EndOfSemesterProject

June 5, 2024

1 End of Semester Project

We've done a lot this semester, you've learned a few things and now you're going to have fun (hopefully) making and thinking about some more advanced plotting.

First, we will introduce additional visualization techniques available to use through the plotly library. Understanding the details of these visualization methods is out of the scope of this course, but they're still pretty cool to look at!

Your end of semester project will be to use these techniques to analyze some data. (More instructions will be given below.)

[1]:
```
import numpy as np
import plotly.express as px
import seaborn as sns
import pandas as pd
```

1.1 Animated Scatter Plots

Our data today comes from the Gapminder Foundation which explores data on poverty, inequality and health around the world.

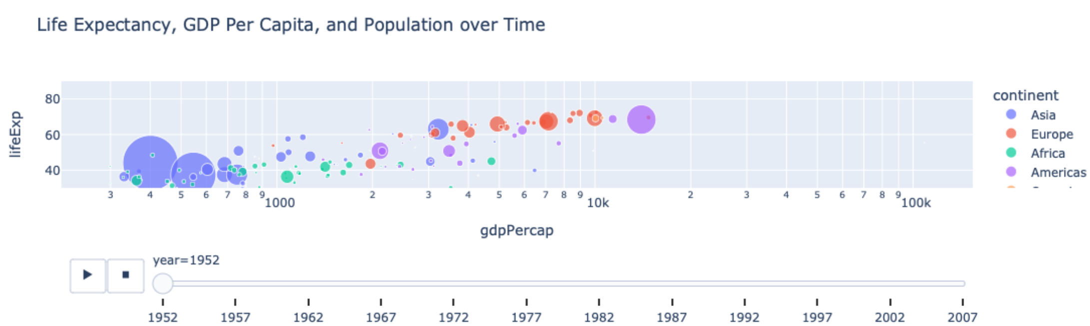
[2]:
```
world = px.data.gapminder()
world.head(10)
```

[2]:
```
       country continent  year  lifeExp       pop   gdpPercap iso_alpha  \
0  Afghanistan      Asia  1952   28.801   8425333  779.445314       AFG
1  Afghanistan      Asia  1957   30.332   9240934  820.853030       AFG
2  Afghanistan      Asia  1962   31.997  10267083  853.100710       AFG
3  Afghanistan      Asia  1967   34.020  11537966  836.197138       AFG
4  Afghanistan      Asia  1972   36.088  13079460  739.981106       AFG
5  Afghanistan      Asia  1977   38.438  14880372  786.113360       AFG
6  Afghanistan      Asia  1982   39.854  12881816  978.011439       AFG
7  Afghanistan      Asia  1987   40.822  13867957  852.395945       AFG
8  Afghanistan      Asia  1992   41.674  16317921  649.341395       AFG
9  Afghanistan      Asia  1997   41.763  22227415  635.341351       AFG

   iso_num
0        4
```

1	4
2	4
3	4
4	4
5	4
6	4
7	4
8	4
9	4

Use the code to generate an animated scatter plot of GDP per capita and life expectancy over time. You can play the animation or scroll to a specific year.

[3]:
```python
px.scatter(world,
           x = 'gdpPercap',
           y = 'lifeExp',
           hover_name = 'country',
           color = 'continent',
           size = 'pop',
           size_max = 60,
           log_x = True,
           range_y = [30, 90],
           animation_frame = 'year',
           title = 'Life Expectancy, GDP Per Capita, and Population over Time'
          )
```

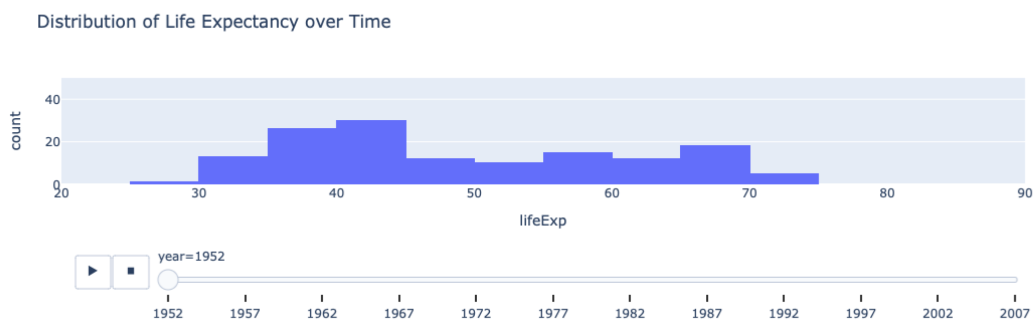


1.2 Animated Histograms

We can do the same with `px.histogram`, using the optional argument `animation_frame`.

[4]:
```python
px.histogram(world,
             x = 'lifeExp',
             animation_frame = 'year',
             range_x = [20, 90],
```

```
             range_y = [0, 50],
             title = 'Distribution of Life Expectancy over Time')
```



1.3 Box Plots

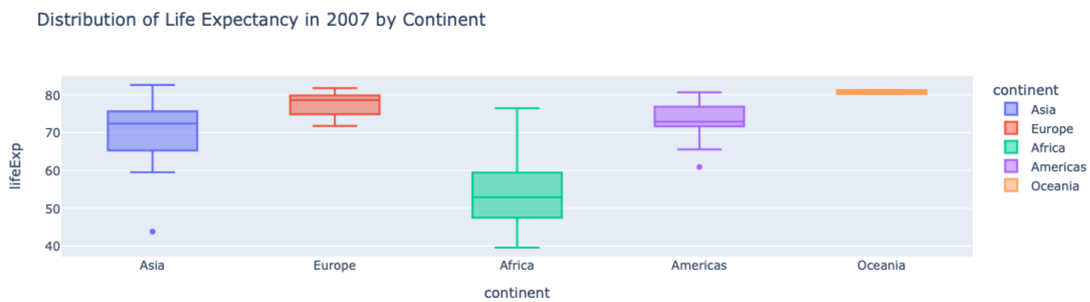
Box plots, also called “box and whisker plots” show the rough distribution of multiple numerical variables. In particular, they show the 25th, 50th (median), and 75th percentiles (the box), as well as 1.5 times the Interquartile Range (the whiskers). This is helpful for identifying outliers.

[5]:
```
worldLatest = world[world['year']== 2007]
worldLatest.head(10)
```

[5]:
```
         country continent  year  lifeExp        pop     gdpPercap iso_alpha  \
11   Afghanistan      Asia  2007   43.828   31889923    974.580338       AFG
23       Albania    Europe  2007   76.423    3600523   5937.029526       ALB
35       Algeria    Africa  2007   72.301   33333216   6223.367465       DZA
47        Angola    Africa  2007   42.731   12420476   4797.231267       AGO
59     Argentina  Americas  2007   75.320   40301927  12779.379640       ARG
71     Australia   Oceania  2007   81.235   20434176  34435.367440       AUS
83       Austria    Europe  2007   79.829    8199783  36126.492700       AUT
95       Bahrain      Asia  2007   75.635     708573  29796.048340       BHR
107   Bangladesh      Asia  2007   64.062  150448339   1391.253792       BGD
119      Belgium    Europe  2007   79.441   10392226  33692.605080       BEL

     iso_num
11         4
23         8
35        12
47        24
59        32
71        36
83        40
95        48
107       50
```

```
119        56
```

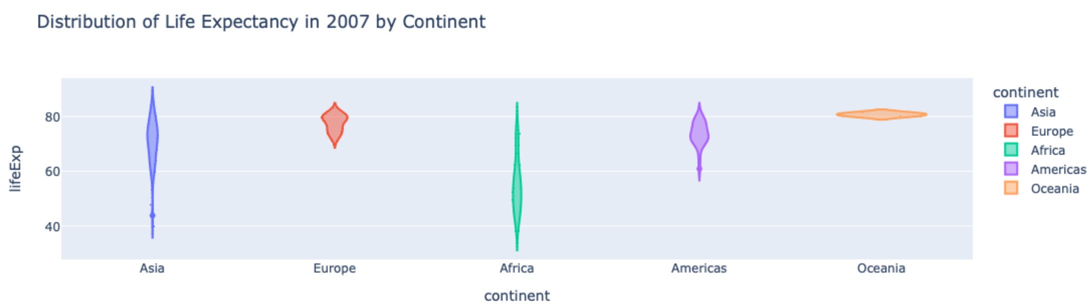
[6]:
```
px.box(worldLatest,
       y = 'lifeExp',
       x = 'continent',
       color = 'continent',
       hover_name = 'country',
       title = 'Distribution of Life Expectancy in 2007 by Continent'
      )
```



1.4 Violin Plots

These are basically the same as box plots, but show the distribution as well

[7]:
```
px.violin(worldLatest,
       y = 'lifeExp',
       x = 'continent',
       color = 'continent',
       hover_name = 'country',
       title = 'Distribution of Life Expectancy in 2007 by Continent'
      )
```



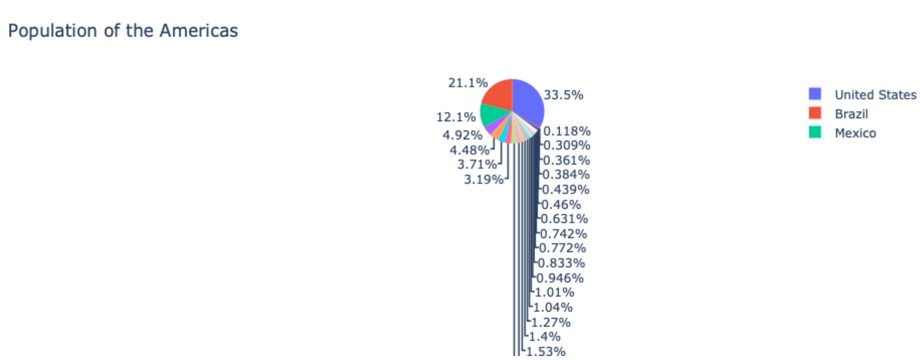
1.5 Pie Charts

Pie charts look cool visually, but they are often hard to analyze. Let's take a closer look at some of them.

[8]:
```
worldLatestAmericas = worldLatest[worldLatest['continent']=='Americas']
worldLatestAmericas.head(10)
```

[8]:
```
                country continent  year  lifeExp        pop     gdpPercap  \
59            Argentina  Americas  2007   75.320   40301927  12779.379640
143             Bolivia  Americas  2007   65.554    9119152   3822.137084
179              Brazil  Americas  2007   72.390  190010647   9065.800825
251              Canada  Americas  2007   80.653   33390141  36319.235010
287               Chile  Americas  2007   78.553   16284741  13171.638850
311            Colombia  Americas  2007   72.889   44227550   7006.580419
359          Costa Rica  Americas  2007   78.782    4133884   9645.061420
395                Cuba  Americas  2007   78.273   11416987   8948.102923
443  Dominican Republic  Americas  2007   72.235    9319622   6025.374752
455             Ecuador  Americas  2007   74.994   13755680   6873.262326

    iso_alpha  iso_num
59        ARG       32
143       BOL       68
179       BRA       76
251       CAN      124
287       CHL      152
311       COL      170
359       CRI      188
395       CUB      192
443       DOM      214
455       ECU      218
```

[9]:
```
px.pie(worldLatestAmericas,
       values = 'pop',
       names = 'country',
       title = 'Population of the Americas'
)
```



We can see the countries showing up as percentages, but there are so many countries, so it's a little bit harder.

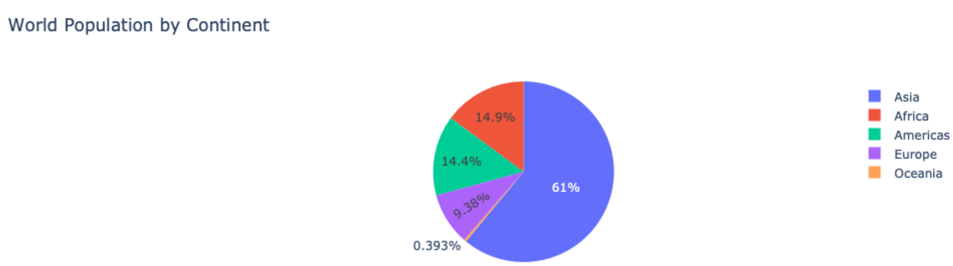
Let's take a look at what happens if we group by continent instead.

[10]:
```
# Grouping by continent and summing up populations
worldLatestByContinent= worldLatest.groupby('continent')['pop'].sum().
 ↪reset_index()
worldLatestByContinent.head(10)
```

[10]:

	continent	pop
0	Africa	929539692
1	Americas	898871184
2	Asia	3811953827
3	Europe	586098529
4	Oceania	24549947

[11]:
```
px.pie(worldLatestByContinent,
      values = 'pop',
      names = 'continent',
      title = 'World Population by Continent')
```



1.6 Animated Pie Charts

This is pretty cool, but do you think we can animate the pie chart? Of course we can!!

First - let's go back to our original `world` DataFrame and get the population for each continent by year. We will use group and retain continent and year.

[12]:
```
worldByContinent = world.groupby(['continent', 'year'])['pop'].sum().
 ↪reset_index()
worldByContinent.head(10)
```

[12]:

	continent	year	pop
0	Africa	1952	237640501
1	Africa	1957	264837738
2	Africa	1962	296516865
3	Africa	1967	335289489
4	Africa	1972	379879541
5	Africa	1977	433061021
6	Africa	1982	499348587
7	Africa	1987	574834110
8	Africa	1992	659081517
9	Africa	1997	743832984

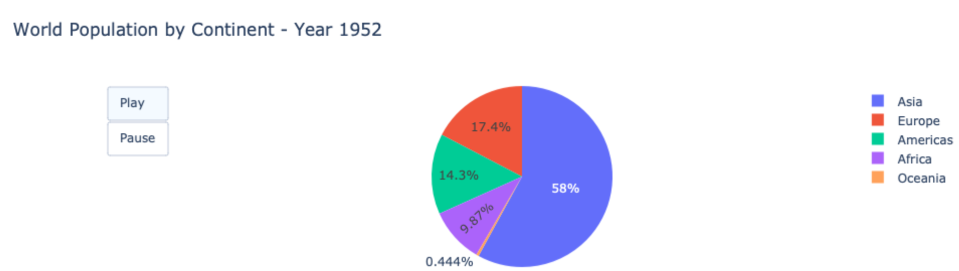
Now we will do the same field we did before in order to animate by year.

However, because there doesn't seem to be the same animate frames we had to do something a little stranger. Create a pie chart for each year, and then animate it with a figure.

It's a little clunkier, but it seems to work.

[13]:
```
import plotly.graph_objs as go

# Assuming worldByContinent already exists

# Create pie charts for each year
frames = []
for year in worldByContinent['year'].unique():
    data_year = worldByContinent[worldByContinent['year'] == year]
    pie = go.Pie(labels=data_year['continent'], values=data_year['pop'],␣
↪name=f'Year {year}')
    frame = go.Frame(data=[pie], name=f'Year {year}')
    frames.append(frame)

# Create figure with the first year's pie chart
fig = go.Figure(data=[go.Pie(labels=worldByContinent[worldByContinent['year']␣
↪== min(worldByContinent['year'])]['continent'],
                             values=worldByContinent[worldByContinent['year']␣
↪== min(worldByContinent['year'])]['pop'],
                             name=f'Year {min(worldByContinent["year"])}')],
                frames=[frames[0]])

# Add frames to animation
fig.frames = frames

# Update layout
fig.update_layout(title=f'World Population by Continent - Year␣
↪{min(worldByContinent["year"])}',
                  updatemenus=[{
                      'type': 'buttons',
```

```
                        'buttons': [{
                            'label': 'Play',
                            'method': 'animate',
                            'args': [None, {
                                'frame': {
                                    'duration': 1000,  # Set the duration for␣
 ↪each frame (milliseconds)
                                    'redraw': True
                                },
                                'fromcurrent': True,
                                'mode': 'immediate'
                            }]
                        }, {
                            'label': 'Pause',
                            'method': 'animate',
                            'args': [[None], {
                                'frame': {
                                    'duration': 0,
                                    'redraw': False
                                },
                                'mode': 'immediate'
                            }]
                        }]
                    }])

# Update each frame's title to include the corresponding year
for i, frame in enumerate(fig.frames):
    fig.frames[i].update(layout_title_text=f'World Population by Continent -␣
 ↪Year {worldByContinent["year"].unique()[i]}')

fig.show()
```



Now, here is a moment that we ask ourselves, was it worth it?

The percentages per continent do not seem to change very much. Perhaps a line graph is a better way to show this.

1.7 Line Graph

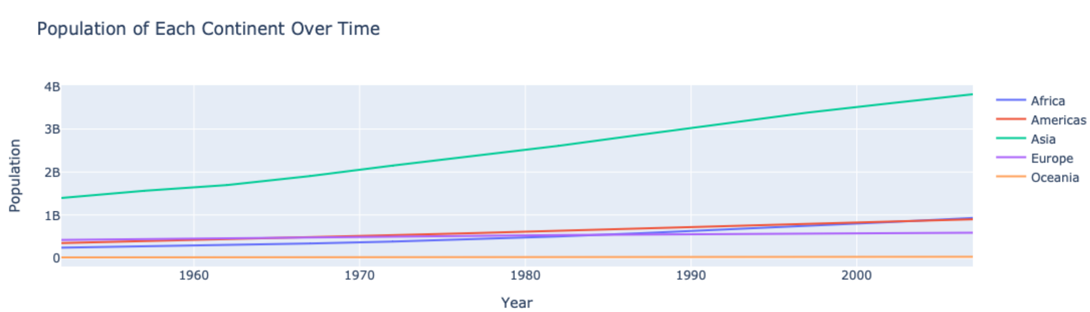
As mentioned before, pie charts are kind of hard to read sometimes. And looking at an animated pie chart like we did above was hard to read. Let's go back to one of the first types of plots we looked at here.

Line plots!!

[14]:
```
worldByContinent['continent'].unique()
```

[14]:
```
array(['Africa', 'Americas', 'Asia', 'Europe', 'Oceania'], dtype=object)
```

[15]:
```
# Define the continents
continents = ['Africa', 'Americas', 'Asia', 'Europe', 'Oceania']

# Create a line graph for each continent
data = []
for continent in continents:
    continent_data = worldByContinent[worldByContinent['continent'] ==␣
 ↪continent]
    data.append(go.Scatter(x=continent_data['year'],
                           y=continent_data['pop'],
                           mode='lines',
                           name=continent))

# Create figure
fig = go.Figure(data=data)

# Update layout
fig.update_layout(title='Population of Each Continent Over Time',
                  xaxis_title='Year',
                  yaxis_title='Population')

fig.show()
```



This gets us some of the way there, but let's look into the percentage of each continent (like our pie chart from above).

Let's first add a column to give the percentage per year that each continent's population has.

[16]:
```
total_population_by_year = worldByContinent.groupby('year')['pop'].sum()

worldByContinent['pop pct'] = worldByContinent.apply(lambda row: row['pop'] /
↪total_population_by_year[row['year']], axis=1)

worldByContinent.head(10)
```

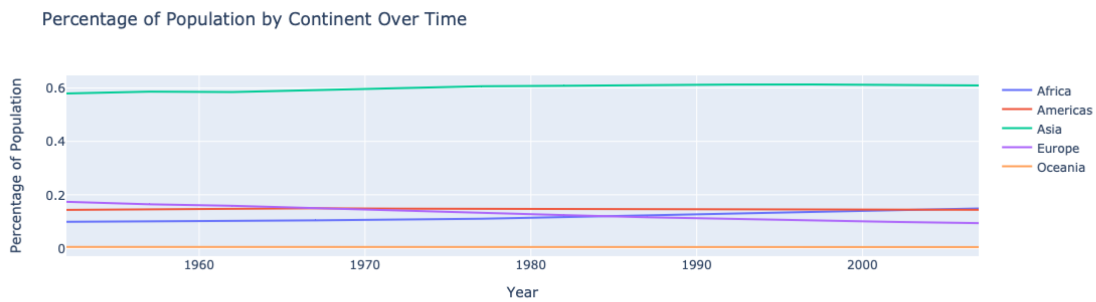
[16]:

	continent	year	pop	pop pct
0	Africa	1952	237640501	0.098731
1	Africa	1957	264837738	0.099398
2	Africa	1962	296516865	0.102255
3	Africa	1967	335289489	0.104209
4	Africa	1972	379879541	0.106201
5	Africa	1977	433061021	0.110192
6	Africa	1982	499348587	0.116414
7	Africa	1987	574834110	0.122527
8	Africa	1992	659081517	0.128961
9	Africa	1997	743832984	0.134870

And now let's plot the proportion of each continent by year.

[17]:
```
# Define the continents
continents = ['Africa', 'Americas', 'Asia', 'Europe', 'Oceania']

# Create a line graph for each continent
data = []
for continent in continents:
    continent_data = worldByContinent[worldByContinent['continent'] ==
↪continent].copy()
    data.append(go.Scatter(x=continent_data['year'],
                           y=continent_data['pop pct'],
                           mode='lines',
                           name=continent))

# Create figure
fig = go.Figure(data=data)

# Update layout
fig.update_layout(title='Percentage of Population by Continent Over Time',
                  xaxis_title='Year',
                  yaxis_title='Percentage of Population')
```

```
fig.show()
```



1.8 Timelines (Gantt Charts)

You may - or may not - have seen these before, but you can illustrate a project timeline with a Gantt chart.

Below this Gantt chart illustrates the timeline of the Suraj Rampure Suraj Rampure, creator of Data 6 at UC Berkeley. (Which we have used for many of the ideas in this course!)

[18]:
```
# Define the phases list
phases = [
    ['Newborn', '1998-11-26', '1999-11-26'],
    ['Toddler, Preschooler', '1999-11-26', '2005-09-03'],
    ['Elementary School Student', '2005-09-03', '2009-06-30'],
    ['Middle School Student', '2009-09-15', '2012-06-15'],
    ['High School Student', '2012-09-05', '2016-05-30'],
    ['Undergrad @ UC Berkeley', '2016-08-22','2020-05-15'],
    ['Masters @ UC Berkeley', '2020-08-25', '2021-05-14'],
    ['Teaching Data 94', '2021-01-20', '2021-05-14']
]

# Define column labels
columns = ['Phase', 'Start', 'End']

# Create a DataFrame from the phases list
phases_df = pd.DataFrame(phases, columns=columns)

# Convert 'Start' and 'End' columns to datetime
phases_df['Start'] = pd.to_datetime(phases_df['Start'])
phases_df['End'] = pd.to_datetime(phases_df['End'])

phases_df.head(10)
```

[18]:

```
                        Phase       Start         End
0                     Newborn 1998-11-26 1999-11-26
1        Toddler, Preschooler 1999-11-26 2005-09-03
2  Elementary School Student 2005-09-03 2009-06-30
3      Middle School Student 2009-09-15 2012-06-15
4        High School Student 2012-09-05 2016-05-30
5     Undergrad @ UC Berkeley 2016-08-22 2020-05-15
6      Masters @ UC Berkeley 2020-08-25 2021-05-14
7           Teaching Data 94 2021-01-20 2021-05-14
```

[ ]:

```
# Create a Gantt chart
fig = px.timeline(phases_df, x_start='Start', x_end='End', y='Phase')

# Update layout
fig.update_layout(title='Phases Timeline',
                  xaxis_title='Date',
                  yaxis_title='Phase')

fig.show()
```

Notice that the lifetime is increasing for the Gantt chart. You can also repeat things in a Gantt chart. For example here is - according to Chat Gpt - the Gantt chart for locations/scenes in `Star Wars IV: A New Hope`

[ ]:

```
# Define the scenes and their corresponding start and end times in the original␣
↪Star Wars movie
scenes = [
    ['Tatooine', '00:00:00', '00:10:00'],
    ['Death Star', '00:10:00', '00:15:00'],
    ['Tatooine', '00:15:00', '00:20:00'],
    ['Tatooine', '00:20:00', '00:25:00'],
    ['Death Star', '00:25:00', '00:30:00'],
    ['Yavin 4', '00:30:00', '00:35:00'],
    ['Death Star', '00:35:00', '00:40:00'],
    ['Tatooine', '00:40:00', '00:45:00'],
    ['Yavin 4', '00:45:00', '01:00:00'],
    ['Death Star', '01:00:00', '01:05:00'],
    ['Yavin 4', '01:05:00', '01:15:00'],
    ['Death Star', '01:15:00', '01:20:00'],
    ['Yavin 4', '01:20:00', '01:25:00']
]

# Convert to DataFrame
scenes_df = pd.DataFrame(scenes, columns=['Location', 'Start', 'End'])

# Convert 'Start' and 'End' columns to datetime
scenes_df['Start'] = pd.to_datetime(scenes_df['Start'], format='%H:%M:%S')
```

```python
scenes_df['End'] = pd.to_datetime(scenes_df['End'], format='%H:%M:%S')

# Calculate duration of each scene
scenes_df['Duration'] = scenes_df['End'] - scenes_df['Start']

# Create a Gantt chart
fig = px.timeline(scenes_df, x_start='Start', x_end='End', y='Location',␣
 ↪title='Star Wars: A New Hope Scene Locations Timeline', hover_data=None)

# Update x-axis tick format to show only time component
fig.update_xaxes(tickformat='%H:%M:%S')

# Update layout
fig.update_layout(xaxis_title='Time', yaxis_title='Location')

fig.show()
```

1.9 Choropleth Maps

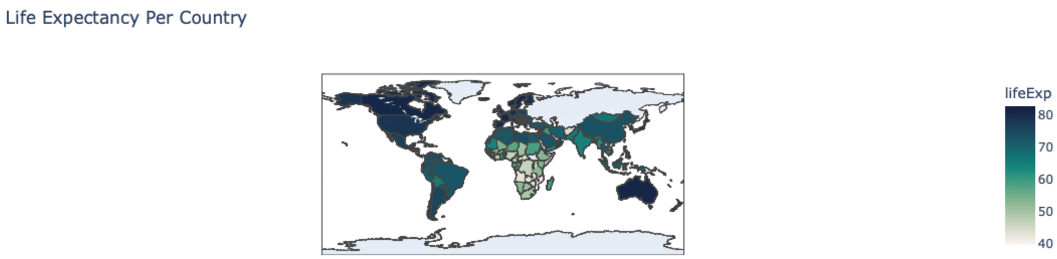
We have already seen Choropleth Maps, but they are super fun!

Let's go back to `worldLatest` which has the population information for 2007.

[20]:
```python
worldLatest.head(10)
```

[20]:
```
         country continent  year  lifeExp        pop     gdpPercap iso_alpha  \
11   Afghanistan      Asia  2007   43.828   31889923    974.580338       AFG
23       Albania    Europe  2007   76.423    3600523   5937.029526       ALB
35       Algeria    Africa  2007   72.301   33333216   6223.367465       DZA
47        Angola    Africa  2007   42.731   12420476   4797.231267       AGO
59     Argentina  Americas  2007   75.320   40301927  12779.379640       ARG
71     Australia   Oceania  2007   81.235   20434176  34435.367440       AUS
83       Austria    Europe  2007   79.829    8199783  36126.492700       AUT
95       Bahrain      Asia  2007   75.635     708573  29796.048340       BHR
107   Bangladesh      Asia  2007   64.062  150448339   1391.253792       BGD
119      Belgium    Europe  2007   79.441   10392226  33692.605080       BEL

     iso_num
11         4
23         8
35        12
47        24
59        32
71        36
83        40
95        48
107       50
119       56
```

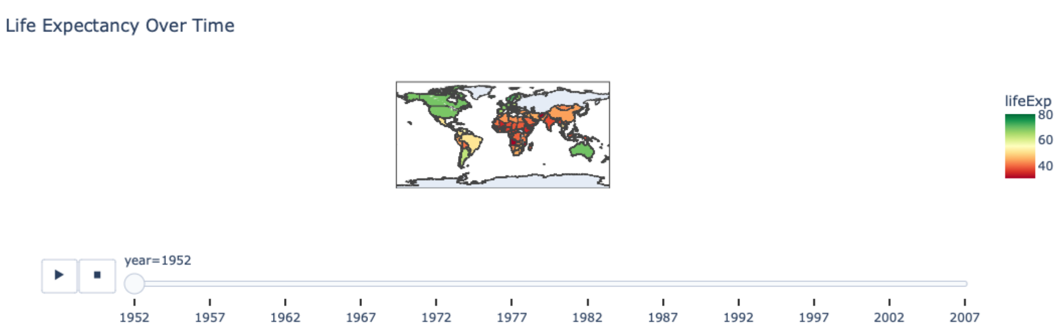
[21]:
```
px.choropleth(worldLatest,
              locations = 'iso_alpha',
              color = 'lifeExp',
              hover_name = 'country',
              title = 'Life Expectancy Per Country',
              color_continuous_scale = px.colors.sequential.tempo
)
```



1.10 Animated Choropleth Maps

And of course, our course wouldn't be complete without animating these maps!

[22]:
```
px.choropleth(world,
             locations="iso_alpha",
             color="lifeExp",
             animation_frame="year",
             color_continuous_scale = px.colors.diverging.RdYlGn,
             title = "Life Expectancy Over Time",
             range_color=(30,80))
```



1.11 3D Scatter Plots

It is also possible to plot points along three dimensions (i.e. with three coordinates).

Let's use a dataset that's part of seaborns, that has recorded data on a number of different penguins. They have recorded the: - Species - Island (they were observed on) - Length of Bill (in milimeters) - Depth of Bill (in milimeters) - Flipper Length (in milimeters) - Body Mass (in grams) - Sex (when possible to determine! Some missing values.)

[23]:
```
penguins = sns.load_dataset('penguins')
penguins.head(10)
```

[23]:
```
  species     island  bill_length_mm  bill_depth_mm  flipper_length_mm  \
0  Adelie  Torgersen            39.1           18.7              181.0
1  Adelie  Torgersen            39.5           17.4              186.0
2  Adelie  Torgersen            40.3           18.0              195.0
3  Adelie  Torgersen             NaN            NaN                NaN
4  Adelie  Torgersen            36.7           19.3              193.0
5  Adelie  Torgersen            39.3           20.6              190.0
6  Adelie  Torgersen            38.9           17.8              181.0
7  Adelie  Torgersen            39.2           19.6              195.0
8  Adelie  Torgersen            34.1           18.1              193.0
9  Adelie  Torgersen            42.0           20.2              190.0

   body_mass_g     sex
0       3750.0    Male
1       3800.0  Female
2       3250.0  Female
3          NaN     NaN
4       3450.0  Female
5       3650.0    Male
6       3625.0  Female
7       4675.0    Male
8       3475.0     NaN
9       4250.0     NaN
```

Let's plot each penguin in 3D space by their: - bill length - bill depth - flipper length

Let's use different colors to indicate their species.

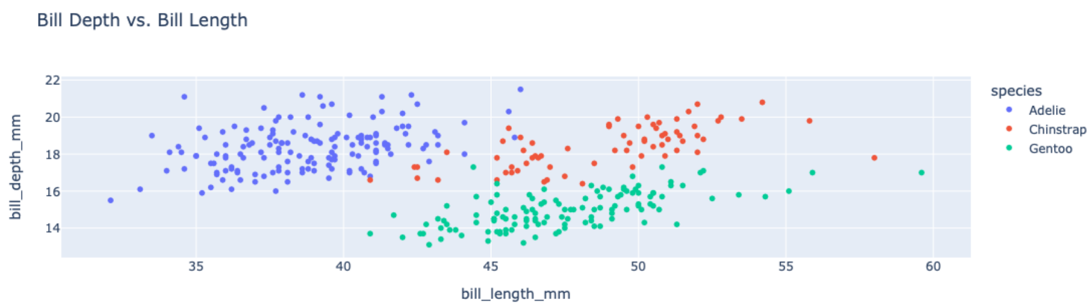
Try dragging the, graph to move around the camera.

[24]:
```
px.scatter_3d(penguins,
              x = 'bill_length_mm',
              y = 'bill_depth_mm',
              z = 'flipper_length_mm',
              color = 'species',
              hover_name = 'island',
              title = 'Flipper Length vs. Bill Depth vs. Bill Length')
```

It is generally hard to work with plots in 3D. So often you need to consider is this really the best way to show the data?

Let's take a look at a 2D scatter plot and see if it's more or less informative.

[25]:
```
px.scatter(penguins,
             x = 'bill_length_mm',
             y = 'bill_depth_mm',
             color = 'species',
             hover_name = 'island',
             title = 'Bill Depth vs. Bill Length')
```



What is something you notice about the different species?

If you had a new penguin with bill length of 45 mm and bill depth of 14 mm, could you make a guess as to what that species would be?

2 Back to the End of Semester Project!!

Okay - so we've had a lot of fun! Now it's time for you to get to work on your own!

Instructions: Your job for the End of Semester Project is to create **4 different plots using Python** (each with a different plot type) and provide a **short description (3 - 5 sentences)**

describing what this plot illustrates to you.

- **What to Submit:** You will need to turn in both a PDF; your ipynb and any data sets you use (you can provide links if they are online or upload the file itself.)
- **What Data Can I Use?** You can choose whatever data set you want for the plots. You can use the same data set for all 4 or different data sets for each. You can use any data set we have used in class, any of the built in python data sets (we will give some examples below) or (as in the example of Suraj Rampure) the data set may come from a personal place. (See below for some examples!)
- **What Kinds of Plots Can I Use?** You can use any type of plot that you want! You just need to be able to make it in Python and provide your code. Make sure you plot is readable.
- **Grading**: This will be worth 40 points in total, 10 points per plot split equally between the plot itself and your written description of it.

2.1 Plot Types:

Feel free to use any of the plot types we have used in class. Here's the ones we have used today.

- Scatter Plot (2 and 3D)
- Line Plot
- Pie Chart
- Histograms
- Box Plots
- Violin Plots
- Choropleth

Your plot can be animated (which would be super cool) but definitely doesn't need to be.

You can even use a plot type we haven't discussed in class such as a Ternary Plots

2.2 Data Sets

You can use whatever data sets you would like! We've used a lot of them in class (Zillow, UC Admission Data, Spotify, Fast Food).

2.2.1 Data Sets from Plotly

But Python itself has a number of really great built in data sets! Here are a few examples from `plotly.express` which we have imported as `px`

1 **Iris Dataset:** > A classic dataset in machine learning and statistics, containing measurements of iris flowers. > > `iris_df = px.data.iris()`

2 **Wind Dataset** > Contains wind speed and direction measurements collected from a meteorological station. > > `wind_df = px.data.wind()`

3 **Carshare Dataset** > Contains data on car sharing usage. > > `carshare_df = px.data.carshare()`

4 **Election Dataset** > Contains data on the 2008 US presidential election results. > >`election_df = px.data.election()`

[26]:
```
#Iris Data Set
iris_df = px.data.iris()

#Wind Dataset:
wind_df = px.data.wind()

#Carshare Dataset:
carshare_df = px.data.carshare()

#Election Dataset:
election_df = px.data.election()
```

2.2.2 Data Sets from Seaborn

We have also seen other built in datasets from Seaborn. Here are some of them that you might also find helpful.

To load seaborn, you'll need to run `import seaborn as sns`

1. **Titanic Dataset:** > Contains data on passengers aboard the Titanic, including information on their survival status, age, sex, class, fare, and more. > > `titanic_df = sns.load_dataset('titanic')`
2. **Functional Magnetic Resonance Imagine (fMRI) FMRI Dataset:** > Contains functional magnetic resonance imaging (fMRI) data, with time series measurements taken at different points in the brain. > > `fmri_df = sns.load_dataset('fmri')`
3. **Diamonds Dataset:** > Contains data on diamonds, including their carat weight, cut, color, clarity, and price. > > `diamonds_df = sns.load_dataset('diamonds')`
4. **Exercise Dataset:** > Contains data from an exercise physiology study, including measurements such as subject IDs, age, weight, and the amount of exercise they performed. > > `exercise_df = sns.load_dataset('exercise')`

[27]:
```
titanic_df = sns.load_dataset('titanic')

fmri_df = sns.load_dataset('fmri')

diamonds_df = sns.load_dataset('diamonds')

exercise_df = sns.load_dataset('exercise')
```

[ ]:
```
```