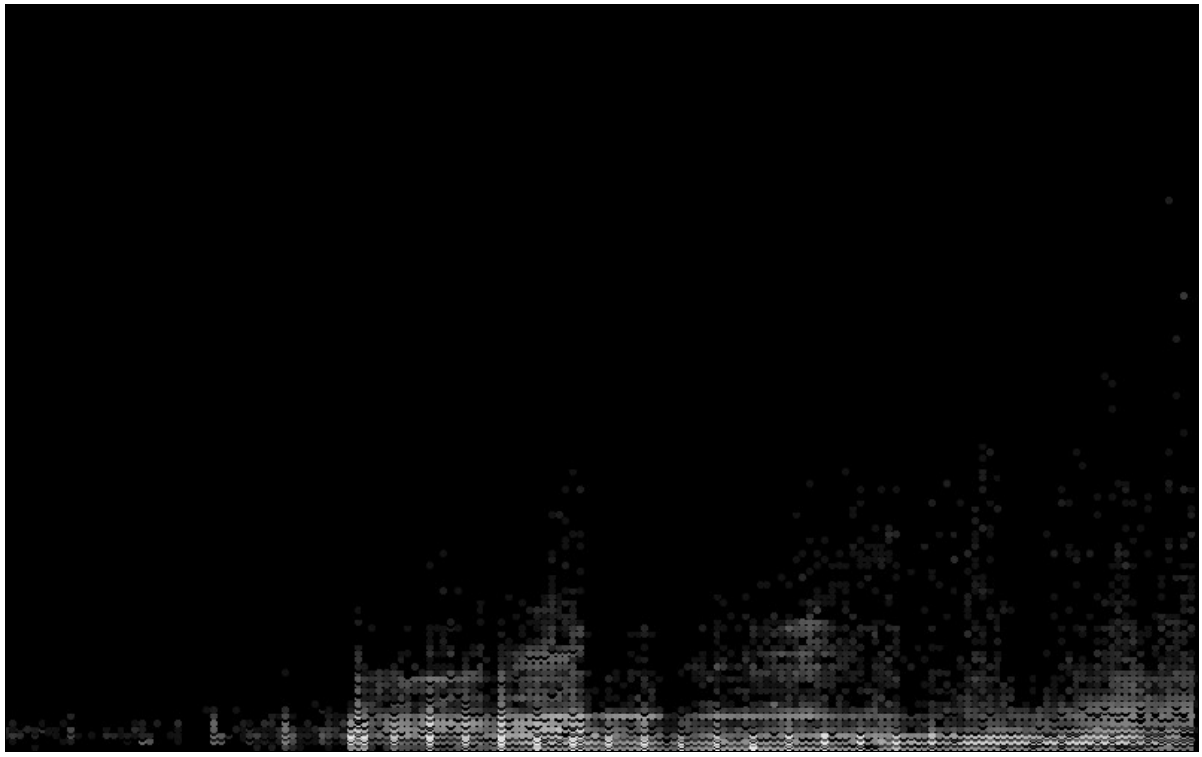
In [7]:
```
yearbins = 200
floorbins = 200

yearBuiltCut = pd.cut(ny['YearBuilt'], np.linspace(ny['YearBuilt'].min(), ny['YearB
uilt'].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(), yearbi
ns))
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)
```

This shows us the distribution, but it's subject to some biases discussed in the Anaconda notebook Plotting Perils (https://anaconda.org/jbednar/plotting_pitfalls/notebook).

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

In [8]:
```
cvs = ds.Canvas(800, 500, x_range = (ny['YearBuilt'].min(), ny['YearBuilt'].max()),
                                  y_range = (ny['NumFloors'].min(), ny['NumFloors'].m
ax()))
agg = cvs.points(ny, 'YearBuilt', 'NumFloors')
view = tf.shade(agg, cmap = cm(Greys9), how='log')
export(tf.spread(view, px=2), 'yearvsnumfloors')
```

Out[8]:

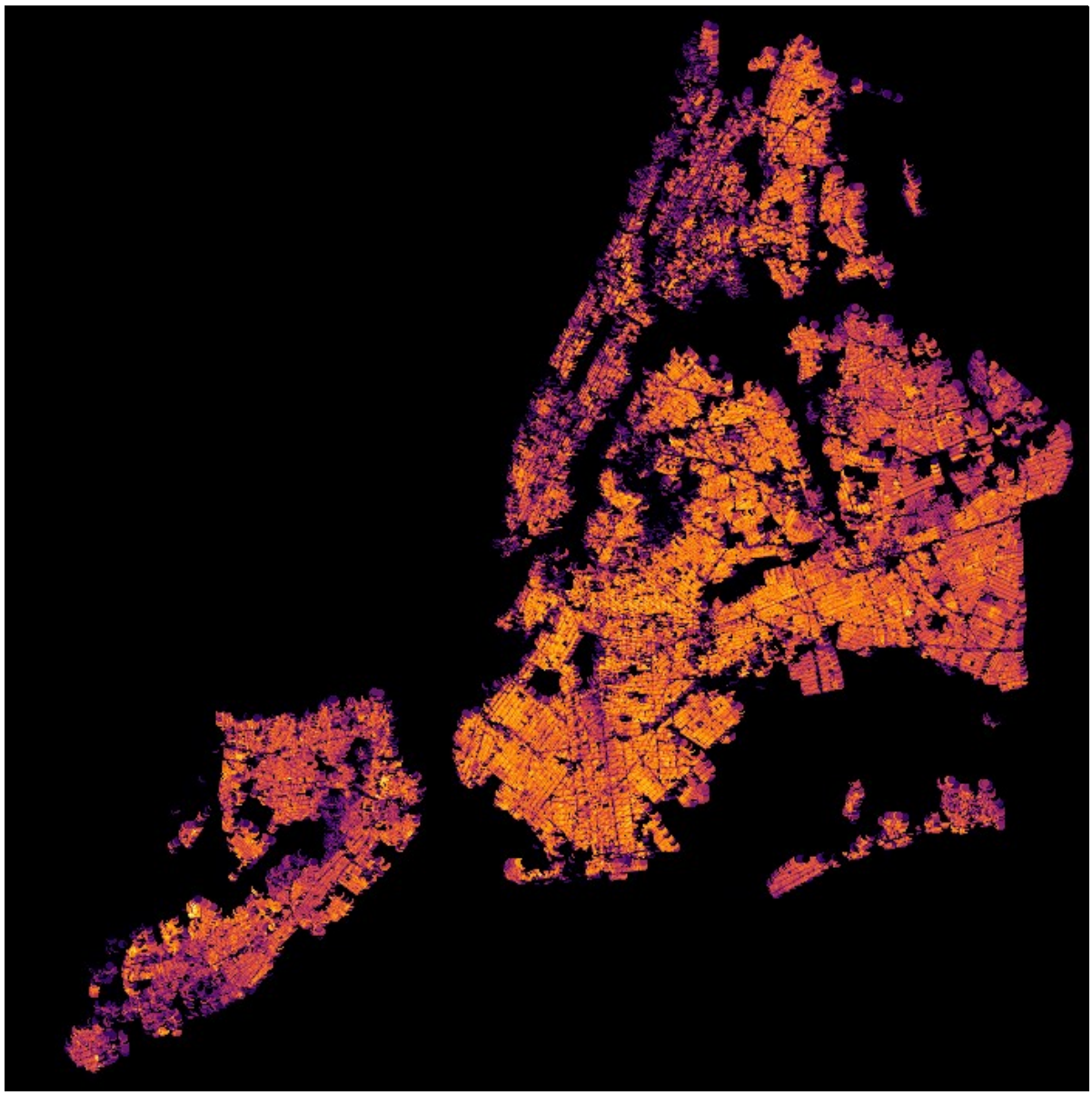


That's technically just a scatterplot, but the points are smartly placed and colored to mimic what one gets in a heatmap. Based on the pixel size, it will either display individual points, or will color the points of denser regions.

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:

In [9]:
```
NewYorkCity   = (( -74.29,  -73.69), (40.49, 40.92))
cvs = ds.Canvas(700, 700, *NewYorkCity)
agg = cvs.points(ny, 'lon', 'lat')
view = tf.shade(agg, cmap = cm(inferno), how='log')
export(tf.spread(view, px=2), 'firery')
```

Out[9]:



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 (https://github.com/bokeh/datashader/tree/master/examples). I would focus on the visualization pipeline (https://anaconda.org/jbednar/pipeline/notebook) and the US Census (https://anaconda.org/jbednar/census/notebook) Example for the question below. Feel free to use my samples as templates as well when you work on this problem.

Question

You work for a real estate developer and are researching underbuilt areas of the city. After looking in the Pluto data dictionary (https://www1.nyc.gov/assets/planning/download/pdf/data-maps/open-data/pluto_datadictionary.pdf?v=17v1_1), 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 (http://www.joshuastevens.net/cartography/make-a-bivariate-choropleth-map/), 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.

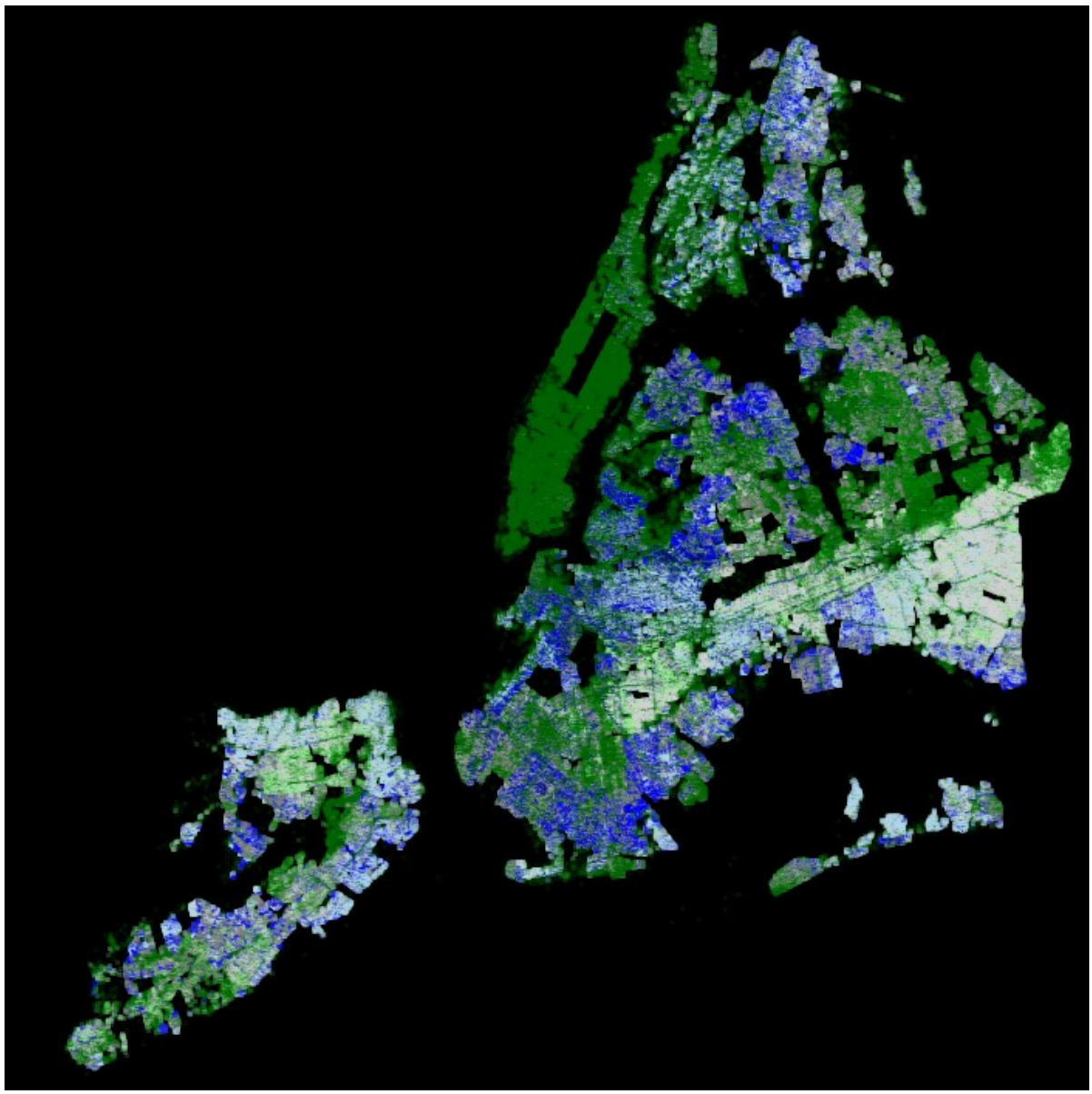
In [10]:
```
assessment = ny[['AssessTot', 'AssessLand','lon','lat']].copy()
assessment['AssessBldg'] = assessment['AssessTot'] - assessment['AssessLand']

#A=overbuilt
#B=Underbuilt
#C=FairValue
labels = [['OverBuilt', 'UnderBuilt', 'FairValue'], ['1', '2', '3']] #https://www.j
oshuastevens.net/cartography/make-a-bivariate-choropleth-map/

#100% - 33% = 67 = 67th percentile
#33% cutoff for each bucket (A,B,C)
cutoff = np.percentile(assessment[['AssessLand', 'AssessBldg']], [33, 67], axis=0)

assessment['Var1_Class'] = pd.cut(assessment['AssessLand'], [0, cutoff[0][0], cutof
f[1][0], np.inf], right=False, labels=labels[0]) #bucket
assessment['Var2_Class'] = pd.cut(assessment['AssessBldg'], [0, cutoff[0][1], cutof
f[1][1], np.inf], right=False, labels=labels[1]) # bucket
assessment['Bi_Class'] = assessment['Var1_Class'].astype(str) + assessment['Var2_Cl
ass'].astype(str)
assessment['Bi_Class'] = pd.Categorical(assessment['Bi_Class'])
assessment
```

Out[10]:

	AssessTot	AssessLand	lon	lat	AssessBldg	Var1_Class	Var2_Class	Bi_Class
1	10156950	834300	-73.987066	40.704619	9322650	FairValue	3	FairValue3
17	351000	129600	-73.989310	40.704109	221400	FairValue	3	FairValue3
20	236700	178200	-73.990608	40.704581	58500	FairValue	3	FairValue3
22	7785450	514350	-73.988008	40.704131	7271100	FairValue	3	FairValue3
23	40980600	972000	-73.987077	40.704101	40008600	FairValue	3	FairValue3
24	1497600	214650	-73.986204	40.703974	1282950	FairValue	3	FairValue3
25	3078450	112500	-73.986212	40.704252	2965950	FairValue	3	FairValue3
26	1388250	241200	-73.985505	40.704175	1147050	FairValue	3	FairValue3
27	2600550	339750	-73.984964	40.704155	2260800	FairValue	3	FairValue3
28	3915900	301050	-73.985130	40.703897	3614850	FairValue	3	FairValue3
29	3024902	98103	-73.985865	40.704048	2926799	FairValue	3	FairValue3
30	252000	119250	-73.984336	40.703870	132750	FairValue	3	FairValue3
31	33300	28350	-73.984325	40.704037	4950	FairValue	1	FairValue1
32	30600	25650	-73.984322	40.704095	4950	FairValue	1	FairValue1
35	3532050	351000	-73.981747	40.704056	3181050	FairValue	3	FairValue3
36	4155300	708300	-73.994979	40.703519	3447000	FairValue	3	FairValue3
37	2652300	514350	-73.993973	40.703887	2137950	FairValue	3	FairValue3
40	712800	324000	-73.991005	40.703407	388800	FairValue	3	FairValue3
41	247386	69672	-73.991211	40.703401	177714	FairValue	3	FairValue3
42	34118550	1818000	-73.991705	40.703624	32300550	FairValue	3	FairValue3
43	3411000	981900	-73.992902	40.703673	2429100	FairValue	3	FairValue3
44	406350	81000	-73.989872	40.703761	325350	FairValue	3	FairValue3
45	10550250	486000	-73.989818	40.703461	10064250	FairValue	3	FairValue3
46	15323847	591747	-73.990244	40.703538	14732100	FairValue	3	FairValue3
47	12949650	688500	-73.989202	40.703654	12261150	FairValue	3	FairValue3
48	2515953	225003	-73.989371	40.703385	2290950	FairValue	3	FairValue3
49	5218561	200703	-73.989072	40.703374	5017858	FairValue	3	FairValue3
51	967950	204750	-73.987874	40.703618	763200	FairValue	3	FairValue3
52	1445400	89100	-73.987806	40.703340	1356300	FairValue	3	FairValue3
53	71586	50405	-73.987474	40.703250	21181	FairValue	2	FairValue2
...	...	...	...	...	...	...	...	...
859192	28663	9728	-74.248634	40.508980	18935	UnderBuilt	2	UnderBuilt2
859193	33748	14581	-74.249438	40.508533	19167	FairValue	2	FairValue2
859194	29738	13831	-74.249603	40.508481	15907	FairValue	2	FairValue2
859195	31296	15275	-74.249826	40.508491	16021	FairValue	2	FairValue2
859196	18987	11360	-74.250016	40.508485	7627	UnderBuilt	1	UnderBuilt1
859197	34038	17064	-74.250214	40.508455	16974	FairValue	2	FairValue2
859198	30672	15746	-74.250415	40.508388	14926	FairValue	1	FairValue1
859199	29020	20426	-74.250584	40.508295	8594	FairValue	1	FairValue1
859200	54809	14659	-74.250774	40.508204	40150	FairValue	3	FairValue3

In [11]:
```
#https://stackoverflow.com/questions/54026510/changing-colormap-for-categorical-data-in-holoviews-datashader
colors = {'FairValue1': 'lightgreen', 'FairValue2': 'green', 'FairValue3': 'darkgreen',
          'OverBuilt1': 'lightblue', 'OverBuilt2': 'blue', 'OverBuilt3': 'darkblue',
          'UnderBuilt1': 'lightgray', 'UnderBuilt2': 'gray', 'UnderBuilt3': 'darkgray'}

NewYorkCity   = (( -74.29,  -73.69), (40.49, 40.92))
cvs = ds.Canvas(700, 700, *NewYorkCity)
agg = cvs.points(assessment, 'lon', 'lat', ds.count_cat('Bi_Class'))
view = tf.shade(agg, color_key=colors,  how='log')
export(tf.spread(view, px=1), 'cloropleth')
```

Out[11]:



In [ ]: