## About Plotly

Plotly is a Data Viz library by the company Plotly based out of Canada with support in languages such as Python, Js, Julia etc.

### Advantages

- Multi language support
- Lot's of graphs
- Interactive plots
- Beautiful plots

Does not work with live data streams. Dash can be explored for that.

## The Plotly Roadmap

- Plotly Go
- Plotly Express
- Dash

In [1]:
```
# import the libraries
import plotly.graph_objects as go
import numpy as np
import pandas as pd
import plotly.express as px
```

In [2]:
```
!pip install plotly
```

```
Requirement already satisfied: plotly in c:\users\user\anaconda3\lib\site-packages (5.13.1)
Requirement already satisfied: tenacity>=6.2.0 in c:\users\user\anaconda3\lib\site-packages (from plotly)
(8.2.2)
```

In [3]:
```
# import datasets
tips = px.data.tips()
iris = px.data.iris()
gap = px.data.gapminder()
```

In [4]:
```
gap.head()
```

Out[4]:

| | country | continent | year | lifeExp | pop | gdpPercap | iso_alpha | iso_num |
|---|---|---|---|---|---|---|---|---|
| **0** | Afghanistan | Asia | 1952 | 28.801 | 8425333 | 779.445314 | AFG | 4 |
| **1** | Afghanistan | Asia | 1957 | 30.332 | 9240934 | 820.853030 | AFG | 4 |
| **2** | Afghanistan | Asia | 1962 | 31.997 | 10267083 | 853.100710 | AFG | 4 |
| **3** | Afghanistan | Asia | 1967 | 34.020 | 11537966 | 836.197138 | AFG | 4 |
| **4** | Afghanistan | Asia | 1972 | 36.088 | 13079460 | 739.981106 | AFG | 4 |

## Scatter plot

we define the x and y values for the scatter plot, create a scatter plot using go.Scatter(), customize the plot layout using update_layout(), and finally display the plot using fig.show()

In [5]:
```
# scatter plot using plotly go

temp_df = gap[gap['year'] == 2007]
temp_df
```

Out[5]:

| | 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 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... |
| **1655** | Vietnam | Asia | 2007 | 74.249 | 85262356 | 2441.576404 | VNM | 704 |
| **1667** | West Bank and Gaza | Asia | 2007 | 73.422 | 4018332 | 3025.349798 | PSE | 275 |
| **1679** | Yemen, Rep. | Asia | 2007 | 62.698 | 22211743 | 2280.769906 | YEM | 887 |
| **1691** | Zambia | Africa | 2007 | 42.384 | 11746035 | 1271.211593 | ZMB | 894 |
| **1703** | Zimbabwe | Africa | 2007 | 43.487 | 12311143 | 469.709298 | ZWE | 716 |

142 rows × 8 columns

In [6]:
```
trace1 = go.Scatter(x=temp_df['lifeExp'],y=temp_df['gdpPercap'],mode='markers')
trace2 = go.Scatter(x=[0,1,2],y=[0,90,30000],mode='lines')

data = [trace1,trace2]

layout = go.Layout(title='Life Exp Vs GDP per Capita for 2007',
                   xaxis={'title':'Life Exp'},yaxis={'title':'GDP'})
fig = go.Figure(data,layout)

fig.show()
```

Life Exp Vs GDP per Capita for 2007
GDP
50k
40k
30k
20k
10k
0

In [7]:
```
# plot life exp and gdp scatter plot -> continent as color -> pop as size -> hover name -> range_x/range_y ->

px.scatter(temp_df, x='lifeExp', y='gdpPercap',
           color='continent',size='pop',size_max=100, hover_name='country')
```

## Plot animation

plot animation is often used to show the evolution of data or to highlight trends and patterns over time. It can be achieved using various libraries and techniques, such as Plotly, Matplotlib, or GIF creation tools, to create visually engaging and informative animated plots.

In [8]:
```
# plot animation of the above curve on the basic of year

px.scatter(gap, x='lifeExp', y='gdpPercap',
           color='continent',size='pop',
           size_max=100, hover_name='country',
           range_x=[30,95],
           animation_frame='year',animation_group='country')
```

## Line Plot

A line plot is a graphical representation of data points connected by straight lines to show the relationship and trends between two continuous variables.

In [9]:
```
# line plot

# plot india pop line plot

temp_df = gap[gap['country'] == 'India']

px.line(temp_df, x='year', y='pop',title='India pop growth')
```

In [10]:
```
# plot india china pak line plot
temp_df = gap[gap['country'].isin(['India','China','Pakistan'])].pivot(index=
                                    'year',columns='country',values='lifeExp')
temp_df
```

Out[10]:

| country | China | India | Pakistan |
|---|---|---|---|
| year | | | |
| **1952** | 44.00000 | 37.373 | 43.436 |
| **1957** | 50.54896 | 40.249 | 45.557 |
| **1962** | 44.50136 | 43.605 | 47.670 |
| **1967** | 58.38112 | 47.193 | 49.800 |
| **1972** | 63.11888 | 50.651 | 51.929 |
| **1977** | 63.96736 | 54.208 | 54.043 |
| **1982** | 65.52500 | 56.596 | 56.158 |
| **1987** | 67.27400 | 58.553 | 58.245 |
| **1992** | 68.69000 | 60.223 | 60.838 |
| **1997** | 70.42600 | 61.765 | 61.818 |
| **2002** | 72.02800 | 62.879 | 63.610 |
| **2007** | 72.96100 | 64.698 | 65.483 |

In [11]:
```
px.line(temp_df, x=temp_df.index, y=temp_df.columns)
```

## 3D Line plot

A 3D line plot is a visualization technique that represents data using lines in a three-dimensional space, showing the relationship between three variables in a single plot.

In [12]:
```
# 3D Line plot

# data to be plotted
df = px.data.tips()

# plotting the figure
fig = px.line_3d(df, x="sex", y="day", z="time", color="sex")

fig.show()
```

## Bar chart

A bar chart is a graphical representation that uses rectangular bars to compare categories or groups of data based on their values.

In [13]:
```
# bar chart

# india's pop over the years

temp_df = gap[gap['country'] == 'India']

px.bar(temp_df,x='year',y='pop')
```

In [14]:
```
# pop comp of 3 countries

temp_df = gap[gap['country'].isin(['India','China','Pakistan'])].pivot(index=
                                    'year',columns='country',values='pop')
temp_df
```

Out[14]:

| country | China | India | Pakistan |
|---|---|---|---|
| year | | | |
| **1952** | 556263527 | 372000000 | 41346560 |
| **1957** | 637408000 | 409000000 | 46679944 |
| **1962** | 665770000 | 454000000 | 53100671 |
| **1967** | 754550000 | 506000000 | 60641899 |
| **1972** | 862030000 | 567000000 | 69325921 |
| **1977** | 943455000 | 634000000 | 78152686 |
| **1982** | 1000281000 | 708000000 | 91462088 |
| **1987** | 1084035000 | 788000000 | 105186881 |
| **1992** | 1164970000 | 872000000 | 120065004 |
| **1997** | 1230075000 | 959000000 | 135564834 |
| **2002** | 1280400000 | 1034172547 | 153403524 |
| **2007** | 1318683096 | 1110396331 | 169270617 |

## Grouped bar chart

A grouped bar chart is a type of visualization that displays multiple bars grouped together, where each group represents a different category or subcategory, allowing for easy comparison between the groups.

In [15]:
```
# grouped bar chart -> text_auto

px.bar(temp_df,    x=temp_df.index,     y=temp_df.columns,
       barmode='group',  log_y=True,  text_auto=True)

# log_y (boolean (default False)) - If True,
# the y-axis is log-scaled in cartesian coordinates.

# text_auto (bool or string (default False)) -
# If True or a string, the x or y or z values will be displayed as text,
# depending on the orientation A string like '.2f' will be interpreted as
# a texttemplate numeric formatting directive.
```

## Stacked bar chart

A stacked bar chart is a type of visualization that displays multiple bars stacked on top of each other, where each bar represents a category or subgroup, and the height of the bar represents the value or proportion associated with that category/subgroup.

In [16]:
```
# stacked bar chart
# pop contribution per country to a continents pop stacked
# for a particular year(2007)

temp_df = gap[gap['year'] == 2007]

px.bar(temp_df, x='continent', y='pop', color='country',log_y=True)
```

## bar chart animation

Bar chart animation refers to the process of creating a dynamic visualization where bar charts change over time to represent evolving data trends.

In [17]:
```
# bar chart animation

px.bar(gap, x='continent',y='pop',color='continent',
       animation_frame='year',animation_group='country',range_y=[0,4000000000])
```

## boxplot

A boxplot is a compact visualization that displays the summary statistics of a dataset, including the median, quartiles, and outliers, using a box-and-whisker representation.

In [18]:
```
# boxplot

# using the tips dataset
df = px.data.tips()

# plotting the box chart
fig = px.box(df, x="day", y="total_bill")

# showing the plot
fig.show()
```

## histogram

A histogram is a graphical representation that organizes data into bins and displays the frequency or count of occurrences of each bin.

In [19]:
```
# histogram
# plot histogram of life expt of all countries in 2007 -> nbins -> text_auto

temp_df = gap[gap['year'] == 2007]

px.histogram(temp_df, x='lifeExp',nbins=10,text_auto=True)
```

## Gantt Chart

Gantt refers to a type of bar chart that visually represents project schedules or timelines by showing tasks or activities as horizontal bars along a time axis.

In [20]:
```
# Gantt Chart

import plotly.figure_factory as ff

# Data to be plotted
df = [dict(Task="A", Start='2020-01-01', Finish='2009-02-02'),
    dict(Task="Job B", Start='2020-03-01', Finish='2020-11-11'),
    dict(Task="Job C", Start='2020-08-06', Finish='2020-09-21')]

# Creating the plot
fig = ff.create_gantt(df)
fig.show()
```

Gantt Chart
1w
1m
6m
YTD
1y
all
Job C
Job B
A

In [21]:
```
# plot histogram of sepal length of all iris species

px.histogram(iris,x='sepal_length',color='species',nbins=30,text_auto=True)
```

## Pie chart

A pie chart is a circular statistical graphic that represents data as slices of a pie, with each slice representing a proportion or percentage of the whole.

In [22]:
```
# Pie -> values -> names
# find the pie chart of pop of european countries in 2007

temp_df = gap[(gap['year'] == 2007) & (gap['continent'] == 'Europe')]

px.pie(temp_df, values='pop', names='country')
```

In [23]:
```
# plot pie chart of world pop in 1952 continent wise ->  -> explode(pull)

temp_df = gap[gap['year'] == 1952].groupby('continent')['pop'].sum().reset_index()

px.pie(temp_df, values='pop', names='continent')
```

## Sunburst plot

A Sunburst plot in Plotly is a circular hierarchical visualization that represents data in a radial form, showing the hierarchy of categories through concentric circles and arcs.

In [24]:
```
# Sunburst plot -> Sunburst plots visualize hierarchical data
# spanning outwards radially from root to leaves. -> color
# path -> [], values

temp_df = gap[gap['year'] == 2007]

px.sunburst(temp_df, path=['continent','country'],values='pop',color='lifeExp')
```

In [25]:
```
px.sunburst(tips,path=['sex','smoker','day','time'],values='total_bill',
            color='size')
```

## Tree Map

A treemap in Plotly is a type of interactive data visualization that represents hierarchical data using nested rectangles, with each rectangle's size proportional to a specific attribute or value associated with it.

In [26]:
```
# Treemap

temp_df = gap[gap['year'] == 2007]

px.treemap(temp_df, path=[px.Constant('World'),'continent','country'],
           values='pop',color='lifeExp')
```

## Heatmap

Heat map in Plotly refers to a graphical representation of data where values are depicted using colors on a grid-like structure, allowing for easy visualization and analysis of patterns, trends, and variations in the data.

In [27]:
```
# Heatmap -> find heatmap of all continents with year on avg life exp

temp_df = gap.pivot_table(index='year',columns='continent',
                          values='lifeExp',aggfunc='mean')
px.imshow(temp_df)
```

## 3D scatter plot

A 3D scatter plot in Plotly is a visual representation of data points in a three-dimensional space, where each point is defined by its x, y, and z coordinates, allowing for the visualization of relationships and patterns in three dimensions.

In [28]:
```
# 3d scatterplot

# plot a 3d scatter plot of all country data for 2007

temp_df = gap[gap['year'] == 2007]

px.scatter_3d(temp_df, x='lifeExp',y='pop',z='gdpPercap',
              log_y=True,color='continent',hover_name='country')
```

In [29]:
```
px.scatter_3d(iris,x='sepal_length',y='sepal_width',
              z='petal_length',color='species')
```

## Scatter_matrix

A scatter matrix in Plotly is a compact grid of scatter plots that displays the relationships and correlations between multiple variables in a single visualization.

In [30]:
```
# scatter_matrix -> dimensions

px.scatter_matrix(iris,dimensions=['sepal_length','sepal_width','petal_length','petal_width']
                  ,color='species')
```

## Facet Plot

Facet plot in Plotly refers to the visualization technique that divides a dataset into subsets based on one or more categorical variables, creating a grid of smaller plots for each subset, allowing for easy comparison and exploration of relationships between variables.

In [31]:
```
# facet plot

px.scatter(tips, x='total_bill', y='tip', facet_col='smoker',
           facet_row='sex',color='time')
```

In [32]:
```
px.histogram(tips,x='total_bill',facet_row='sex')
```

In [33]:
```
px.scatter(gap, x='lifeExp', y='gdpPercap', facet_col='year', facet_col_wrap=3)
```

**3D Surface plot**

A 3D surface plot in Plotly refers to the visualization of data as a three-dimensional surface, where the x, y, and z values are plotted in a three-dimensional space to represent the variation of a variable across multiple dimensions.

In [34]:
```
# 3d Surface plot
# can not be created using Plotly express
# we will use plotly graph object -> go

x = np.linspace(-10,10,100)
y = np.linspace(-10,10,100)

xx, yy = np.meshgrid(x,y)

z = xx**2 + yy**2
# z = np.sin(xx) + np.tan(yy)
# z = np.sqrt(xx**2 + yy**2)


trace = go.Surface(x=x,y=y,z=z)

data = [trace]

layout = go.Layout(title='3D Surface Plot')

fig = go.Figure(data,layout)

fig.show()
```

## 3D Surface Plot

In [35]:
```
# 3D surface Plot

# Data to be plotted
x = np.outer(np.linspace(-2, 2, 30), np.ones(30))
y = x.copy().T
z = np.cos(x ** 2 + y ** 2)

# plotting the figure
fig = go.Figure(data=[go.Surface(x=x, y=y, z=z)])

fig.show()
```

## Contour Plot

A contour plot in Plotly is a graphical representation that displays the 2D variation of a continuous variable through contour lines.

In [36]:
```
# Contour plot
x = np.linspace(-10,10,100)
y = np.linspace(-10,10,100)

xx, yy = np.meshgrid(x,y)

# z = xx**2 + yy**2
z = np.sin(xx) + np.cos(yy)
# z = np.sqrt(xx**2 + yy**2)


trace = go.Contour(x=x,y=y,z=z)

data = [trace]

layout = go.Layout(title='3D Surface Plot')

fig = go.Figure(data,layout)

fig.show()
```

## SubPlots

Subplots in Plotly refer to the arrangement of multiple plots or charts within a single figure, allowing for side-by-side or stacked visualizations.

In [37]:
```
# Subplots
from plotly.subplots import make_subplots
```

In [38]:
```
fig = make_subplots(rows=2,cols=2)
```

In [39]:
```
fig.add_trace(
    go.Scatter(x=[1,9,5],y=[2,10,1]),
    row = 1,
    col = 1
)

fig.add_trace(
    go.Histogram(x=[1,9,5,22,109,134,56,78,12,34,89]),
    row = 1,
    col = 2

)

fig.update_layout(title='Subplot Demo')

fig.show()
```

## Scatter_geo

`scatter_geo` in Plotly is a function used to create scatter plots on geographical maps.

In [40]:
```
df = px.data.gapminder().query("year == 2007")

plot = px.scatter_geo(df, locations="iso_alpha")
plot.show()
```

In [41]:
```
df = px.data.gapminder().query("year == 2007")

plot = px.scatter_geo(df, locations="iso_alpha",size="gdpPercap", color = "country")
plot.show()
```

## Scatter Polar

Scatter Polar in Plotly is a visualization technique that represents data points in a polar coordinate system, where the distance from the center represents one variable, and the angle represents another variable.

In [42]:
```
# scatter Polar
import plotly.express as px

df = px.data.tips()

plot = px.scatter_polar(df, r = 'day',
                        theta = 'total_bill',
                        color = 'time',
                        symbol = 'tip')
plot.show()
```

## Scatter terenary

Scatter ternary in Plotly refers to the creation of triangular scatter plots where data points are represented within a ternary diagram. This type of plot allows the visualization of three variables that add up to a constant sum, typically represented by the three vertices of the triangle. It provides insights into the relative proportions and relationships between the variables.

In [43]:
```
df = px.data.iris()

plot = px.scatter_ternary(df, a = 'sepal_width',
                          b = 'sepal_length',
                          c='petal_width',
                          color = 'species',
                          size = 'petal_length')
plot.show()
```

In [44]:
```
df = px.data.iris()

plot = px.scatter_ternary(df, a = 'sepal_width',
                          b = 'sepal_length',
                          c = 'petal_width',
                          color = 'species',
                          size = 'petal_length',
                          symbol = 'species_id')
plot.show()
```

## 3D Cones

3D cones in Plotly are graphical representations of cone-shaped objects in a three-dimensional space, used for visualizing data or geometric concepts.

In [45]:
```
fig = go.Figure(data=go.Cone(x=[11,31, 12],
                             y=[12,32, 21],
                             z=[13,41, 15],
                             u=[14,16, 17],
                             v=[15,27, 10],
                             w=[10,29, 21]))

fig.show()
```

## 3D Volume Plots

3D volume plots in Plotly refer to the creation of visual representations that depict volumetric data in a three-dimensional space, allowing for the visualization of complex structures and distributions.

In [46]:
```
import plotly.graph_objects as go
import plotly.express as px
import numpy as np

df = px.data.tips()

x1 = np.linspace(-4, 4, 9)
y1 = np.linspace(-5, 5, 11)
z1 = np.linspace(-5, 5, 11)

X, Y, Z = np.meshgrid(x1, y1, z1)

values = (np.sin(X**2 + Y**2))/(X**2 + Y**2)

fig = go.Figure(data=go.Volume(
    x=X.flatten(),
    y=Y.flatten(),
    z=Z.flatten(),
    value=values.flatten(),
    opacity=0.1,
    caps= dict(x_show=False, y_show=True, z_show=False),
))
fig.show()
```

```
C:\Users\user\AppData\Local\Temp/ipykernel_2640/3749250901.py:15: RuntimeWarning:

invalid value encountered in true_divide
```

## 3D streamtube plots

3D streamtube plots in Plotly refer to the creation of visualizations that depict fluid flow using streamlines that are enclosed within a tube-like structure, providing a three-dimensional representation of the flow behavior.

In [47]:
```
# 3D streamtube plots

x, y, z = np.mgrid[0:20, 0:20, 0:20]
x = x.flatten()
y = y.flatten()
z = z.flatten()

u = np.zeros_like(x)
v = np.zeros_like(y)
w = z**2

fig = go.Figure(data=go.Streamtube(x=x, y=y, z=z, u=u, v=v, w=w))
fig.show()
```

## 3D Mesh Plots

3D mesh plots in Plotly refer to visualizations that represent a surface or a mesh-like structure using a combination of x, y, and z coordinates in a three-dimensional space.

In [48]:
```
import plotly.graph_objects as go
import numpy as np

# Data for three-dimensional scattered points
z = 15 * np.random.random(100)
x = np.sin(z) + 0.1 * np.random.randn(100)
y = np.cos(z) + 0.1 * np.random.randn(100)

fig = go.Figure(data=[go.Mesh3d(
x=x, y=y, z=z, color='red', opacity=0.20)])

fig.show()
```

## 3D Scatter points

3D Scatter points in Plotly refer to the visual representation of data points in a three-dimensional space using markers or symbols. It allows for the plotting of data points with three different variables, where each variable corresponds to the position of the point along the x, y, and z axes.

In [49]:
```
# Data for three-dimensional scattered points

z = 15 * np.random.random(100)
x = np.sin(z) + 0.1 * np.random.randn(100)
y = np.cos(z) + 0.1 * np.random.randn(100)

fig = go.Figure(data=[go.Mesh3d(x=x, y=y, z=z, color='green',
                                opacity=0.20, alphahull=3)])

fig.show()
```

## Create tables

Creating tables in Plotly refers to the process of generating tabular data structures with customizable formatting and styling for visual representation and analysis purposes.

In [50]:
```
# How to create tables in Plotly

import plotly.graph_objects as go

fig = go.Figure(data=[go.Table(
    header=dict(values=['A', 'B','C']),
    cells=dict(values=[[10, 20, 30, 40],
    [40, 20, 10, 50],[11, 22, 32, 40]]))
])
fig.show()
```

| A | B | C |
|---|---|---|
| 10 | 40 | 11 |
| 20 | 20 | 22 |
| 30 | 10 | 32 |
| 40 | 50 | 40 |

In [51]:
```
color1 = 'lightgreen'
color2 = 'lightblue'

fig = go.Figure(data=[go.Table(
   header=dict(values=['A', 'B']),
   cells=dict(values=[[10, 20, 30, 40],[40, 20, 10, 50]],
   fill_color=[[color1, color2, color1,color2, color1]*2],))
])
fig.show()
```

| A | B |
|---|---|
| 10 | 40 |
| 20 | 20 |
| 30 | 10 |
| 40 | 50 |

## Sankey Diagram

A sankey diagram is a visualization used to depict a flow from one set of values to another.

In [52]:
```
# Sankey Diagram


plot = go.Figure(go.Sankey(
       node = {"label": ["A", "B", "C"],
       "x": [0.5, 0.2, 0.1],
       "y": [0.4, 0.3, 0.7],'pad':5},
link = {"source": [1, 0, 1],
        "target": [2, 3, 4],
        "value": [4, 2, 1]}))

plot.show()
```

## Dendograms

A dendrogram is a diagram representing a tree. The figure factory called create_dendrogram performs hierarchical clustering on data and represents the resulting tree. Values on the tree depth axis correspond to distances between clusters.

In [53]:
```
# Creating Dendogram


from plotly.figure_factory import create_dendrogram

X = np.random.rand(10,10)
fig = create_dendrogram(X)
fig.show()
```

In [54]:
```
import plotly.graph_objects as go


plot = go.Figure(go.Sankey(
    node = {"label": ["Hello", "Hi", "H", "IIIH"],
        "x": [0.5, 0.2, 0.1, 0.9],"y": [0.6, 0.8, 0.7],
        "color": "green",
        'pad':5},
link = {
        "source": [3, 2, 1],
         "target": [5, 3, 7],
         "value": [6, 1, 2]}))

plot.show()
```

## Candle Stick

The candlestick chart is a style of financial chart that describes o for a given x coordinate (most likely time). The boxes represent the spread between the open and close values and the lines represent the spread between the low and high values.

In [55]:
```
# candlestick

from datetime import datetime

open_data = [33.0, 33.3, 33.5, 33.0, 34.1]
high_data = [33.1, 33.3, 33.6, 33.2, 34.8]
low_data = [32.7, 32.7, 32.8, 32.6, 32.8]
close_data = [33.0, 32.9, 33.3, 33.1, 33.1]
dates = [datetime(year=2013, month=10, day=10),
        datetime(year=2013, month=11, day=10),
        datetime(year=2013, month=12, day=10),
        datetime(year=2014, month=1, day=10),
        datetime(year=2014, month=2, day=10)]

fig = go.Figure(data=[go.Candlestick(x=dates,
                    open=open_data, high=high_data,
                    low=low_data, close=close_data)])

fig.show()
```

34.5
34
33.5
33
32.5
Oct 2013
Nov 2013
Dec 2013
Jan 2014
Feb 2014

In [56]:
```
from plotly.figure_factory import create_candlestick

from datetime import datetime

# Add data

open_data = [33.0, 33.3, 33.5, 33.0, 34.1]
high_data = [33.1, 33.3, 33.6, 33.2, 34.8]
low_data = [32.7, 32.7, 32.8, 32.6, 32.8]
close_data = [33.0, 32.9, 33.3, 33.1, 33.1]

dates = [datetime(year=2013, month=10, day=10),
        datetime(year=2013, month=11, day=10),
        datetime(year=2013, month=12, day=10),
        datetime(year=2014, month=1, day=10),
 datetime(year=2014, month=2, day=10)]

# Create ohlc
fig = create_candlestick(open_data, high_data,low_data, close_data, dates=dates)

fig.show()
```

## streamline plot

streamline plots are based on the representation on a 2-D vector field which is explained as velocity fields, which are consist of closed curves that are tangent to the velocity field. Streamlining is the fastest technique and more efficient for getting the data. Velocity values are interpolated when determining the streamlines. Streamlines are initialized on the boundary of the x-y domain.

In [57]:
```
x = np.linspace(-1, 2, 50)
y = np.linspace(-1, 1, 50)
Y, X = np.meshgrid(x, y)
u = np.cos(X)*Y
v = np.cos(y)*X

# Source for x and y coordinate
# of scatter plot
X, Y = 0, 0

# Create streamline figure
fig = ff.create_streamline(x, y, u, v, arrow_scale=.1)

fig.add_trace(go.Scatter(x=[X], y=[Y],
                         mode='markers',
                         marker_size=15,
 ))

fig.show()
```

## Wind Rose and Polar Bar Charts

The wind rose chart are graphical charts that show the speed and direction of winds at a location over a period of time. This chart is represented in circular format and the circle indicates the amount of time that the wind blows from a particular direction. The wind rose chart is also known as the polar bar chart.

In [58]:
```
# using the wind dataset
df = px.data.wind()

fig = px.bar_polar(df, r="strength", theta="direction",
                   color="frequency",)
fig.show()
```

## Bullet Chart

This method is used to create bullet charts. This function can take both dataframes or a sequence of dictionaries.

Syntax: plotly.figure_f

In [59]:
```
import plotly.figure_factory as ff
import pandas as pd


data = [
   {"title": "Revenue",
   "subtitle": "US$, in thousands",
    "ranges": [150, 225, 300],
    "measures":[220, 270],
    "markers":[250]},

    {"title": "Profit",
    "subtitle": "%",
    "ranges": [20, 25, 30],
    "measures":[21, 23],
     "markers":[26]},

    {"title": "Order Size",
    "subtitle": "US$, average",
     "ranges": [350, 500, 600],
     "measures":[100, 320],
     "markers":[550]},

    {"title": "New Customers",
    "subtitle": "count",
    "ranges": [1400, 2000, 2500],
    "measures":[1000, 1650],
    "markers":[2100]},

   {"title": "Satisfaction",
   "subtitle": "out of 5",
    "ranges": [3.5, 4.25, 5],
     "measures":[3.2, 4.7],
     "markers":[4.4]}
]

fig = ff.create_bullet(
    data, titles='title',
    markers='markers',
    measures='measures',
    orientation='v',
    measure_colors=['rgb(14, 52, 75)', 'rgb(31, 141, 127)'],
    scatter_options={'marker': {'symbol': 'circle'}},
```

```
width=700)

fig.show()
```

In [61]:
```
# ChatGPT example for 3D animation

import plotly.graph_objects as go
import numpy as np

# Create sample data
t = np.linspace(0, 10, 100)
x = np.cos(t)
y = np.sin(t)
z = t

# Create a 3D scatter plot
fig = go.Figure(data=[go.Scatter3d(
    x=x,
    y=y,
    z=z,
    mode='markers'
)])

# Define animation frames
frames = [go.Frame(data=[go.Scatter3d(
    x=x[:i],
    y=y[:i],
    z=z[:i],
    mode='markers',
    marker=dict(size=5)
)]) for i in range(2, len(t))]

# Add frames to the figure
fig.frames = frames

# Update animation settings
fig.update_layout(
    title='3D Animation Example',
    scene=dict(
        xaxis_title='X-axis',
        yaxis_title='Y-axis',
        zaxis_title='Z-axis'
    ),
    updatemenus=[
        dict(
            type='buttons',
            buttons=[
```

```
                dict(
                    label='Play',
                    method='animate',
                    args=[None, {'frame': {'duration': 50, 'redraw': True}, 'fromcurrent': True, 'transition'
                ),
                dict(
                    label='Pause',
                    method='animate',
                    args=[[None], {'frame': {'duration': 0, 'redraw': False}, 'mode': 'immediate', 'transitio
                )
            ],
        )
    ]
)

# Display the animation
fig.show()
```

## 3D Animation Example

Play

Pause

In [62]:
```
# Asked ChatGPT For Complex 3D Animation

import plotly.graph_objects as go
import numpy as np

# Create sample data
t = np.linspace(0, 10, 100)
x = np.cos(t)
y = np.sin(t)
z = t

# Define colors for each point
colors = np.sin(t)

# Create a 3D scatter plot
fig = go.Figure(data=[go.Scatter3d(
    x=x,
    y=y,
    z=z,
    mode='markers',
    marker=dict(
        size=5,
        color=colors,
        colorscale='Viridis',
        cmin=np.min(colors),
        cmax=np.max(colors),
        opacity=0.8
    )
)])

# Define animation frames
frames = [go.Frame(data=[go.Scatter3d(
    x=x[:i],
    y=y[:i],
    z=z[:i],
    mode='markers',
    marker=dict(
        size=5,
        color=colors[:i],
        colorscale='Viridis',
        cmin=np.min(colors),
        cmax=np.max(colors),
        opacity=0.8
```

```
    )
)]) for i in range(2, len(t))]

# Add frames to the figure
fig.frames = frames

# Update animation settings
fig.update_layout(
    title='3D Animation with Multiple Colors',
    scene=dict(
        xaxis_title='X-axis',
        yaxis_title='Y-axis',
        zaxis_title='Z-axis'
    ),
    updatemenus=[
        dict(
            type='buttons',
            buttons=[
                dict(
                    label='Play',
                    method='animate',
                    args=[None, {'frame': {'duration': 50, 'redraw': True}, 'fromcurrent': True, 'transition'
                ),
                dict(
                    label='Pause',
                    method='animate',
                    args=[[None], {'frame': {'duration': 0, 'redraw': False}, 'mode': 'immediate', 'transitio
                )
            ],
        )
    ]
)

# Display the animation
fig.show()
```

## 3D Animation with Multiple Colors

Play

Pause

In [63]:
```
import plotly.express as px
import pandas as pd

# Create sample data (using gapminder dataset as an example)
df = px.data.gapminder()

# Create an animated scatter plot
fig = px.scatter(df, x='lifeExp', y='gdpPercap', color='continent',
                 size='pop', size_max=100, hover_name='country',
                 animation_frame='year', range_x=[20, 90], range_y=[0, 80000])

# Customize the plot layout
fig.update_layout(
    title='Animated Scatter Plot Example',
    xaxis_title='Life Expectancy',
    yaxis_title='GDP per Capita'
)

# Display the animation
fig.show()
```

Animated Scatter Plot Example
GDP per Capita
80k
70k
60k
50k
40k
30k
20k
10k
0
20
30
40
50
60
70
80
Life Expectancy
year=2007

In [64]:
```
import plotly.express as px
import pandas as pd

# Create sample data (using iris dataset as an example)
df = px.data.iris()

# Create an animated scatter plot
fig = px.scatter(df, x='sepal_width', y='sepal_length', color='species',
                 size='petal_length', hover_name='species',
                 animation_frame='petal_width', range_x=[1.5, 5.5], range_y=[4, 8])

# Customize the plot layout
fig.update_layout(
    title='Animated Scatter Plot Example',
    xaxis_title='Sepal Width',
    yaxis_title='Sepal Length'
)

# Display the animation
fig.show()
```

Animated Scatter Plot Example
Sepal Length
8
7.5
7
6.5
6
5.5
5
4.5
4
1.5
2
2.5
3
3.5
4
4.5
5
Sepal Width
petal_width=2.3

In [65]:
```
import plotly.express as px
import pandas as pd

# Create sample data (using tips dataset as an example)
df = px.data.tips()

# Create an animated scatter plot with different bubble shapes
fig = px.scatter(df, x='total_bill', y='tip', color='sex',
                 size='size', hover_name='day',
                 animation_frame='time', range_x=[0, 60], range_y=[0, 12],
                 symbol='sex')

# Customize the plot layout
fig.update_layout(
    title='Animated Scatter Plot Example',
    xaxis_title='Total Bill',
    yaxis_title='Tip'
)

# Display the animation
fig.show()
```

Animated Scatter Plot Example
Tip
Total Bill
0
2
4
6
8
10
12
0
10
20
30
40
50
time=Lunch

In [ ]: