Visualization Wizardry: Let your Charts Talk Through Plotly!

In [1]:

```
# !pip install plotly
```

In [2]:

```
import plotly.express as px
import plotly.graph_objects as go

import pandas as pd
import numpy as np
%matplotlib inline
```

Table of Contents

2.Basic Plots

- Line Plots
- Bar Plots
- Scatter Plots
- Pie Charts
- Histograms

3.Advanced Plots

- Box Plot
- Violin Plot
- Density Heatmap
- 3D Plots
- Scatter Matrix
- Facet Grid
- Animated Plots

Important Plotly Modules

If you haven't used Plotly before, you are really missing out on visualization wizardry, Plotly shines as a powerful Python library that enables you to create interactive and visually appealing charts and graphs for data visualization with very low code.

Mastering Plotly can make you understand your data much more clearly and most importantly it can take your data storytelling to the next level through its interactiveness! In this article, we'll delve deep into the world of Plotly, exploring its most important spells (I mean methods), features, and capabilities, Assuming that you know the theory of these plots. Here's a glimpse of what lies ahead.

Plotly has various modules for different purposes, let's explore them.

1. **Plotly.express:** It is a high-level API for creating quick and easy visualizations. It is used for creating simple charts with minimal code, import plotly.express as px to use Plotly express. When using this, first you can pass the dataframe, and for x and y parameters you can just pass the column name, as it is compatable with pandas dataframes.
2. **plotly.graph_objects:** This module is part of Plotly's core functionality and is used for creating a wide range of charts and graphs, import plotly.graph_objects as go to use Plotly graph objects. When using this, we cannot pass the data, we can only pass the series for x and y parameters.
3. **plotly.offline:** In some cases, you might want to work with Plotly offline, especially if you're generating visualizations for offline use or within Jupyter notebooks.

Basic Plots

we can create all the basic plots with low code and amazing interactivity such as:

1. Hover information displays details when you mouse over the data points.
2. while zooming and panning enable closer examination of the plot.
3. You can select specific data series or data points by clicking on them in the legend or on the plot itself.
4. You can add custom buttons or controls to the plot that trigger specific actions or updates when clicked, and much more.

Line Plots

- **Using Plotly Express for Simple Plots**: Let's take a dataset available in Plotly to explore a few line plots.

In [3]:

```
# Stock data available in the plotly express module
df_stocks = px.data.stocks()
df_stocks.head()
```

Out[3]:

	date	GOOG	AAPL	AMZN	FB	NFLX	MSFT
0	2018-01-01	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
1	2018-01-08	1.018172	1.011943	1.061881	0.959968	1.053526	1.015988
2	2018-01-15	1.032008	1.019771	1.053240	0.970243	1.049860	1.020524
3	2018-01-22	1.066783	0.980057	1.140676	1.016858	1.307681	1.066561
4	2018-01-29	1.008773	0.917143	1.163374	1.018357	1.273537	1.040708

In [4]:

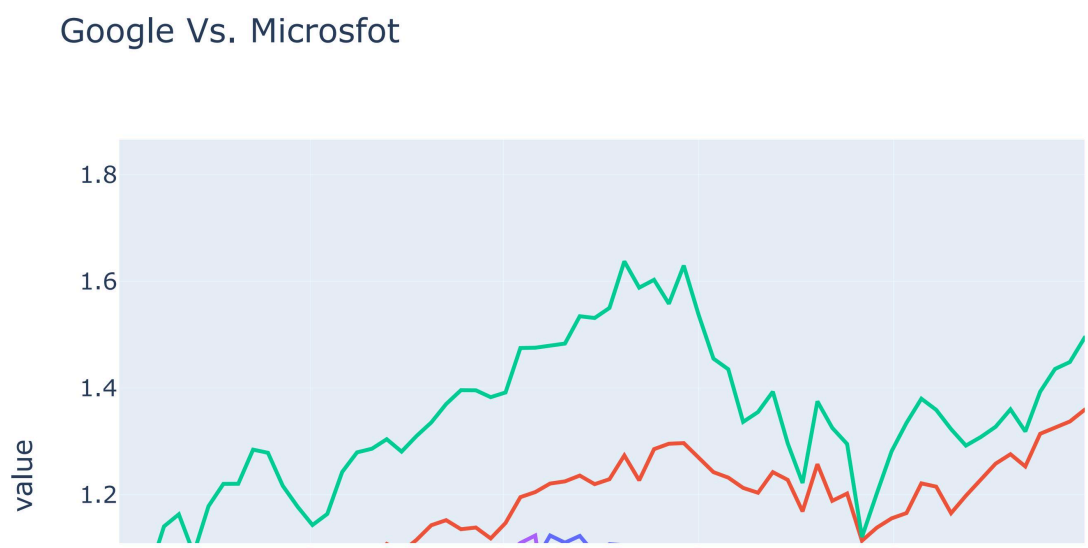
```
# Exploring Facebook stocks through an interactive line plot
px.line(df_stocks, x='date', y='FB')
```

By running the single above code line, you will be able to generate such a beautifully interactive plot like this.
By running the single above code line, you will be able to generate such a beautifully interactive plot like this.

- **Generating Multiple Line plots by just providing a list**

In [5]:

```
# Generate multiple line plots by just providing a list of series names
px.line(df_stocks, x='date', y=['GOOG','MSFT','AMZN','FB'], title='Google Vs. Microsfo
t')
```



Now, if you only want to see two or three, you can disable the rest by just clicking on the label name on the right side, this is so cool, right?

- **Using Graph Objects for further customizations**

To achieve these kinds of things, we need to use the graph_objects, which is why it was said that we need the graph_objects module for more complex charts and plotly_express for simpler charts.

1. Create a figure object using fig=go.Figure().
2. To add a plot to the figure use fig.add_trace()and pass the graph object to the figure object.
3. Use go.scatter(x,y,mode)to create graph objects for scatter plots and line plots.
4. And here we need to use the mode parameter to specify the plot style, by default it will be assigned to lines which will give a line plot. And for the Facebook, as we gave mode='lines+markers' it will have the lines along with markers for the points, easy styling!
5. For line styling, use theline parameter and assign a dictionary with various stylings you need.

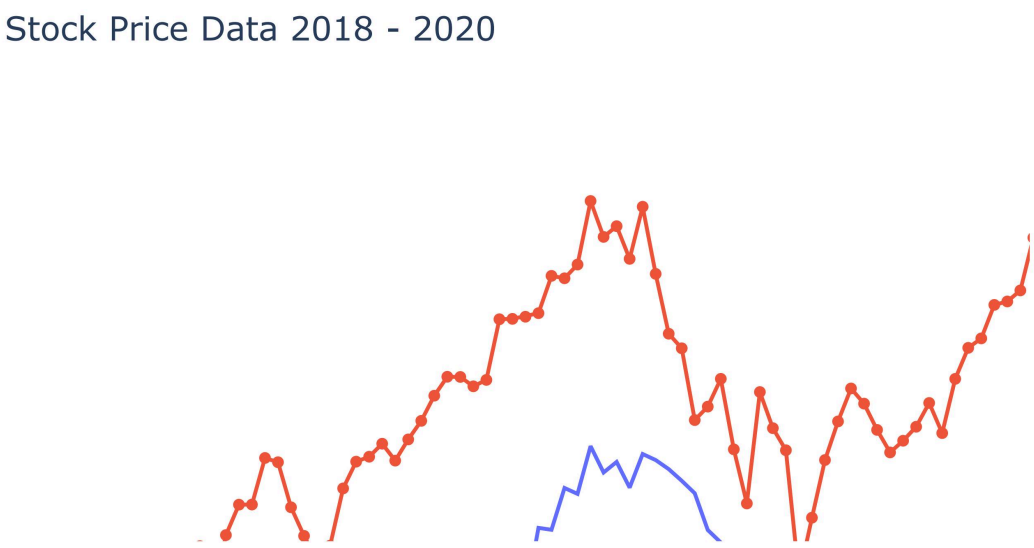
In [6]:

```
# Create a figure object for which we can later add the plots
fig = go.Figure()

# pass the graph objects to the add trace method, assign a series to x and y parameters
of graph objects.
fig.add_trace(go.Scatter(x=df_stocks.date, y=df_stocks.AAPL, mode='lines', name='Appl
e'))
fig.add_trace(go.Scatter(x=df_stocks.date, y=df_stocks.AMZN, mode='lines+markers', name
='Amazon'))

# Customizing a particular line
fig.add_trace(go.Scatter(x=df_stocks.date, y=df_stocks.GOOG,
                         mode='lines+markers', name='Google',
                         line=dict(color='darkgreen', dash='dot')))

# Further style the figure
fig.update_layout(title='Stock Price Data 2018 - 2020',
                      # customize axis by using xaxis/yaxis '_' and style
                      xaxis_title='Price',
                      # customize various styles axis by passing a dictionary
                      yaxis=dict(
                          showgrid=False,
                          zeroline=False,
                          showline=False,
                          showticklabels=False,
                      ),
                      # customize entire plot style
                      plot_bgcolor='white')
```



Bar Charts

Using Plotly Express for simple Bar Plots:

Use px.bar(data,x,y) to create your bar chart. You can add an extra dimension to the bar plot by simply using the parameter, in that case you will get the stacked bar plot, if you want it side by side, you can change it with the parameter barmode='group' . The below snippets are just images.

In [8]:

```
# Tips data avaialbe in plotly module
tips = px.data.tips()
tips.head()
```

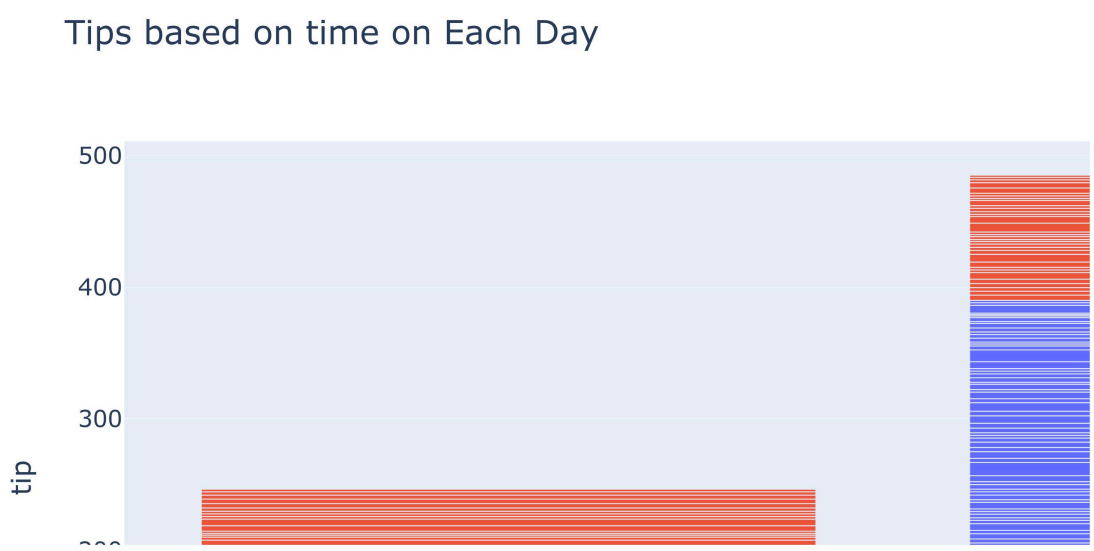
Out[8]:

	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

Using Plotly Express for Simple Bar plots

In [11]:

```
# Stacked bar plot
px.bar(tips, x='sex', y='tip', color='time', title='Tips based on time on Each Day')
```



In [13]:

```
# Place bars next to each other
px.bar(tips, x="sex", y="tip", color='time', barmode='group')
```

customized Bar plots

In [14]:

```
# Filtering data for countries in 2007 greater than 50000000
population = px.data.gapminder()
filtered_data = population[(population.year==2007) & (population['pop']>50000000)]
filtered_data.head()
```

Out[14]:

	country	continent	year	lifeExp	pop	gdpPercap	iso_alpha	iso_num
107	Bangladesh	Asia	2007	64.062	150448339	1391.253792	BGD	50
179	Brazil	Americas	2007	72.390	190010647	9065.800825	BRA	76
299	China	Asia	2007	72.961	1318683096	4959.114854	CHN	156
335	Congo, Dem. Rep.	Africa	2007	46.462	64606759	277.551859	COD	180
467	Egypt	Africa	2007	71.338	80264543	5581.180998	EGY	818

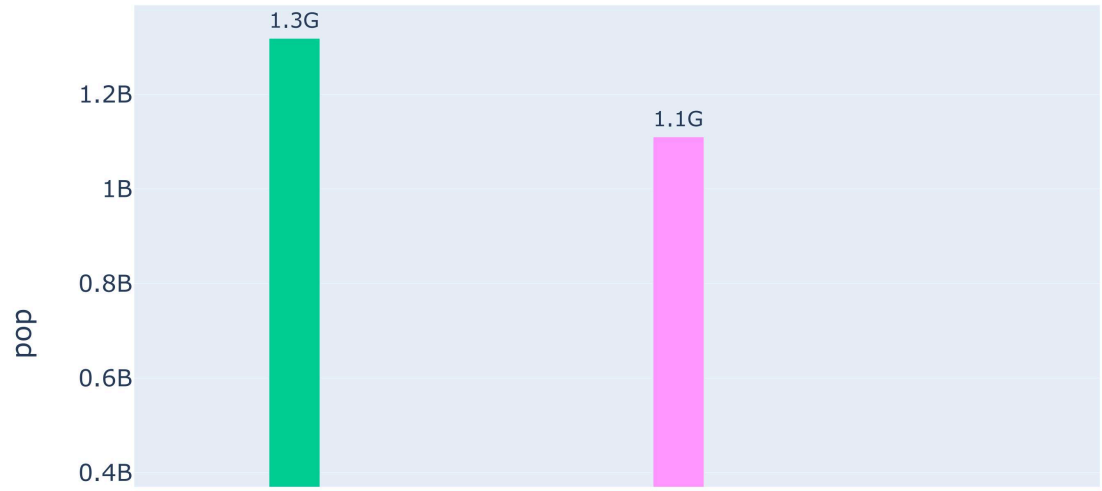
Instead of directly printing the plotly express object, if we save to figure, then we can make more customizations using the update_trace and update_layout methods of the figure.

In [15]:

```
# Using colors to distinguish a dimension
fig = px.bar(filtered_data, y='pop', x='country', text='pop', color='country')

# Put bar total value above bars with 2 values of precision
fig.update_traces(texttemplate='%{text:.2s}', textposition='outside')

# Rotate labels 45 degrees
fig.update_layout(xaxis_tickangle=-45)
```



If we hadn't given the color dimension to be country, all of the bars would have the same color. And by using text parameter we were able to set the total population at the top of the bars.

Scatter Plot

Use px.scatter(data,x,y) to create simple scatter plots. You can add more dimension using color and size parameters. By using the graph object, use go.scatter(data,x,y,mode='markers') to create a scatter plot, and we can even add a numeric variable as a dimension with color a parameter.

In [16]:

```
# Use included Iris data set
df_iris = px.data.iris()

# Create a customized scatter
fig = go.Figure()
fig.add_trace(go.Scatter(
    x=df_iris.sepal_width, y=df_iris.sepal_length,
    mode='markers',
    marker_color=df_iris.sepal_width,
    text=df_iris.species,
    marker_showscale=True
))
fig.update_traces(marker_line_width=2, marker_size=10)
```

Pie Charts

Use px.Pie(data,values=series_name) to create a pie chart. For further customizations, you can use graph object pie chart with go.Pie() and we use the update_traces of the figure object to customize the hover info, text, and pull amount (which is the same as explode, if you are familiar with seaborn).

In [17]:

```
population.head()
```

Out[17]:

	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

In [18]:

```
population.continent.value_counts()
```

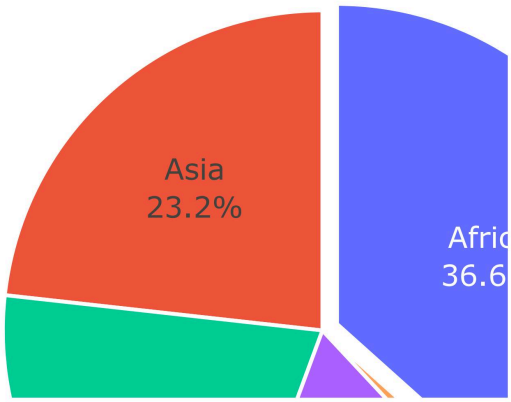
Out[18]:

```
Africa      624
Asia        396
Europe      360
Americas    300
Oceania      24
Name: continent, dtype: int64
```

In [19]:

```
# Customize pie chart
fig = go.Figure(go.Pie(labels=population.continent.value_counts().index,
                       values=population.continent.value_counts()))

# use update traces to customise the hover info, text, pull amount for each pie slice,
and stroke
fig.update_traces(hoverinfo='label+percent', textfont_size=15,
                  textinfo='label+percent', pull=[0.05, 0, 0, 0, 0],
                  marker_line=dict(color='#FFFFFF', width=2))
```

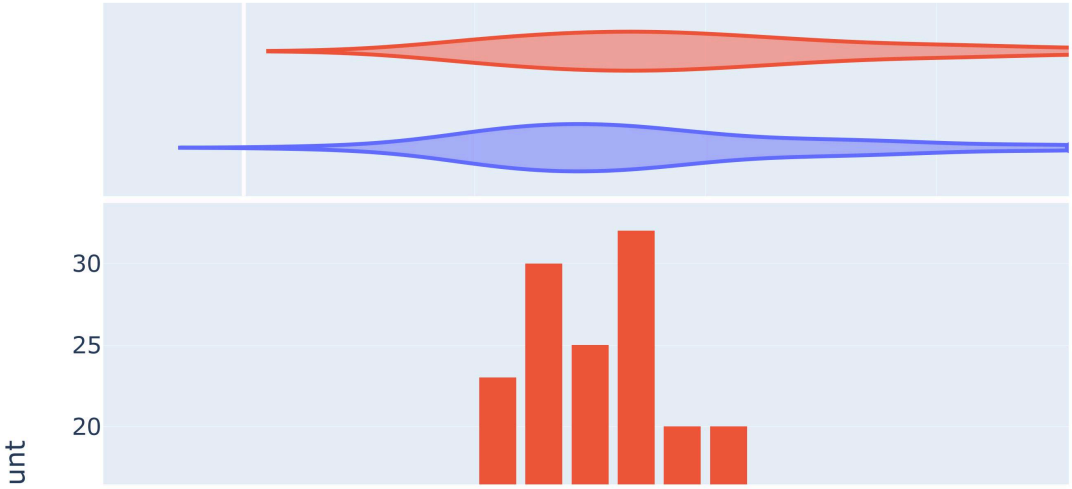


Histograms

Use px.histogram(values) to create a histogram, and you can stack another histogram on top of that using thecolor parameter. And here is an amazing with histogram plots, you can use marginal parameter to add a layer of another plot on the top, such as box, violin, or rug. Here's an example of adding a violin plot on top of the stack bar chart.

In [20]:

```
# Stack histograms based on different column data
df_tips = px.data.tips()
fig = px.histogram(df_tips, x="total_bill", color="sex", marginal='violin')
fig.update_layout(bargap=0.2)
```



- You can hover over the bar charts to see more details and interact with them.
- Hover on the violin plot to see the quartile details.
- And you can click on the side labels to enable or disable them, so cool, huh!

Box Plots

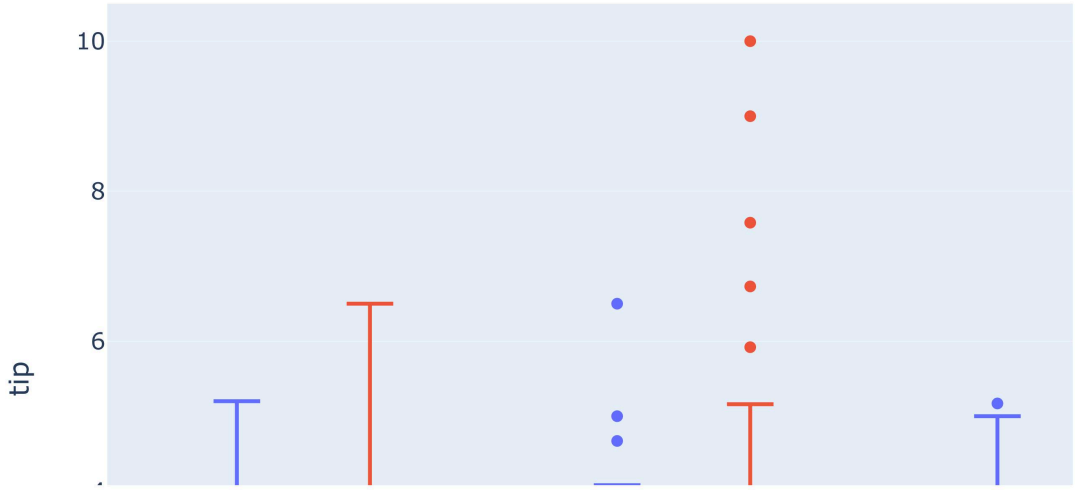
Use px.box(data,x,y) to create the box plot, you can use the color parameter to add another dimension for the box plot. And using the points='all'parameter will show all the points scattered to the left. On hover, you can see the quartile values.

In [21]:

```
df_tips = px.data.tips()
# We can see which sex tips the most, points displays all the data points
px.box(df_tips, x='sex', y='tip', points='all')
```

In [22]:

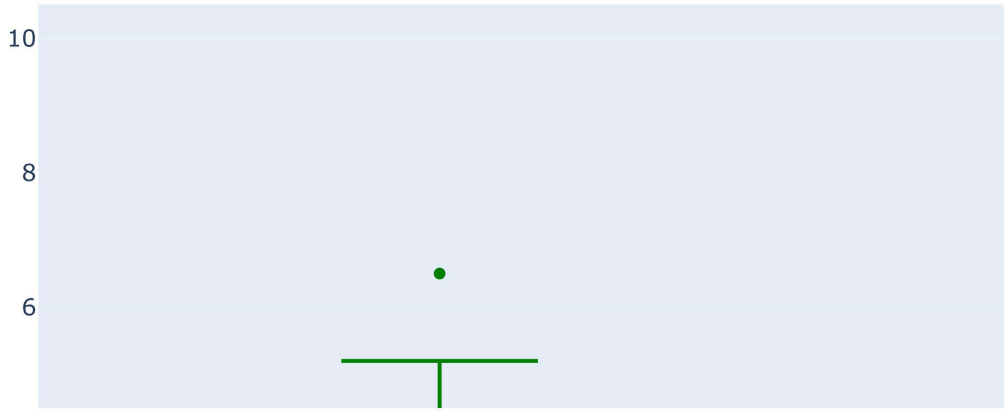
```
# Display tip sex data by day
px.box(df_tips, x='day', y='tip', color='sex')
```



You can do further customizations using go.Box(x,y), Using that you can also add mean and standard deviation with the parameter boxmean='sd' . Remember when using graph object, we won't be passing the data, but just the series. Points parameters is not used in the graph object box method.

In [23]:

```
# Adding standard deviation and mean
fig = go.Figure()
fig.add_trace(go.Box(x=df_tips.sex, y=df_tips.tip, marker_color='green',boxmean='sd'))
```

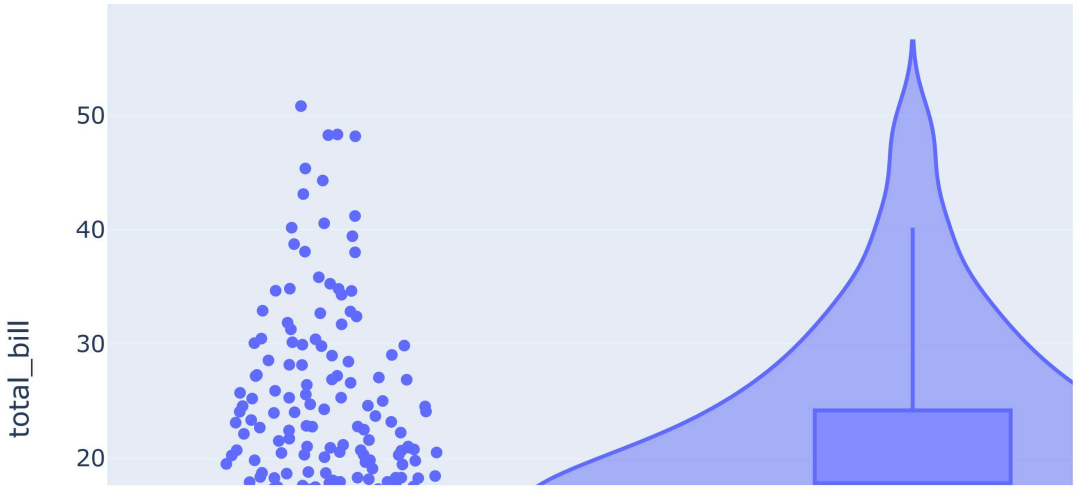


Violin Plots

Use px.violin(data,x,y) to create violin plots. By just giving y parameters you can create a violin plot for one numeric variable. But if you want to create violin plots for a numeric variable based on a category column, then you can specify the category column in the x parameter. And additionaly you can add one more category column dimension using the color parameter. Use box=True parameter to show the box in the plot.

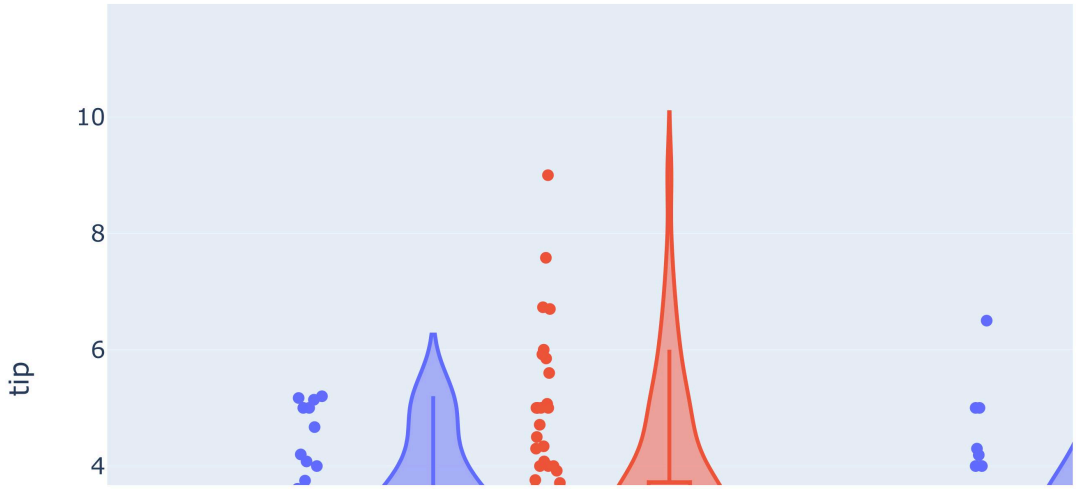
In [24]:

```
# Violin plot for the total bill
df_tips = px.data.tips()
px.violin(df_tips, y="total_bill", box=True, points='all')
```



In [25]:

```
# Multiple plots
px.violin(df_tips, y="tip", x="smoker", color="sex", box=True, points="all",
          hover_data=df_tips.columns)
```

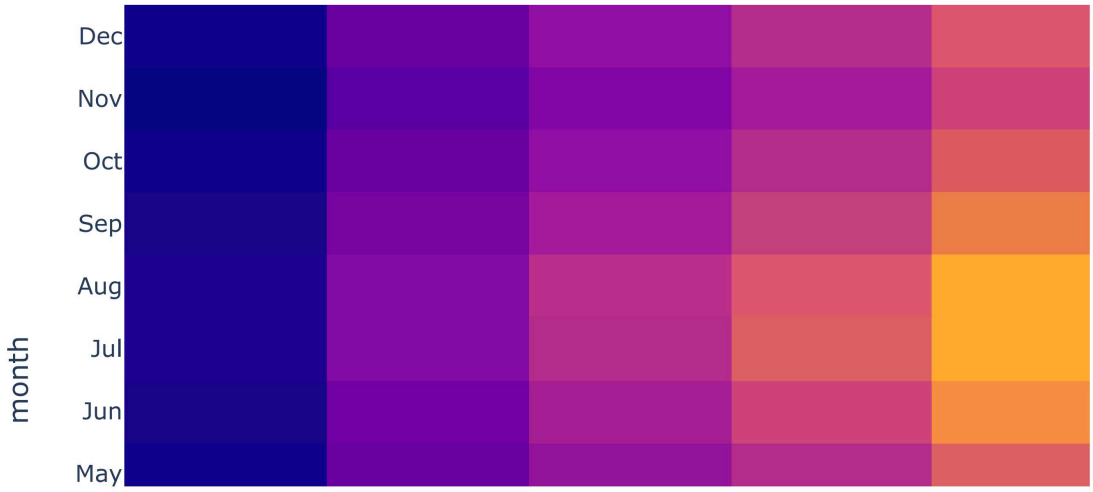


Density Heatmaps

If you need to do an analysis for three variables, you can easily opt for the density heatmaps. Use px.density_heatmap(data,x,y,z) to create a density heat map.

In [28]:

```
# Create a heatmap using Seaborn data
import seaborn as sns
flights = sns.load_dataset("flights")
px.density_heatmap(flights, x='year', y='month', z='passengers')
```



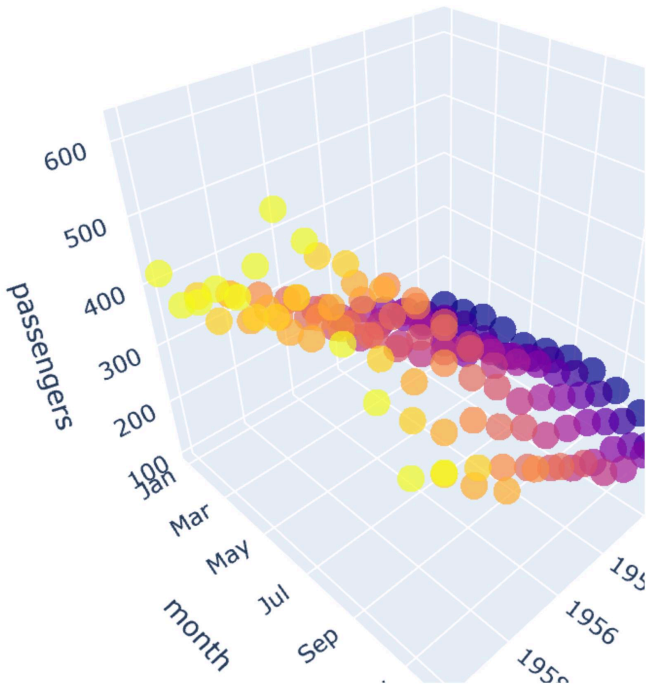
3D Plots

3D Scatter plot

To create a 3D Scatter plot, you can use px.scatter_3d(data,x,y,z,color)

In [29]:

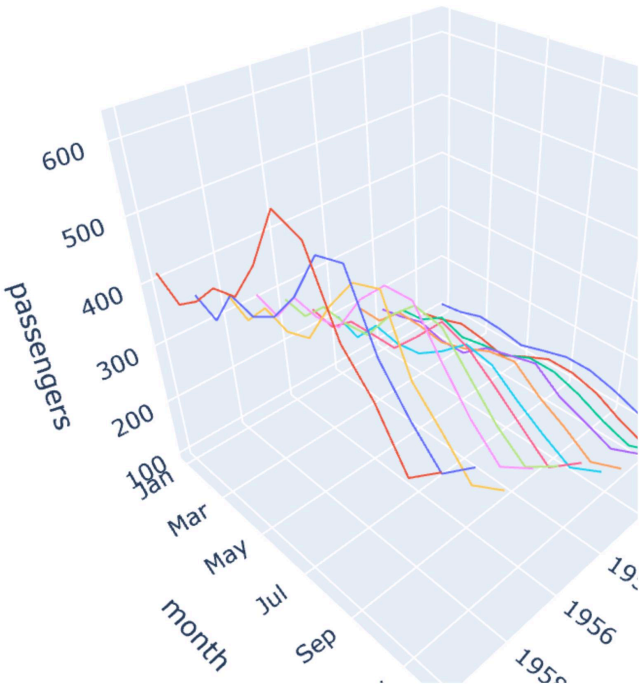
```
# Create a 3D scatter plot using flight data
px.scatter_3d(flights, x='year', y='month', z='passengers', color='year', opacity=0.7)
```



Similarly, you can also create a 3D Line plot using px.line_3d(data,x,y,z,color)

In [30]:

```
px.line_3d(flights, x='year', y='month', z='passengers', color='year')
```

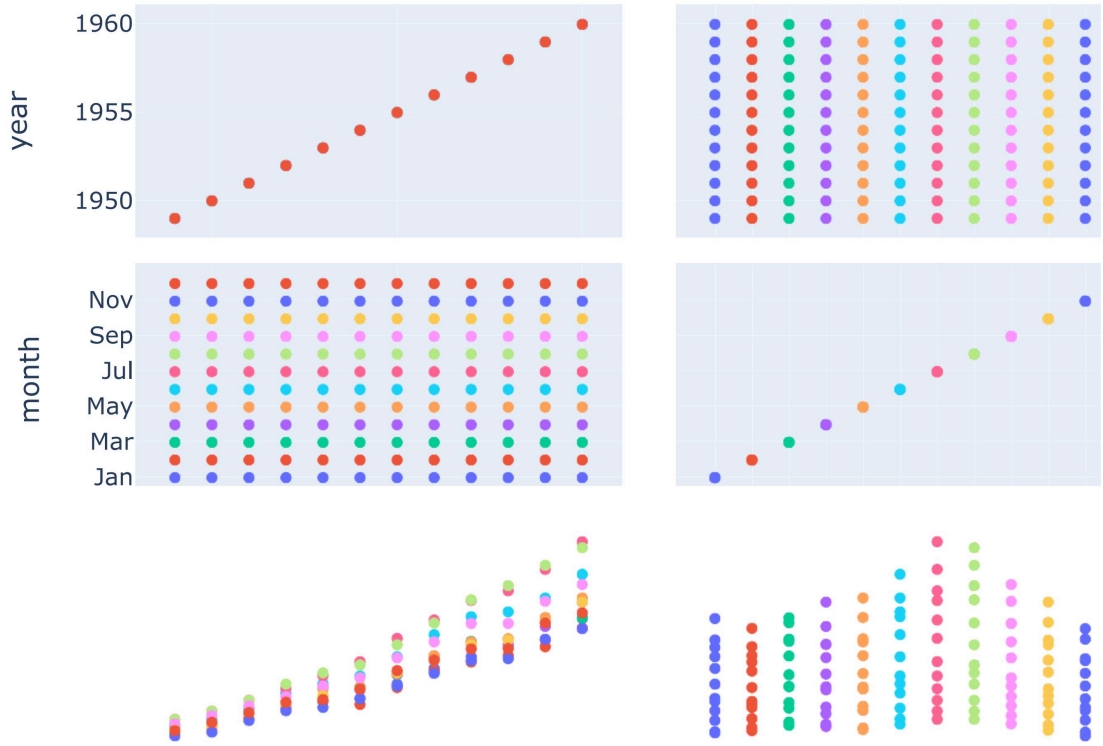


Scatter Matrix

Scatter matrix is similar to the paiplot from sns, which gives an exhaustive idea of the numeric variables. Use px.scatter_matrix(data) to display scatter matrix. To differentiate further based on any category, as always you an use the color parameter.

In [31]:

```
px.scatter_matrix(flights, color='month')
```

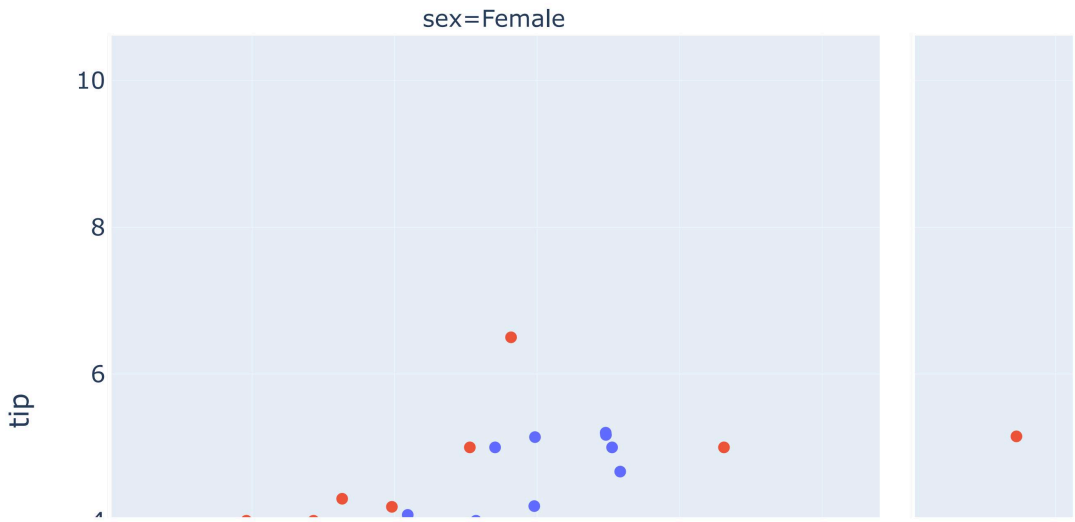


Facet Grids

For all the plotly express plots we have seen so far, we can add the facet grids, by using the parameters face_col and face_row parameters.

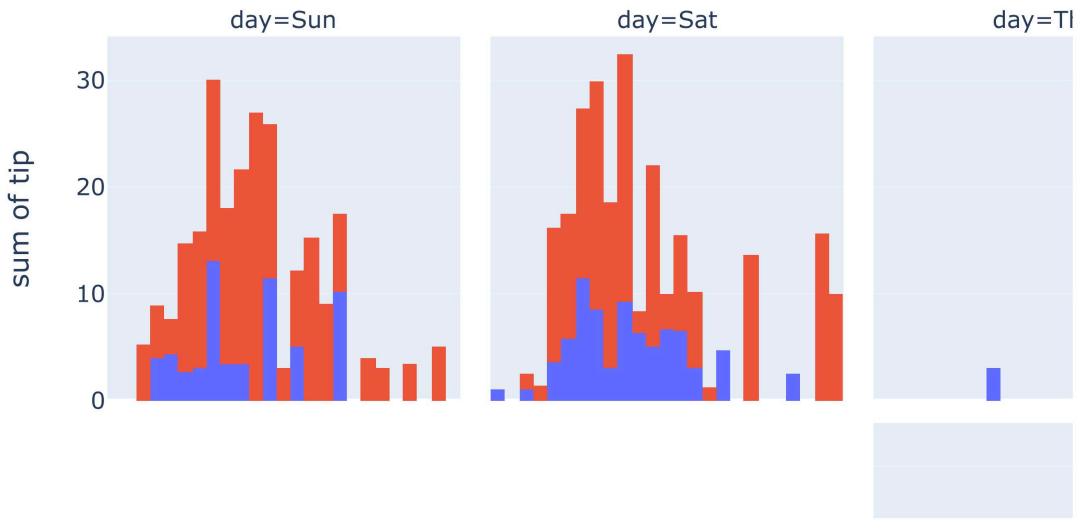
In [32]:

```
df_tips = px.data.tips()
px.scatter(df_tips, x="total_bill", y="tip", color="smoker", facet_col="sex")
```



In [33]:

```
# We can line up data in rows and columns
px.histogram(df_tips, x="total_bill", y="tip", color="sex", facet_row="time", facet_col
="day")
```

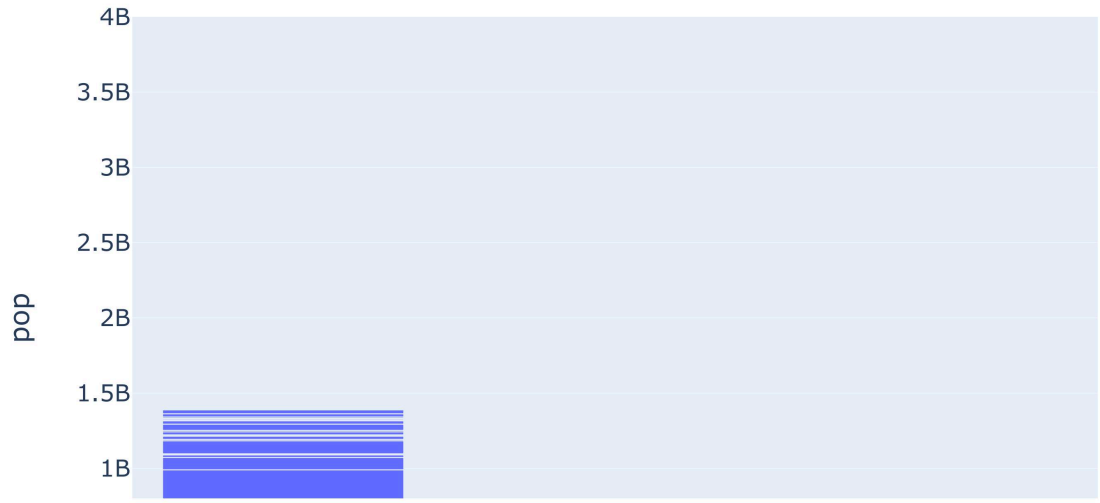


Animated Plots

You can add animation to your plotly express plots. The animation_frame parameter specifies which variable in your dataset should be used to create animation frames. The animation_group parameter is used to group data points within each animation frame. For example, if you're visualizing the movement of objects over time, you might use animation_frame to specify the time steps and animation_group to group objects by their IDs or labels.

In [34]:

```
# Watch as bars chart population changes
population = px.data.gapminder()
px.bar(population, x="continent", y="pop", color="continent",
  animation_frame="year", range_y=[0,4000000000])
```



Conclusion

I guess now you would agree how powerful yet simple plotly is, With Plotly, you're not limited to static graphs but invited to weave stories, paint insights, and orchestrate data's symphony. From the elegance of scatter plots to the drama of animated charts, Plotly lets you be the virtuoso of your data narrative. So, embrace Plotly, and let your data dance, and captivate your audience. It's not just a tool; it's your brush to paint the future of data visualization. Unleash your creativity with plotly, Happy Learning

RaviTeja G

Follow me for more Detailed Notes and Articles Related to Data Science :)