Data Visualisation With Bokeh

Last week we looked at how statistics could inform our understanding of data. In this week, we look at how data can be visualised, in particular using the **Bokeh (http://bokeh.pydata.org/en/latest/)** library.

We used some of the basic **`Chart` (http://bokeh.pydata.org/en/latest/docs/reference/charts.html)** functions last week to visualise distributions. This week will focus on geographical data and how this can be represented on a map.

In [1]:

```
# You don't need to write anything here
# Add the initial imports
import pandas as pd
from bokeh.plotting import Figure
from bokeh.io import show, output_notebook, push_notebook
from bokeh.models import *
#from bokeh.plotting import figure
from bokeh.tile_providers import WMTSTileSource
from ipywidgets import *
import ipywidgets as widgets


import numpy as np
output_notebook()

# We will need this function to calculate a position on the map when given lattitude
def wgs84_to_web_mercator(df, lon="lon", lat="lat"):
    #위치정보를 위해 코드 변경을 위해
    """
    Converts decimal longitude/latitude to Web Mercator format
    Source https://github.com/bokeh/bokeh-notebooks/blob/master/tutorial/11%20-%20ge
    """
    k = 6378137
    df["x"] = df[lon] * (k * np.pi/180.0)
    df["y"] = np.log(np.tan((90 + df[lat]) * np.pi/360.0)) * k
    return df
```

(http://bokeh.pydata.org) BokehJS successfully loaded.

Widgets

Widgets are functions which convert **Python** code to **HTML** code for output in the notebook. Bokeh has its own widgets, but for this exercise we will focus on the widgets used within the Jupyter Notebook:**`ipywidgets` (http://ipywidgets.readthedocs.io/en/latest/index.html)**. These widgets could range from a simple text description to interactive widgets which may modify the appearance of Bokeh visualisations in real time.

We will start with a **'slider' (http://bokeh.pydata.org/en/latest/docs/gallery/slider.html)** widget, which we will use to modify a simple line graph. We will use the **`interact` (http://ipywidgets.readthedocs.io/en/latest/examples/Using%20Interact.html)** function to do this. It works by passing a function name as a parameter, and every time that the slider is moved, it calls the function. We demonstrate this in the following cell:

Typesetting math: 100%

In [2]:

```
# You don't need to write anything here

# This function is called every time we change the value on the slider
# Notice that we are calling the function 'interact' which creates an interactive w
# 'interact' is a function called from ipywidgets
def f(x):
    print("Move the slider!", x)
    return x
# x=2 means that the first time it calls the function, the `x` parameter of `f` wil
interact(f, x=3)
```

```
Move the slider! 3

3
```

Out[2]:

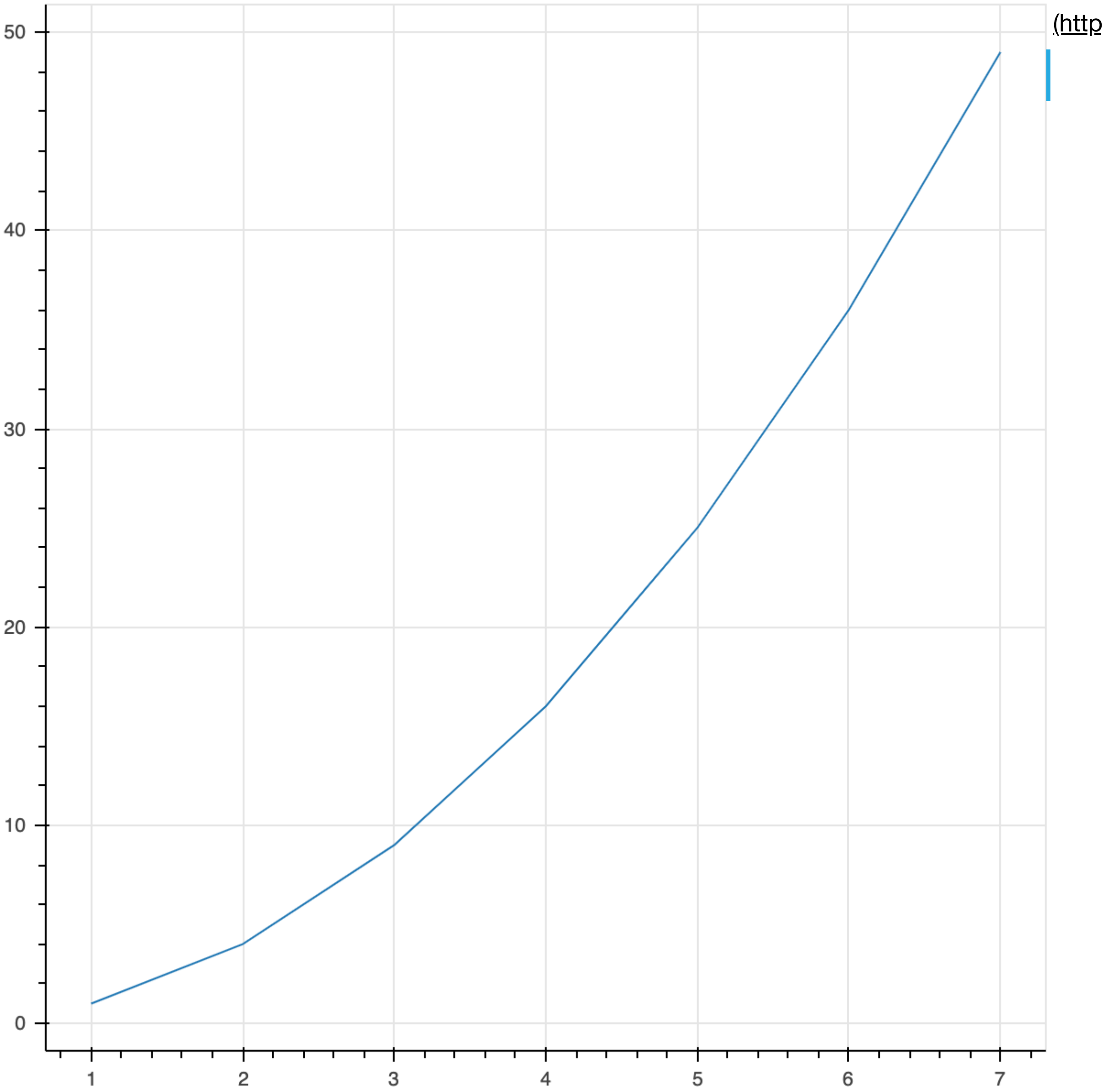
```
<function __main__.f>
```

In the next cell, we introduce the **`Figure` class (http://bokeh.pydata.org/en/latest/docs/reference/plotting.html#bokeh.plotting.figure.Figure)** to create our line graph. We briefly demonstrated this in **week 3** when demonstrating the Iris dataset for linear regression. For this week, we will go into a bit more about it, since we will be using it in our visualisation in the assignment.

We use the **`line` method (http://bokeh.pydata.org/en/0.10.0/docs/reference/plotting.html#bokeh.plotting.Figure.line)** to add our line to the graph. On this occasion, we are presenting a simple graph of $y = x^2$, and use the **`show`** function to display the graph. Notice that we pass it **`notebook_handle=true` (http://bokeh.pydata.org/en/latest/docs/user_guide/notebook.html#working-in-the-notebook)**. This gives other methods a means of dynamically updating it later.

In [4]:

```
# You don't need to write anything here
# Here we create an empty figure to which we add a line representing y=x**2

fig = Figure()
#피규어 함수
li = fig.line(x=pd.Series([1,2,3,4,5,6,7]), y=pd.Series([1,4,9,16,25,36,49]))
#line sub 함수나 sub 라이브러리를 사용 중임 시리즈 형태여야됨
show(fig, notebook_handle=True)
```



Out[4]:

```
<Bokeh Notebook handle for In[4]>
```

If we want to **update** the figure, we can make use of the **`handle`** **(http://bokeh.pydata.org/en/latest/docs/user_guide/notebook.html#notebook-handles)** we passed to the **`Figure`**. Bokeh has a function for this called **`push_notebook`** **(http://bokeh.pydata.org/en/latest/docs/reference/io.html)**, which will ***push*** any changes made to the notebook since the last time it was called.

Typesetting math: 100%

To update the data, we are going to update the data source of the line, by changing the values of the $y$ axis, so we have an equation of $y = x^3$.

So, if we call as follows, it will update the figure above:

In [5]:

```
# You don't need to write anything here
# Now, we can adjust the value in the line figure above, and see the data change
li.data_source.data['y'] = li.data_source.data['x'] **3
push_notebook()
#푸쉬 놋북 함수를 통하여 수정해주기 !
```

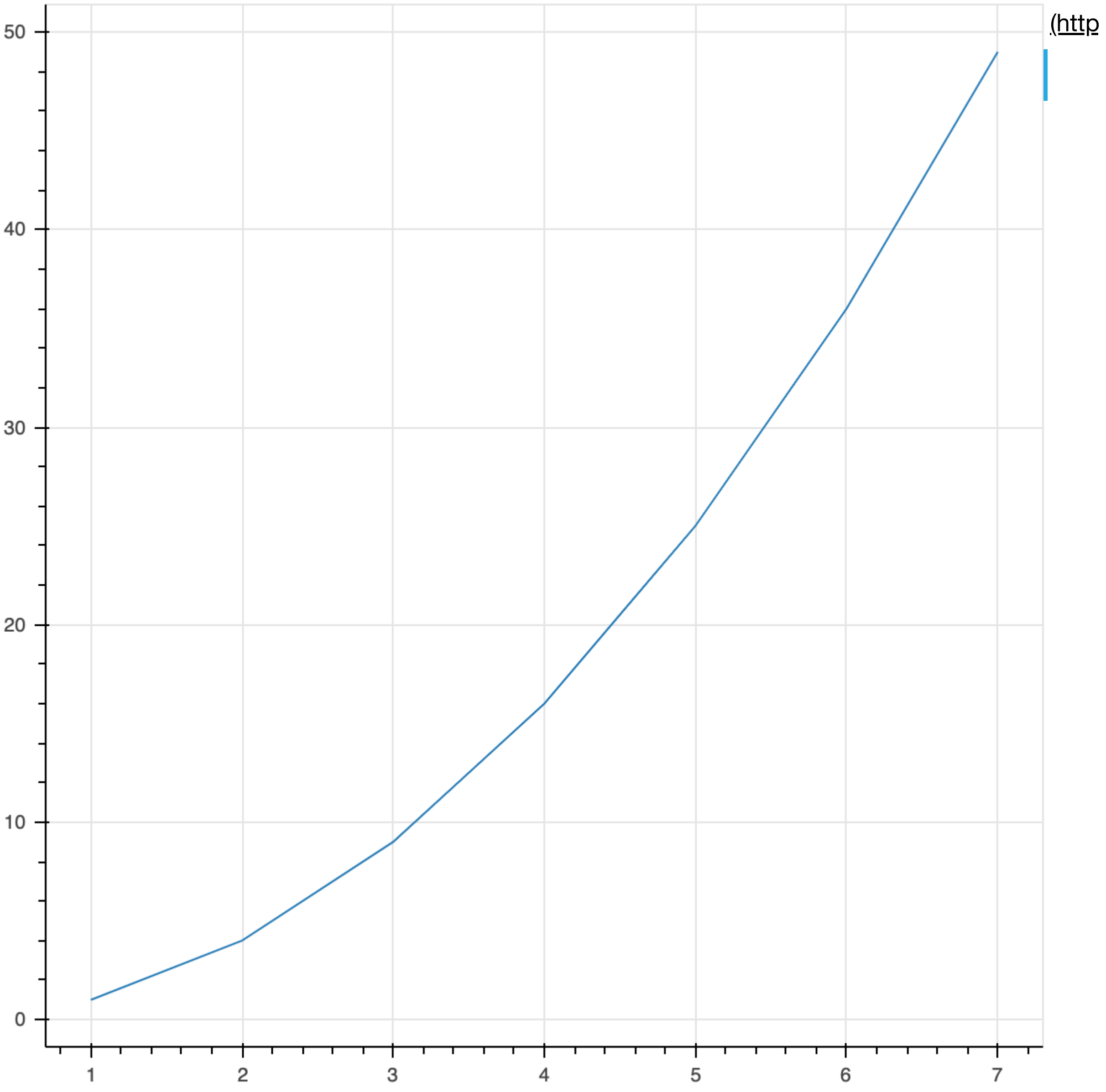
That shows the two elements required to make an interactive visualisation. Now, try and put it all together:

- Create a slider using `interact`, which calls function `g(x)`
- `g(x)` should update the line of a plot to be x to the power `g(x)`
- If you can make a slider work with integers, try and update using a `FloatSlider` (http://ipywidgets.readthedocs.io/en/latest/examples/Using%20Interact.html) which increases in value 0.1 for every change instead of an `IntSlider` as in the example above.

Typesetting math: 100%

In [59]:

```
# YOUR CODE HERE
# Place your figure code in this cell
fig = Figure()
li = fig.line(x=pd.Series([1,2,3,4,5,6,7]), y=pd.Series([1,4,9,16,25,36,49]))
#x가지고 y도 가지기, 구획하기 위해선 ys가 필요
show(fig, notebook_handle=True)
#데이터나 정보를 업데이트 할 수 있도록 놋북핸들트루!
```



Out[59]:

```
<Bokeh Notebook handle for In[59]>
```

Typesetting math: 100%

In [60]:

```
# YOUR CODE HERE
# Place your slider code in this cell
def g(x):
    li.data_source.data['y'] = li.data_source.data['x'] **x
    #우리는 데이터소스 y를 가지고 있고, 이는 x의 power x를 말하고 있다.
    push_notebook() #정보를 노트에 넣고
interact(g, x=3)#괄호가 있는 함수인 상호 작용을 사용 할 것임.
#우리는 g(x)함수를 정의하고 있기 때문에 g를 사용하고, x벨류를 3으로 설정
#--> 슬라이더가 나온다.
```

Out[60]:

```
<function __main__.g>
```

Tools

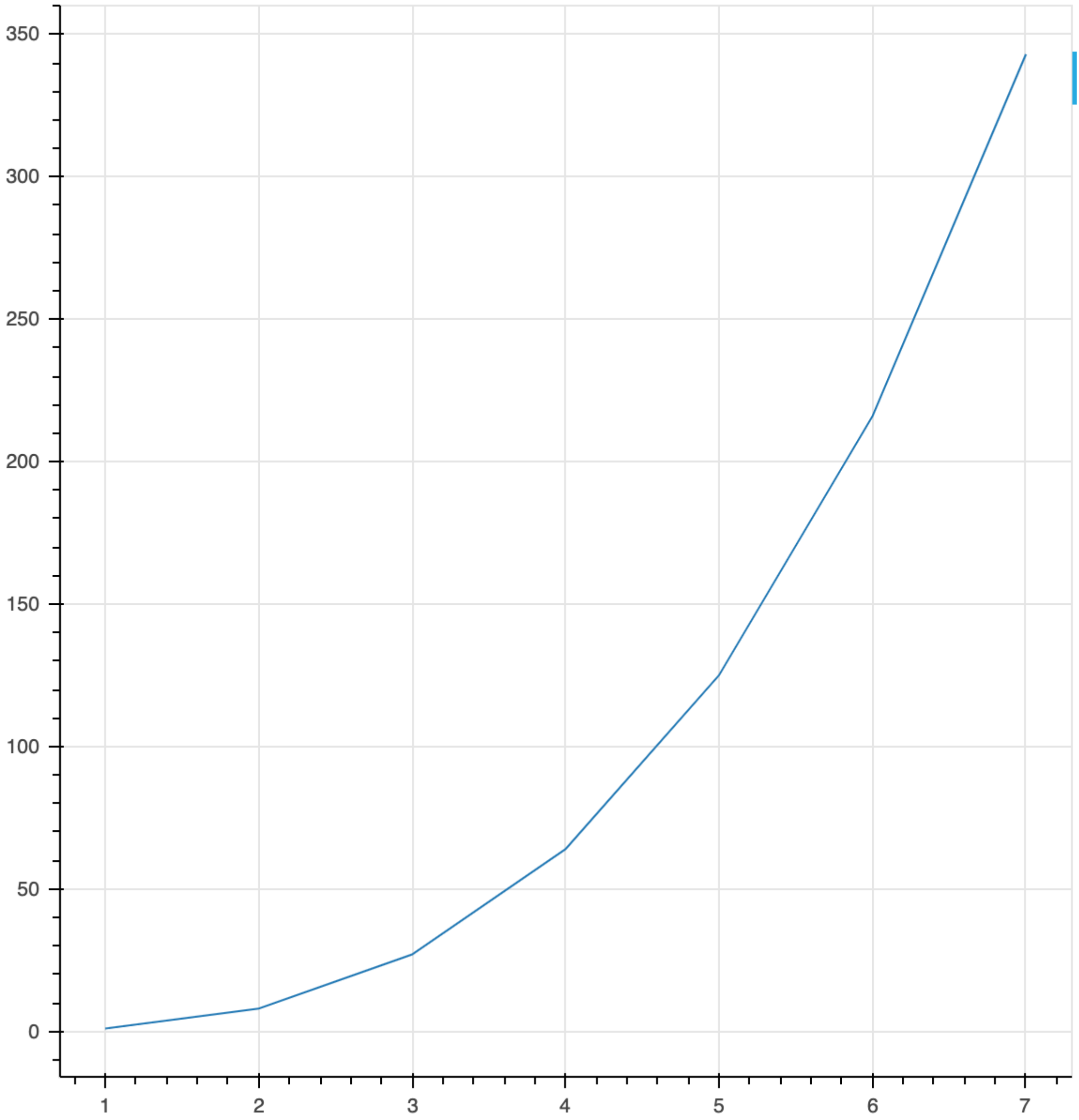
Notice on the side of the `Figure` object, there are a series of buttons. These allow interactive exploration of the figure. By default, Bokeh adds some to our figure, but we can add them ourselves in two ways.

Firstly, we can use the **`add_tool`** **(http://bokeh.pydata.org/en/latest/docs/user_guide/tools.html#specifying-tools)** function for each tool manually. For example, we might want to add the **`LassoSelectTool`**, we could simply pass an instance of this class to the **`add_tool`** function as follows:

Typesetting math: 100%

In [61]:

```
# fig = Figure()
fig.add_tools(LassoSelectTool())
#add_tool 함수를 이용하여 올가미 도구를 추가하여 사용
show(fig)
```



(http

The **easier way** of doing this is to pass to the tools parameter when the `Figure` is created, which can be done in one of two ways:

- By passing a list of `Tool` instances
- A comma separated string of different tools

See the two examples of code doing the same thing on empty `Figure` instances as follows:

Typesetting math: 100%

In [62]:

```
tools_fig_1 = Figure(tools=[WheelZoomTool(), BoxSelectTool()], height=100)
show(tools_fig_1)
```

(http

In [63]:

```
tools_fig_2 = Figure(tools='wheel_zoom,box_select', height=100)
                                            #쉼표로 구분 가능
show(tools_fig_2)
```

(http

The full list of **available tools (http://bokeh.pydata.org/en/latest/docs/user_guide/tools.html#specifying-tools)** and their usage can be seen in the **Bokeh tools user guide (http://bokeh.pydata.org/en/latest/docs/user_guide/tools.html)**. If you don't remember the name of the tool you want and enter the wrong value, Bokeh will warn you and give you some suggestions for tools you might like to add.

Data Sources

This (updating the graph/data) was possible, because when we created the line to display on our graph, we gave it the variable `li`, and we were still able to access the `li` variable. This was a `Glyph` (http://bokeh.pydata.org/en/latest/docs/user_guide/plotting.html#plotting-with-basic-glyphs), which has associated with it a `ColumnDataSource` (http://bokeh.pydata.org/en/latest/docs/reference/models/sources.html#bokeh.models.sources.ColumnDataSou type created from the values inserted.

We have already seen a Pandas `DataFrame`, which is a generic data structure for holding data. The `ColumnDataSource` is part of `Bokeh` rather than `Pandas`, and is used specifically as a means of storing data for a graph.

This object can be accessed as the `data_source` of the `Glyph`, and the `data` attribute is a series of key/value pairs derived from the source data.

Typesetting math: 100%

In [64]:

```
# You don't need to write anything here
# Show the variables associated with the data source of the `ColumnDataSource`
print(vars(li.data_source))
#y값이 무엇인지 물어보기
#결과값  : x값은 변경되지 않았지만 y값은 x값을 부여받았기 때문에 앞서 x^3를 부여받았당

# Show a column of the source data
print('\n`y` data from the graph:\n', li.data_source.data['y'])
#온리 y값만 출력하게 할 수 있다 (li의 데이터 소스 값만 출력하게 하는거지))
```

```
{'_id': '51bd8b19-be1e-4756-a7d3-ecd7175d7841', '_property_values':
{'callback': None, 'data': {'y': 0      1
1      8
2     27
3     64
4    125
5    216
6    343
dtype: int64, 'x': 0    1
1    2
2    3
3    4
4    5
5    6
6    7
dtype: int64}, 'js_callbacks': {}, 'column_names': ['y', 'x']}, '_docu
ment': <bokeh.document.Document object at 0x7ff8bfaa0978>, '_callback
s': {}}

`y` data from the graph:
 0      1
1      8
2     27
3     64
4    125
5    216
6    343
dtype: int64
```

Having introduced the concept of widgets, figures, and the ColumnDataSource, we are now going to make use of the **Bokeh map tiling (http://geo.holoviews.org/Working_with_Bokeh.html)** feature. We will use sample data from Bokeh, based on states in the USA, and we will match each of these states to the winner in the US presidential election of 2016.

We will create a map of the USA, which will add the correct colour to an individual state when we select from a checkbox.

To begin, we will import the data, and add colour for a single state (California):

In [8]:

```
#이 정보를 실제로 어떻게 표시할 수 있으며 어떻게 정보를 사용할 수 있는가 !

# You don't need to write anything here
# Here we import our data and make a copy
# These data includes the co-ordinates of the US state borders
from bokeh.sampledata.us_states import data
us_states = data
#가변적인 미국 언더 스코어 상태를 data라고 하는 이 정보 집합에 할당 !
# These data is the winners from the 2016 election by state
election_winners = pd.read_csv('election.csv')
#판다 리드 CSV를 사용하기
#--> 변수 설정 완료
print(data)
```

```
{'WY': {'lats': [42.0564, 42.10051, 42.14132, 42.19558, 42.20782, 42.2
8126, 42.3709, 42.45085, 42.66754, 42.92784, 42.98878, 43.02641, 43.08
959, 43.14791, 43.18826, 43.49991, 43.68192, 43.72274, 43.78102, 43.80
987, 43.91411, 44.00827, 44.12492, 44.26798, 44.46779, 44.47407, 44.47
555, 44.93488, 44.99233, 44.99994, 45.00276, 45.00389, 44.99958, 44.99
997, 45.00032, 45.00004, 45.00061, 45.00132, 45.00123, 45.00067, 44.99
626, 44.99565, 44.99428, 44.99358, 45.00034, 45.00026, 45.00039, 44.99
998, 44.99964, 44.99901, 44.99835, 44.99738, 44.99738, 44.99736, 44.89
596, 44.8008, 44.76792, 44.70237, 44.67308, 44.57694, 44.53558, 44.427
58, 44.33037, 44.2584, 44.18038, 44.11429, 44.02684, 43.94686, 43.8749
8, 43.75174, 43.64966, 43.5756, 43.50396, 43.42891, 43.35212, 43.2881
6, 43.2032, 43.12598, 43.00077, 43.00059, 42.99997, 42.82387, 42.6501
9, 42.61705, 42.47312, 42.26135, 42.19867, 42.14711, 42.12262, 42.0961
6, 42.02571, 41.99693, 41.97263, 41.929, 41.875, 41.84185, 41.80402, 4
1.7338, 41.63839, 41.54938, 41.49972, 41.44736, 41.41683, 41.2782, 41.
18064, 41.01681, 41.00141, 41.00139, 41.00162, 41.00153, 40.99826, 40.
9983, 40.99818, 40.99766, 40.99722, 40.99701, 40.997, 40.99927, 41.002
13, 41.00315, 41.00305, 41.00283, 41.00305, 41.00228, 41.00197, 41.002
05, 41.00139, 41.00011, 41.00008, 40.99996, 41.0001, 41.00013, 41.0005
1, 41.00066, 41.00069, 41.00064, 40.99964, 40.99871, 40.9983, 40.9984
```

In [9]:

```
# You don't need to write anything here
# Here we are preparing a map as a `Tile` which we will use as a background on the
from bokeh.plotting import figure
from bokeh.tile_providers import WMTSTileSource

# Create a figure which has co-ordinates centred on the USA
x_range,y_range = ((-13884029,-7453304), (2698291,6455972))
#미국 중심 좌표를 가진 그림 만들기, 그림을 만들고 싶다면 중심에 두기

# Create the figure
fig = figure(tools='pan, wheel_zoom', x_range=x_range, y_range=y_range)
fig.axis.visible = False
```

Typesetting math: 100%

In [10]:

```
# You don't need to write anything here
# In this cell we add a map tile to the figure, adding a URL in a standard format to
url = 'http://a.basemaps.cartocdn.com/dark_all/{Z}/{X}/{Y}.png'
#우리는 실제로 타일 소스와 관련하여 정보가 저장된 위치에 대한 URL을 제공하기
attribution = "Map tiles by Carto, under CC BY 3.0. Data by OpenStreetMap, under ODk
fig.add_tile(WMTSTileSource(url=url, attribution=attribution))
#에트리뷰션을 가지고 있음을 주목!
show(fig)
```

CANADA
UNITED STATES
MEXICO
CUBA
GUATEMALA
PANAMA
VENEZUELA
SURINAME
COLOMBIA
ECUADOR
PERU
BRAZIL
Pacific Ocean

(http

Map tiles by Carto, under CC BY 3.0. Data by OpenStreetMap, under ODbL

Typesetting math: 100%

In [11]:

```
# You don't need to write anything here
states = []
lons = []
lats = [] #리스트 작성하기
# The sample data is in a slightly difficult format, so we will change it to be in a
# We don't mind about the State being repeated, as long as we have all the latitudes
#     Lat       Lon         State
# 0 -82.88318   -82.88318   FL
# 1 -82.87484   -82.87484   FL
# 2 -82.86562   -82.86562   FL

for s in us_states: #US에서 시작
    # The amount of longitudes is the same as the latitudes, so this is safe
    # Iterate through each lat/lon pair

    for data in range(len(us_states[s]['lons'])):
        #범위 또는 길이는 우리가 여기에 있는 수 또는 길이를 기반으로 한다.
        states.append(s)
        lons.append(us_states[s]['lons'][data]) #경도 데이터 추가
        lats.append(us_states[s]['lats'][data]) #위도 데이터 추가


# We created 3 lists of equal length, now we create a
df = pd.DataFrame({'state': states, 'lat': lats, 'lon': lons})#dict(state=states, lo
print(df)
```

```
            lat        lon state
0      42.05640 -111.04694    WY
1      42.10051 -111.04691    WY
2      42.14132 -111.04711    WY
3      42.19558 -111.04711    WY
4      42.20782 -111.04732    WY
5      42.28126 -111.04718    WY
6      42.37090 -111.04713    WY
7      42.45085 -111.04694    WY
8      42.66754 -111.04444    WY
9      42.92784 -111.04399    WY
10     42.98878 -111.04407    WY
11     43.02641 -111.04406    WY
12     43.08959 -111.04401    WY
13     43.14791 -111.04397    WY
14     43.18826 -111.04418    WY
15     43.49991 -111.04549    WY
16     43.68192 -111.04607    WY
17     43.72274 -111.04639    WY
18     43.78102 -111.04659    WY
19     43.80987 -111.04663    WY
20     43.91411 -111.04654    WY
21     44.00827 -111.04806    WY
22     44.12492 -111.04912    WY
23     44.26798 -111.04960    WY
24     44.46779 -111.04898    WY
25     44.47407 -111.04897    WY
26     44.47555 -111.04897    WY
27     44.93488 -111.05553    WY
28     44.99233 -110.57457    WY
29     44.99994 -110.15603    WY
...         ...        ...   ...
```

Typesetting math: 100%

```
14178  33.01547  -92.81184    AR
14179  33.01662  -92.91975    AR
14180  33.01774  -93.10172    AR
14181  33.01813  -93.20515    AR
14182  33.01817  -93.31438    AR
14183  33.01832  -93.37442    AR
14184  33.01864  -93.47987    AR
14185  33.01928  -93.66171    AR
14186  33.01943  -93.84706    AR
14187  33.01947  -93.97693    AR
14188  33.01921  -94.04273    AR
14189  33.01922  -94.04296    AR
14190  33.13883  -94.04311    AR
14191  33.27941  -94.04299    AR
14192  33.35086  -94.04307    AR
14193  33.38629  -94.04313    AR
14194  33.40633  -94.04313    AR
14195  33.42372  -94.04309    AR
14196  33.43760  -94.04311    AR
14197  33.44967  -94.04312    AR
14198  33.46722  -94.04312    AR
14199  33.49066  -94.04311    AR
14200  33.55175  -94.04484    AR
14201  33.55404  -94.07140    AR
14202  33.58216  -94.07521    AR
14203  33.55937  -94.13238    AR
14204  33.59046  -94.17241    AR
14205  33.57438  -94.19515    AR
14206  33.55209  -94.23197    AR
14207  33.56763  -94.26926    AR

[14208 rows x 3 columns]
```

Now we have the data set up into two data frames: One with the lat/lng co-ordinates of the borders of states in the USA, the other with the winner of that state in the 2016 US Presidental election.

We have seen what the geographical data look like, now display the first few rows of data about the winners using the variable `election_winners`:

In [12]:

```
# YOUR CODE HERE
election_winners.head()
#선거 당첨자 중 처음 다섯 명, 선거 당첨자의 처음 다섯 행이 기능함
```

Out[12]:

	state	winner
0	AK	trump
1	MI	trump
2	ME	clinton
3	PA	trump
4	NY	clinton

Typesetting math: 100%

To work with the `tile` we used, we need to include `x` and `y` columns in Web Mercator format. In the cell below, modify the dataset as follows:

- Call the `wgs84_to_web_mercator` function to add extra columns `x` and `y` to the `DataFrame`

In [13]:

```
# YOUR CODE HERE
#타일로 작업 Mercator 방식으로 x 및 y열을 얻었기 때문에 이것을 위도와 경도로 바꿈
df = wgs84_to_web_mercator(df, 'lon', 'lat')
df.head()
```

Out[13]:

	lat	lon	state	x	y
0	42.05640	-111.04694	WY	-1.236169e+07	5.169432e+06
1	42.10051	-111.04691	WY	-1.236169e+07	5.176047e+06
2	42.14132	-111.04711	WY	-1.236171e+07	5.182172e+06
3	42.19558	-111.04711	WY	-1.236171e+07	5.190322e+06
4	42.20782	-111.04732	WY	-1.236173e+07	5.192161e+06

Using the election dataset, we now want to add an extra column to the `DataFrame` to give the colour of the state depending on the victor. We are going to set the state to blue if Clinton won it, or red if Trump won it.

To do this, we are going to use **`loc` (http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.loc.html)**. This specifies criteria for which rows we are to select, and then provides the name of a column to include the output:

In [14]:

```
# You don't need to write anything here
for e in range(len(election_winners)): #범위는 선거 승자의 길이
    winner = election_winners['winner'][e]
    colour = ''
    if winner == 'clinton':
        colour = 'blue'
    else:
        colour = 'red'

election_winners.loc[election_winners['winner'] == 'clinton', 'colour'] = 'blue'
#레이블 기반 위치인 .loc레이블을 기준으로 데이터의 위치를 찾을 수 있다.
#위너 클린턴, 컬러 블루
election_winners.loc[election_winners['winner'] == 'trump', 'colour'] = 'red'
election_winners.head()
```

Out[14]:

	state	winner	colour
0	AK	trump	red
1	MI	trump	red
2	ME	clinton	blue
3	PA	trump	red
4	NY	clinton	blue

Typesetting math: 100%

Now we can start using our data to overlay `Glyphs` (http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.loc.html) onto a map. We will start by creating a new map centred over the USA. Create a new map with the same attributes as the previous map you created. Call the `Figure` variable `fig`.

In [15]:

```
# YOUR CODE HERE
USA = x_range,y_range = ((-13884029,-7453304), (2698291,6455972))

url = 'http://a.basemaps.cartocdn.com/dark_all/{Z}/{X}/{Y}.png'
attribution = "Map tiles by Carto, under CC BY 3.0. Data by OpenStreetMap, under ODb

fig = Figure(tools='pan, wheel_zoom', x_range=x_range, y_range=y_range)
fig.axis.visible = False
fig.add_tile(WMTSTileSource(url=url, attribution=attribution))
```

Out[15]:

TileRenderer(id = 'f45b11a7-d4a2-4fae-aafe-d25db381d38d', …)

Now try and add a `Glpyh` to the `Figure` which gives the outline of California, and colours it in blue. Use the same strategy as you used above for the line graph. The glyph in question is `Patch` (http://bokeh.pydata.org/en/latest/docs/user_guide/plotting.html#patch-glyphs), which uses the function `patch`.

In [20]:

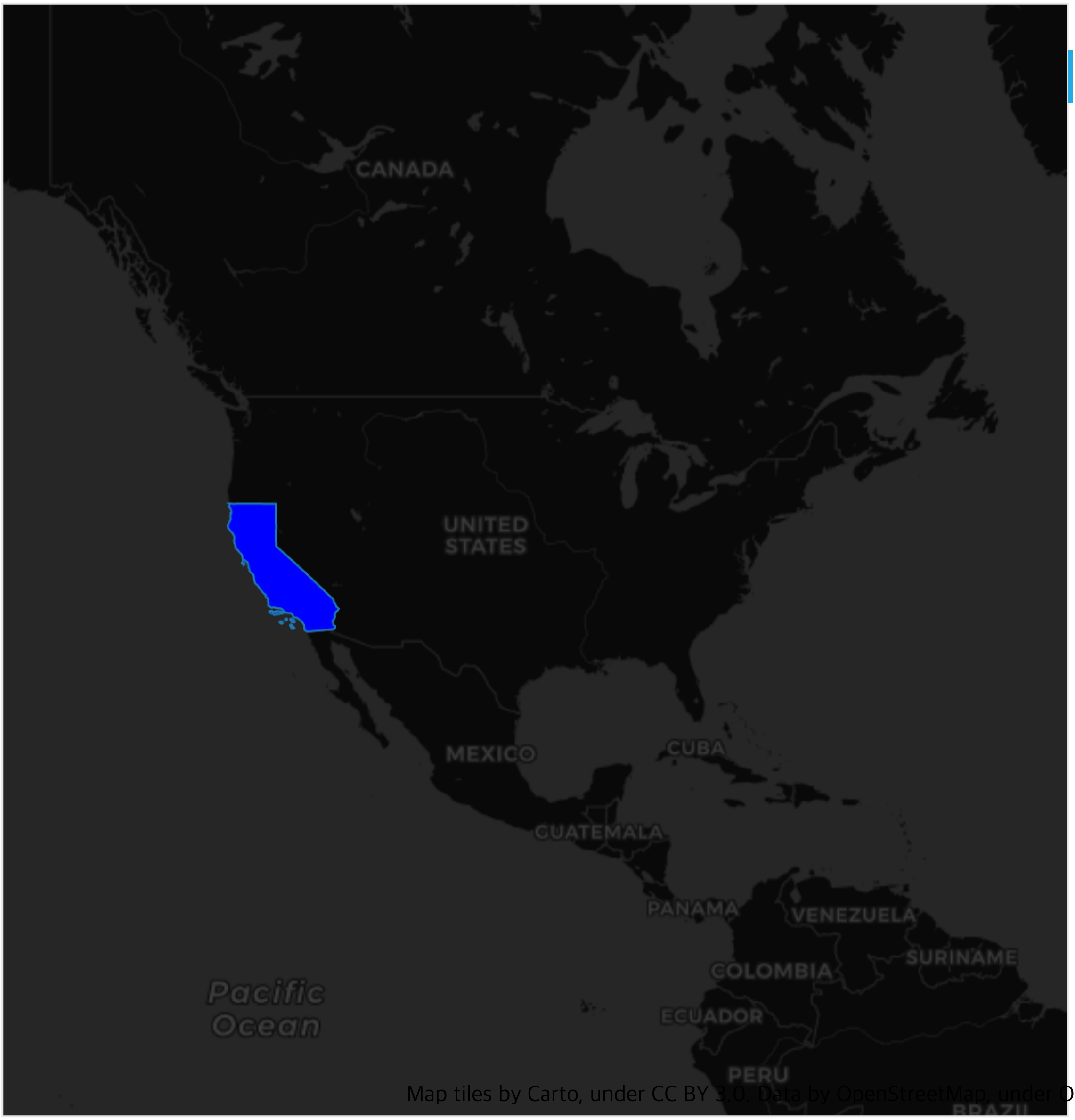
```
#이제 피규어에 윤곽을 제공하는 글리프를 추가 !

# YOUR CODE HERE
data = df.loc[df['state'] == 'CA'] #정보 세트나 데이터 프레임의 상태 필드(state)
colour = election_winners.loc[election_winners['state'] == 'CA']['colour'].iloc[0]
                    #색상은 CA와 일치하는 색상이 됩니다.

ca = fig.patch(data['x'], data['y'], fill_color=colour) #패치하기
```

Now we can show the map of the USA with the California Glyph:

Typesetting math: 100%

In [21]:

```
show(fig)
```



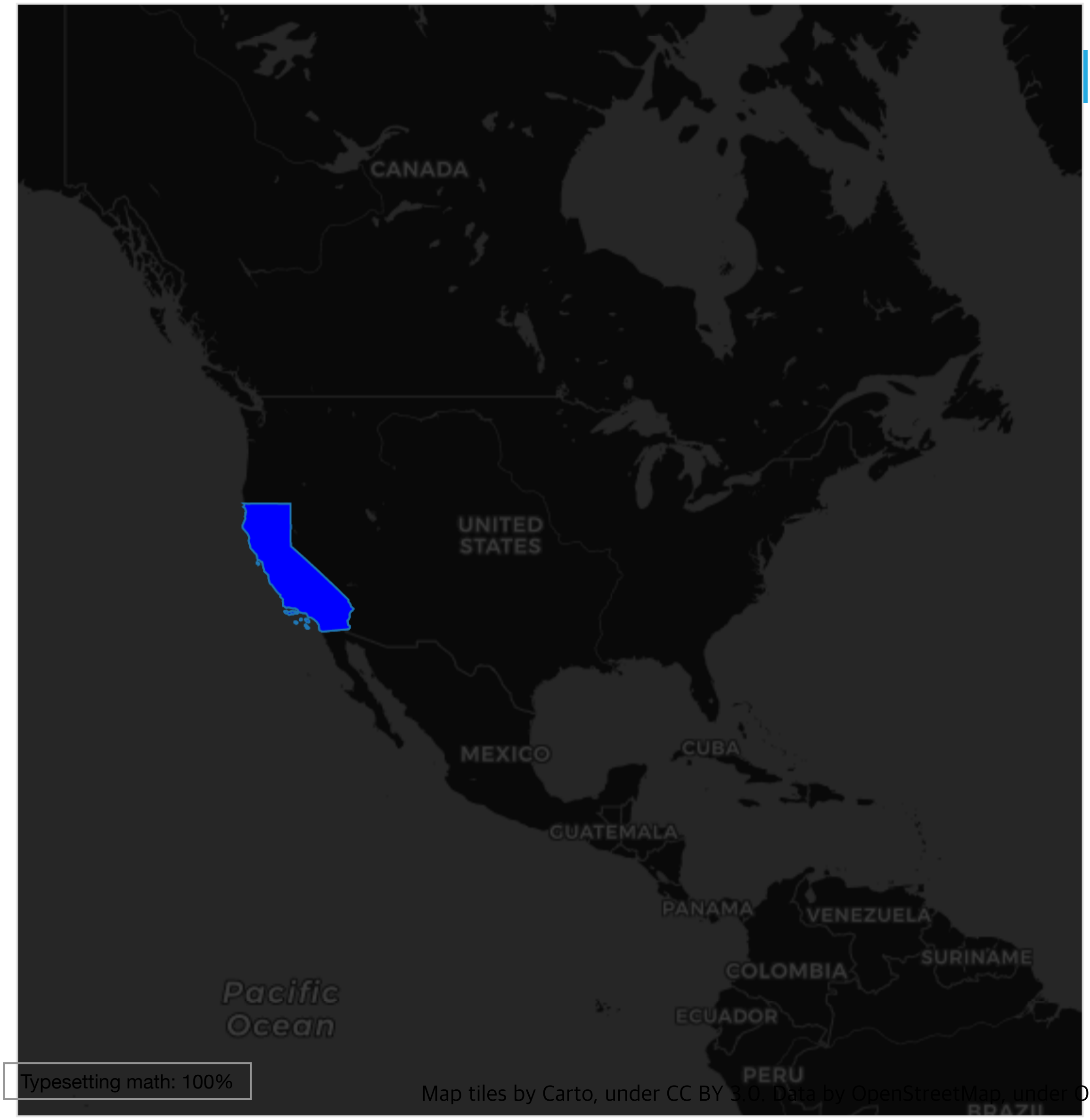
(http

Map tiles by Carto, under CC BY 3.0. Data by OpenStreetMap, under ODbL

Typesetting math: 100%

In [22]:

```
patches_x = []
patches_y = []
colors = []
list_of_states = ['AL', 'AK', 'AZ', 'AR', 'CA', 'CO', 'CT', 'DE', 'DC', 'FL', 'GA',
                  'IA', 'KS', 'KY', 'LA', 'ME', 'MD', 'MA', 'MI', 'MN', 'MS', 'MO',
                  'NJ', 'NM', 'NY', 'NC', 'ND', 'OH', 'OK', 'OR', 'PA', 'RI', 'SC',
                  'VT', 'VA', 'WA', 'WV', 'WI', 'WY']


# Making a copy of a subset of the original data.  This way, if we make a change, i
data = df.loc[df['state'] == 'CA'].copy()
#CA와 동일한 df state를 사용하여 df.loc을 할당 합니다.
#이 복사본을 변수 데이터에 할당합니다.


# We're using the patch glyph to colour fill the states
# The .iloc[0] signifies the first row in the dataset, because we only need one colo

ca = fig.patch(data['x'], data['y'], fill_color=colour)

show(fig)
```



(http

Typesetting math: 100%

Map tiles by Carto, under CC BY 3.0. Data by OpenStreetMap, under ODbL

Putting it all Together

Now we have built individual components which we can modify to generate our interactive visualisation. To put them together, we will:

- Create and display a widget which allows text to be entered to select a state
- Create a function for them to call which will modify the selected regions on the map
- Display the map

First, we are going to try and generalise the code we saw above into one which works for all states into the function `callback`, and display a new map.

Typesetting math: 100%

In [23]:

```
def callback(state):
    # Filter the data here from `df` using `loc` to get the individual state
    # Then, access the data_source of `ca` to change it to the new data
    # YOUR CODE HERE
    data = df.loc[df['state'] == state]
    ca.data_source.data['x'] = data['x']
    ca.data_source.data['y'] = data['y']

    # Get the colour of the new state, and update the fill_color of the glyph
    colour = election_winners.loc[election_winners['state'] == state]['colour'].iloc
    ca.glyph.fill_color = colour
    # Update the map
    # YOUR CODE HERE
    push_notebook()

fig = Figure(tools='pan, wheel_zoom', x_range=x_range, y_range=y_range)
fig.axis.visible = False
fig.add_tile(WMTSTileSource(url=url, attribution=attribution))
ca = fig.patch(data['x'], data['y'], fill_color=colour)
show(fig, notebook_handle=True)
```

CANADA
UNITED STATES
MEXICO
CUBA
GUATEMALA
PANAMA
VENEZUELA
SURINAME
COLOMBIA
ECUADOR
PERU
BRAZIL
Pacific Ocean
Map tiles by Carto, under CC BY 3.0. Data by OpenStreetMap, under ODbL

(http

Out[23]:

Typesetting math: 100%

```
<Bokeh Notebook handle for In[23]>
```

Now we'll add a text box interactive widget, so that by entering a state we can see the map update. This still uses the `interactive` function, except the parameter to the `callback` function is a string rather than a number.

The response to the call for interactive is being assigned to a variable - in this case `i` - to later use.

In [26]:

```
# You don't need to write anything here
i = interactive(callback, state='')
i
```

```
---------------------------------------------------------------------------
-----
IndexError                                Traceback (most recent call
 last)
<ipython-input-23-9c17acf398cc> in callback(state)
      8 
      9     # Get the colour of the new state, and update the fill_col
or of the glyph
---> 10     colour = election_winners.loc[election_winners['state'] == 
state]['colour'].iloc[0]
     11     ca.glyph.fill_color = colour
     12     # Update the map

/opt/conda/lib/python3.5/site-packages/pandas/core/indexing.py in __ge
titem__(self, key)
   1310             return self._getitem_tuple(key)
   1311         else:
-> 1312             return self._getitem_axis(key, axis=0)
   1313 
   1314     def _getitem_axis(self, key, axis=0):

/opt/conda/lib/python3.5/site-packages/pandas/core/indexing.py in _get
item_axis(self, key, axis)
   1626 
   1627                 # validate the location
-> 1628                 self._is_valid_integer(key, axis)
   1629 
   1630             return self._get_loc(key, axis=axis)

/opt/conda/lib/python3.5/site-packages/pandas/core/indexing.py in _is_
valid_integer(self, key, axis)
   1540         l = len(ax)
   1541         if key >= l or key < -l:
-> 1542             raise IndexError("single positional indexer is out
-of-bounds")
   1543         return True
   1544 

IndexError: single positional indexer is out-of-bounds
```

Layout

Typesetting math: 100%

Finally, the widgets need to be set out. Both Jupyter and Bokeh have their own widgets for layout, and they are not yet compatible. To lay them out, we suggest that you keep the plot in one cell, and the widgets in the cell above or below.

The functions `VBox` and `HBox` allow widgets to be laid out in a way which they align either vertically (for `VBox`) or horizontally (for `HBox`). The function takes a list of `ipywidgets` widgets:

In [92]:

```
HBox(
    # We can put HBox and VBox inside each other as well
    [ HBox([i, i]), VBox([i,i, i]) ]
)

    # Note - we may get an error warning message here as an indexer is out of bounds
```

```
/opt/conda/lib/python3.5/site-packages/bokeh/models/sources.py:81: Bok
ehUserWarning: ColumnDataSource's columns must be of the same length
  lambda: warnings.warn("ColumnDataSource's columns must be of the sam
e length", BokehUserWarning))
```

Have a look at the list of widgets on the ipywidgets documentation (http://ipywidgets.readthedocs.io/en/latest/examples/Widget%20List.html) and experiment with displaying those in the cell below:

In [ ]:

```
# YOUR CODE HERE
```