First Tutorial

In [1]:

```
import pandas as pd
import numpy as np
import scipy as sp
```

In [2]:

```
from plotly.offline import iplot
import plotly.figure_factory as ff
```

In [3]:

```
df = pd.read_csv("https://raw.githubusercontent.com/plotly/datasets/master/school_earnings.
```

In [4]:

```
table = ff.create_table(df)
iplot(table, filename='jupyter-table1')
```

School	Women	Men
MIT	94	152
Stanford	96	151
Harvard	112	165
U.Penn	92	141
Princeton	90	137
Chicago	78	118
Georgetown	94	131
Tufts	76	112
Yale	79	114
Columbia	86	119
Duke	93	124
Dartmouth	84	114
NYU	67	94

In [5]:

```
#Taking a columns and finding the first element
schools = df.School
schools[0]
```

Out[5]:

```
'MIT'
```

In [6]:

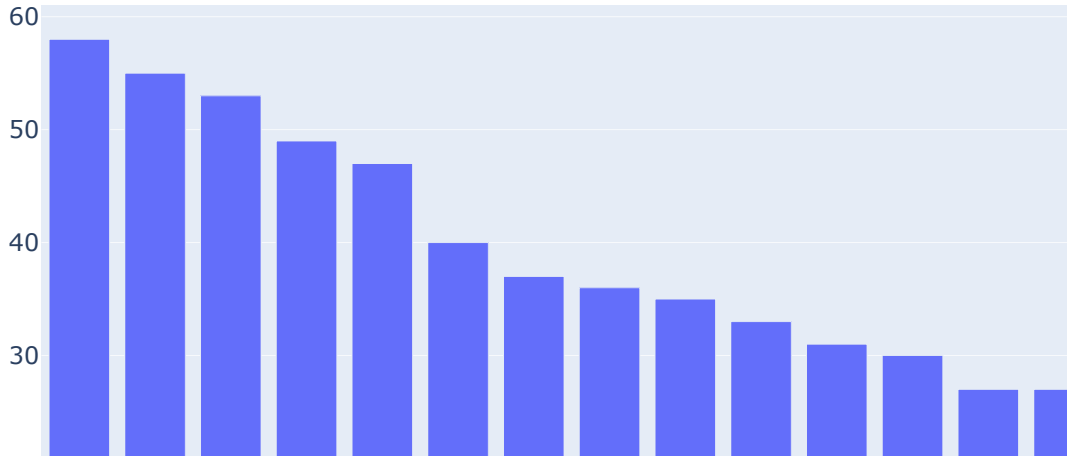
```
#finding the standard deviation for every column
df.std()
```

Out[6]:

```
Women    12.813683
Men      25.705289
Gap      14.137084
dtype: float64
```

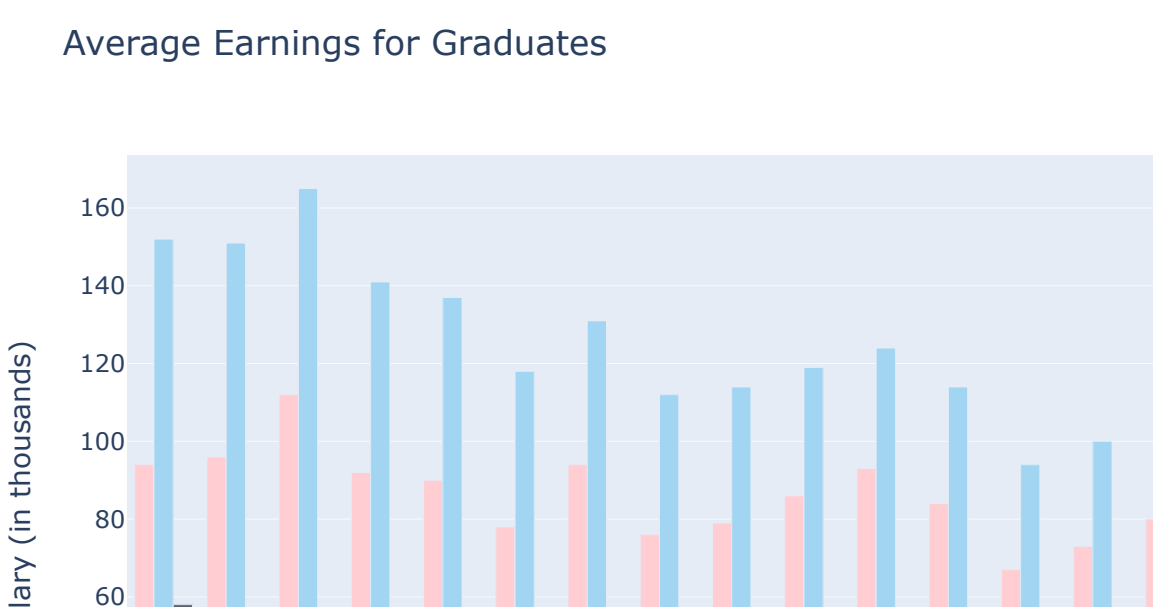
In [7]:

```
import plotly.graph_objects as go
#Plotting a barchart. x axis is the school (Categorical) y axis is the Gap (Numerical)

data = [go.Bar(x=df.School,
               y=df.Gap)]
iplot(data, filename='jupyter-basic_bar')
```



In [8]:

```
#Plotting a barchart. x axis is the school (Categorical) y axis is the Women (Numerical)
trace_women = go.Bar(x=df.School,
                     y=df.Women,
                     name='Women',
                     marker=dict(color='#ffcdd2'))

#Plotting a barchart. x axis is the school (Categorical) y axis is the Men (Numerical)
trace_men = go.Bar(x=df.School,
                   y=df.Men,
                   name='Men',
                   marker=dict(color='#A2D5F2'))

#Plotting a barchart. x axis is the school (Categorical) y axis is the Gap (Numerical)
trace_gap = go.Bar(x=df.School,
                   y=df.Gap,
                   name='Gap',
                   marker=dict(color='#59606D'))

#Putting the different barplot on a list
data = [trace_women, trace_men, trace_gap]

#Changing the title with go.layout, the xaxis and y axis
layout = go.Layout(title="Average Earnings for Graduates",
                   xaxis=dict(title='School'),
                   yaxis=dict(title='Salary (in thousands)'))

fig = go.Figure(data=data, layout=layout)

#Plotting
iplot(fig, filename='jupyter-styled_bar')
```



In [9]:

```
df = pd.read_csv('https://raw.githubusercontent.com/plotly/datasets/master/Nuclear%20Waste%
```

In [10]:

```
#Having an idea of what is the data.
df.head()
```

Out[10]:

	lat	lon	text
0	35.888827	-106.305022	Acid/Pueblo Canyon
1	39.503487	-84.743859	Alba Craft Shop
2	44.620822	-123.120917	"Albany, Oregon, FUSRAP Site"
3	40.641371	-80.242936	Aliquippa Forge
4	39.361063	-84.540750	Associated Aircraft Tool and Manufacturing Co.

In [11]:

```
#Creating variable for each column
site_lat = df.lat
site_lon = df.lon
locations_name = df.text
```

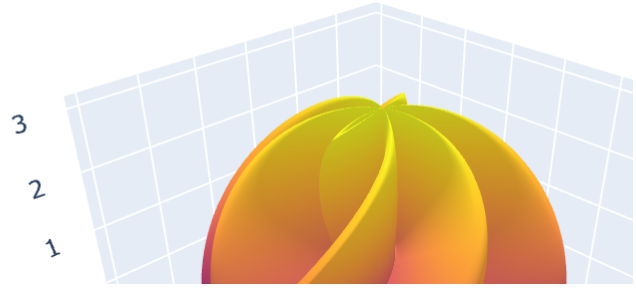
In [12]:

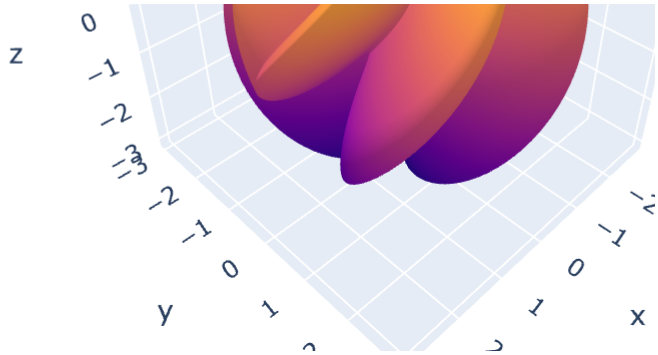
```
#access token from the mapbox
mapbox_access_token = 'pk.eyJ1IjoiamFzZWlkIiwiYSI6ImNrandtazBoMjBqb2gybmxzYmxmeDBtYWYifQ.zC
#Loading the dataframe
df = pd.read_csv('https://raw.githubusercontent.com/plotly/datasets/master/Nuclear%20Waste%
#Takeing the following coluns individually
site_lat = df.lat
site_lon = df.lon
locations_name = df.text
#Bulding the plot using the data lat and long
data = [
    go.Scattermapbox(
        lat=site_lat,
        lon=site_lon,
        mode='markers',
        marker=dict(
            size=17,
            color='rgb(255, 0, 0)',
            opacity=0.7
        ),
        text=locations_name,
        hoverinfo='text'
    ),
    go.Scattermapbox(
        lat=site_lat,
        lon=site_lon,
        mode='markers',
        marker=dict(
            size=8,
            color='rgb(242, 177, 172)',
            opacity=0.7
        ),
        hoverinfo='none'
    )]

#Naming and customizing the plot
layout = go.Layout(
    title='Nuclear Waste Sites on Campus',
    autosize=True,
    hovermode='closest',
    showlegend=False,
    mapbox=dict(
        accesstoken=mapbox_access_token,
        bearing=0,
        center=dict(
            lat=38,
            lon=-94
        ),
        pitch=0,
        zoom=3,
        style='light'
    ),
)

fig = dict(data=data, layout=layout)

iplot(fig, filename='jupyter-Nuclear Waste Sites on American Campuses')
```

Nuclear Waste Sites on Campus

In [14]:

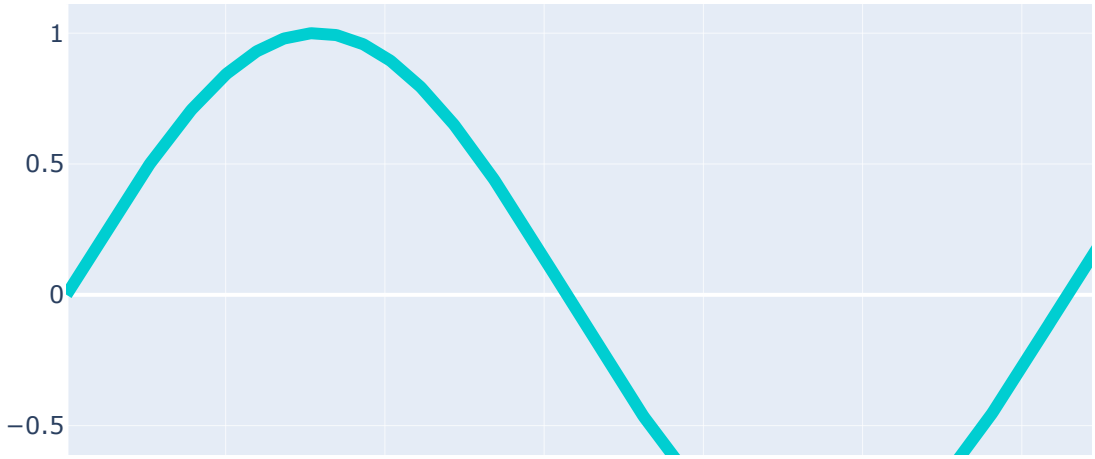
```
#Creating a list of numbers that are important for the form of the viz
s = np.linspace(0, 2 * np.pi, 240)
t = np.linspace(0, np.pi, 240)
tGrid, sGrid = np.meshgrid(s, t)

#Here, we are applying some mathematical equation to get the 3 variables needed to plot the
r = 2 + np.sin(7 * sGrid + 5 * tGrid)  # r = 2 + sin(7s+5t)
x = r * np.cos(sGrid) * np.sin(tGrid)  # x = r*cos(s)*sin(t)
y = r * np.sin(sGrid) * np.sin(tGrid)  # y = r*sin(s)*sin(t)
z = r * np.cos(tGrid)                  # z = r*cos(t)

#Building the viz
surface = go.Surface(x=x, y=y, z=z)
data = [surface]
#Changing the layout, with new title,
layout = go.Layout(
    title='Parametric Plot',
    scene=dict(
        xaxis=dict(
            gridcolor='rgb(255, 255, 255)',
            zerolinecolor='rgb(255, 255, 255)',
            showbackground=True,
            backgroundcolor='rgb(230, 230,230)'
        ),
        yaxis=dict(
            gridcolor='rgb(255, 255, 255)',
            zerolinecolor='rgb(255, 255, 255)',
            showbackground=True,
            backgroundcolor='rgb(230, 230,230)'
        ),
        zaxis=dict(
            gridcolor='rgb(255, 255, 255)',
            zerolinecolor='rgb(255, 255, 255)',
            showbackground=True,
            backgroundcolor='rgb(230, 230,230)'
        )
    )
)

fig = go.Figure(data=data, layout=layout)
iplot(fig, filename='jupyter-parametric_plot')
```




z
0
−1
−2
−3
−3
−2
−1
0
1
y
−1
0
1
x

In [67]:

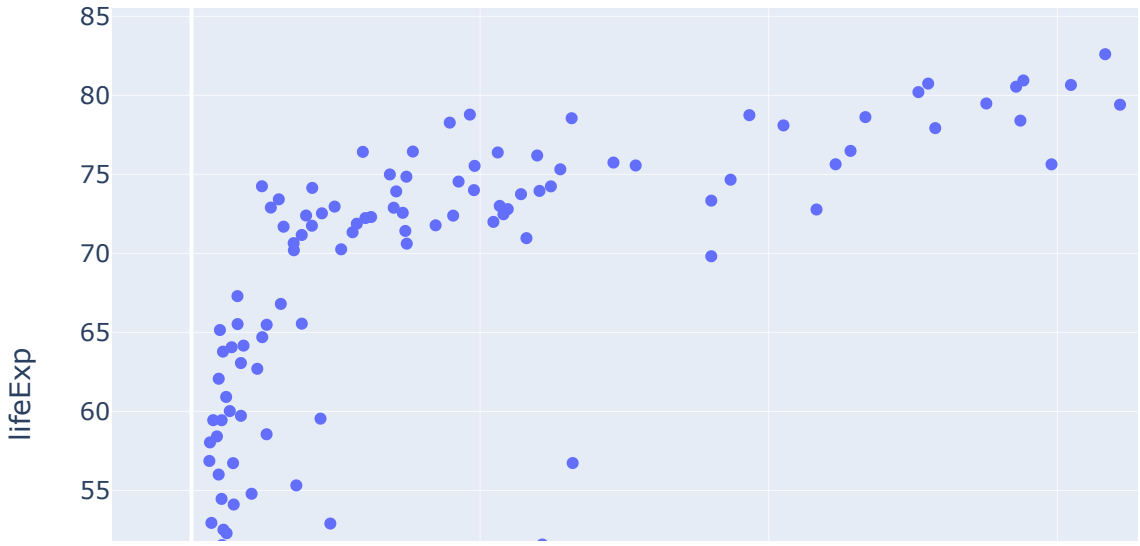
```
data = [dict(
        visible = False,
        line=dict(color='#00CED1', width=6),
        name = 'ν = '+str(step),
        x = np.arange(0,10,0.01),
        y = np.sin(step*np.arange(0,10,0.01))) for step in np.arange(0,5,0.1)]
data[10]['visible'] = True

steps = []
for i in range(len(data)):
    step = dict(
        method = 'restyle',
        args = ['visible', [False] * len(data)],
    )
    step['args'][1][i] = True # Toggle i'th trace to "visible"
    steps.append(step)

sliders = [dict(
    active = 10,
    currentvalue = {"prefix": "Frequency: "},
    pad = {"t": 50},
    steps = steps
)]

layout = dict(sliders=sliders)
fig = dict(data=data, layout=layout)
iplot(fig, filename='Sine Wave Slider')
```



Second Tutorial

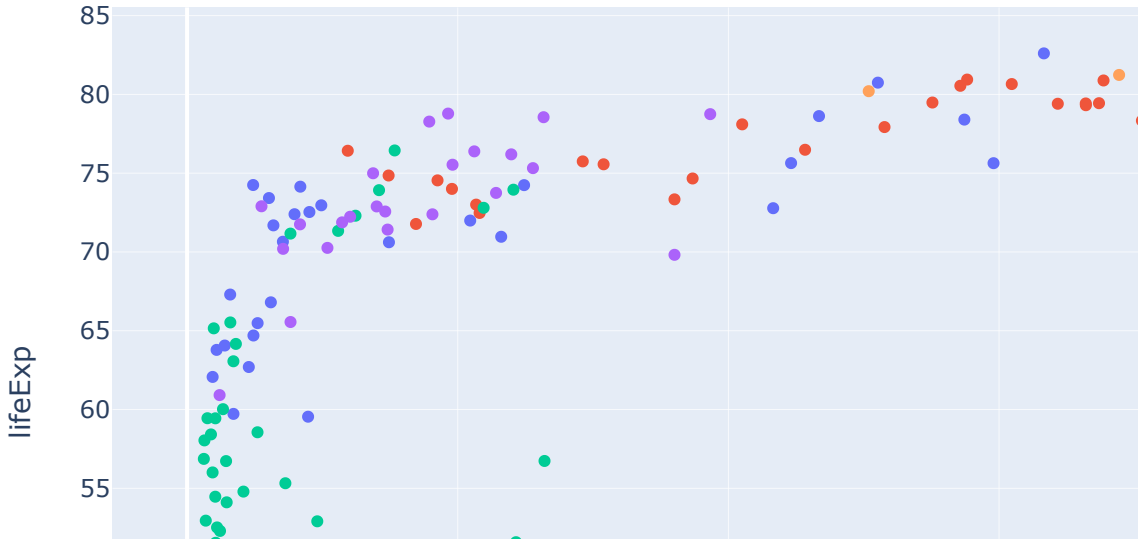
In [68]:

```
import plotly.express as px
#Getting the data
gapminder = px.data.gapminder()
#Geeting only for the year 2007
gapminder2007 = gapminder.query("year == 2007")

#Creating a scatterplot with x the gdppercap and y lifeExp
px.scatter(gapminder2007, x="gdpPercap", y="lifeExp")
```



In [69]:

```
# We added the color to change depending on the continent
px.scatter(gapminder2007, x="gdpPercap", y="lifeExp", color="continent")
```



In [70]:

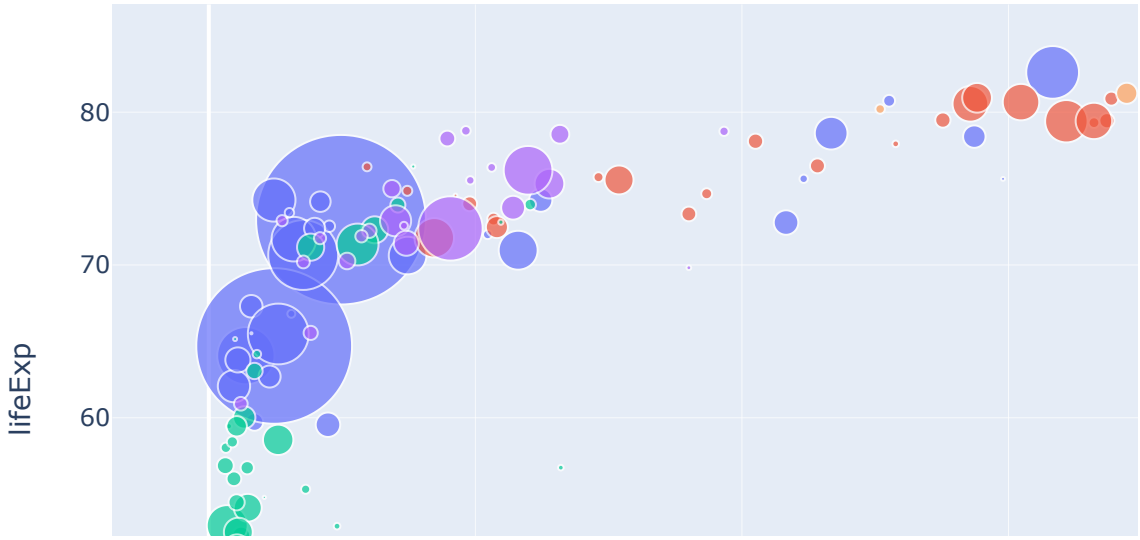
```
gapminder2007.head()
```

Out[70]:

	country	continent	year	lifeExp	pop	gdpPercap	iso_alpha	iso_num
11	Afghanistan	Asia	2007	43.828	31889923	974.580338	AFG	4
23	Albania	Europe	2007	76.423	3600523	5937.029526	ALB	8
35	Algeria	Africa	2007	72.301	33333216	6223.367465	DZA	12
47	Angola	Africa	2007	42.731	12420476	4797.231267	AGO	24
59	Argentina	Americas	2007	75.320	40301927	12779.379640	ARG	32

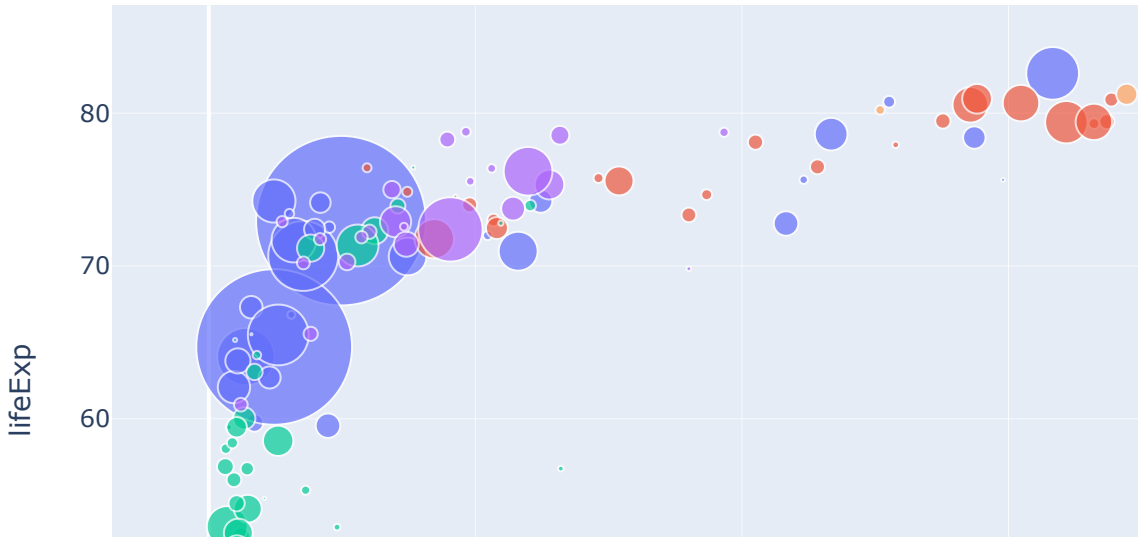
In [71]:

```
# We added the size to depend on the population which is in fact a column. The maximum size
px.scatter(gapminder2007, x="gdpPercap", y="lifeExp", color="continent", size="pop", size_m
```



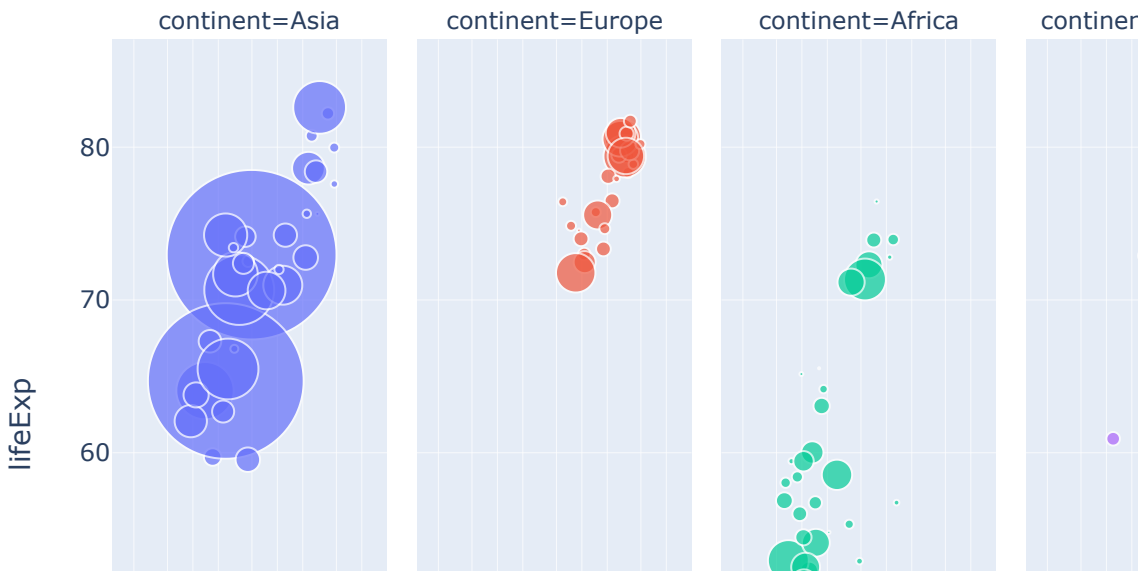
In [72]:

```
#hover_name allows you to check which coutry we are dealing with by moving the mouse to it
px.scatter(gapminder2007, x="gdpPercap", y="lifeExp", color="continent", size="pop", size_m
```



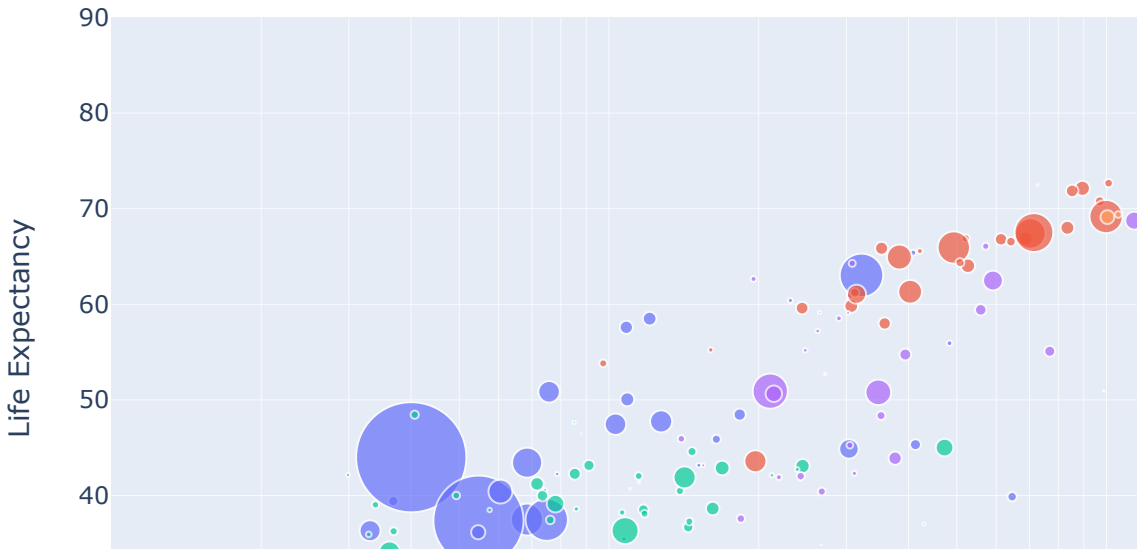
In [73]:

```
#Splitting the data in 6 different scatterplots depending on the continent and activatinhg
px.scatter(gapminder2007, x="gdpPercap", y="lifeExp", color="continent", size="pop", size_m
           hover_name="country", facet_col="continent", log_x=True)
```



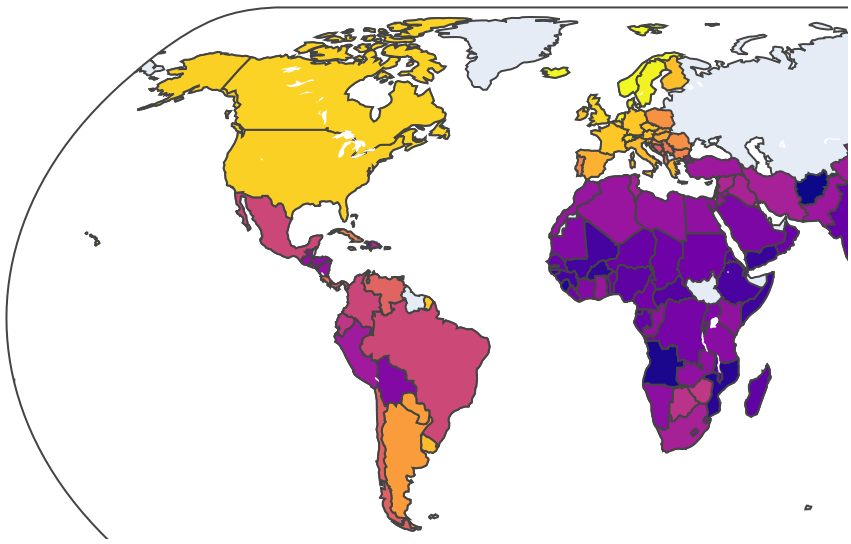
In [74]:

```
# We took this the initial data and made animation_frame and animatio_group to check the ev
px.scatter(gapminder, x="gdpPercap", y="lifeExp",size="pop", size_max=60, color="continent"
           animation_frame="year", animation_group="country", log_x=True, range_x=[100,1000
           labels=dict(pop="Population", gdpPercap="GDP per Capita", lifeExp="Life Expectar
```



In [75]:

```
#Here, we plotted the evolution on the map.
px.choropleth(gapminder, locations="iso_alpha", color="lifeExp", hover_name="country", anim
              color_continuous_scale=px.colors.sequential.Plasma, projection="natural earth
```



In [76]:

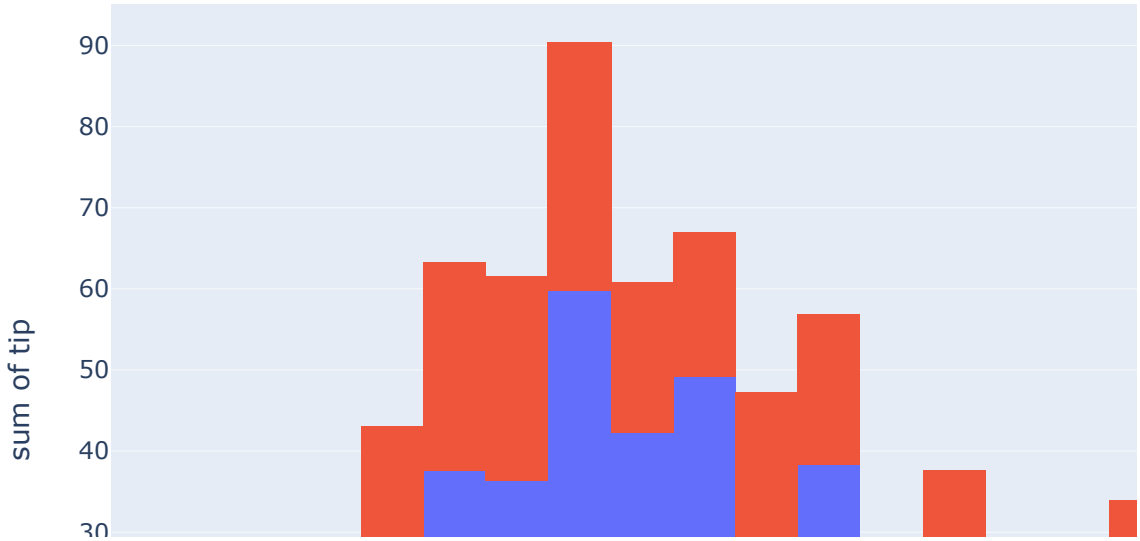
```
#Importing the data and exploring it
tips = px.data.tips()
tips.head()
```

Out[76]:

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

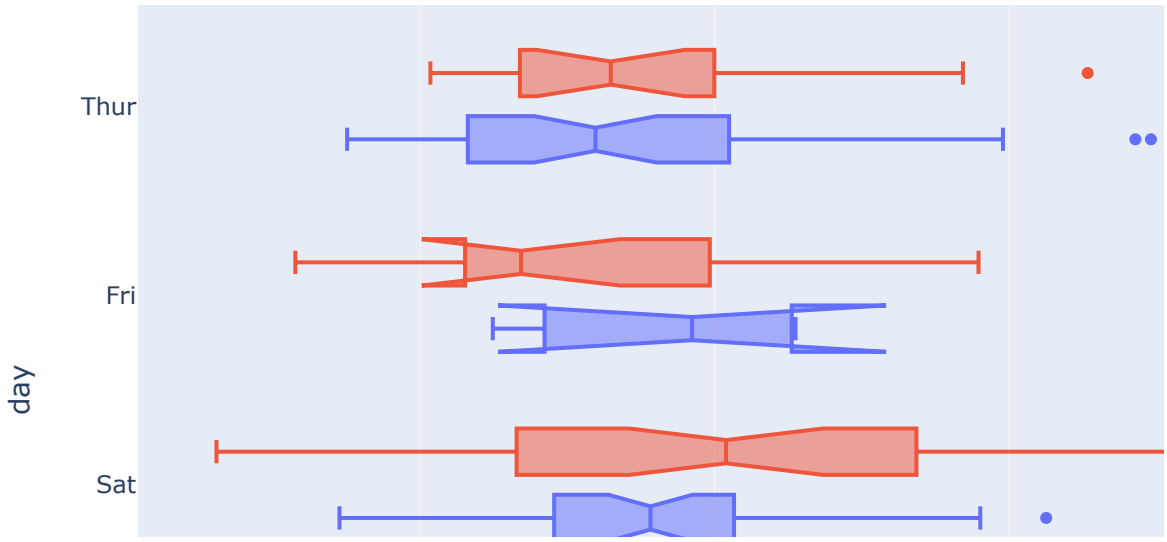
In [77]:

```
#Building a histogram with x as the number of bill and y as to how many has been tipped. Al
px.histogram(tips, x="total_bill", y="tip", histfunc="sum", color="smoker")
```



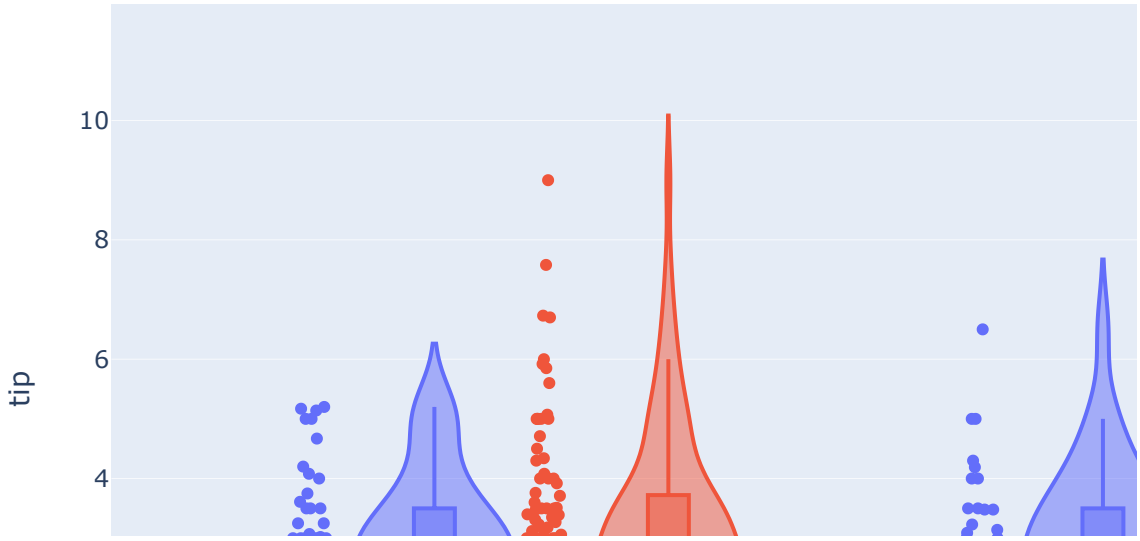
In [78]:

```
#Creating boxplot but the broken down into 4 days and in each day we have two categories. T
#category is giving. Orientation here makes it horizontal. notched makes the boxplot change
px.box(tips, x="total_bill", y="day", orientation="h", color="smoker", notched=True,
       category_orders={"day": ["Thur", "Fri", "Sat", "Sun"]})
```



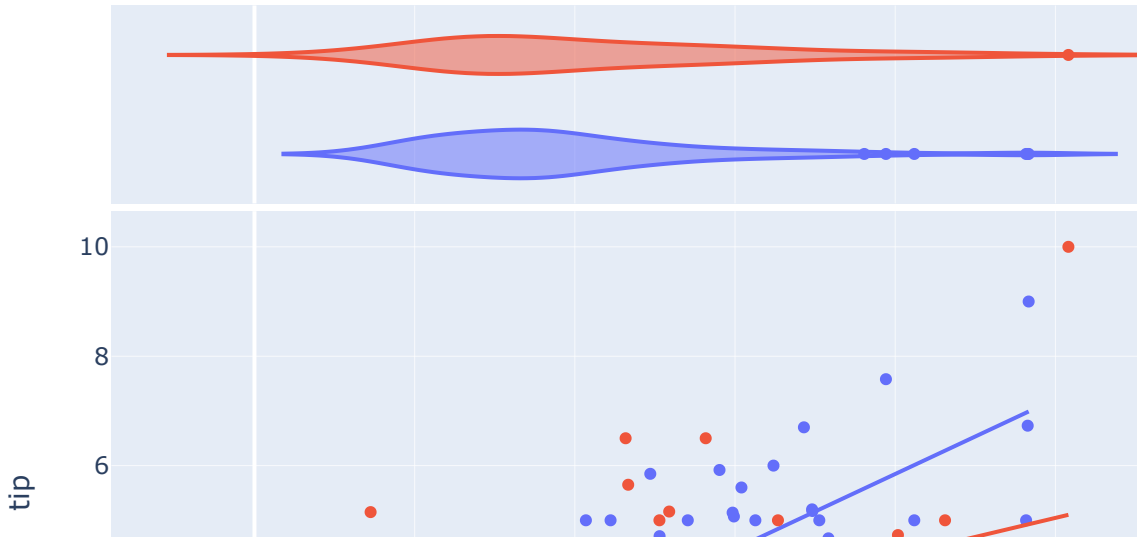
In [79]:

```
px.violin(tips, y="tip", x="smoker", color="sex", box=True, points="all")
```



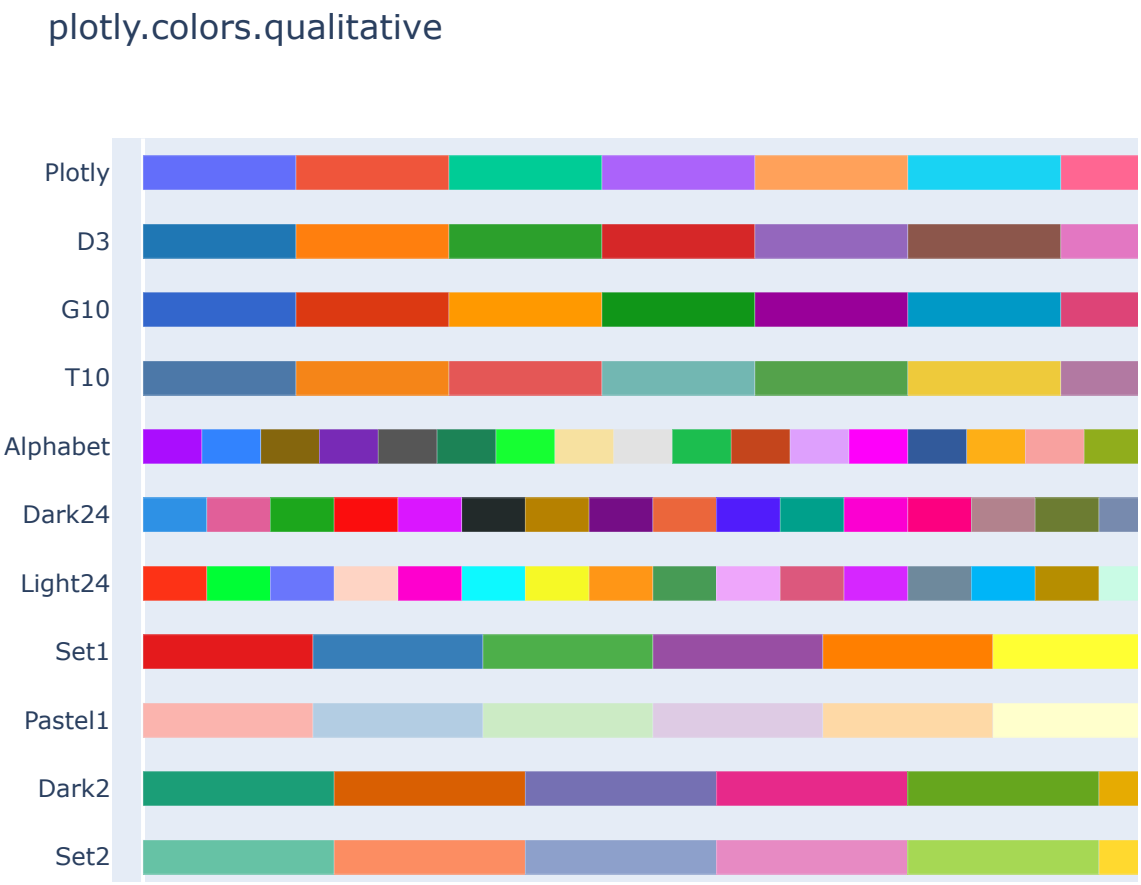
In [80]:

```
#We build a scatter plot to check if there is a linear relation between total_bill and tips
# to this for both categories. Wel aslo did a boxplot and violin using marginal_x and margi
px.scatter(tips, x="total_bill", y="tip", color="smoker", trendline="ols", marginal_x="viol
```



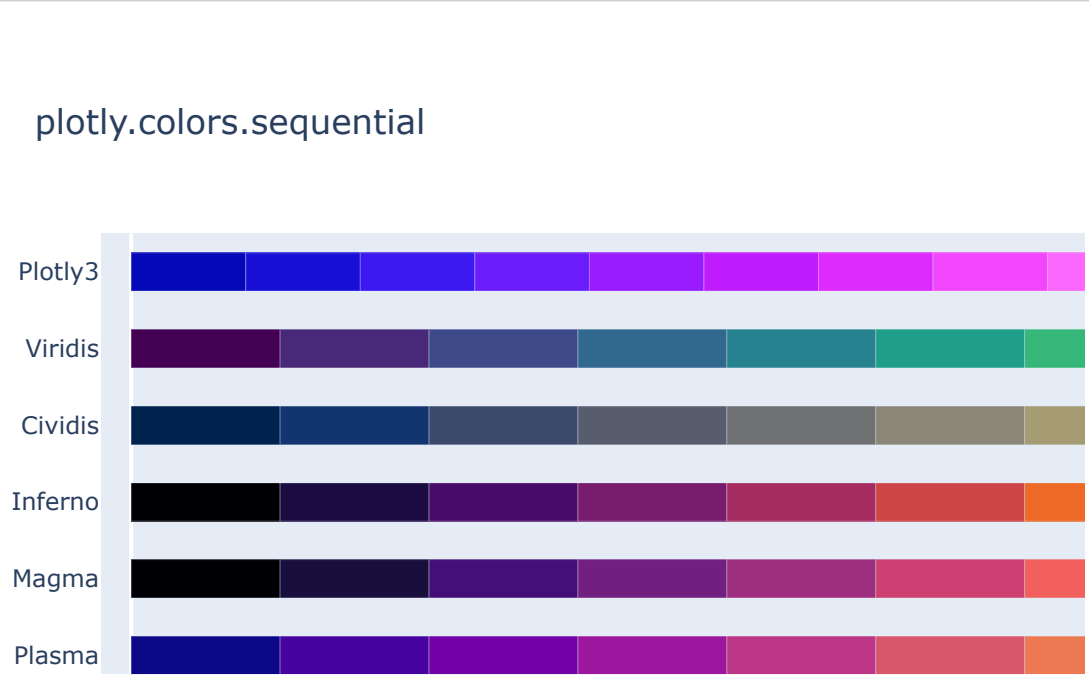
In [81]:

```
px.colors.qualitative.swatches()
```



In [82]:

```
px.colors.sequential.swatches()
```



In [83]:

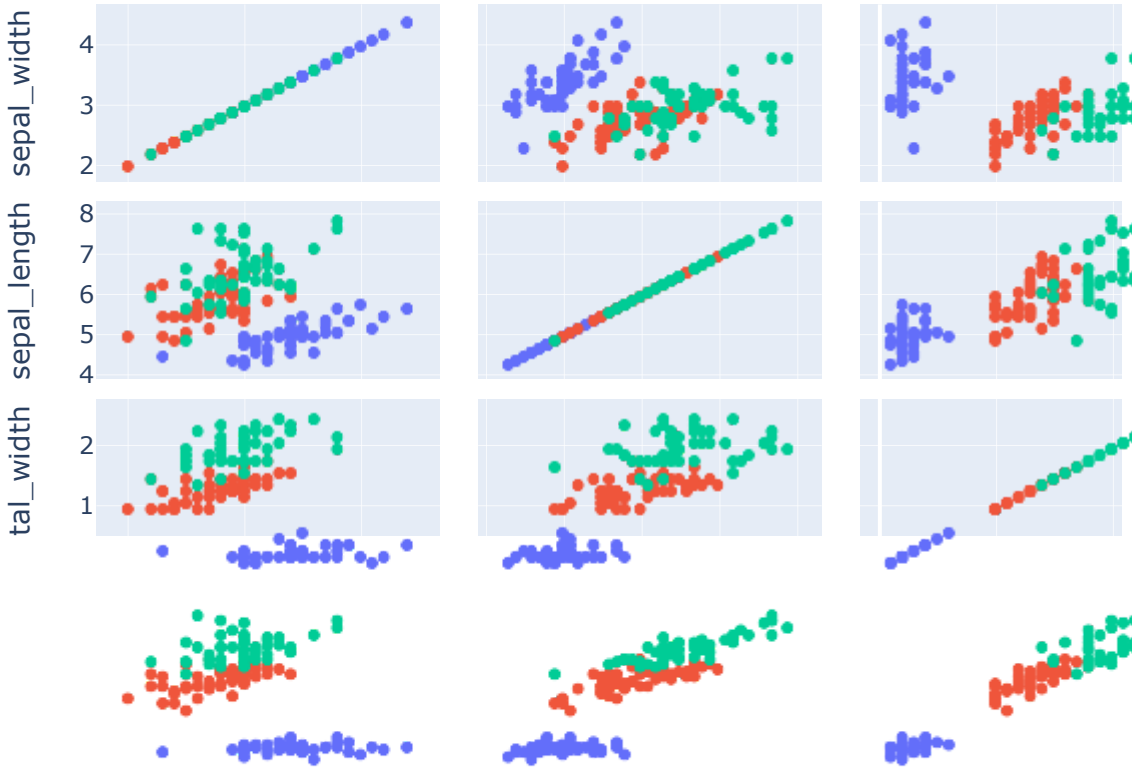
```
iris = px.data.iris()
iris.head()
```

Out[83]:

	sepal_length	sepal_width	petal_length	petal_width	species	species_id
0	5.1	3.5	1.4	0.2	setosa	1
1	4.9	3.0	1.4	0.2	setosa	1
2	4.7	3.2	1.3	0.2	setosa	1
3	4.6	3.1	1.5	0.2	setosa	1
4	5.0	3.6	1.4	0.2	setosa	1

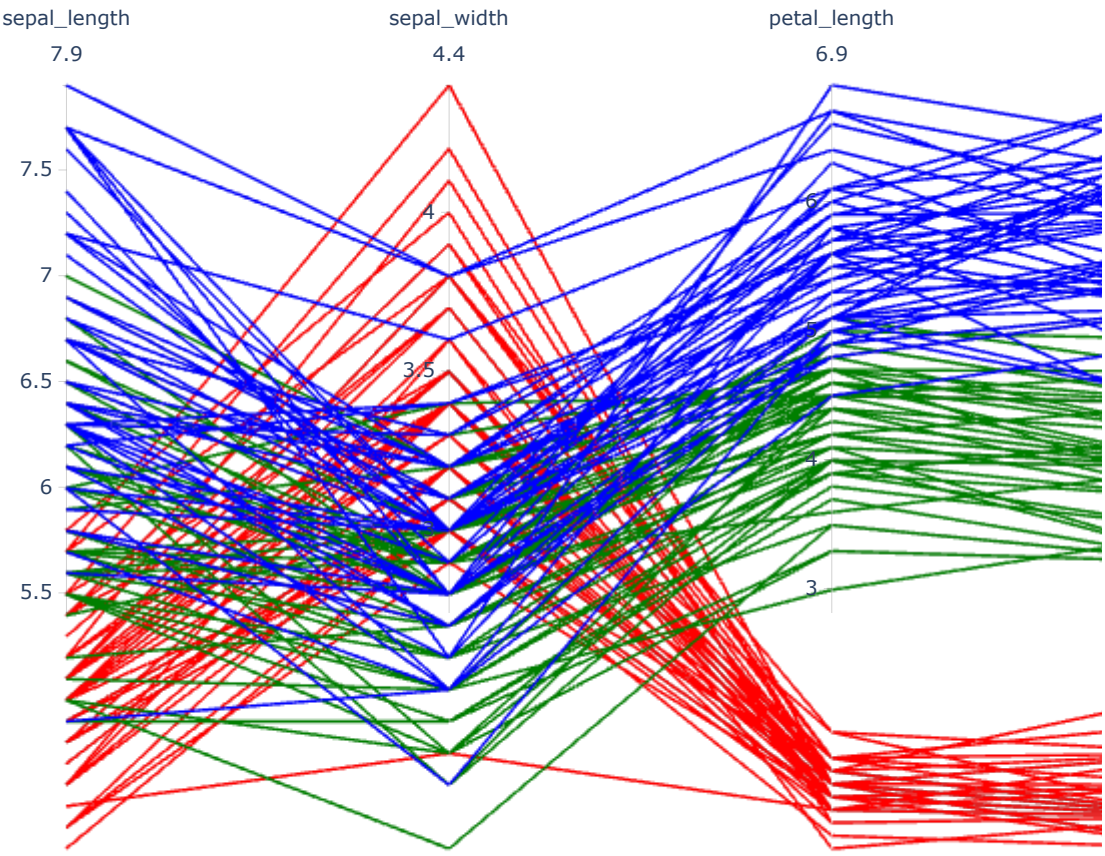
In [84]:

```
px.scatter_matrix(iris, dimensions=["sepal_width","sepal_length","petal_width","petal_lengt
```



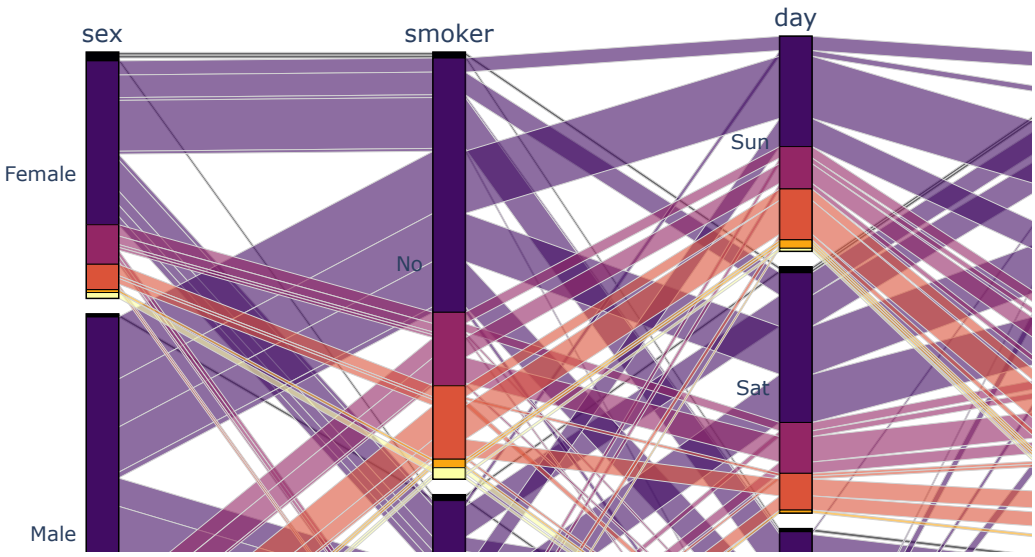
In [85]:

```
px.parallel_coordinates(iris, color="species_id", color_continuous_scale=["red","green","bl
```



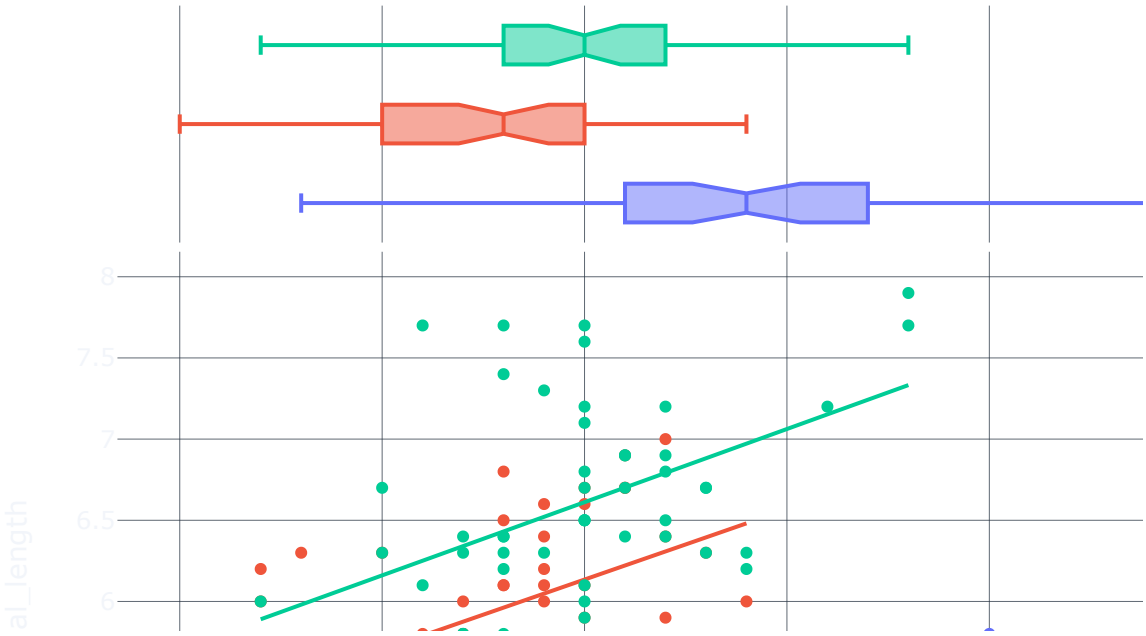
In [86]:

```
px.parallel_categories(tips, color="size", color_continuous_scale=px.colors.sequential.Infe
```



In [87]:

```
px.scatter(iris, x="sepal_width",y="sepal_length",color="species", marginal_y="histogram",
           trendline="ols", template="plotly_dark")
```



In [89]:

```
fig=px.scatter(px.data.iris(), x="sepal_width",y="sepal_length",color="species")
fig.update(layout=dict(legend =dict(orientation="h", y=1.1, x=0.5),
                       annotations=[go.layout.Annotation(text="This one is iteresting", x=3.
```

In [ ]:

In [ ]:

In [ ]:

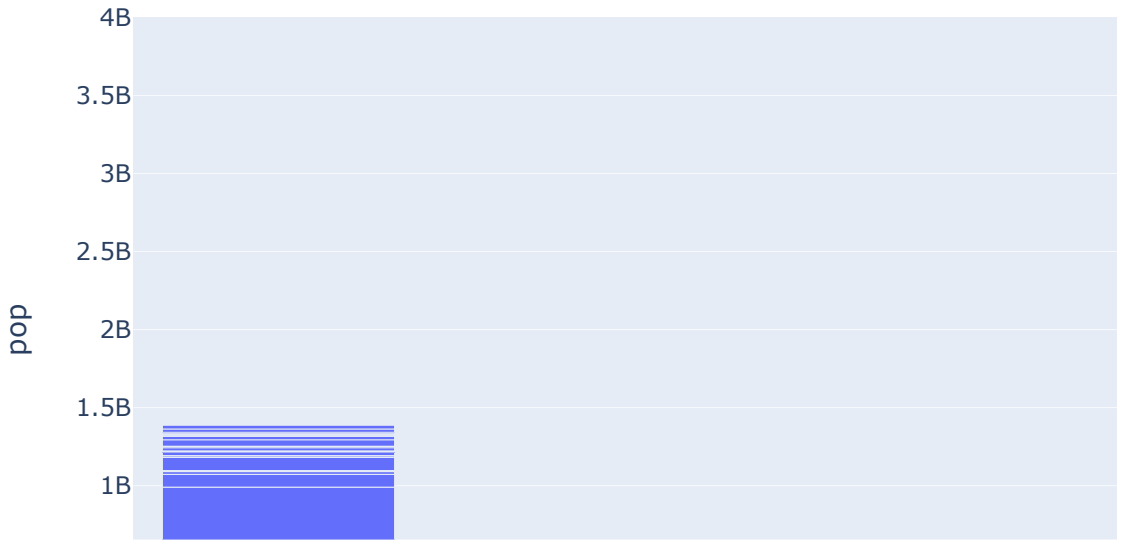
3rd link

In [58]:

```
import dash
import dash_core_components as dcc
import dash_html_components as html
from dash.dependencies import Input, Output
import plotly.express as px
```

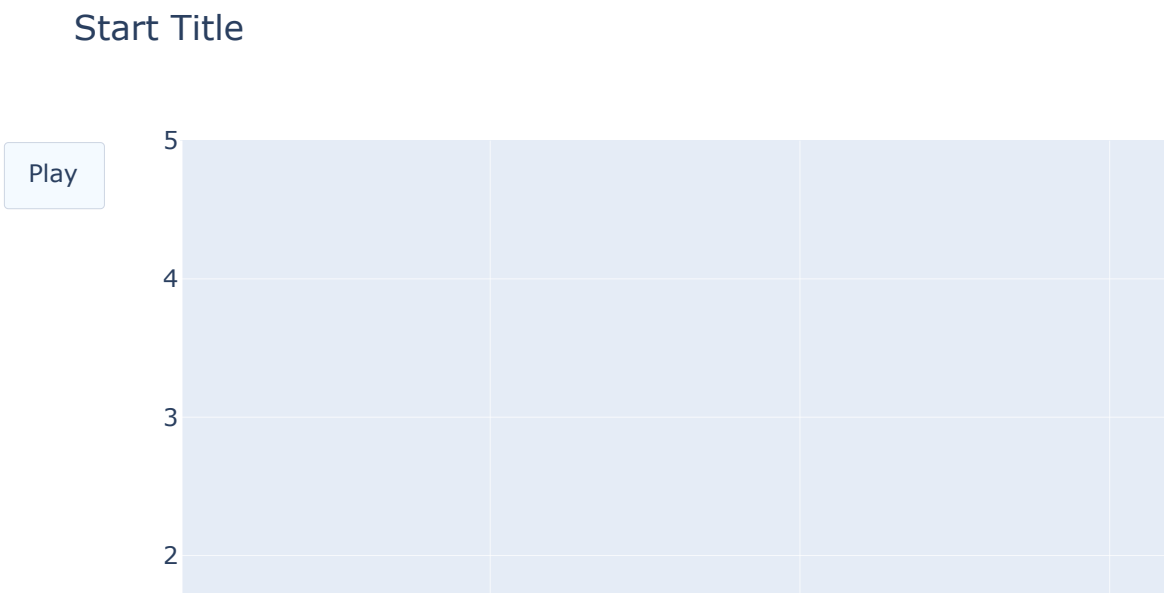
In [59]:

```
import plotly.express as px

df = px.data.gapminder()
#Building a barchart. Also we used animation_fram and animation group to iterate through ye
fig = px.bar(df, x="continent", y="pop", color="continent",
  animation_frame="year", animation_group="country", range_y=[0,4000000000])
fig.show()
```



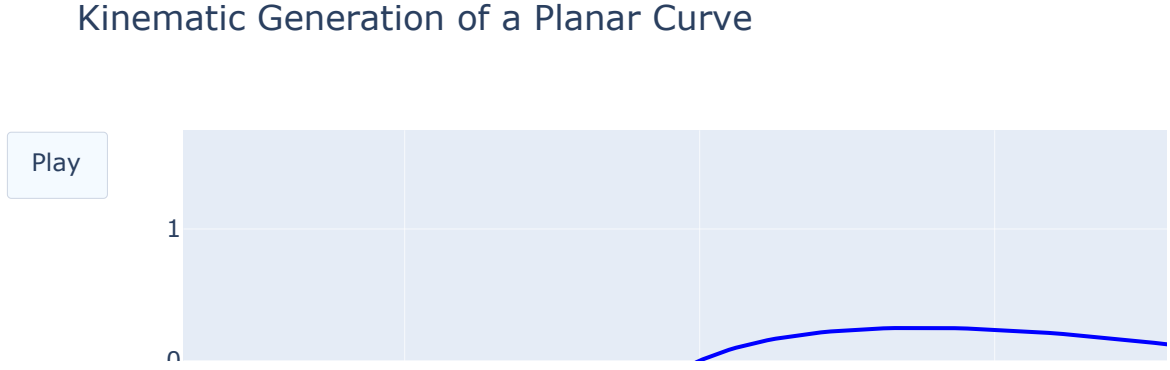
In [60]:

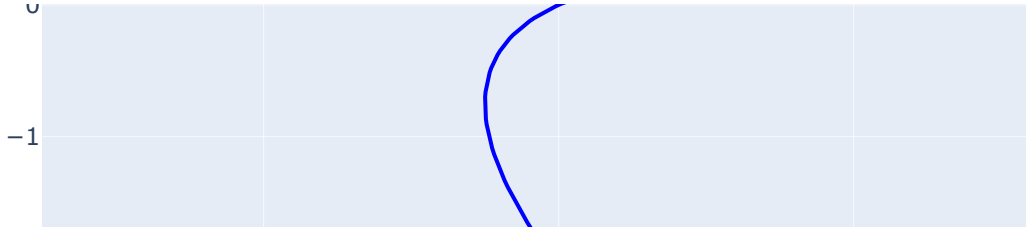
```
import plotly.graph_objects as go

fig = go.Figure(
    data=[go.Scatter(x=[0, 1], y=[0, 1])],# scatter plot with only two points
    layout=go.Layout(
        xaxis=dict(range=[0, 5], autorange=False), #Range is betweent 0 and 5 for x axis
        yaxis=dict(range=[0, 5], autorange=False), #Range is between 0 and 5 for y axis
        title="Start Title", #Title
        updatemenus=[dict(
            type="buttons", #Forming the buttons that we want to press
            buttons=[dict(label="Play", #We name the button play
                          method="animate", # We choose the animate methods
                          args=[None])])]
    ),
    frames=[go.Frame(data=[go.Scatter(x=[1, 2], y=[1, 2])]), #Here we are forming the other
            go.Frame(data=[go.Scatter(x=[1, 4], y=[1, 4])]),
            go.Frame(data=[go.Scatter(x=[3, 4], y=[3, 4])],
                     layout=go.Layout(title_text="End Title"))]
)

fig.show()
```



In [64]:

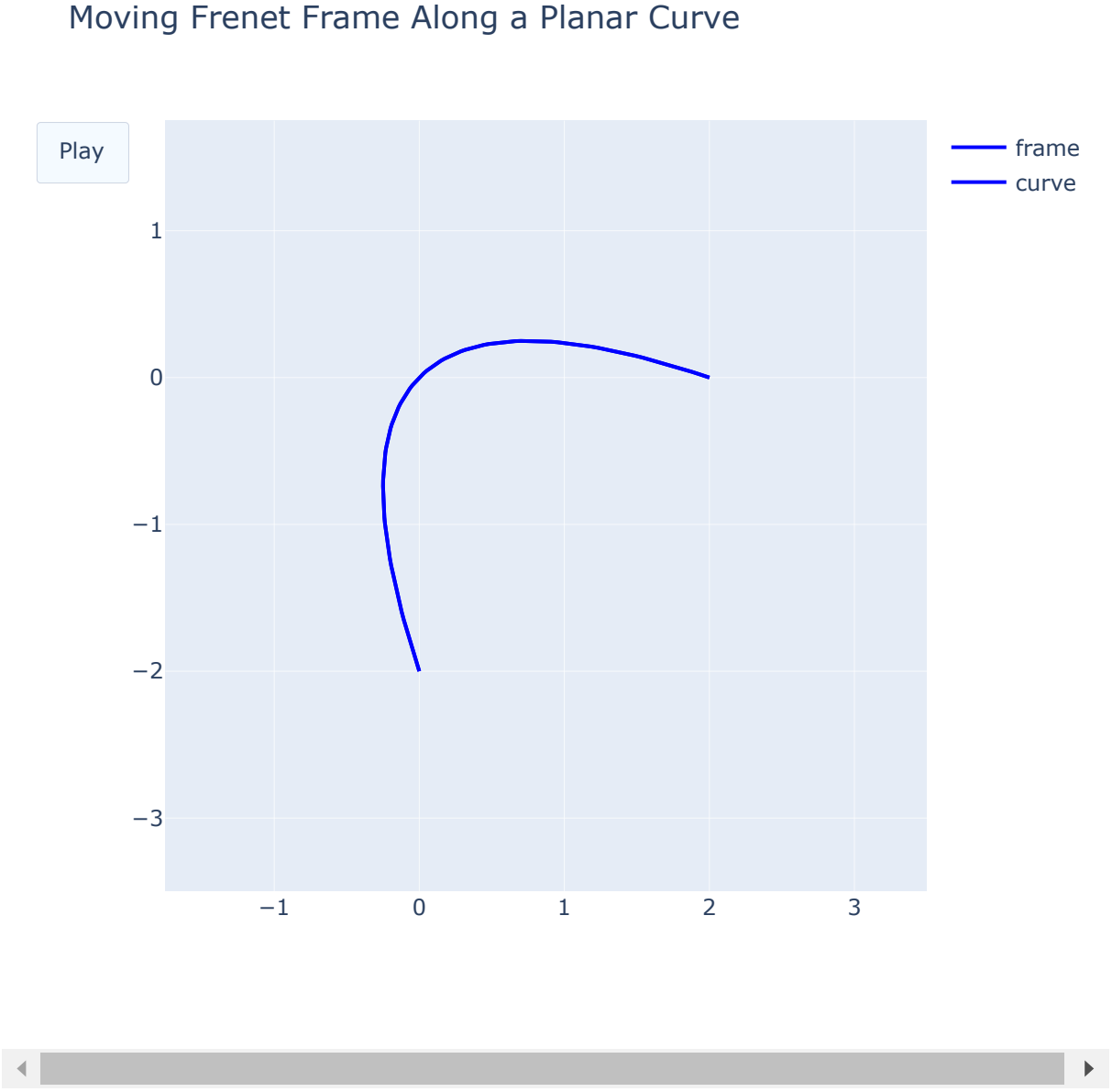
```
import plotly.graph_objects as go

import numpy as np

# Generate curve data
t = np.linspace(-1, 1, 100)
x = t + t ** 2
y = t - t ** 2
xm = np.min(x) - 1.5
xM = np.max(x) + 1.5
ym = np.min(y) - 1.5
yM = np.max(y) + 1.5
N = 50
s = np.linspace(-1, 1, N)
xx = s + s ** 2
yy = s - s ** 2


# Create figure
fig = go.Figure(
    data=[go.Scatter(x=x, y=y, #The data for curve
                     mode="lines",#We want a line
                     line=dict(width=2, color="blue")), #Customizing the line
          go.Scatter(x=x, y=y,
                     mode="lines",
                     line=dict(width=2, color="blue"))],
    layout=go.Layout(
        xaxis=dict(range=[xm, xM], autorange=False, zeroline=False), #Specifying the range
        yaxis=dict(range=[ym, yM], autorange=False, zeroline=False), #Specifying the range
        title_text="Kinematic Generation of a Planar Curve", hovermode="closest",
        updatemenus=[dict(type="buttons", #Creating the button
                          buttons=[dict(label="Play",
                                        method="animate",
                                        args=[None])])]),
    frames=[go.Frame(
        data=[go.Scatter(
            x=[xx[k]],
            y=[yy[k]],
            mode="markers", #The points
            marker=dict(color="red", size=10))]) #Customizing the points

        for k in range(N)]
)

fig.show()
```




0
−1

In [65]:

```
## This is the same as before only difference is that we have the tangente moving

# Generate curve data
t = np.linspace(-1, 1, 100)
x = t + t ** 2
y = t - t ** 2
xm = np.min(x) - 1.5
xM = np.max(x) + 1.5
ym = np.min(y) - 1.5
yM = np.max(y) + 1.5
N = 50
s = np.linspace(-1, 1, N)
xx = s + s ** 2
yy = s - s ** 2
vx = 1 + 2 * s
vy = 1 - 2 * s  # v=(vx, vy) is the velocity
speed = np.sqrt(vx ** 2 + vy ** 2)
ux = vx / speed  # (ux, uy) unit tangent vector, (-uy, ux) unit normal vector
uy = vy / speed

xend = xx + ux  # end coordinates for the unit tangent vector at (xx, yy)
yend = yy + uy

xnoe = xx - uy  # end coordinates for the unit normal vector at (xx,yy)
ynoe = yy + ux


# Create figure
fig = go.Figure(
    data=[go.Scatter(x=x, y=y,
                     name="frame",
                     mode="lines",
                     line=dict(width=2, color="blue")),
          go.Scatter(x=x, y=y,
                     name="curve",
                     mode="lines",
                     line=dict(width=2, color="blue"))
          ],
    layout=go.Layout(width=600, height=600,
                     xaxis=dict(range=[xm, xM], autorange=False, zeroline=False),
                     yaxis=dict(range=[ym, yM], autorange=False, zeroline=False),
                     title="Moving Frenet Frame Along a Planar Curve",
                     hovermode="closest",
                     updatemenus=[dict(type="buttons",
                                       buttons=[dict(label="Play",
                                                     method="animate",
                                                     args=[None])])]),

    frames=[go.Frame(
        data=[go.Scatter(
            x=[xx[k], xend[k], None, xx[k], xnoe[k]],
            y=[yy[k], yend[k], None, yy[k], ynoe[k]],
            mode="lines",
            line=dict(color="red", width=2))
        ]) for k in range(N)]
)

fig.show()
```

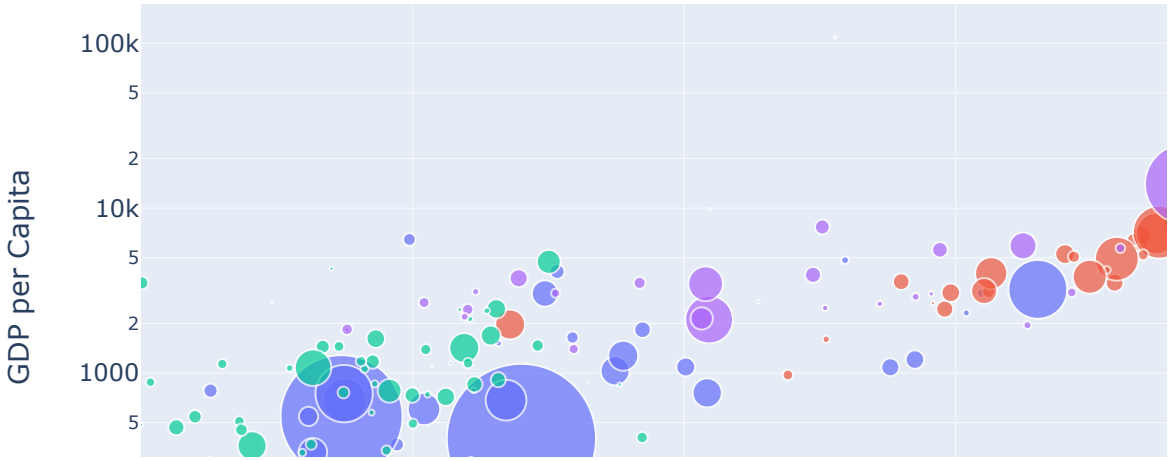

Moving Frenet Frame Along a Planar Curve
Play
frame
curve
1
0
−1
−2
−3
−1
0
1
2
3

In [66]:

```
import plotly.graph_objects as go

import pandas as pd

url = "https://raw.githubusercontent.com/plotly/datasets/master/gapminderDataFiveYear.csv"
dataset = pd.read_csv(url)

years = ["1952", "1962", "1967", "1972", "1977", "1982", "1987", "1992", "1997", "2002",
         "2007"]

# make list of continents
continents = []
for continent in dataset["continent"]:
    if continent not in continents:
        continents.append(continent)
# make figure
fig_dict = {
    "data": [],
    "layout": {},
    "frames": []
}

# fill in most of layout
fig_dict["layout"]["xaxis"] = {"range": [30, 85], "title": "Life Expectancy"}
fig_dict["layout"]["yaxis"] = {"title": "GDP per Capita", "type": "log"}
fig_dict["layout"]["hovermode"] = "closest"
fig_dict["layout"]["updatemenus"] = [
    {
        "buttons": [
            {
                "args": [None, {"frame": {"duration": 500, "redraw": False},
                                "fromcurrent": True, "transition": {"duration": 300,
                                                                    "easing": "quadratic-ir
                "label": "Play",
                "method": "animate"
            },
            {
                "args": [[None], {"frame": {"duration": 0, "redraw": False},
                                  "mode": "immediate",
                                  "transition": {"duration": 0}}],
                "label": "Pause",
                "method": "animate"
            }
        ],
        "direction": "left",
        "pad": {"r": 10, "t": 87},
        "showactive": False,
        "type": "buttons",
        "x": 0.1,
        "xanchor": "right",
        "y": 0,
        "yanchor": "top"
    }
]

sliders_dict = {
    "active": 0,
    "yanchor": "top",
    "xanchor": "left",
```

```
    "currentvalue": {
        "font": {"size": 20},
        "prefix": "Year:",
        "visible": True,
        "xanchor": "right"
    },
    "transition": {"duration": 300, "easing": "cubic-in-out"},
    "pad": {"b": 10, "t": 50},
    "len": 0.9,
    "x": 0.1,
    "y": 0,
    "steps": []
}

# make data
year = 1952
for continent in continents:
    dataset_by_year = dataset[dataset["year"] == year]
    dataset_by_year_and_cont = dataset_by_year[
        dataset_by_year["continent"] == continent]

    data_dict = {
        "x": list(dataset_by_year_and_cont["lifeExp"]),
        "y": list(dataset_by_year_and_cont["gdpPercap"]),
        "mode": "markers",
        "text": list(dataset_by_year_and_cont["country"]),
        "marker": {
            "sizemode": "area",
            "sizeref": 200000,
            "size": list(dataset_by_year_and_cont["pop"])
        },
        "name": continent
    }
    fig_dict["data"].append(data_dict)

# make frames
for year in years:
    frame = {"data": [], "name": str(year)}
    for continent in continents:
        dataset_by_year = dataset[dataset["year"] == int(year)]
        dataset_by_year_and_cont = dataset_by_year[
            dataset_by_year["continent"] == continent]

        data_dict = {
            "x": list(dataset_by_year_and_cont["lifeExp"]),
            "y": list(dataset_by_year_and_cont["gdpPercap"]),
            "mode": "markers",
            "text": list(dataset_by_year_and_cont["country"]),
            "marker": {
                "sizemode": "area",
                "sizeref": 200000,
                "size": list(dataset_by_year_and_cont["pop"])
            },
            "name": continent
        }
        frame["data"].append(data_dict)

    fig_dict["frames"].append(frame)
    slider_step = {"args": [
        [year],
        {"frame": {"duration": 300, "redraw": False},
```

```
            "mode": "immediate",
            "transition": {"duration": 300}}
        ],
            "label": year,
            "method": "animate"}
        sliders_dict["steps"].append(slider_step)


fig_dict["layout"]["sliders"] = [sliders_dict]

fig = go.Figure(fig_dict)

fig.show()
```



In [ ]: