Simple Geographic Scatter Plots with Plotly Express

In this post, I'd like to introduce a quick but powerful visualization option available with Plotly Express. While the module provides over 30 functions for creating visualizations, today I will be focusing on creating geographic scatter plots using scatter_geo().

As a quick background, Plotly is a free open source library built on top of the Plotly JavaScript library (a high-level, declarative charting library). It allows you to build interactive visualizations that can be displayed in Jupyter Notebooks or saved in HTML files. The Plotly Express module is available in the Plotly library and makes it easy to quickly create powerful interactive visualizations with just a few lines of code.

Plotly can be installed using pip or conda and has support for both Jupyter Labs and Jupyter Notebooks. Additional details on installation steps can be found at: https://plotly.com/python/getting-started/.

Geographic Plots

Before we get started with an example, I'd like to show some basic capabilities of scatter_geo(). The first thing we need to do is import the Plotly Express module. We will then use a blank dataframe to see a default visualization.

In [1]:
```
#Import plotly.express and pandas
import plotly.express as px
import pandas as pd
import plotly
plotly.offline.init_notebook_mode()

#Create a dummy dataframe and show the figure
df=pd.DataFrame({'data':[0]})
fig = px.scatter_geo(df, size='data')
fig.show(renderer='notebook')
```

That's amazing! With only a few lines of code we have a background of the world map. Not only that, but the figure is **interactive**. Grab the map with your mouse and try moving it around. Test the controls in the upper right of the output. With a single click you can capture of photo of the map, pan, and zoom in and out. My only issue with the output is the logo link to the Plotly site in the upper right that promotes additional commercial products. The Plotly library is free so this is only a minor nuisance. All in all, not a bad start.

Now let's play around with some parameters. We can use "projection" to change the style of the map. Let's see a few options. Note that each of the following have the same level of interactivity as our first figure.

In [2]:
```
#List a few options for the projection parameter and loop through them
some_options = ['orthographic', 'natural earth','conic equal area']
for option in some_options:
    fig = px.scatter_geo(df, size='data', projection=option)
    fig.show()
```

Where are the UFOs?

We are off to a good start but now let's look at an example. In this scenario, we would like to visually represent which US states have recorded the most UFO sightings. We will be looking at total accumulated time of UFO sighting by state using the Kaggle data set "UFO Sightings around the world". The data includes over 80,000 sightings dating back to 1949 and can be found at: https://www.kaggle.com/camnugent/ufo-sightings-around-the-world.

We need to make some simple changes to the data so will begin by selecting the columns we need from the original csv file. This includes the state abbreviation and the length of each recorded UFO encounter. We remove missing values and then change the abbreviations as the scatter_geo() function uses uppercase abbreviations to identify states. We also need to convert encounter times to a numeric value for calculations.

In [3]:
```
#Read csv file, remove NaN values and select states and length of each encounter
ufos = pd.read_csv('ufo_sighting_data.csv', low_memory=False)
ufos.dropna(inplace=True)
ufos = ufos[['state/province', 'length_of_encounter_seconds']]

#Format data as floats and make state abbreviations uppercase
ufos['length_of_encounter_seconds'] = ufos['length_of_encounter_seconds'].astype(float)
ufos['state/province'] = ufos['state/province'].str.upper()
```

We now want to aggregate the total time by each state using the groupby operation. To improve readability, we convert the accumulated time to total days as it is currently in total seconds. Finally, for readability, we will change the "state/province" column name to just "state".

In [4]:
```
#Aggregate the total time of encounters by state and convert to total days, and rename the state/providence column name
ufos_by_state = ufos.groupby('state/province', as_index=False)['length_of_encounter_seconds'].sum()
ufos_by_state['days']=ufos_by_state['length_of_encounter_seconds']/(6000*24).astype('int64')
ufos_by_state.rename(columns = {'state/province':'state'}, inplace = True)
```

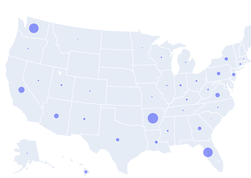
Let's Visualize

We are making great progress. Now we can use our ufos_by_state dataframe with scatter_geo(). We need to specify our dataframe, the attribute we use to size the data (days) and which data includes our locations (state). We also need to let the function know that we are identifying regions by USA state using the locationmode parameter. Let's see how that looks.

In [5]:
```
fig = px.scatter_geo(ufos_by_state, size="days", locations="state", locationmode='USA-states')
fig.show()
```

Interesting but it might be more useful to zoom in by including the "scope" parameter. We can do this by setting the scope attribute to USA.

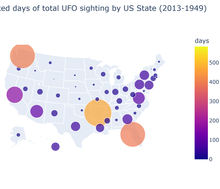
In [6]:
```
fig = px.scatter_geo(ufos_by_state, size="days", locations="state", locationmode='USA-states', scope="usa")
fig.show()
```



That looks better. We can easily see that states like Arkansas, and Washington, and Florida have accumulated the most UFO sighting time. Move your mouse arrow over the bubble to see the actual values. State's in the upper-Midwest have much smaller values. If you want to see a UFO, I'd recommend a vacation in Arkansas. If you are worried about a sighting, North Dakota is your best bet. Something is going on here that warrants more investigation but I'll leave that to another post.

Let's make a couple of finishing touches. First, while we can visually see the differences across states, let's include a scale to add clarity to the results. This is done by using the color parameter and setting it to the amount of days accumulated. We will also scale up the size of the bubbles using the size_max parameter and scale the size of our visualization with height and width. Lastly, let's add a title and see our final figure.

In [7]:
```
fig = px.scatter_geo(ufos_by_state, size="days", locations="state", locationmode='USA-states', \
                     scope="usa", color='days', size_max=50, height=480, width=640,\
                     title='Accumulated days of total UFO sighting by US State (2013-1949)", )
fig.show()
```



Conclusion

Now that is a nice visualization! I would like to reiterate that all this is done with one function and, essentially, one line of code. Not only do we have an appealing figure, we also have the built-in mapping, interactivity, and screen capture features with no additional coding. I have provided a small glimpse of the scatter_geo() possibilities and encourage you to try it out for yourself. For more information on the function and associated parameters, you can visit: https://plotly.com/python-api-reference/generated/plotly.express.scatter_geo.html. In conclusion, the Plotly Express scatter_geo() function makes it quick and easy to build advanced geographic scatter plots and should be considered for your next project.

In [ ]:
```
```