San Francisco Restaurant Inspections

Description

NOTE: This is taken from a project that I completed for a Data Science course (DS100) at University of California, Berkeley. This is an addendum that I added because I wished to learned more about different ways of visualizing data for analysis.

In this project, I investigated restaurant food safety scores for restaurants in San Francisco. The scores and violation information have been made available by the San Francisco Department of Public Health (https://data.sfgov.org/Health-and-Social-Services/Restaurant-Scores-LIVES-Standard/pyih-qa8i).

In cleaning and exploring the data, I gained practice with:

- Pandas
- Data cleaning: identifying type of data collected, missing values, anomalies, etc.
- Data Analysis
- Exploring different ways of visualizing data: distributions, mapping

Process and Reflection

I focused a lot of the time on the project learning about new ways to visualize data. In the process, I also learned about different ways of cleaning data in order to get the correct format or remove missing data.

Some new tools that I discovered include:

- DATA CLEANING
  - **geopy + geocoders:** An issue that I noticed while cleaning points to graph was that a lot of businesses were missing longitude, latitutde.
    - I wanted to looked at ways to use the address in order to find longitude and latitude points, which led me to finding geopy and its Nominatim library.
    - *ISSUES:* I didn't end up using this idea because of the following reasons:
      - A problem I ran into here was how inefficient it was in terms of finding all the geocode from the address. Given how much data was missing, it would have taken at least half an hour to process.
      - In addition, I found that if the addresses were not exact, whether due to commas or different ways of writing numbers (i.e. leading with 0 for single digit numbers), the library would not work. This was an issue because this was the case for many addresses.
      - *SOLUTION:* Instead, I found that zipcodes were much more readily available for each address. Thus, I decided that mapping scores based on zipcode was a better idea.
- VISUALIZATIONS
  - **shapefile + geopandas:** My goal was to visualize the location of the inspections and the scores to get a look at the possibility of any patterns related to location and inspection scores.
    - Thus, I discovered the use of shapefiles and geopandas that enable mappings of data points using longitude and latitude.
    - Another thing I learned is that after using geopandas to import and graph the shapefiles, you can use any other plotting libraries such as seaborn to plot points (as long as you have longitude, latitude points.
  - **geojson and bokeh:** I wanted to visualize the median scores within each zipcode.
    - I found geojson to get the polygon shapes for each zipcode and overlay it onto a map.
    - I also found other libraries from bokeh to help create a color map using the our data points.
    - I hope to use bokeh more in the future. I think it has a lot of cool features that are great for data visualization.

In [1]:
```
%%capture install
! pip install shapely
! pip install geopandas
! pip install descartes
! pip install geopy
! pip install bokeh
```

In [ ]:
```
```

Libraries

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns

import geopandas as gpd
import descartes
from shapely.geometry import Point, Polygon

from geopy.geocoders import Nominatim

import json
from bokeh.io import output_notebook, show, output_file
from bokeh.plotting import figure
from bokeh.models import GeoJSONDataSource, LinearColorMapper, ColorBar
from bokeh.palettes import brewer


sns.set()
plt.style.use('fivethirtyeight')

%matplotlib inline

import zipfile
import os # Used to interact with the file system
from pathlib import Path
```

Obtaining the Data

In [3]:
```
dsDir = Path('data')

bus = pd.read_csv(dsDir/'bus.csv', encoding = 'ISO-8859-1')
ins2vio = pd.read_csv(dsDir/'ins2vio.csv')
ins = pd.read_csv(dsDir/'ins.csv')
vio = pd.read_csv(dsDir/'vio.csv')
```

In [4]:
```
display(bus.head())
display(ins.head())
display(vio.head())
```

	business id column	name	address	city	state	postal_code	latitude	longitude
0	1000	HEUNG YUEN RESTAURANT	3279 22nd St	San Francisco	CA	94110	37.755282	-122.420493
1	100010	ILLY CAFFE SF_PIER 39	PIER 39 K-106-B	San Francisco	CA	94133	-9999.000000	-9999.000000
2	100017	AMICI'S EAST COAST PIZZERIA	475 06th St	San Francisco	CA	94103	-9999.000000	-9999.000000
3	100026	LOCAL CATERING	1566 CARROLL AVE	San Francisco	CA	94124	-9999.000000	-9999.000000
4	100030	OUI OUI! MACARON	2200 JERROLD AVE STE C	San Francisco	CA	94124	-9999.000000	-9999.000000

	iid	date	score	type
0	100010_20190329	03/29/2019 12:00:00 AM	-1	New Construction
1	100010_20190403	04/03/2019 12:00:00 AM	100	Routine - Unscheduled
2	100017_20190417	04/17/2019 12:00:00 AM	-1	New Ownership
3	100017_20190816	08/16/2019 12:00:00 AM	91	Routine - Unscheduled
4	100017_20190826	08/26/2019 12:00:00 AM	-1	Reinspection/Followup

	description	risk_category	vid
0	Consumer advisory not provided for raw or unde...	Moderate Risk	103128
1	Contaminated or adulterated food	High Risk	103108
2	Discharge from employee nose mouth or eye	Moderate Risk	103117
3	Employee eating or smoking	Moderate Risk	103118
4	Food in poor condition	Moderate Risk	103123

In [5]:
```
display(ins2vio.head())
```

	iid	vid
0	97975_20190725	103124
1	85986_20161011	103114
2	95754_20190327	103124
3	77005_20170429	103120
4	4794_20181030	103138

I. Data Cleaning: bus and ins

bus: Renaming Business ID

The `bus` dataframe contains a column called `business id column` which probably corresponds to a unique business id. We renamed the column to `bid` to assist with readibility.

In [6]:
```
bus = bus.rename(columns={'business id column': 'bid'})
```

bus: Postal Code

Here we examine the number of restaurants per zipcode.

In [7]:

```
zip_counts = bus.groupby('postal_code').size().sort_values(ascending = False)
print(zip_counts.to_string())
```

```
postal_code
94103          562
94110          555
94102          456
94107          408
94133          398
94109          382
94111          259
94122          255
94105          249
94118          231
94115          230
94108          229
94124          218
94114          200
-9999          194
94112          192
94117          189
94123          177
94121          157
94104          142
94132          132
94116           97
94158           90
94134           82
94127           67
94131           49
94130            8
94143            5
94013            2
94188            2
CA               2
94301            2
94101            2
95122            1
941033148        1
95133            1
95132            1
94102-5917       1
94014            1
941              1
94080            1
94105-2907       1
92672            1
64110            1
00000            1
94105-1420       1
941102019        1
95117            1
95112            1
95109            1
95105            1
94901            1
94621            1
94602            1
94544            1
94518            1
```

```
94117-3504      1
94120           1
94122-1909      1
94123-3106      1
94124-1917      1
94129           1
Ca              1
```

I noticed that there were a lot of missing, invalid and differently formatted zip codes.

So next I want to get a new column that gets the first 5 numbers of zip codes and have None for those with invalid or missing zipcodes.

In [8]:
```
#list of valid zipcodes in sf
valid_zips = pd.read_json('data/sf_zipcodes.json',dtype= str)['zip_code
s']
valid_zips.head(5)
```

Out[8]:
```
0    94102
1    94103
2    94104
3    94105
4    94107
Name: zip_codes, dtype: object
```

In [9]:
```
bus['postal5'] = bus['postal_code'].str[:5]
invalid_postal5 = bus[~bus['postal5'].isin(valid_zips)]['postal5'].uniqu
e()
bus['postal5'].replace(invalid_postal5, [None] * len(invalid_postal5), i
nplace = True)
bus.head()
```

Out[9]:

	bid	name	address	city	state	postal_code	latitude	longitude	p
0	1000	HEUNG YUEN RESTAURANT	3279 22nd St	San Francisco	CA	94110	37.755282	-122.420493	
1	100010	ILLY CAFFE SF_PIER 39	PIER 39 K-106-B	San Francisco	CA	94133	-9999.000000	-9999.000000	
2	100017	AMICI'S EAST COAST PIZZERIA	475 06th St	San Francisco	CA	94103	-9999.000000	-9999.000000	
3	100026	LOCAL CATERING	1566 CARROLL AVE	San Francisco	CA	94124	-9999.000000	-9999.000000	
4	100030	OUI OUI! MACARON	2200 JERROLD AVE STE C	San Francisco	CA	94124	-9999.000000	-9999.000000	

ins: Extract bid from each Inspection id

In [10]:
```
ins.head(5)
```

Out[10]:

	iid	date	score	type
0	100010_20190329	03/29/2019 12:00:00 AM	-1	New Construction
1	100010_20190403	04/03/2019 12:00:00 AM	100	Routine - Unscheduled
2	100017_20190417	04/17/2019 12:00:00 AM	-1	New Ownership
3	100017_20190816	08/16/2019 12:00:00 AM	91	Routine - Unscheduled
4	100017_20190826	08/26/2019 12:00:00 AM	-1	Reinspection/Followup

We notice that the column `iid` probably corresponds to an inspection id and has two numbers. The first number likely is the `bid` for the inspection. Next we are creating a new bid column in the ins data frame.

In [11]:
```
ins['bid'] = ins['iid'].str.split("_").apply(lambda b: int(b[0]))
ins
```

Out[11]:

	iid	date	score	type	bid
0	100010_20190329	03/29/2019 12:00:00 AM	-1	New Construction	100010
1	100010_20190403	04/03/2019 12:00:00 AM	100	Routine - Unscheduled	100010
2	100017_20190417	04/17/2019 12:00:00 AM	-1	New Ownership	100017
3	100017_20190816	08/16/2019 12:00:00 AM	91	Routine - Unscheduled	100017
4	100017_20190826	08/26/2019 12:00:00 AM	-1	Reinspection/Followup	100017
...	...	...	...	...	...
26658	999_20180924	09/24/2018 12:00:00 AM	-1	Routine - Scheduled	999
26659	999_20181102	11/02/2018 12:00:00 AM	-1	Reinspection/Followup	999
26660	999_20190909	09/09/2019 12:00:00 AM	80	Routine - Unscheduled	999
26661	99_20171207	12/07/2017 12:00:00 AM	82	Routine - Unscheduled	99
26662	99_20180808	08/08/2018 12:00:00 AM	84	Routine - Unscheduled	99

26663 rows × 5 columns

ins: Year Column

We want to get the year for each inspection for data analysis.

In [12]:
```python
ins_date_type = type(ins['date'][0])
ins['timestamp'] = pd.to_datetime(ins['date'])
ins['year'] = ins['timestamp'].dt.year
ins.head()
```

Out[12]:

	iid	date	score	type	bid	timestamp	year
0	100010_20190329	03/29/2019 12:00:00 AM	-1	New Construction	100010	2019-03-29	2019
1	100010_20190403	04/03/2019 12:00:00 AM	100	Routine - Unscheduled	100010	2019-04-03	2019
2	100017_20190417	04/17/2019 12:00:00 AM	-1	New Ownership	100017	2019-04-17	2019
3	100017_20190816	08/16/2019 12:00:00 AM	91	Routine - Unscheduled	100017	2019-08-16	2019
4	100017_20190826	08/26/2019 12:00:00 AM	-1	Reinspection/Followup	100017	2019-08-26	2019

Types of Inspections per Year 2016-19

In [13]:
```
ins_pivot = ins.pivot_table(index = 'type', columns = 'year', values =
'iid', aggfunc = 'count', fill_value = 0)
ins_pivot['Total'] = ins_pivot.sum(numeric_only=True, axis = 1)

ins_pivot_sorted = ins_pivot.sort_values('Total', ascending = False)

ins_pivot_sorted
```

Out[13]:

year type	2016	2017	2018	2019	Total
Routine - Unscheduled	966	4057	4373	4681	14077
Reinspection/Followup	445	1767	1935	2292	6439
New Ownership	99	506	528	459	1592
Complaint	91	418	512	437	1458
New Construction	102	485	218	189	994
Non-inspection site visit	51	276	253	231	811
New Ownership - Followup	0	45	219	235	499
Structural Inspection	1	153	50	190	394
Complaint Reinspection/Followup	19	68	70	70	227
Foodborne Illness Investigation	1	29	50	35	115
Routine - Scheduled	0	9	8	29	46
Administrative or Document Review	2	1	1	0	4
Multi-agency Investigation	0	0	1	2	3
Special Event	0	3	0	0	3
Community Health Assessment	1	0	0	0	1

ins: Missing Scores

In [14]:
```
ins['score'].value_counts().head()
```

Out[14]:
```
-1       12632
 100      1993
 96       1681
 92       1260
 94       1250
Name: score, dtype: int64
```

Something that I noticed was that there are a large number of inspections with the `'score'` of `-1`. This is probably a placeholder for missing scores.

Let's see which types of inspections had missing scores.

In [15]:
```
ins_tf = ins
ins_tf['Missing Score'] = ins_tf['score'] == -1
ins_missing_score_pivot = ins_tf.pivot_table(index = 'type', columns = 'Missing Score', aggfunc = 'count',values = 'iid', fill_value = 0)
ins_missing_score_pivot['Total'] = ins_missing_score_pivot.sum(numeric_only=True, axis = 1)
ins_missing_score_pivot = ins_missing_score_pivot.sort_values('Total', ascending = False)
ins_missing_score_pivot
```

Out[15]:

Missing Score	False	True	Total
type			
Routine - Unscheduled	14031	46	14077
Reinspection/Followup	0	6439	6439
New Ownership	0	1592	1592
Complaint	0	1458	1458
New Construction	0	994	994
Non-inspection site visit	0	811	811
New Ownership - Followup	0	499	499
Structural Inspection	0	394	394
Complaint Reinspection/Followup	0	227	227
Foodborne Illness Investigation	0	115	115
Routine - Scheduled	0	46	46
Administrative or Document Review	0	4	4
Multi-agency Investigation	0	3	3
Special Event	0	3	3
Community Health Assessment	0	1	1

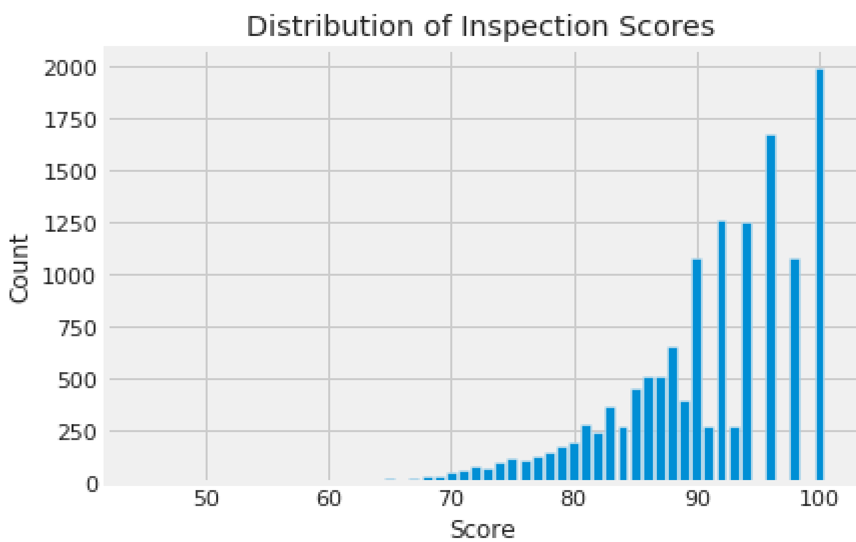
I noticed that inspection scores appear only to be assigned to `Routine - Unscheduled` inspections. It is reasonable that for inspection types such as `New Ownership` and `Complaint` to have no associated inspection scores, but we might be curious why there are no inspection scores for the `Reinspection/Followup` inspection type.

II. Data Analysis: Inspections

Here, I want to analyze the distribution of inspection scores.

In [16]:
```
filtered_ins = ins[ins['Missing Score'] == False]
plt.bar(filtered_ins['score'].value_counts().keys(), filtered_ins['scor
e'].value_counts())
plt.xlabel("Score")
plt.ylabel("Count")
plt.title("Distribution of Inspection Scores")
```

Out[16]:
```
Text(0.5, 1.0, 'Distribution of Inspection Scores')
```



OBSERVATION: The distribution of scores is skewed to the right of the graph, with almost all scores being greater than 60. This will be useful when we are visualizing 'high' and 'low' scores relative to each other. The scores that have the highest count are ones in the range of 90-100, which is good because we want scores for restaurants to be higher. We also see that there are also gaps within scores on the top. This can largely be dued to the point deductions from violations being even numbers.

II. Visualizations

In [17]:
```
bus.head()
```

Out[17]:

	bid	name	address	city	state	postal_code	latitude	longitude	p
0	1000	HEUNG YUEN RESTAURANT	3279 22nd St	San Francisco	CA	94110	37.755282	-122.420493	
1	100010	ILLY CAFFE SF_PIER 39	PIER 39 K-106-B	San Francisco	CA	94133	-9999.000000	-9999.000000	
2	100017	AMICI'S EAST COAST PIZZERIA	475 06th St	San Francisco	CA	94103	-9999.000000	-9999.000000	
3	100026	LOCAL CATERING	1566 CARROLL AVE	San Francisco	CA	94124	-9999.000000	-9999.000000	
4	100030	OUI OUI! MACARON	2200 JERROLD AVE STE C	San Francisco	CA	94124	-9999.000000	-9999.000000	

Getting relevant columns from business dataframe

We want:

- `bid`
- `name`
- `longitude`
- `latitude`
- `postal5`

In [18]:
```
bus_clean = bus[["bid","name", "latitude","longitude", "postal5"]]
bus_clean
```

Out[18]:

	bid	name	latitude	longitude	postal5
0	1000	HEUNG YUEN RESTAURANT	37.755282	-122.420493	94110
1	100010	ILLY CAFFE SF_PIER 39	-9999.000000	-9999.000000	94133
2	100017	AMICI'S EAST COAST PIZZERIA	-9999.000000	-9999.000000	94103
3	100026	LOCAL CATERING	-9999.000000	-9999.000000	94124
4	100030	OUI OUI! MACARON	-9999.000000	-9999.000000	94124
...	...	...	...	...	...
6248	99948	SUSIECAKES BAKERY	-9999.000000	-9999.000000	94118
6249	99988	HINODEYA SOMA	-9999.000000	-9999.000000	94107
6250	99991	TON TON	-9999.000000	-9999.000000	94102
6251	99992	URBAN EXPRESS KITCHENS LLC	-9999.000000	-9999.000000	94103
6252	99993	THE BRIXTON SOUTH	-9999.000000	-9999.000000	94102

6253 rows × 5 columns

Joining inspection with business information

In [19]:
```
ins_named = ins.merge(bus_clean, how = 'left', left_on = 'bid', right_on
='bid')
ins_named
```

Out[19]:

	iid	date	score	type	bid	timestamp	year	Miss Sc
0	100010_20190329	03/29/2019 12:00:00 AM	-1	New Construction	100010	2019-03-29	2019	·
1	100010_20190403	04/03/2019 12:00:00 AM	100	Routine - Unscheduled	100010	2019-04-03	2019	F
2	100017_20190417	04/17/2019 12:00:00 AM	-1	New Ownership	100017	2019-04-17	2019	·
3	100017_20190816	08/16/2019 12:00:00 AM	91	Routine - Unscheduled	100017	2019-08-16	2019	F
4	100017_20190826	08/26/2019 12:00:00 AM	-1	Reinspection/Followup	100017	2019-08-26	2019	·
...	...	...	...	...	...	...	...	
26658	999_20180924	09/24/2018 12:00:00 AM	-1	Routine - Scheduled	999	2018-09-24	2018	·
26659	999_20181102	11/02/2018 12:00:00 AM	-1	Reinspection/Followup	999	2018-11-02	2018	·
26660	999_20190909	09/09/2019 12:00:00 AM	80	Routine - Unscheduled	999	2019-09-09	2019	F
26661	99_20171207	12/07/2017 12:00:00 AM	82	Routine - Unscheduled	99	2017-12-07	2017	F
26662	99_20180808	08/08/2018 12:00:00 AM	84	Routine - Unscheduled	99	2018-08-08	2018	F

26663 rows × 12 columns

Getting the shapefile

I found the shapefile from [datasf.org (datasf.org)](datasf.org)

In [20]:
```
street_map = gpd.read_file('geo_export_b35327a2-c448-435e-b713-677e799d2
ba5.shp')
```

Visualizing Median Score of Routine - Unscheduled Inspections

Get the median score for each business

In [21]:
```
ins_median = ins_named[ins_named['Missing Score'] == False].groupby(
    ['bid',
     'longitude',
     'latitude'],
    as_index = False)['score'].median().rename(columns = {'score':'media
n score'})
ins_median
```

Out[21]:

	bid	longitude	latitude	median score
0	19	-122.421547	37.786848	95.0
1	24	-122.403135	37.792888	98.0
2	31	-122.419004	37.807155	95.0
3	45	-122.413641	37.747114	88.0
4	48	-122.465749	37.764013	90.5
...	...	...	...	...
5719	101853	-9999.000000	-9999.000000	100.0
5720	102067	-9999.000000	-9999.000000	100.0
5721	102257	-9999.000000	-9999.000000	94.0
5722	102336	-9999.000000	-9999.000000	82.0
5723	102398	-9999.000000	-9999.000000	90.0

5724 rows × 4 columns

GeoPandas

Here we are going to use GeoPandas to help with mapping points.

In [22]:
```
geometry = [Point(xy) for xy in zip (ins_median["longitude"], ins_median
['latitude'])]
```

In [23]:
```
geo_df = gpd.GeoDataFrame(ins_median,
                          crs = 4326,
                          geometry = geometry)
geo_df.head()
```

Out[23]:

	bid	longitude	latitude	median score	geometry
0	19	-122.421547	37.786848	95.0	POINT (-122.42155 37.78685)
1	24	-122.403135	37.792888	98.0	POINT (-122.40314 37.79289)
2	31	-122.419004	37.807155	95.0	POINT (-122.41900 37.80716)
3	45	-122.413641	37.747114	88.0	POINT (-122.41364 37.74711)
4	48	-122.465749	37.764013	90.5	POINT (-122.46575 37.76401)

Geocode Bounds for SF

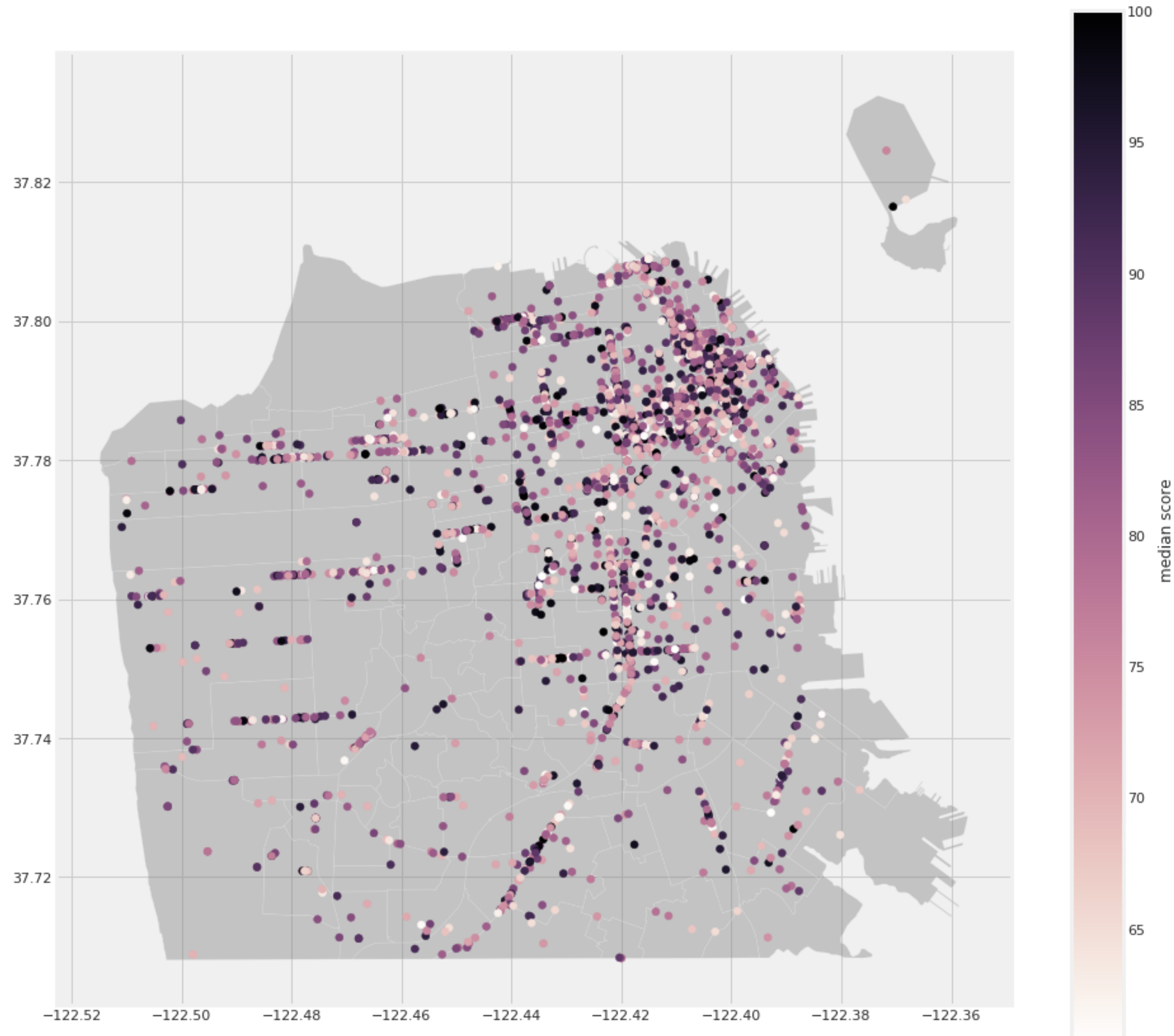
In [24]:
```
sf_ulat = 37.84
sf_llat = 37.70
sf_ulon = -122.34
sf_llon = -122.52
```

In [25]:
```
clean_geo = geo_df.loc[(geo_df['latitude'].between(sf_llat, sf_ulat))
                      & (geo_df['longitude'].between(sf_llon, sf_ulon))]
print(len(clean_geo))
```

```
2719
```

Map of Median Inspection Scores in SF

In [26]:
```
fig, axs = plt.subplots(figsize = (15,15))
streets_fig = street_map.plot(ax = axs, alpha = 0.4, color = 'grey')

cmap = sns.cubehelix_palette(light = 1, dark = 0,as_cmap=True)

point_fig = clean_geo.plot(ax = axs,
                           c = 'median score',
                           cmap = cmap,)

norm = plt.Normalize(clean_geo['median score'].min(), clean_geo['median
 score'].max())

sm = plt.cm.ScalarMappable(cmap = cmap, norm=norm)
sm.set_array([])
# Remove the legend and add a colorbar
axs.figure.colorbar(sm).set_label('median score')
```



Inspection Type Visualization and Analysis

Let's look at the different types of inspections.

In [27]:
```
ins_types = pd.Series(ins_pivot.index)
ins_types
```

Out[27]:
```
0      Administrative or Document Review
1            Community Health Assessment
2                              Complaint
3          Complaint Reinspection/Followup
4          Foodborne Illness Investigation
5               Multi-agency Investigation
6                         New Construction
7                            New Ownership
8                 New Ownership - Followup
9                Non-inspection site visit
10                   Reinspection/Followup
11                     Routine - Scheduled
12                   Routine - Unscheduled
13                           Special Event
14                   Structural Inspection
Name: type, dtype: object
```

In [28]:
```
clean_type = ins_named.loc[(ins_named['latitude'].between(sf_llat, sf_ul
at))
                      & (ins_named['longitude'].between(sf_llon, sf_ulon
)),
                           ['bid', 'name', 'longitude', 'latitude','typ
e','year',]]
clean_type
```

Out[28]:

	bid	name	longitude	latitude	type	year
59	1000	HEUNG YUEN RESTAURANT	-122.420493	37.755282	Reinspection/Followup	2016
60	1000	HEUNG YUEN RESTAURANT	-122.420493	37.755282	Routine - Unscheduled	2017
61	1000	HEUNG YUEN RESTAURANT	-122.420493	37.755282	Reinspection/Followup	2017
62	1000	HEUNG YUEN RESTAURANT	-122.420493	37.755282	Routine - Unscheduled	2018
63	1000	HEUNG YUEN RESTAURANT	-122.420493	37.755282	Reinspection/Followup	2018
...	...	...	...	...	...	...
26658	999	SERRANO'S PIZZA II	-122.420534	37.756997	Routine - Scheduled	2018
26659	999	SERRANO'S PIZZA II	-122.420534	37.756997	Reinspection/Followup	2018
26660	999	SERRANO'S PIZZA II	-122.420534	37.756997	Routine - Unscheduled	2019
26661	99	J & M A-1 CAFE RESTAURANT LLC	-122.405967	37.794293	Routine - Unscheduled	2017
26662	99	J & M A-1 CAFE RESTAURANT LLC	-122.405967	37.794293	Routine - Unscheduled	2018

12571 rows × 6 columns

New Constructions Per Year

Let's look at the number of new construction inspections per year.

In [29]:
```
ins_newcon = ins_named[ins_named['type'] == 'New Construction'].loc[:,[
'bid',
     'name',
     'longitude',
     'latitude',
     'year']]
ins_newcon
```

Out[29]:

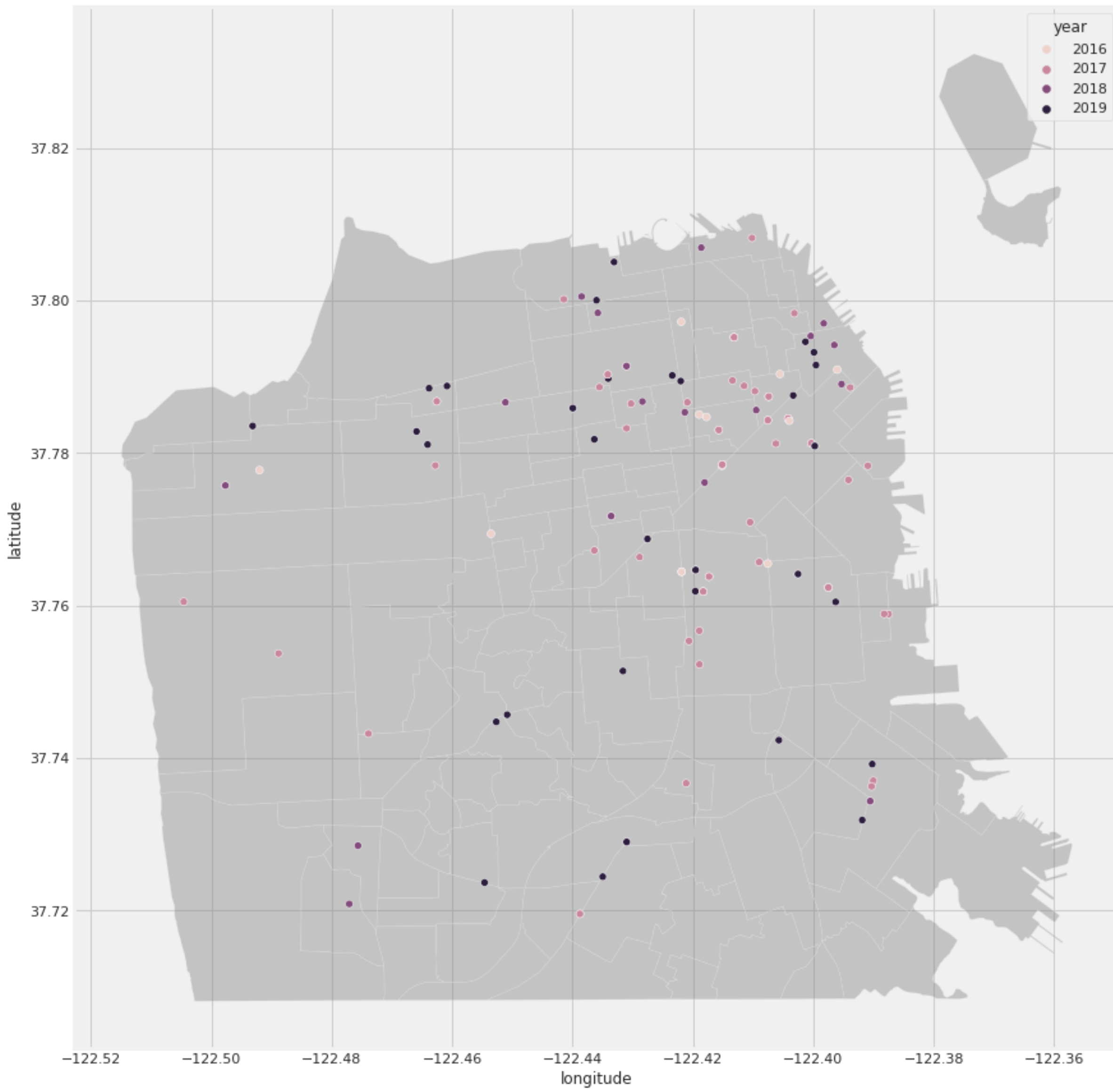
	bid	name	longitude	latitude	year
0	100010	ILLY CAFFE SF_PIER 39	-9999.0	-9999.0	2019
25	100059	DUMPLING ALLEY	-9999.0	-9999.0	2019
26	100059	DUMPLING ALLEY	-9999.0	-9999.0	2019
27	100059	DUMPLING ALLEY	-9999.0	-9999.0	2019
39	100081	THE MATTERHORN RESTAURANT AND BAKERY	-9999.0	-9999.0	2019
...	...	...	...	...	...
26638	99948	SUSIECAKES BAKERY	-9999.0	-9999.0	2019
26639	99948	SUSIECAKES BAKERY	-9999.0	-9999.0	2019
26640	99948	SUSIECAKES BAKERY	-9999.0	-9999.0	2019
26649	99993	THE BRIXTON SOUTH	-9999.0	-9999.0	2019
26650	99993	THE BRIXTON SOUTH	-9999.0	-9999.0	2019

994 rows × 5 columns

Visual: New Constructions Locations by Year

I decided to use seaborn to plot points here to explore other ways to use the shapefile.

```
In [30]: fig, axs = plt.subplots(figsize = (15,15))
streets_fig = street_map.plot(ax = axs, alpha = 0.4, color = 'grey')

newcon_fig = sns.scatterplot(ax = axs,
                             data = clean_type[clean_type['type'] == 'Ne
w Construction'],
                             x = 'longitude',
                             y = 'latitude',
                             hue = 'year',)
```

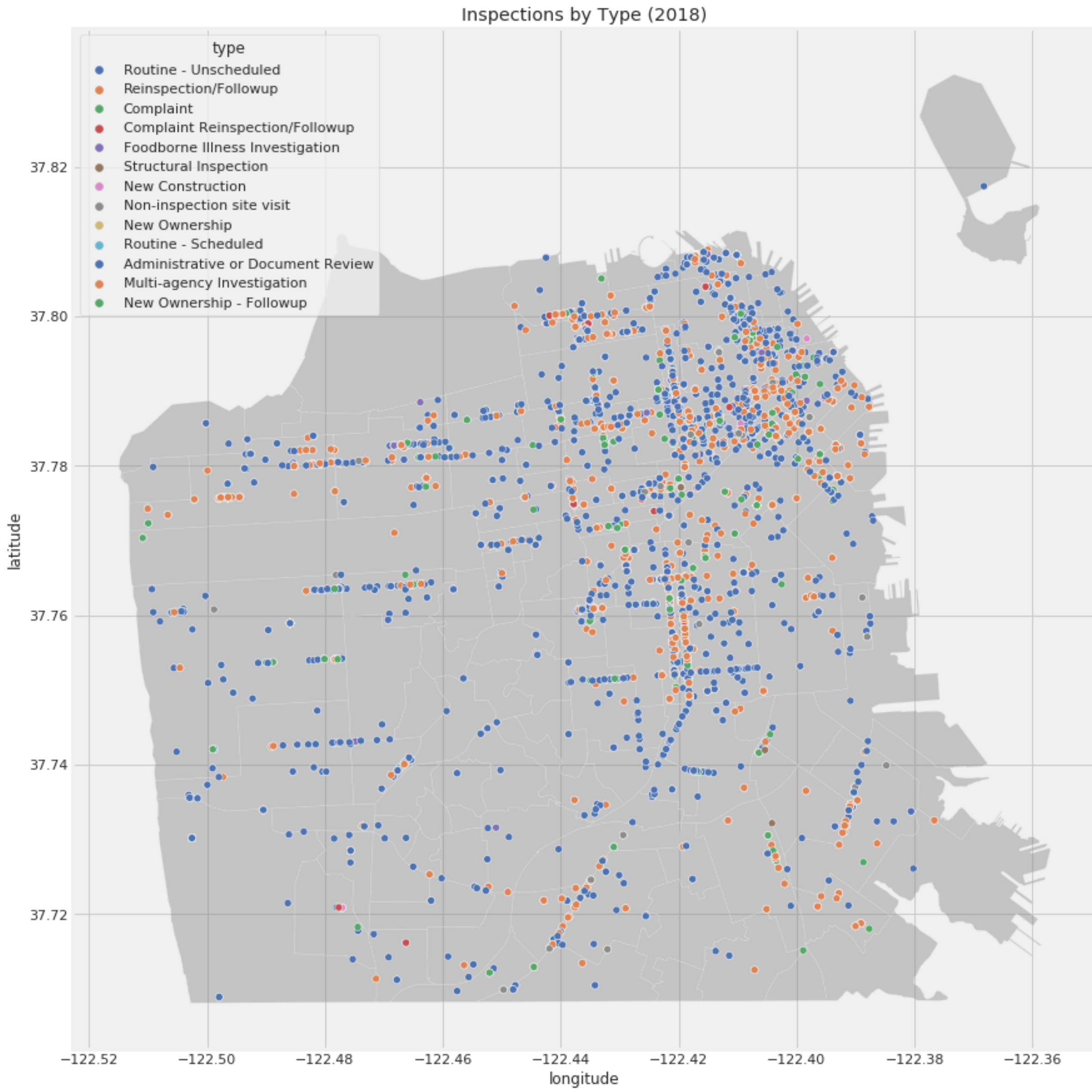


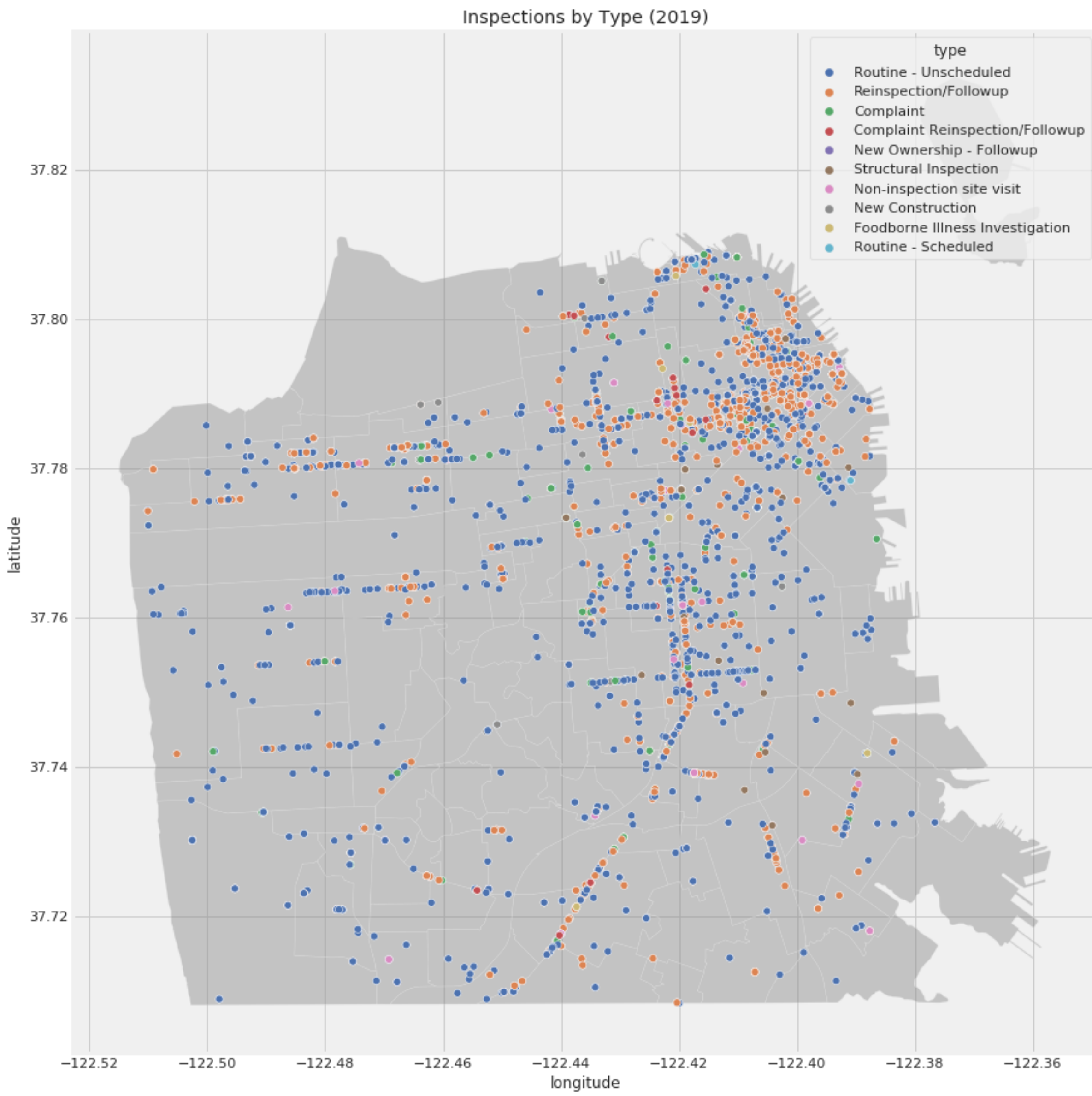
OBSERVATION: A lot of the inspections that were New Construction did not have longitude and latitude points. Thus, we are missing a lot of data on where a lot of New Construction are located.

Inspections types location (2018 and 2019)

In [31]:
```
fig, axs = plt.subplots(figsize = (15,15))
streets_fig = street_map.plot(ax = axs, alpha = 0.4, color = 'grey')

newcon_fig = sns.scatterplot(ax = axs,
                             data = clean_type[clean_type['year'] == 201
8],
                             x = 'longitude',
                             y = 'latitude',
                             hue = 'type',
                             palette = 'deep',)
plt.title("Inspections by Type (2018)")

fig, axs = plt.subplots(figsize = (15,15))
streets_fig = street_map.plot(ax = axs, alpha = 0.4, color = 'grey')

newcon_fig = sns.scatterplot(ax = axs,
                             data = clean_type[clean_type['year'] == 201
9],
                             x = 'longitude',
                             y = 'latitude',
                             hue = 'type',
                             palette = 'deep',)
plt.title("Inspections by Type (2019)")
```

```
Out[31]: Text(0.5, 1, 'Inspections by Type (2019)')
```



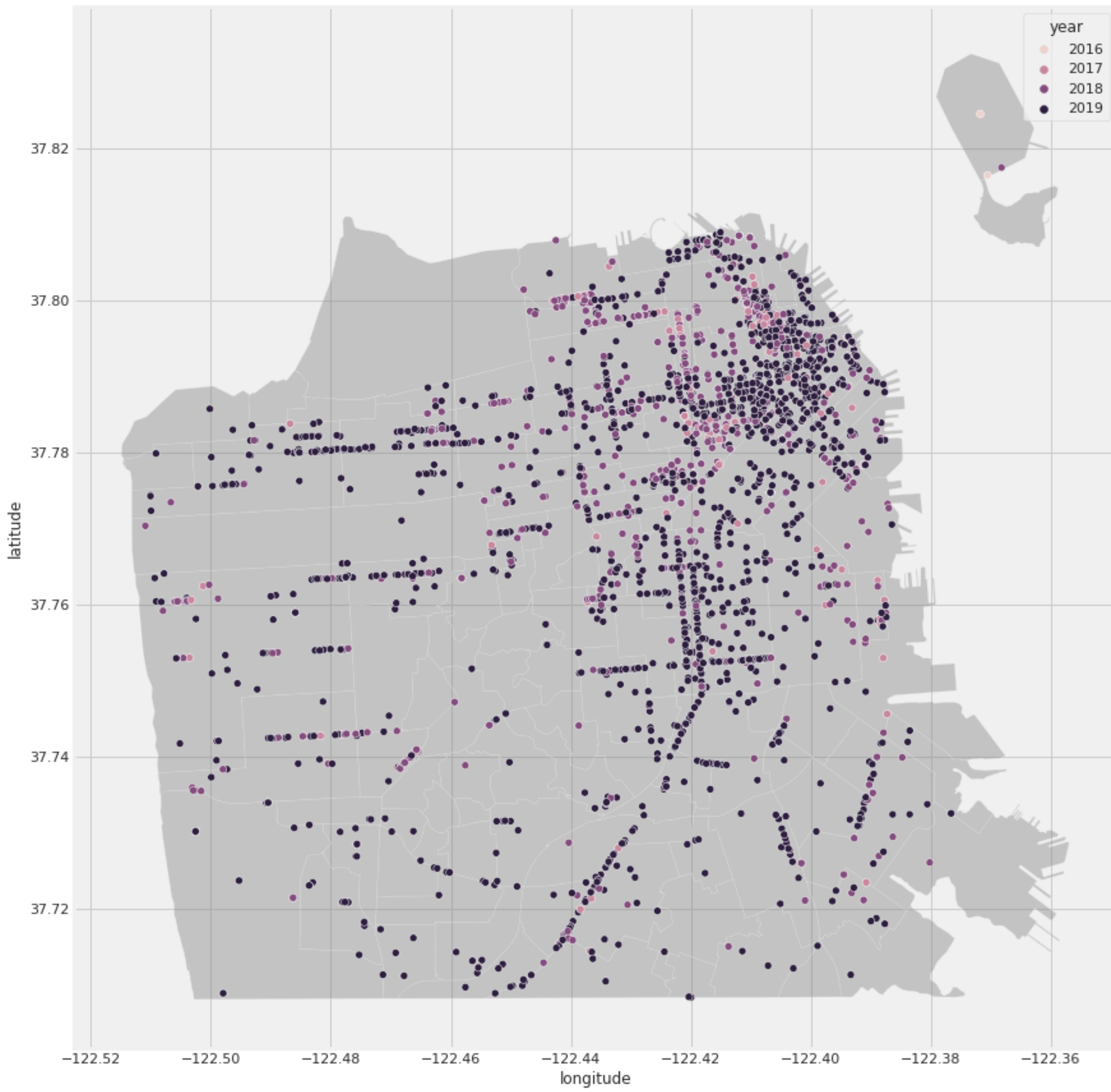


Inspections Locations per Year

I looked next at where inspections took place over the years. For each business, I chose to look at the their latest inspection.

In [32]:
```
recent_ins = clean_type.sort_values("year", ascending = False).groupby([
'bid','longitude', 'latitude']).first().reset_index()
```

In [33]:
```
fig, axs = plt.subplots(figsize = (15,15))
streets_fig = street_map.plot(ax = axs, alpha = 0.4, color = 'grey')


newcon_fig = sns.scatterplot(ax = axs,
                             data = recent_ins,
                             x = 'longitude',
                             y = 'latitude',
                             hue = 'year',)
```



In [ ]:

III. Visualization: Median Score per Zip Code (2019)

In [34]:
```
valid_zips = pd.read_json('data/sf_zipcodes.json',dtype= str)['zip_codes']

valid_postal5 = bus[bus['postal5'].isin(valid_zips)]
valid_postal5
```

Out[34]:

	bid	name	address	city	state	postal_code	latitude	longitu
0	1000	HEUNG YUEN RESTAURANT	3279 22nd St	San Francisco	CA	94110	37.755282	-122.4204
1	100010	ILLY CAFFE SF_PIER 39	PIER 39 K-106-B	San Francisco	CA	94133	-9999.000000	-9999.0000
2	100017	AMICI'S EAST COAST PIZZERIA	475 06th St	San Francisco	CA	94103	-9999.000000	-9999.0000
3	100026	LOCAL CATERING	1566 CARROLL AVE	San Francisco	CA	94124	-9999.000000	-9999.0000
4	100030	OUI OUI! MACARON	2200 JERROLD AVE STE C	San Francisco	CA	94124	-9999.000000	-9999.0000
...	...	...	...	...	...	...	...	
6248	99948	SUSIECAKES BAKERY	3509 CALIFORNIA ST	San Francisco	CA	94118	-9999.000000	-9999.0000
6249	99988	HINODEYA SOMA	303 02nd ST STE 102	San Francisco	CA	94107	-9999.000000	-9999.0000
6250	99991	TON TON	422 GEARY ST	San Francisco	CA	94102	-9999.000000	-9999.0000
6251	99992	URBAN EXPRESS KITCHENS LLC	475 06th ST	San Francisco	CA	94103	-9999.000000	-9999.0000
6252	99993	THE BRIXTON SOUTH	701 02nd St	San Francisco	CA	94102	-9999.000000	-9999.0000

6032 rows × 10 columns

In [35]:
```
bus_clean_zip = bus[["bid", "postal5"]]
bus_clean_zip
```

Out[35]:

	bid	postal5
0	1000	94110
1	100010	94133
2	100017	94103
3	100026	94124
4	100030	94124
...	...	...
6248	99948	94118
6249	99988	94107
6250	99991	94102
6251	99992	94103
6252	99993	94102

6253 rows × 2 columns

In [36]:
```
ins_zip_score = ins_named[(ins_named['Missing Score'] == False) & (ins_n
amed['year'] == 2019)].groupby(
    'postal5',
    as_index = False)['score'].median().rename(columns = {'score':'media
n_score'})
ins_zip_score.head()
```

Out[36]:

	postal5	median_score
0	94102	92.0
1	94103	91.0
2	94104	90.0
3	94105	90.0
4	94107	96.0

In [37]:
```
data = 'SanFrancisco.Neighborhoods.json'
gdf = gpd.read_file(data)
gdf.head()
```

Out[37]:

	id	neighborhood	geometry
0	94105	Rincon Hill	GEOMETRYCOLLECTION (POLYGON ((-122.39170 37.79...
1	94107	South Beach	GEOMETRYCOLLECTION (POLYGON ((-122.38777 37.78...
2	94108	Chinatown	GEOMETRYCOLLECTION (POLYGON ((-122.40496 37.79...
3	94109	Nob Hill	GEOMETRYCOLLECTION (POLYGON ((-122.42043 37.80...
4	94112	Ingleside	GEOMETRYCOLLECTION (POLYGON ((-122.42070 37.73...

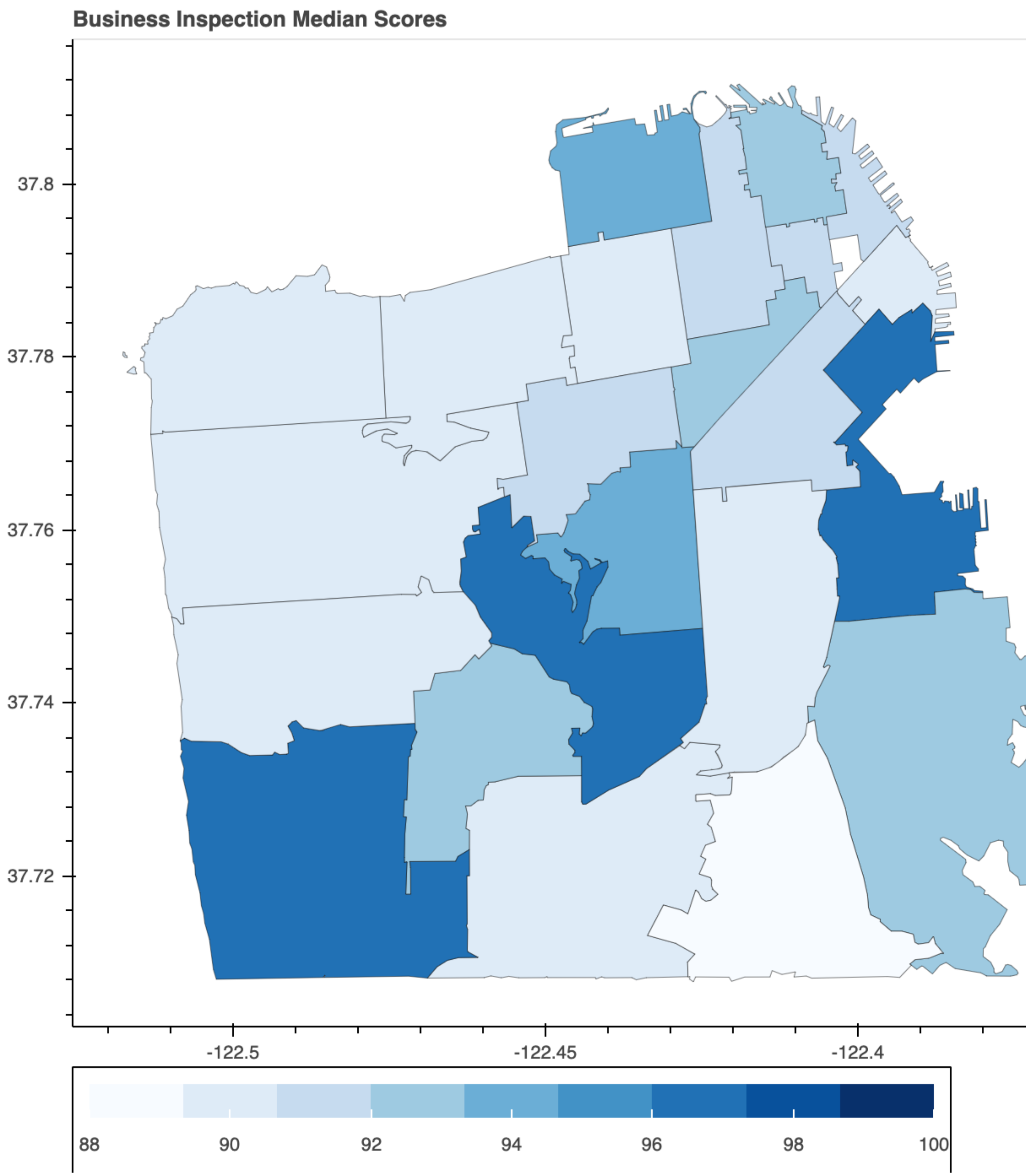
In [38]:
```
ins_zip_score['id'] = ins_zip_score['postal5'].astype(str)
```

In [39]:
```
merge = gdf.merge(ins_zip_score, how='left', on='id')

merged_json = json.loads(merge.to_json())
json_data = json.dumps(merged_json)
```

In [40]:

```
geosource = GeoJSONDataSource(geojson = json_data)
#set the color palette
palette = brewer['Blues'][9]
palette = palette[::-1]
color_mapper = LinearColorMapper(palette = palette, low = 88, high = 100
,  nan_color = '#d9d9d9')
color_bar = ColorBar(color_mapper=color_mapper, label_standoff=8,width =
500, height = 20,
border_line_color='black',location = (0,0), orientation ='horizontal')
#Set the size and title of the graph
p = figure(title = 'Business Inspection Median Scores', plot_height = 70
0 , plot_width = 700, toolbar_location = None,
          tooltips=[
         ("Name", "@neighborhood"),
         ("Zip Code","@id"),
         ("Median Score", "@median_score")])
#Makes it so there are no grid lines
p.xgrid.grid_line_color = None
p.ygrid.grid_line_color = None
p.patches('xs','ys', source = geosource,fill_color = {'field':'median_sc
ore', 'transform' : color_mapper},
         line_color = 'black', line_width = 0.25, fill_alpha = 1)
p.add_layout(color_bar, 'below')
output_notebook()
show(p)
```

(https://bokeh.org) BokehJS 2.2.3 successfully loaded.



In [ ]:

In [ ]: