Day 7, Part 2 - Heatmap

One more fun info viz thing - we'll do a few things with `bqplot` which is the plotting library used by bloomberg.

Please note that this codebase very new (Bloomberg just released it) is constantly being updated so is less developed than `matplotlib` or others and therefore might be a little buggy.

In [100]:
```
# lets import our usual stuff
import pandas as pd
import numpy as np
import ipywidgets
%matplotlib inline
#%matplotlib notebook
```

Now we'll import bqplot. You'll have to install it with pip or conda if you don't have it installed already. You will probably have to restart the kernel and/or

In [101]:
```
# if you get a "No module named bqplot" -> install!

#!pip install bqplot
#!conda install -c conda-forge bqplot --yes # try first

# it is possible you'll need:
#!jupyter nbextension enable --py --sys-prefix bqplot
#!jupyter nbextension enable --py --sys-prefix widgetsnbextension
```

In [102]:
```
import bqplot
```

Test heatmap

Lets start thinking about heatmaps with some random data:

In [103]:
```
data = np.random.random((10, 10))
data
```

Out[103]:
```
array([[0.42385505, 0.60639321, 0.0191932 , 0.30157482, 0.66017354,
        0.29007761, 0.61801543, 0.4287687 , 0.13547406, 0.29828233],
       [0.56996491, 0.59087276, 0.57432525, 0.65320082, 0.65210327,
        0.43141844, 0.8965466 , 0.36756187, 0.43586493, 0.89192336],
       [0.80619399, 0.70388858, 0.10022689, 0.91948261, 0.7142413 ,
        0.99884701, 0.1494483 , 0.86812606, 0.16249293, 0.61555956],
       [0.12381998, 0.84800823, 0.80731896, 0.56910074, 0.4071833 ,
        0.069167  , 0.69742877, 0.45354268, 0.7220556 , 0.86638233],
       [0.97552151, 0.85580334, 0.01171408, 0.35997806, 0.72999056,
        0.17162968, 0.52103661, 0.05433799, 0.19999652, 0.01852179],
       [0.7936977 , 0.22392469, 0.34535168, 0.92808129, 0.7044144 ,
        0.03183893, 0.16469416, 0.6214784 , 0.57722859, 0.23789282],
       [0.934214  , 0.61396596, 0.5356328 , 0.58990998, 0.73012203,
        0.311945  , 0.39822106, 0.20984375, 0.18619301, 0.94437239],
       [0.7395508 , 0.49045881, 0.22741463, 0.25435648, 0.05802916,
        0.43441663, 0.31179588, 0.69634349, 0.37775184, 0.17960368],
       [0.02467873, 0.06724963, 0.67939277, 0.45369684, 0.53657921,
        0.89667129, 0.99033895, 0.21689698, 0.6630782 , 0.26332238],
       [0.020651  , 0.75837865, 0.32001715, 0.38346389, 0.58831711,
        0.83104846, 0.62898184, 0.87265066, 0.27354203, 0.79804683]])
```

In [104]:
```
# lets start by generating a quick heat map

# (1)
# create our first scale of our plot: just a color scale
col_sc = bqplot.ColorScale()
# now we'll use bqplot's gridheatmap function
#  with our randomly generated data & our scales to
#  make a heatmap like so:
heat_map = bqplot.GridHeatMap(color = data,
                              scales = {'color': col_sc})
# put our marks into our figure and lets go!
fig = bqplot.Figure(marks = [heat_map])

# (2) ok, this is fine and all, but lets add some reference for our
#  color scheme with a colorbar & also lets choose a different
#  color scheme
col_sc = bqplot.ColorScale(scheme = "Reds")
# lets plot some axes on our plot as well, in this case
#  our axis will be a color bar, vertically on the right
#  of our heatmap
c_ax = bqplot.ColorAxis(scale = col_sc,
                        orientation = 'vertical',
                        side = 'right')
# put it all together and lets take a look!
heat_map = bqplot.GridHeatMap(color = data,
                              scales = {'color': col_sc})
# generate fig!
fig = bqplot.Figure(marks = [heat_map], axes = [c_ax])

# (3) finally, lets add some axes labels on the x & y axis,
#  we need to add their scales first
# this scale will just count up the boxes in the vertical
#   & horizontal direction
x_sc = bqplot.OrdinalScale()
y_sc = bqplot.OrdinalScale()
# add our axes objects
x_ax = bqplot.Axis(scale = x_sc)
y_ax = bqplot.Axis(scale = y_sc,
                   orientation = 'vertical')
heat_map = bqplot.GridHeatMap(color = data,
                              scales = {'color': col_sc,
                                        'row': y_sc,
                                        'column':x_sc})
fig = bqplot.Figure(marks = [heat_map],
                    axes = [c_ax, y_ax, x_ax])


fig
```

Note: if no figure shows, try restarting the kernel and/or refreshing the page

So, while this indeed a lovely heatmap, it isn't interactive in any way! boo to that!

In [105]:
```
# keep data from last time
#import IPython
#IPython.OutputArea.auto_scroll_threshold = 9999;
from IPython import display

# now add scales - colors, x & y
col_sc = bqplot.ColorScale(scheme = "Reds")
x_sc = bqplot.OrdinalScale()
y_sc = bqplot.OrdinalScale()

# create axis - for colors, x & y
c_ax = bqplot.ColorAxis(scale = col_sc,
                        orientation = 'vertical',
                        side = 'right')
x_ax = bqplot.Axis(scale = x_sc)
y_ax = bqplot.Axis(scale = y_sc,
                   orientation = 'vertical')

# lets now re-do our heat map & add in some interactivity:
heat_map = bqplot.GridHeatMap(color = data,
                              scales = {'color': col_sc,
                                        'row': y_sc,
                                        'column': x_sc},
                              interactions = {'click': 'select'},
                             anchor_style={"fill":"blue"})
#NOTE: anchor_style seems not to do anything now...

# stir and combine into 1 figure
fig = bqplot.Figure(marks = [heat_map],
                    axes = [c_ax, y_ax, x_ax])

#display(fig)
#ipywidgets.HBox([fig])
fig
```

In [106]:
```
#  Ok fine, but our selection isn't linked to anything!
#  lets check out what heat_map selected is
heat_map.selected
#  note if I select a different box & re-run this cell,
#  I get out different values
```

Out[106]:
```
[]
```

In [107]:
```
# so now, lets write a little function that links the data value
#  to the selected & lets print this in a little ipywidgets label
mySelectedLabel = ipywidgets.Label()

# (1)
# lets write our linking function
# there are a few ways to link this,
#  here is a simple way first
def get_data_value(change):
    i,j = heat_map.selected[0]
    v = data[i,j] # grab data value
    mySelectedLabel.value = str(v) # set our label

# (2) this is maybe in-elegant as we are
#  explicitly calling our origininal heat map!
#  so, lets instead remind ourselves what "change" is here
def get_data_value(change):
    print(change)
    i,j = heat_map.selected[0]
    v = data[i,j] # grab data value
    mySelectedLabel.value = str(v) # set our label
# now we see when we click we get back a whole
#  dictionary of information - if we recall,
#  "owner" here is our heat_map which "owns"
# this change.
#  If we want to be able to apply our function to
#  this or any other heatmap figure we generate,
#  we can re-write the above function as follows:

# (3)
#def get_data_value(change,mylab):
def get_data_value(change):
    #print(change['owner'].selected)
    i,j = change['owner'].selected[0]
    v = data[i,j] # grab data value
    mySelectedLabel.value = str(v) # set our label
    #mylab.value = str(v) # set our label
# so, this now is applied to any map that we choose to input

# regenerate our heatmap to use in our fig canvas
heat_map = bqplot.GridHeatMap(color = data,
                              scales = {'color': col_sc,
                                        'row': y_sc,
                                        'column': x_sc},
                              interactions = {'click': 'select'},
                              anchor_style = {'fill':'blue'},
                              selected_style = {'opacity': 1.0},
                              unselected_style = {'opacity': 0.8})

# make sure we check out
heat_map.observe(get_data_value, 'selected')
#heat_map.observe(self, mySelectedLabel)
fig = bqplot.Figure(marks = [heat_map],
                    axes = [c_ax, y_ax, x_ax])

ipywidgets.VBox([mySelectedLabel, fig])
```

Dashboarding: Heatmap + plot

We'll now combine this idea with kepler data. We'll replace the randomly selected data with 2D bins of radius and eccentricity from the Kepler data.

First, let's make a 2D histogram of the radius and eccentricity of Kepler data. To do that, we first have to read the data set in!

In [108]:
```
# now let's read in the kepler confirmed planets dataset
planets = pd.read_csv('https://jnaiman.github.io/csci-p-14110/lesson06/data/planets_2019.07.12_17.16.25.csv',
                      sep=",", comment="#")
```

Let's remind ourselves of the column names. Last time we used `columns`, now we'll use `keys` but they do the same thing:

In [109]:
```
planets.keys()
```

Out[109]:
```
Index(['pl_hostname', 'pl_letter', 'pl_name', 'pl_discmethod',
       'pl_controvflag', 'pl_pnum', 'pl_orbper', 'pl_orbpererr1',
       'pl_orbpererr2', 'pl_orbperlim', 'pl_orbsmax', 'pl_orbsmaxerr1',
       'pl_orbsmaxerr2', 'pl_orbsmaxlim', 'pl_orbeccen', 'pl_orbeccenerr1',
       'pl_orbeccenerr2', 'pl_orbeccenlim', 'pl_orbincl', 'pl_orbinclerr1',
       'pl_orbinclerr2', 'pl_orbincllim', 'pl_bmassj', 'pl_bmassjerr1',
       'pl_bmassjerr2', 'pl_bmassjlim', 'pl_bmassprov', 'pl_radj',
       'pl_radjerr1', 'pl_radjerr2', 'pl_radjlim', 'pl_dens', 'pl_denserr1',
       'pl_denserr2', 'pl_denslim', 'ra_str', 'ra', 'dec_str', 'dec',
       'st_dist', 'st_disterr1', 'st_disterr2', 'st_distlim', 'gaia_dist',
       'gaia_disterr1', 'gaia_disterr2', 'gaia_distlim', 'st_optmag',
       'st_optmagerr', 'st_optmaglim', 'st_optband', 'gaia_gmag',
       'gaia_gmagerr', 'gaia_gmaglim', 'st_teff', 'st_tefferr1', 'st_tefferr2',
       'st_tefflim', 'st_mass', 'st_masserr1', 'st_masserr2', 'st_masslim',
       'st_rad', 'st_raderr1', 'st_raderr2', 'st_radlim', 'pl_massj',
       'pl_massjerr1', 'pl_massjerr2', 'pl_massjlim', 'pl_rade', 'pl_radeerr1',
       'pl_radeerr2', 'pl_radelim', 'pl_disc', 'pl_mnum', 'st_sp', 'st_spstr',
       'st_sperr', 'st_splim', 'st_lum', 'st_lumerr1', 'st_lumerr2',
       'st_lumlim', 'st_age', 'st_ageerr1', 'st_ageerr2', 'st_agelim'],
      dtype='object')
```

We want to make a 2D histogram of radii and eccentricities so:

In [110]:
```
# pl_orbsmax is in AU, semi-major axis
planets[['pl_orbeccen','pl_orbsmax']]
```

Out[110]:

	pl_orbeccen	pl_orbsmax
0	0.2310	1.290000
1	0.0800	1.530000
2	0.0000	0.830000
3	0.3700	2.930000
4	0.6800	1.660000
5	0.0800	2.600000
6	NaN	330.000000
7	0.0420	0.190000
8	0.0900	1.333000
9	0.2900	2.080000
10	NaN	52.000000
11	NaN	156.000000
12	NaN	15.000000
13	NaN	46.000000
14	NaN	0.920000
15	0.2600	NaN
16	NaN	230.000000
17	0.2900	0.990000
18	0.4320	0.870000
19	0.3800	1.190000
20	0.0320	2.100000
21	0.0980	3.600000
22	0.1600	11.600000
23	0.4500	12.000000
24	0.0130	0.052700
25	0.0034	0.115227
26	0.0200	0.241376
27	0.0190	5.503000
28	NaN	0.015440
29	0.3050	0.788000
...	...	...
3986	0.1100	1.300000
3987	0.7020	3.390000
3988	0.1510	1.930000

	pl_orbeccen	pl_orbsmax
3989	0.1440	1.190000
3990	0.0490	2.050000
3991	0.2100	1.240000
3992	0.0570	2.170000
3993	0.7124	1.275000
3994	NaN	55.000000
3995	0.1670	2.650000
3996	0.0900	1.100000
3997	0.1256	1.900000
3998	0.1650	6.100000
3999	0.1060	1.100000
4000	0.1910	0.830000
4001	0.1300	3.900000
4002	0.0000	0.068390
4003	0.4000	4.430000
4004	0.0373	0.219600
4005	0.0520	0.412300
4006	0.0110	0.049000
4007	0.1800	0.538000
4008	0.1600	1.334000
4009	0.0600	0.133000
4010	0.2300	0.243000
4011	0.0310	1.170000
4012	0.0215	0.059222
4013	0.2596	0.827774
4014	0.2987	2.513290
4015	0.0000	0.680000

4016 rows × 2 columns

We can use numpy to make our histogram (or pandas). Note we have some missing numbers indicated by an NaN. We only want to select entries that have both eccentricity and semi-major axis so we need to do some data cleaning:

In [111]:
```
x = planets['pl_orbeccen']
y = planets['pl_orbsmax']

# only entries for no NaNs:
xplot = x[~np.isnan(x) & ~np.isnan(y)]
yplot = y[~np.isnan(x) & ~np.isnan(y)]
```

In [112]:
```
myHist, xedges, yedges = np.histogram2d(xplot, yplot,
                                        bins=[10,10])


myHist
```

Out[112]:
```
array([[521.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.],
       [244.,   1.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.],
       [150.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.],
       [ 86.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.],
       [ 54.,   1.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.],
       [ 45.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.],
       [ 23.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.],
       [ 19.,   1.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.],
       [ 16.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.],
       [  5.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   1.]])
```

In [113]:
```
mySelectedLabel = ipywidgets.Label()

col_sc = bqplot.ColorScale(scheme = "Reds")
x_sc = bqplot.LinearScale()
y_sc = bqplot.LinearScale()

# create axis - for colors, x & y
c_ax = bqplot.ColorAxis(scale = col_sc,
                        orientation = 'vertical',
                        side = 'right')
x_ax = bqplot.Axis(scale = x_sc)
y_ax = bqplot.Axis(scale = y_sc,
                   orientation = 'vertical')

def get_data_value(change):
    i,j = change['owner'].selected[0]
    v = myHist[i,j] # grab data value
    mySelectedLabel.value = str(v) # set our label

# regenerate our heatmap to use in our fig canvas
heat_map = bqplot.GridHeatMap(color = myHist,
                              scales = {'color': col_sc,
                                        'row': y_sc,
                                        'column': x_sc},
                              interactions = {'click': 'select'},
                              anchor_style = {'fill':'blue'},
                              selected_style = {'opacity': 1.0},
                              unselected_style = {'opacity': 0.8})

# make sure we check out
heat_map.observe(get_data_value, 'selected')
fig = bqplot.Figure(marks = [heat_map],
                    axes = [c_ax, y_ax, x_ax])

ipywidgets.VBox([mySelectedLabel, fig])
```

Now we can use the x&y edges output from our histogram and re-put them into bin centers with some fancy in-line programming:

In [114]:
```
# this is becuase the edges are bin edges, not centers
y_centers = [(yedges[i]+yedges[i+1])*0.5 for i in range(len(yedges)-1)]
x_centers = [(xedges[i]+xedges[i+1])*0.5 for i in range(len(xedges)-1)]
```

In [115]:
```
mySelectedLabel = ipywidgets.Label()

col_sc = bqplot.ColorScale(scheme = "Reds")
x_sc = bqplot.LinearScale()
y_sc = bqplot.LinearScale()

# create axis - for colors, x & y
c_ax = bqplot.ColorAxis(scale = col_sc,
                        orientation = 'vertical',
                        side = 'right')
x_ax = bqplot.Axis(scale = x_sc,
                   label='Semi-major axis in AU')
y_ax = bqplot.Axis(scale = y_sc,
                    orientation = 'vertical',
                   label='Eccentricity')

def get_data_value(change):
    i,j = change['owner'].selected[0]
    v = myHist[i,j] # grab data value
    mySelectedLabel.value = str(v) # set our label

# regenerate our heatmap to use in our fig canvas
heat_map = bqplot.GridHeatMap(color = myHist,
                              row=x_centers,
                              column=y_centers,
                              scales = {'color': col_sc,
                                        'row': y_sc,
                                        'column': x_sc},
                              interactions = {'click': 'select'},
                              anchor_style = {'fill':'blue', 'stroke':
'blue'},
                             opacity=0.5)

# make sure we check out
heat_map.observe(get_data_value, 'selected')
fig = bqplot.Figure(marks = [heat_map],
                    axes = [c_ax, y_ax, x_ax])

ipywidgets.VBox([mySelectedLabel, fig])
```

Exercise

Clearly, most of the interesting stuff is happening at the 20 AU, low eccentricity point on this heatmap.

How can you "zoom in" and only plot this region?

Hint: there are several ways, one is by changing the bins for the 2D histogram (look at the documentation for histogram2d!)

Heatmap + Trajectory = Dashboard

Let's now combine this heatmap with our trajectory plot before to make an interactive dashboard for our data. In this case, we'll copy exactly what we had before, but will add an extra `bqplot.Lines` plot that will *also* be updated when a selection on the heatmap is made.

In [116]:
```
theta = np.arange(0, 2*np.pi, 0.001)

# Line's plot
x_l_sc = bqplot.LinearScale()
y_l_sc = bqplot.LinearScale()

line_plot = bqplot.Lines(x=[], y=[], # start empty
                         scales = {'x': y_l_sc,
                                   'y': x_l_sc})

x_l_ax = bqplot.Axis(scale = x_l_sc,
                     label='x in AU')
y_l_ax = bqplot.Axis(scale = y_l_sc,
                     orientation = 'vertical',
                     label='y in AU')
fig_lines = bqplot.Figure(marks = [line_plot],
                          axes = [y_l_ax, x_l_ax])

# quick check it out
#fig_lines
```

In [117]:
```
mySelectedLabel = ipywidgets.Label()

col_sc = bqplot.ColorScale(scheme = "Reds")
x_sc = bqplot.LinearScale()
y_sc = bqplot.LinearScale()

# create axis - for colors, x & y
c_ax = bqplot.ColorAxis(scale = col_sc,
                        orientation = 'vertical',
                        side = 'right')
x_ax = bqplot.Axis(scale = x_sc,
                   label='Semi-major axis in AU')
y_ax = bqplot.Axis(scale = y_sc,
                    orientation = 'vertical',
                   label='Eccentricity')

def get_data_value(change):
    i,j = change['owner'].selected[0]
    v = myHist[i,j] # grab data value
    mySelectedLabel.value = str(v) # set our label
    # NOW ALSO update line data
    semiMaj = y_centers[j]
    ecc = x_centers[i]
    #print(ecc)
    # from our plot_ellipse function, with a not b
    b = semiMaj*ecc
    r = b/np.sqrt(1-ecc*(np.cos(theta))**2)
    x = r*np.cos(theta)
    y = r*np.sin(theta)
    line_plot.x = x
    line_plot.y = y

# regenerate our heatmap to use in our fig canvas
heat_map = bqplot.GridHeatMap(color = myHist,
                              row=x_centers,
                              column=y_centers,
                              scales = {'color': col_sc,
                                        'row': y_sc,
                                        'column': x_sc},
                              interactions = {'click': 'select'},
                              anchor_style = {'fill':'blue'}, # note: th
ese below may not work
                              selected_style = {'opacity': 1.0, 'color':
'blue'},
                              unselected_style = {'opacity': 0.1})

# make sure we check out
heat_map.observe(get_data_value, 'selected')
fig = bqplot.Figure(marks = [heat_map],
                    axes = [c_ax, y_ax, x_ax])

ipywidgets.VBox([mySelectedLabel, ipywidgets.HBox([fig,fig_lines])])
```

Exercise

How can you make this have the same x & y coords for all plots? Hint: look up "bqplot LinearScale" to see if there may be helpful parameters to use. Alternatively you can try `bqplot.LinearScale?` in the notebook.

Can you plot more than one system for comparison? Note: more than one square is selected when you `SHIFT-CLICK` .

Extensions:

Remake this dashboard with another set of Kepler data parameters.

Allow for a selection to run a simulation and plot both analytical and numeric solutions.

How can you plot a multi-body system using interactivity with the Kepler dataset? What assumptions do you need to make? Try just analytical solutions and then later add numerical.

In [ ]: