THE PUBLIC LIBRARY
of Cincinnati and Hamilton County

CincyPy Meetup: Data Visualizations

ray.voelker@gmail.com

In this notebook, we'll explore some tools that are part of the PyViz conda metapackage

In [1]:

```
# read the google maps API from the env
# * remember to export the key before starting the notebook...
# source ~/google_maps_api_key
GOOGLE_API_KEY = %env GOOGLE_API_KEY

%matplotlib inline

# pandas DataFrame object is at the heart of most of these visualizations
import pandas as pd

import matplotlib.pyplot as plt

# note that seaborn isn't part of the dataviz conda metapackage, and you'll have
to pip install it
import seaborn as sns

# bokeh produces interactive plots / visualizations
from bokeh.io import output_file, output_notebook, show
from bokeh.models import ColumnDataSource, GMapOptions, Label
from bokeh.plotting import gmap, figure, output_file, show

output_notebook()
```

(https://bokeh.pydata.org) BokehJS 1.2.0 successfully loaded.

We previously geo-coded data about patrons ...

We ran data through the smartystreets service to clean up address information, and to get geo-coded values from the address info. This data is now in 3 .csv files:

1. `patron_data.csv` : basic library **patron data** (including geo-coded locations) of patrons **having circulation activity**
2. `patron_data_active_no_circs.csv` : geo-coded locations of **patrons having no circulation activity**, but some other type of activity or interaction with the library
3. `branches.csv` : geo-coded locations of library branches

In [41]:
```
# show the head of the .csv file about patrons ...
# (data has been fuzzed to obscure patron data)
!head -n 5 data/patron_data_fuzz.csv
```

```
patron_record_id,patron_zipcode,patron_full_zipcode,patron_latitude,patron_lon
gitude,count_checkouts,count_checkins
481037331234,45233,45233-1460,39.11,-84.68,7,8
481037331235,45039,45039-9222,39.33,-84.23,498,486
481037331236,45230,45230-1606,39.09,-84.38,92,95
481037331237,45208,45208-1760,39.14,-84.41,92,91
```

In [42]:

```
# show the head of the .csv file about patrons (no physical-item circulation
s)...
# (data has been fuzzed to obscure patron data)
!head -n 5 data/patron_data_active_no_circs_fuzz.csv
```

```
patron_record_id,patron_zipcode,patron_full_zipcode,patron_latitude,patron_lon
gitude
481037331234,45247,45247-6958,39.19,-84.61
481037331235,45249,45249-1246,39.29,-84.34
481037331236,45014,45014-3516,39.32,-84.56
481037331237,45140,45140-2815,39.25,-84.27
```

In [4]:

```
# show the head of the .csv file with branch location information...
!head -n 5 data/branches.csv
```

```
location_code,branch_latitude,branch_longitude
1,39.10577,-84.51331
an,39.08623,-84.35268
av,39.14699,-84.48798
ba,39.2298,-84.37373
```

Read that data!

Now that we have the modules and some environment stuff set, lets read some data into **DataFrames**

In [5]:

```
# create pandas dataframe of: patrons with circulations
patron_data = pd.read_csv(
    './data/patron_data.csv')

# create pandas dataframe of: patrons with activity
# (patrons with login to account or download of e-books, etc but no pysical circ
s)
patron_data_active_no_circs = pd.read_csv(
    './data/patron_data_active_no_circs.csv')

# create pandas dataframe of: branch locations
branches = pd.read_csv(
    './data/branches.csv')
```

Get some info about the records we just read in…

In [6]:

```
# count the number of records we have in the: patrons with circulations
patron_data['patron_record_id'].count()
```

Out[6]:

```
103814
```

In [7]:

```
# count the number of records we have in the: patrons with circulations
patron_data_active_no_circs['patron_record_id'].count()
```

Out[7]: 181993

In [8]:
```
# count the number of records we have in the: branch locations
branches['location_code'].count()
```
Out[8]:
```
42
```

Using pandas dataframes, we can also quickly get some additional data from our datasets...

Computing what the median, and standard deviations of the latitude, and longitude values may be useful ...

In [9]:

```
# median latitudes from both our data sets...
print('\n'.join((
    str(patron_data['patron_latitude'].median()),
    str(patron_data_active_no_circs['patron_latitude'].median())
)))
```

```
39.18468
39.17685
```

In [10]:

```
# median longitudes from both our data sets...
print('\n'.join((
    str(patron_data['patron_longitude'].median()),
    str(patron_data_active_no_circs['patron_longitude'].median())
)))
```

```
-84.47425
-84.50079000000001
```

In [11]:

```
# standard deviations
print('\n'.join((
    str(patron_data['patron_longitude'].std()),
    str(patron_data_active_no_circs['patron_longitude'].std())
)))
```

```
0.17051549657832074
0.30648479764761066
```

I thought this was a presentation about data visualizations?!?

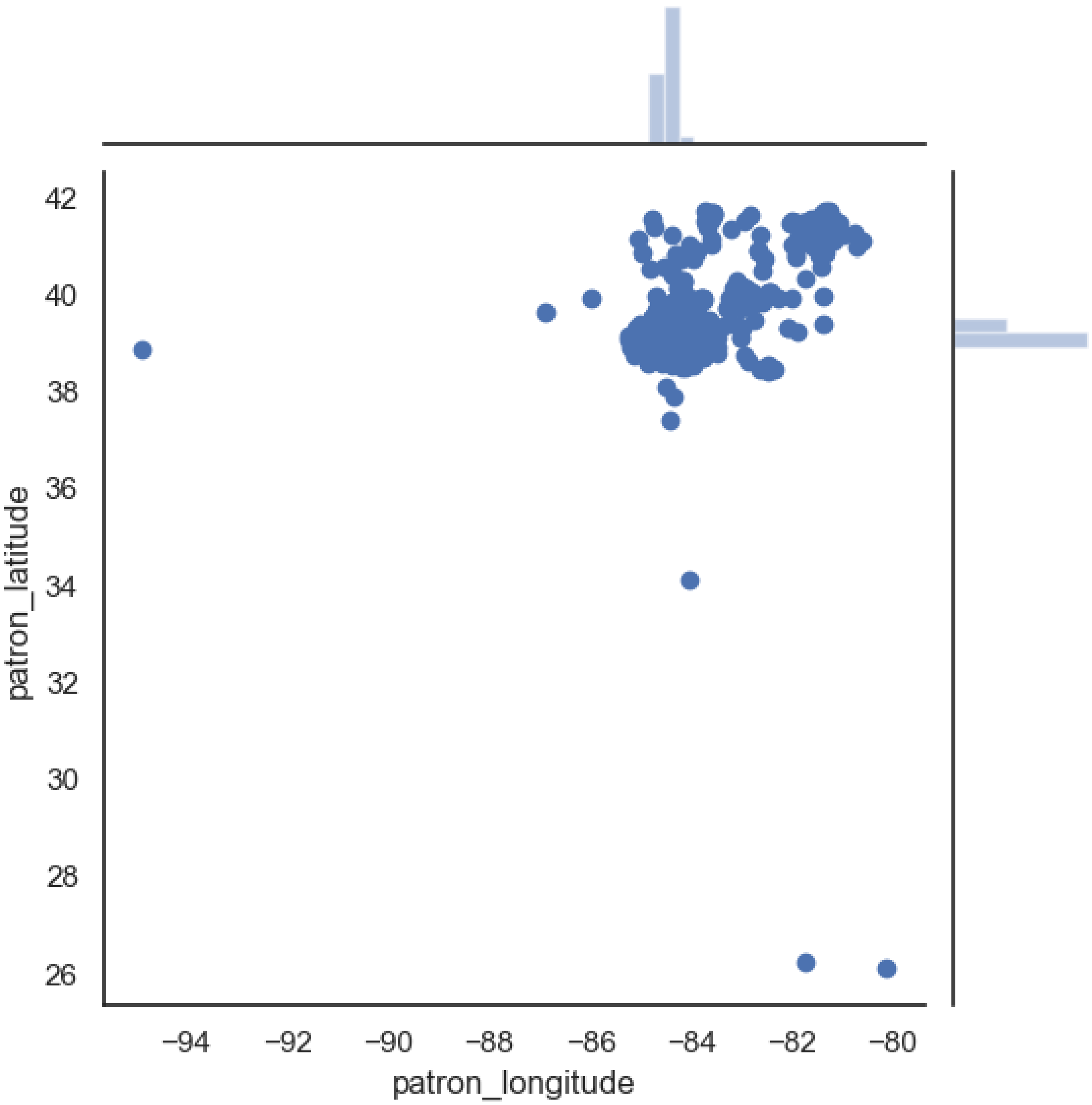
Using Matplotlib, and another plotting module, called `seaborn` , we can find out some basic info about our data set...

In [12]:

```
plt.rcParams['figure.dpi'] = 100
sns.set(style="white", color_codes=True)
sns.jointplot("patron_longitude", "patron_latitude", data=patron_data)
```

Out[12]:

```
<seaborn.axisgrid.JointGrid at 0x7f5dc9d736a0>
```



42
40
38
36
34
32
30
28
26
patron_latitude
−94
−92
−90
−88
−86
−84
−82
−80
patron_longitude

It looks like we have a few outliers ... maybe we want to take a closer look...

In [13]:
```
patron_data['patron_longitude'].describe()
```

Out[13]:
```
count    103814.000000
mean        -84.472887
std           0.170515
min         -94.864150
25%         -84.578440
50%         -84.474250
75%         -84.375790
max         -80.118610
Name: patron_longitude, dtype: float64
```

In [14]:
```
patron_data['patron_latitude'].describe()
```

Out[14]:
```
count    103814.000000
mean         39.190854
std           0.122342
min          26.143680
25%          39.135260
50%          39.184680
75%          39.239860
max          41.723930
Name: patron_latitude, dtype: float64
```

OK, great ... so, we know must of our data is within the range `39.135260` (25th percentile,) and `39.239860` (75th percentile)

In [15]:

```
patron_data['patron_latitude'].quantile([.25, .75])
```

Out[15]:

```
0.25    39.13526
0.75    39.23986
Name: patron_latitude, dtype: float64
```

In [16]:

```
# filter the latitude values
patron_data_filtered = patron_data[
        patron_data['patron_latitude'].between(39.13526,
                                               39.23986)]

# now, lets see what it looks like
patron_data_filtered.describe()
```

Out[16]:

	patron_record_id	patron_zipcode	patron_latitude	patron_longitude	count_checkouts	count_checkins
count	5.190600e+04	51906.000000	51906.000000	51906.000000	51906.000000	51906.000000
mean	4.810379e+11	45216.008092	39.184200	-84.501022	33.267907	32.903691
std	4.380452e+05	58.254889	0.031353	0.114090	74.463601	71.971787
min	4.810373e+11	45001.000000	39.135270	-85.258940	0.000000	0.000000
25%	4.810375e+11	45212.000000	39.154230	-84.590400	3.000000	3.000000
50%	4.810377e+11	45227.000000	39.184680	-84.492350	10.000000	10.000000
75%	4.810383e+11	45239.000000	39.211780	-84.412800	33.000000	32.000000
max	4.810387e+11	47037.000000	39.239860	-83.692790	3476.000000	3475.000000

In [17]:

```
# now lets do the same for longitude
patron_data['patron_longitude'].describe()
```

Out[17]:

```
count    103814.000000
mean        -84.472887
std           0.170515
min         -94.864150
25%         -84.578440
50%         -84.474250
75%         -84.375790
max         -80.118610
Name: patron_longitude, dtype: float64
```

In [18]:

```python
patron_data['patron_longitude'].quantile([.25, .75])
```

Out[18]:

```
0.25   -84.57844
0.75   -84.37579
Name: patron_longitude, dtype: float64
```

In [19]:

```
# remember that we're filtering the previous filtered dataframe
patron_data_filtered = patron_data_filtered[
    patron_data_filtered['patron_longitude'].between(-84.57844,
                                                      -84.37579)]

patron_data_filtered.describe()
```

Out[19]:

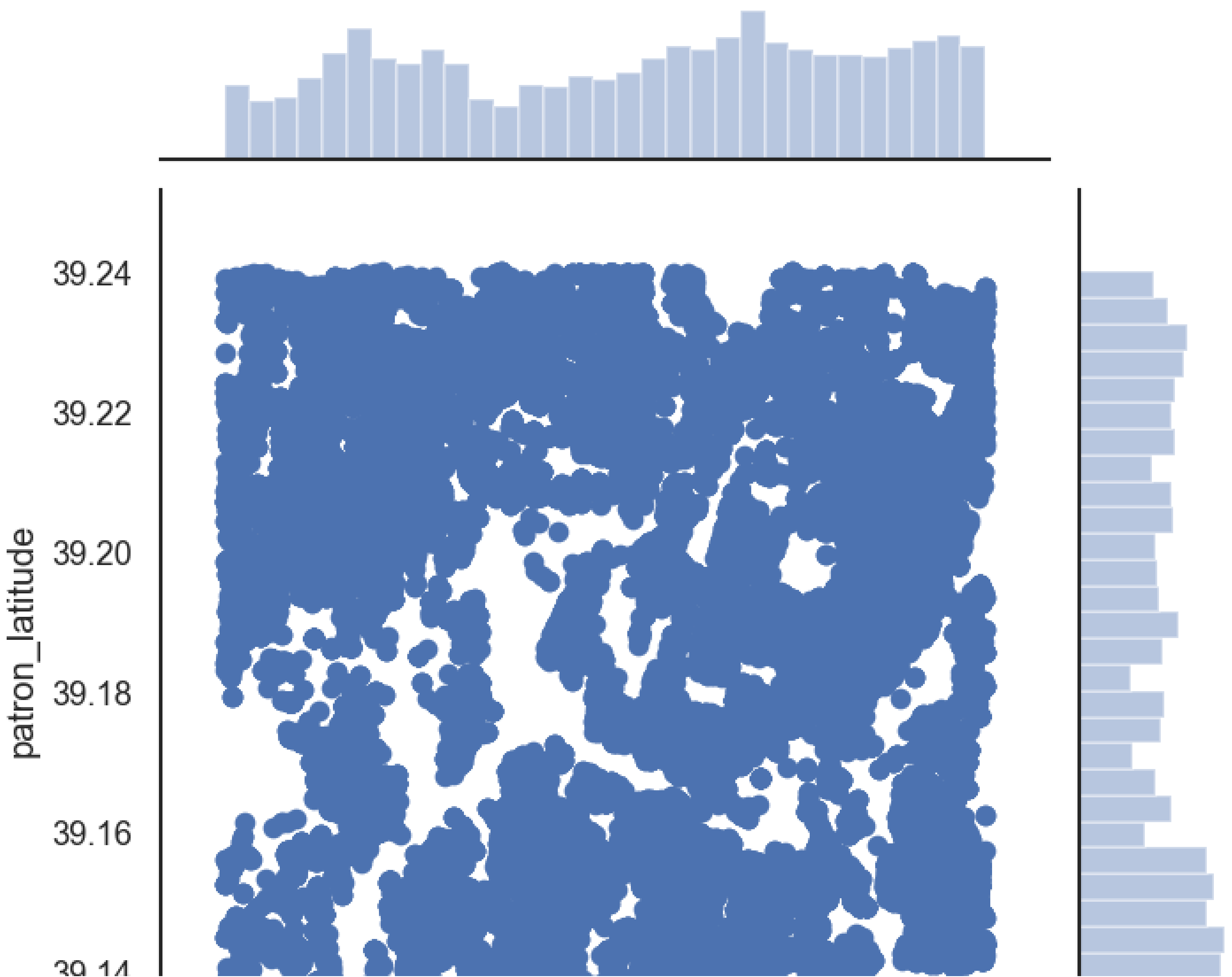
	patron_record_id	patron_zipcode	patron_latitude	patron_longitude	count_checkouts	count_checkins
count	3.096300e+04	30963.000000	30963.000000	30963.000000	30963.000000	30963.000000
mean	4.810379e+11	45223.627652	39.184639	-84.469508	33.698737	33.325679
std	4.470725e+05	10.702195	0.031867	0.058260	76.027779	75.045064
min	4.810373e+11	45206.000000	39.135270	-84.578440	0.000000	0.000000
25%	4.810375e+11	45215.000000	39.153930	-84.523620	3.000000	3.000000
50%	4.810377e+11	45224.000000	39.184940	-84.461690	10.000000	10.000000
75%	4.810384e+11	45232.000000	39.213950	-84.420180	33.000000	32.000000
max	4.810387e+11	45275.000000	39.239860	-84.375800	3476.000000	3475.000000

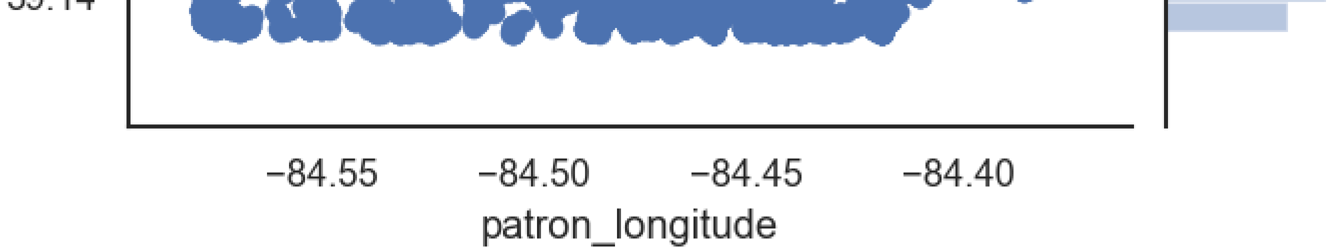
Plot-em! (using Seaborn / Matplotlib)

Let's re-run the plot from above, but now with the filtered results...

In [20]:
```
plt.rcParams['figure.dpi'] = 120
sns.set(style="white", color_codes=True)
sns.jointplot("patron_longitude", "patron_latitude", data=patron_data_filtered)
```

Out[20]:
```
<seaborn.axisgrid.JointGrid at 0x7f5dc77a4208>
```




−84.55
−84.50
−84.45
−84.40
patron_longitude

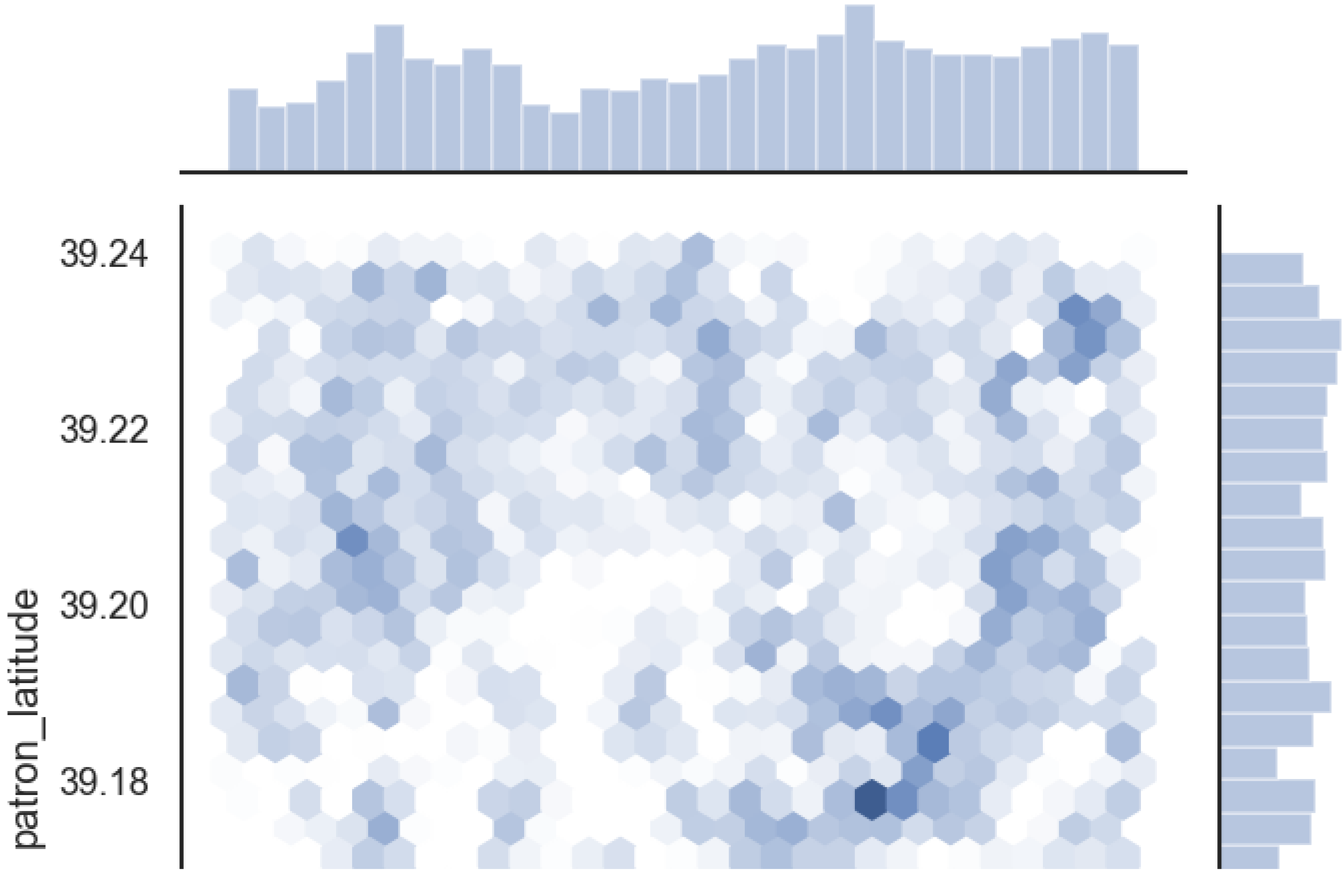
The previous plot doesn't do a great job of showing overplotting ... try using a plot that uses the concept of "binning"

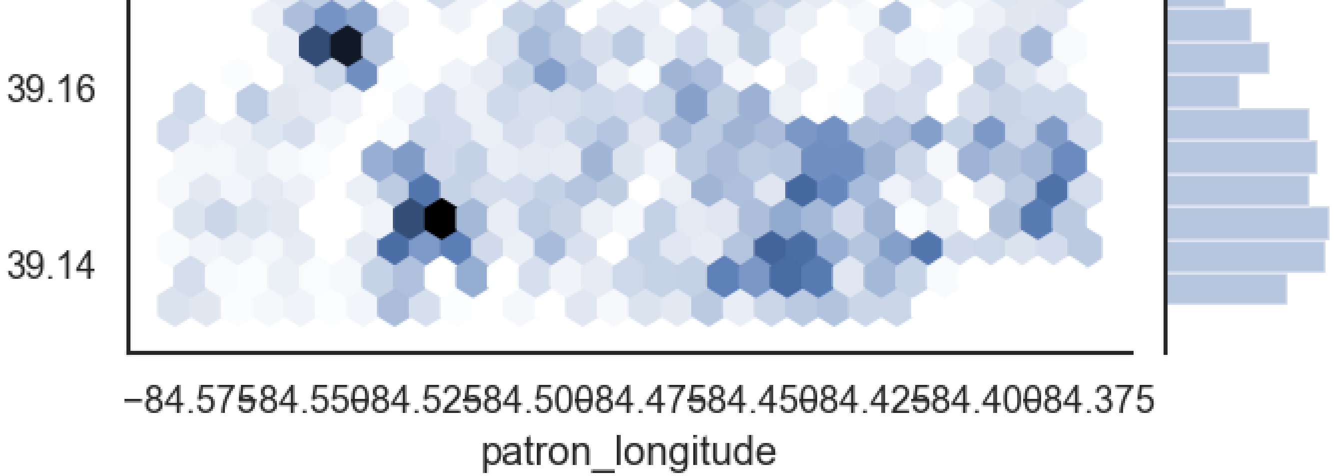
Hexbin plots

... shows the counts of observations that fall within hexagonal bins.

In [21]:
```
# TODO: fix the xlabels
plt.figure(figsize = (16,10))
sns.jointplot("patron_longitude",
              "patron_latitude",
              data=patron_data_filtered,
              kind='hex'
)
```

Out[21]:
```
<seaborn.axisgrid.JointGrid at 0x7f5dc76aada0>

<Figure size 1920x1200 with 0 Axes>
```

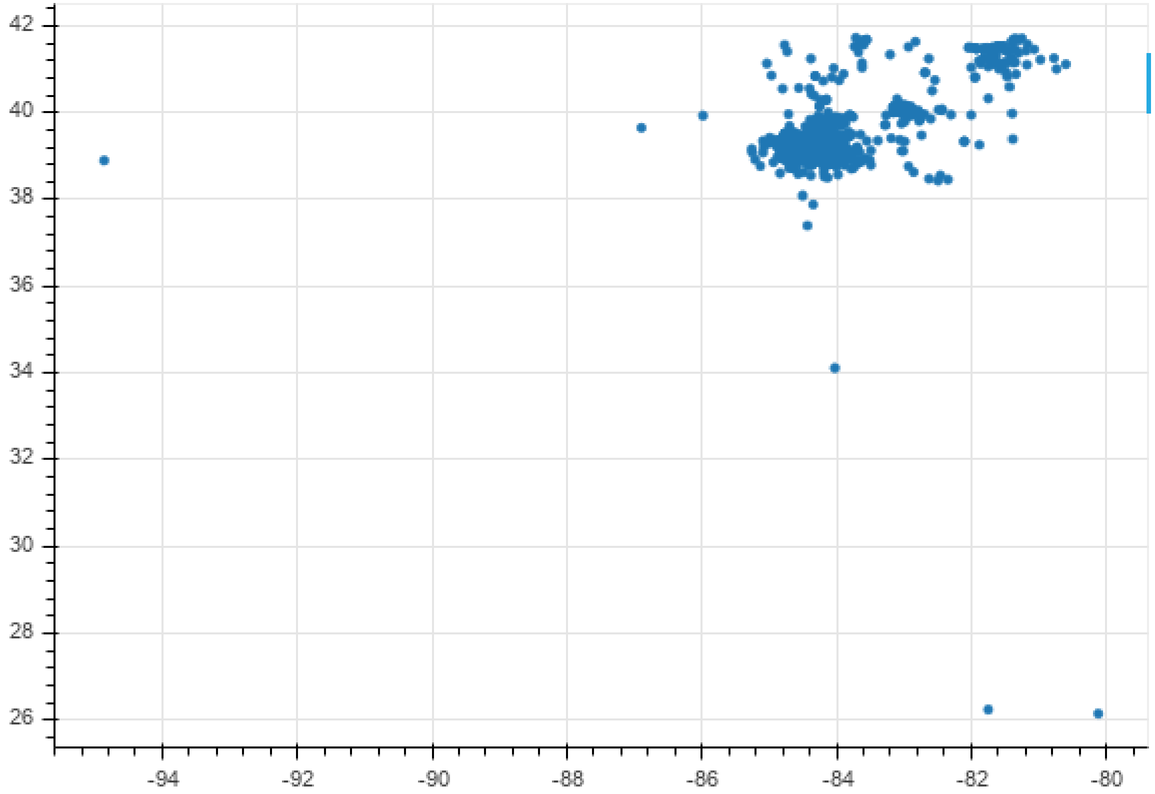



39.16
39.14
−84.575
−84.550
−84.525
−84.500
−84.475
−84.450
−84.425
−84.400
−84.375
patron_longitude

Bokeh Plot

Below is an interactive bokeh plot of that same data...

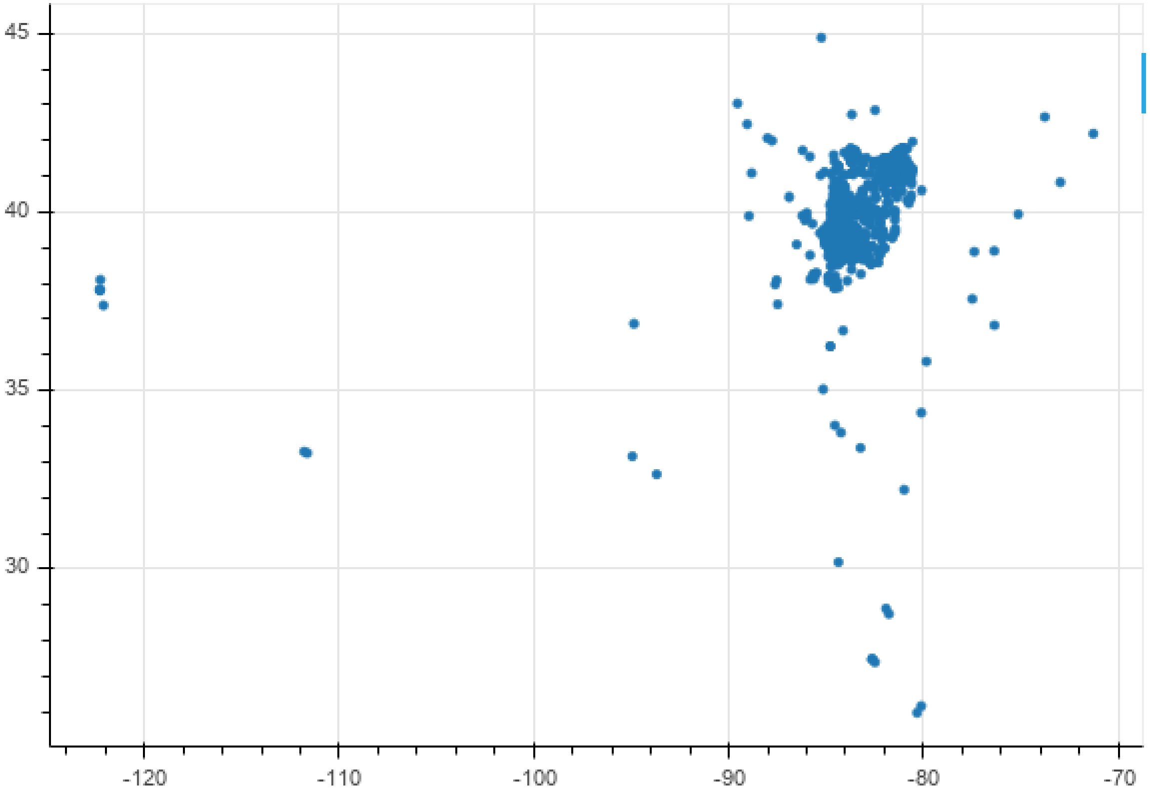
In [22]:
```
# basic scatter plot of patrons with physical-item ciculations using longitude,
 latitude...
plot = figure(plot_width=600, plot_height=400)
plot.scatter(patron_data.patron_longitude,
             patron_data.patron_latitude)
show(plot)
```



(ht

In [23]:

```
# basic scatter plot of patrons with NO physical-item ciculations using longitud
e, latitude...
plot = figure(plot_width=600, plot_height=400)
plot.scatter(patron_data_active_no_circs.patron_longitude,
             patron_data_active_no_circs.patron_latitude)
show(plot)
```



(ht

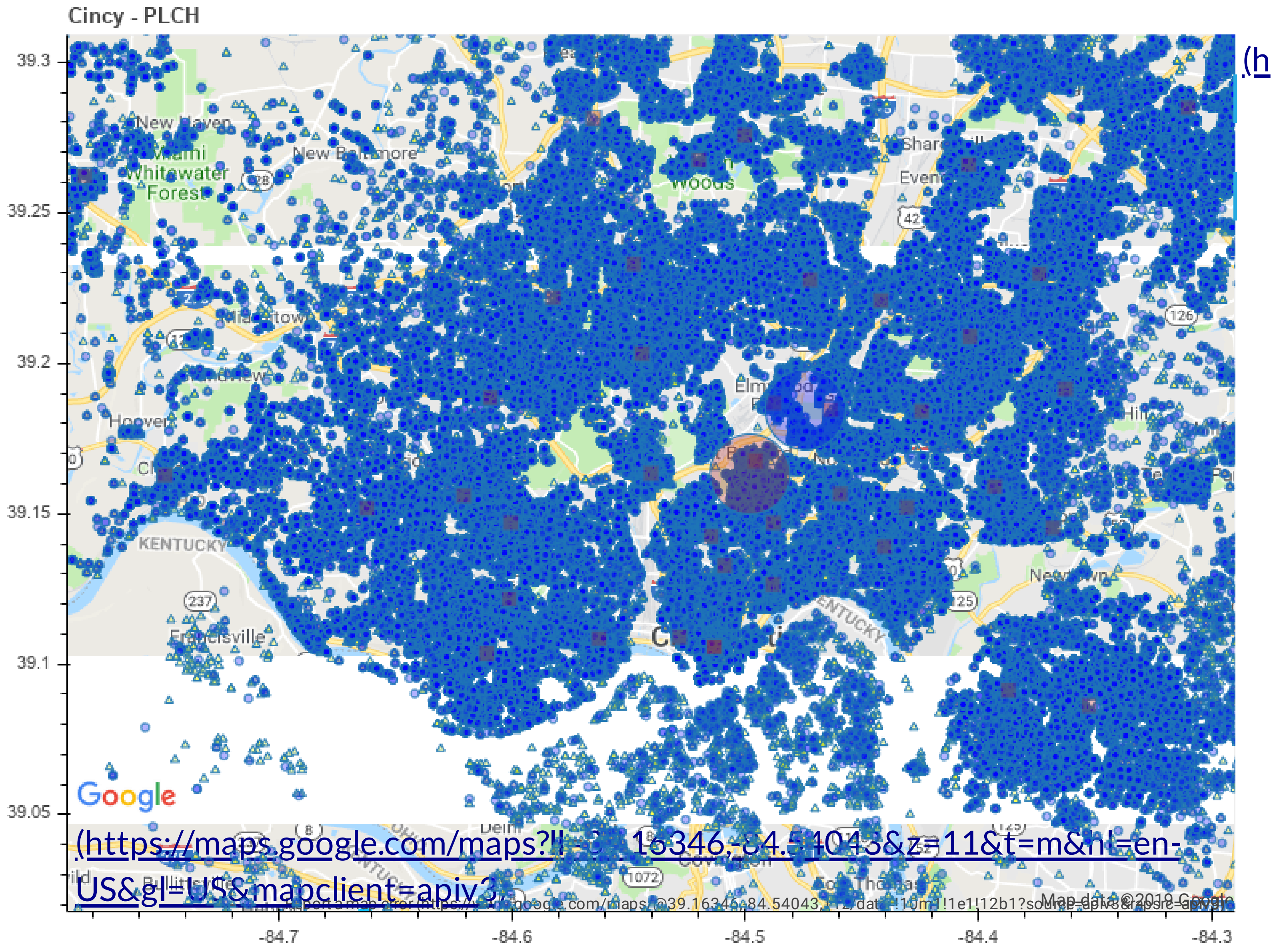
In [24]:

```
# comment / uncomment depending on if we want output to external file
#output_file("gmap.html")

map_options = GMapOptions(lat=39.16346, lng=-84.54043, map_type="roadmap", zoom=
11)

#           some other useful dimensions...
#           plot_width=1920,
#           plot_height=1080,
#            plot_width=1024,
#            plot_height=768,
#           plot_width=3840,
#           plot_height=2160,

p = gmap(GOOGLE_API_KEY,
         map_options,
         title="Cincy - PLCH",
         plot_width=800,
         plot_height=600,
         tools="wheel_zoom, reset, pan, save, box_zoom",
         active_drag="pan",
         active_scroll="wheel_zoom"
)

# plot the patrons with activity, but no circulations
# patron_data_active_no_circs
source = ColumnDataSource(
    data=dict(lat=patron_data_active_no_circs.patron_latitude,
              lon=patron_data_active_no_circs.patron_longitude)
)
p.triangle(x="lon", y="lat", size=5, fill_color="yellow",
           fill_alpha=0.3, source=source)

# plot the patrons with circulations
source = ColumnDataSource(
    data=dict(lat=patron_data.patron_latitude,
              lon=patron_data.patron_longitude)
```

```
)
p.circle(x="lon", y="lat", size=5, fill_color="blue",
         fill_alpha=0.3, source=source)

# plot the brances
source = ColumnDataSource(
    data=dict(lat=branches.branch_latitude,
              lon=branches.branch_longitude)
)

p.square(x="lon", y="lat", size=10, fill_color="firebrick",
         fill_alpha=0.3, source=source)

# plot the median location of the branches
source = ColumnDataSource(
    data=dict(lat=[branches.branch_latitude.median()],
              lon=[branches.branch_longitude.median()]
    )
)
p.circle(x="lon", y="lat", size=50, fill_color="firebrick",
         fill_alpha=0.3, source=source)


# plot the median location of the patrons
source = ColumnDataSource(
    data=dict(lat=[patron_data.patron_latitude.median()],
              lon=[patron_data.patron_longitude.median()]
    )
)
p.circle(x="lon", y="lat", size=50, fill_color="blue",
         fill_alpha=0.3, source=source)
```

Out[24]: **GlyphRenderer**(id = '1268', …)

In [25]:
```
show(p)
```


Cincy - PLCH
39.3
39.25
39.2
39.15
39.1
39.05
-84.7
-84.6
-84.5
-84.4
-84.3
New Haven
Miami Whitewater Forest
New Baltimore
Woods
Evendale
Elmwood
Hoover
KENTUCKY
Francisville
Newtown
Google
Map data ©2019 Google

Heatmaps

Seaborn has some good tools for creating heatmap visualizations, so lets explore some of those:

This first group of data is **juvinile books** checked out by **branch** and **iso week number**

In [26]:

```
# Previously I ran a query to export checkouts of juvinile items...
# including the location code(library branch), iso week number, and count checko
uts of that group:
!head -n 10 ./data/checkout_location_by_juv_itype_and_iso_week.csv
```

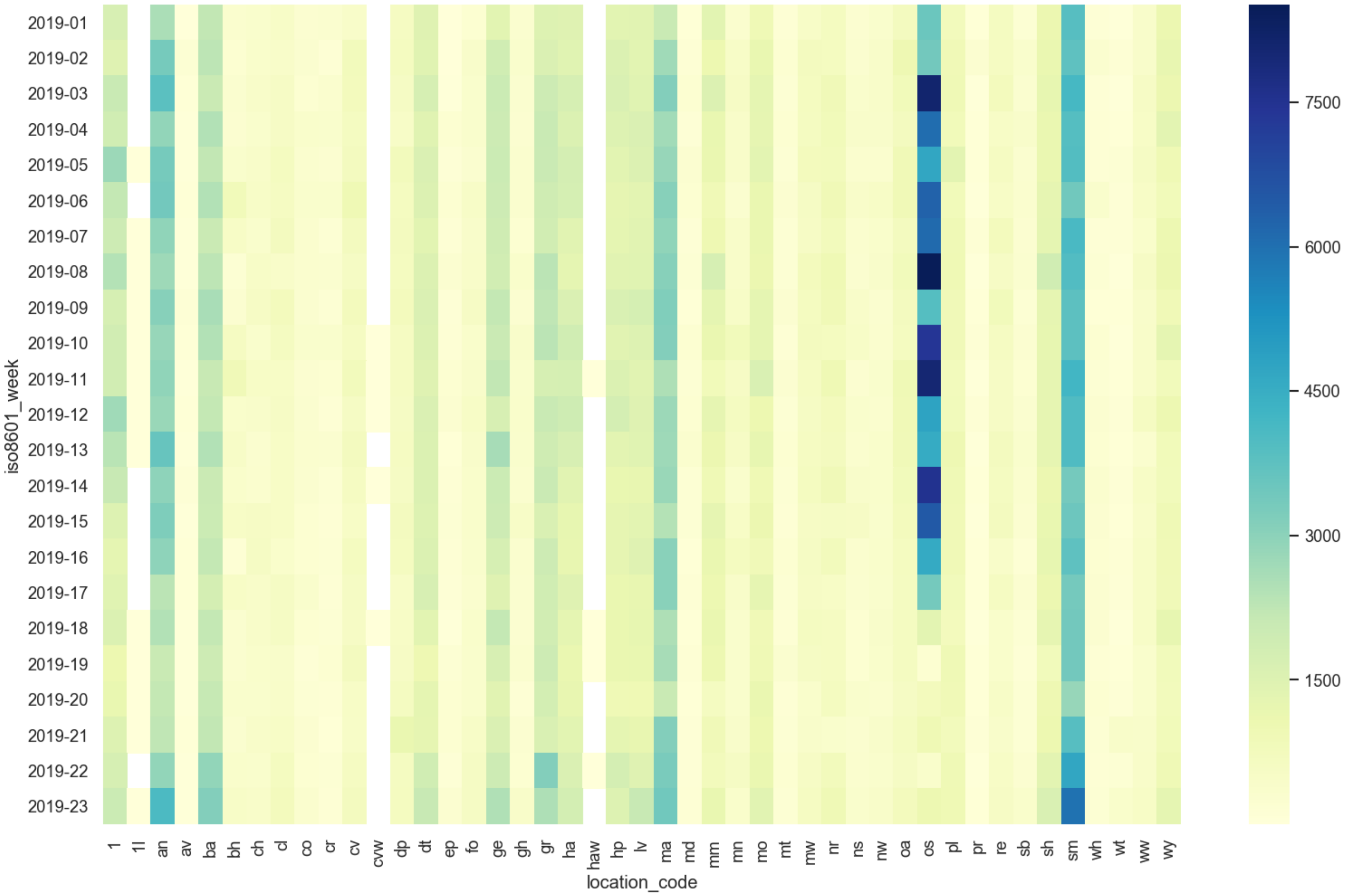
```
location_code,iso8601_week,checkouts
1,2019-01,1673
1,2019-02,1487
1,2019-03,2077
1,2019-04,1870
1,2019-05,2776
1,2019-06,2176
1,2019-07,1987
1,2019-08,2424
1,2019-09,1711
```

In [27]:

```
checkout_location_by_juv_itype_and_iso_week = pd.read_csv(
    './data/checkout_location_by_juv_itype_and_iso_week.csv'
)

# pivot the data
checkout_location_by_juv_itype_and_iso_week = checkout_location_by_juv_itype_and
_iso_week.pivot(
    'iso8601_week', 'location_code', 'checkouts'
)
```

In [28]:
```python
plt.figure(figsize = (16,10))
sns.heatmap(checkout_location_by_juv_itype_and_iso_week, cmap="YlGnBu")
```

Out[28]: <matplotlib.axes._subplots.AxesSubplot at 0x7f5dc5710b70>



Next, lets take a look at *all* the **item types** by **branch**

In [29]:

```
!head -n 10 ./data/itype_by_location.csv
```

```
item_type,location,checkouts
Book,1,129727
Book,1l,84
Book,an,96659
Book,av,2431
Book,ba,71794
Book,bh,6113
Book,ch,15394
Book,cl,28893
Book,co,14944
```

In [30]:

```
itype_by_locations = pd.read_csv(
    './data/itype_by_location.csv')

# pivot the data
itype_by_locations = itype_by_locations.pivot('item_type', 'location', 'checkout
s')
```

In [31]:
```
plt.figure(figsize = (30,15))
sns.heatmap(itype_by_locations, cmap="YlGnBu")
```

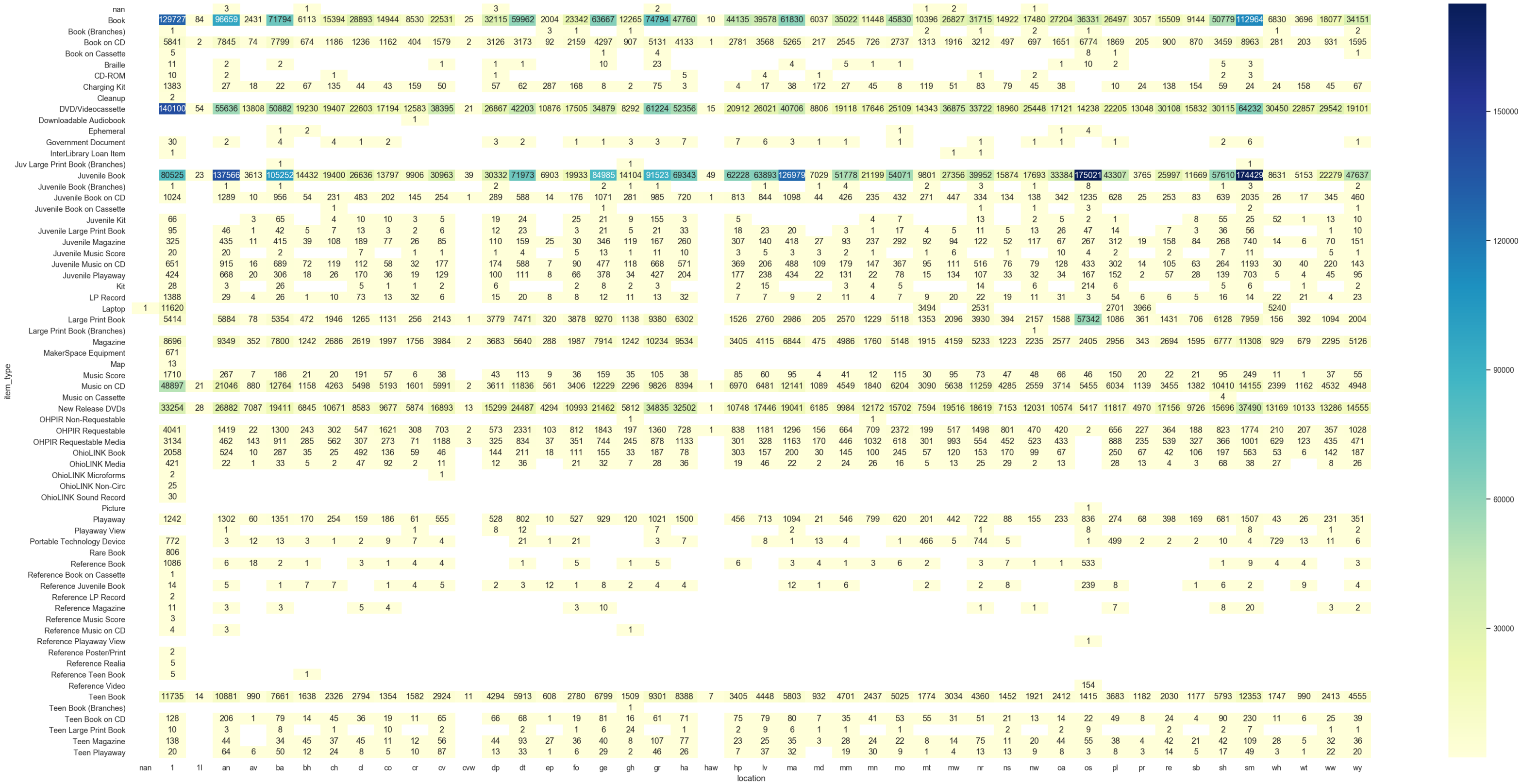
Out[31]: <matplotlib.axes._subplots.AxesSubplot at 0x7f5dc56fe748>

In [32]:

```python
# include some numbers in there, just for the heck of it
# width, height in inches
plt.figure(figsize = (40,20))
sns.heatmap(itype_by_locations, cmap="YlGnBu", annot=True, fmt='g')
```

Out[32]:

```
&lt;matplotlib.axes._subplots.AxesSubplot at 0x7f5dc4942b00&gt;
```



Examine source and destination for items delivered to branch pickup locations (hold shelf)

In [33]:

```
!head -n 5 ./data/source_destination_hold_shelf.csv
```

In [34]:

```
source_destination_hold_shelf = pd.read_csv(
    './data/source_destination_hold_shelf.csv')

# pivot the data
source_destination_hold_shelf = source_destination_hold_shelf.pivot(
    's_location_code', 'pickup_location_code', 'counted'
)
```

In [35]:
```
plt.figure(figsize = (30,15))
sns.heatmap(source_destination_hold_shelf, cmap="YlGnBu")
```

Out[35]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5dc53e78d0>
```