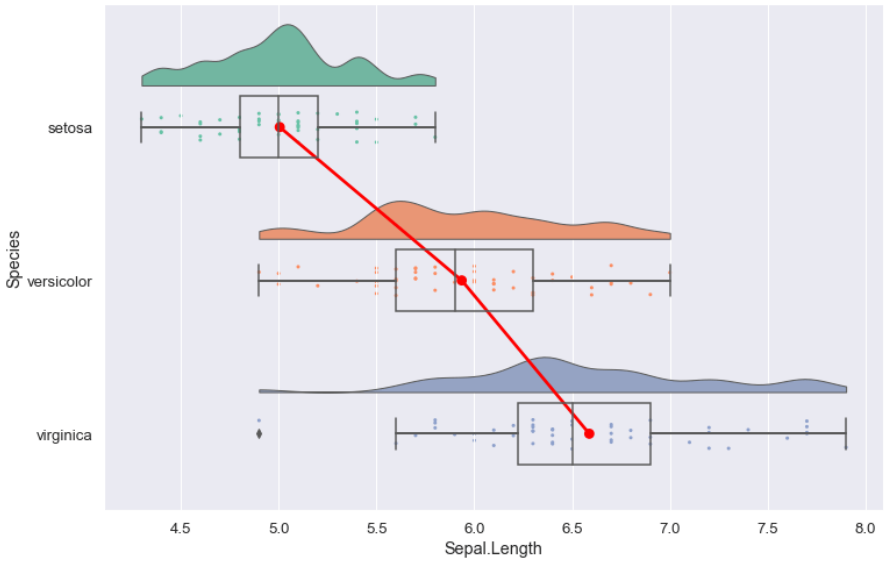
Interactive Visualization Libraries

By

Veer Pal Singh



1. **Bokeh Library**
2. **Pygal Library**
3. **Altair Library**
4. **Holoview Library**
5. **Plotly Library**

What is Data Visualization?

Data visualization is a scientific method of finding out the patterns, anomalies, and outliers in the data. It is a method that is used to create the graphs and plots which helps in finding out the trend in the data because it makes data simpler in terms of understanding by the human brain and we identify trends with naked eyes.

What is Interactive Data Visualization?

Interactive visualization is the use of data analysis software to manipulate and explore graphical representations of data. It is an evolution of the concept of "dataviz" (data visualization) based on the incorporation of interaction tools.

As such, interactive data visualization, using of the tools and processes of interactive libraries, produces a visual representation of data which can be explored and analyzed directly within the visualization itself. This interaction can help uncover insights which lead to better, data-driven decisions.

Representing data graphically and interactively in the form of graphs, charts, and maps helps users quickly spot trends and assess the status of underlying patterns. Allowing direct interaction helps users dig deeper to identify patterns and find new relationships in their data.

Content

- BOKEH LIBRARY
- PYGAL LIBRARY
- ALTAIR LIBRARY
- HOLOVIEWS LIBRARY
- PLOTLY LIBRARY

BOKEH LIBRARY

What is Bokeh?

Bokeh is an interactive visualization library in python. The best feature which bokeh provides is highly interactive graphs and plots that target modern web browsers for presentations. Bokeh helps us to make elegant, and concise charts with a wide range of various charts.

Bokeh primarily focuses on converting the data source into JSON format which then uses as input for BokehJS. Some of the best features of Bokeh are:

- Flexibility: Bokeh provides simple charts and customs charts too for complex use-cases.
- Productivity: Bokeh has an easily compatible nature and can work with Pandas and Jupyter notebooks.
- Styling: We have control of our chart and we can easily modify the charts by using custom Javascript.
- Open source: Bokeh provide a large number of examples and idea to start with and it is distributed under Berkeley Source Distribution (BSD) license. With bokeh, we can easily visualize large data and create different charts in an attractive and elegant manner.

Where to use Bokeh charts?

There are plenty of visualization libraries why do we need to use bokeh only? Let’s see why.

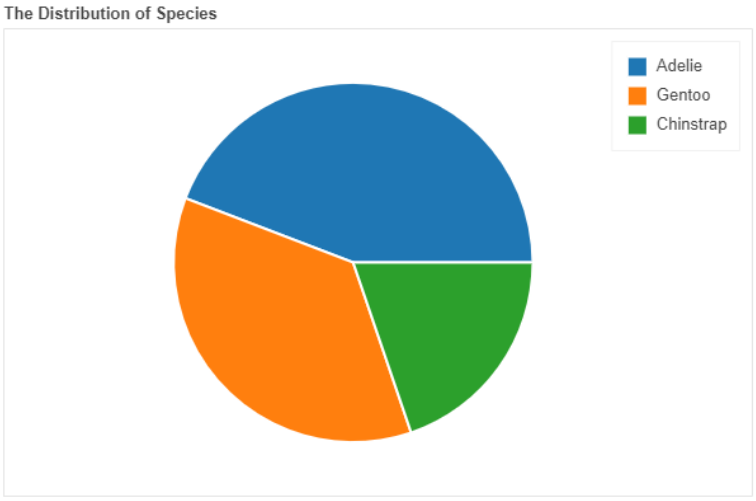
We can use the bokeh library to embed the charts on the web page. With bokeh, we can embed the charts on the web, make a live dashboard, and apps. Bokeh provides its own styling option and widgets for the charts. This is the advantage of embedding the bokeh charts on and website using Flask or Django.

Let's have a quick look at the parameters:

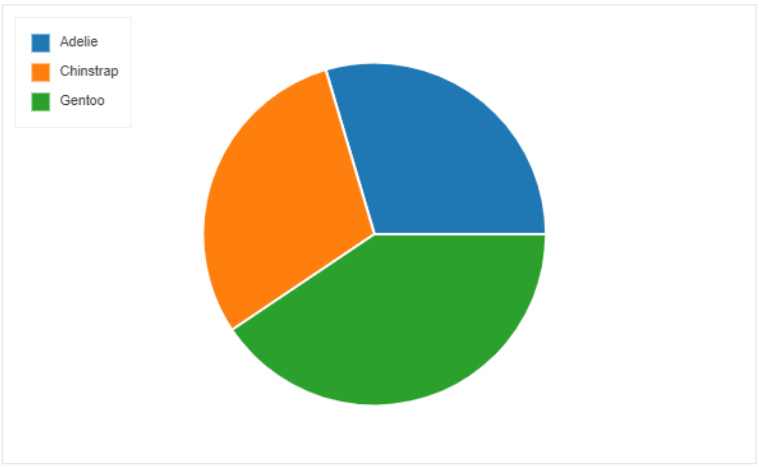
- kind what type of chart do you want to plot? Currently, pandas_bokeh supports the following chart types: line, point, step, scatter, bar, histogram, area, pie and map.
- x and y Simply pass in the column name(s) of the Pandas dataframe
- xlabel and ylabel The label of the x-axis and y-axis relatively
- title The title of the chart

Let's first create a simple Pie Chart using Bokeh library:

```
species_number = penguins.species.value_counts()
df_species = pd.DataFrame(species_number)
print(pd.DataFrame(species_number).T)

species_number.plot_bokeh(kind='pie',
                      title='The Distribution of Species');
```

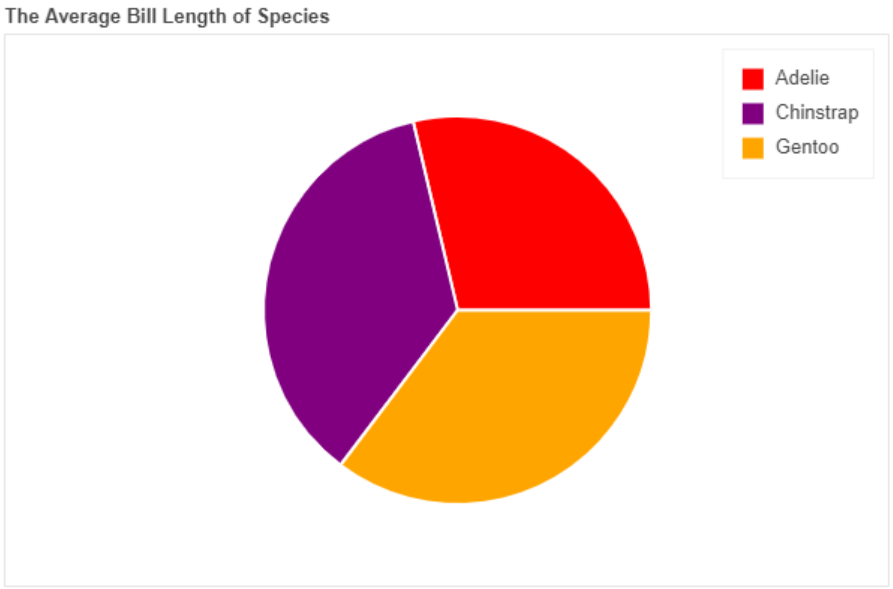



```
print(avg_body_mass)

avg_body_mass.plot_bokeh.pie(figsize=(600, 400),
                             title="The Average Body Mass of Species",
                             fontsize_title=16,
                             legend = "top_left",
                             fontsize_legend=8)
```



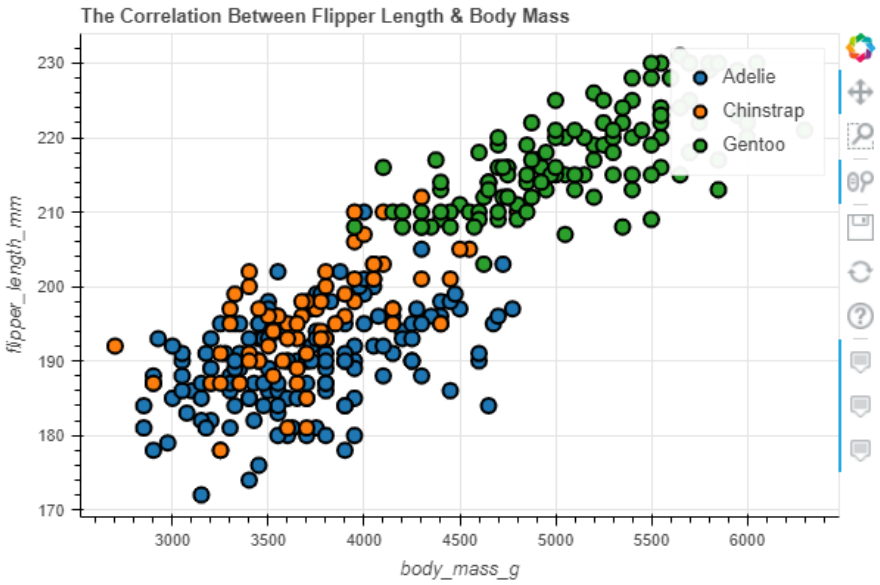
Let us play with colors at Bar chart.

```
print(avg_bill_length)

avg_bill_length.plot_bokeh.pie(x="species",
                               y="bill_length_mm",
                               colormap=["red", "purple", "orange"],
                               title="The Average Bill Length of Species")
```



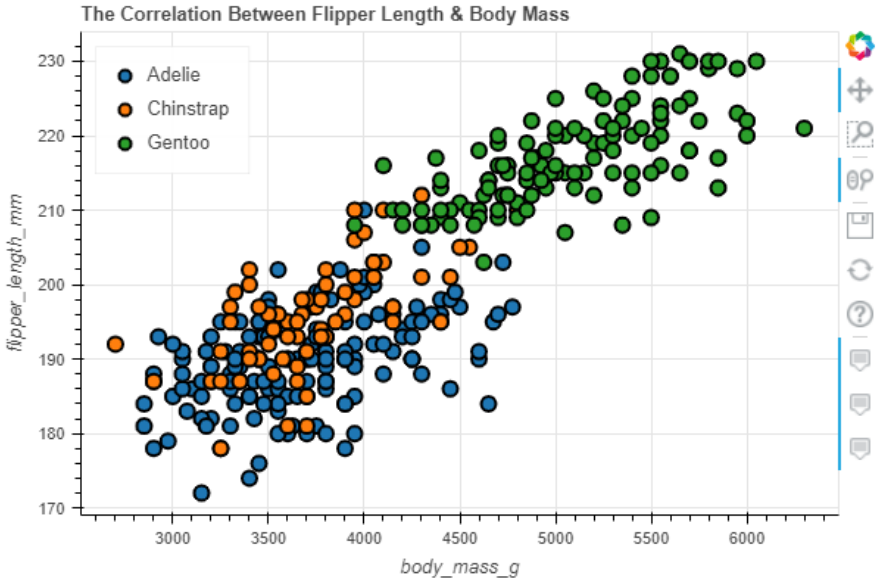
Let's create a basic Scatter Chart using Bokeh library:

```
from bokeh.plotting import figure, show

penguins.plot_bokeh.scatter(x='body_mass_g',
                            y='flipper_length_mm',
                            category='species',
                            title='The Correlation Between Flipper Length & Body Mass')
```



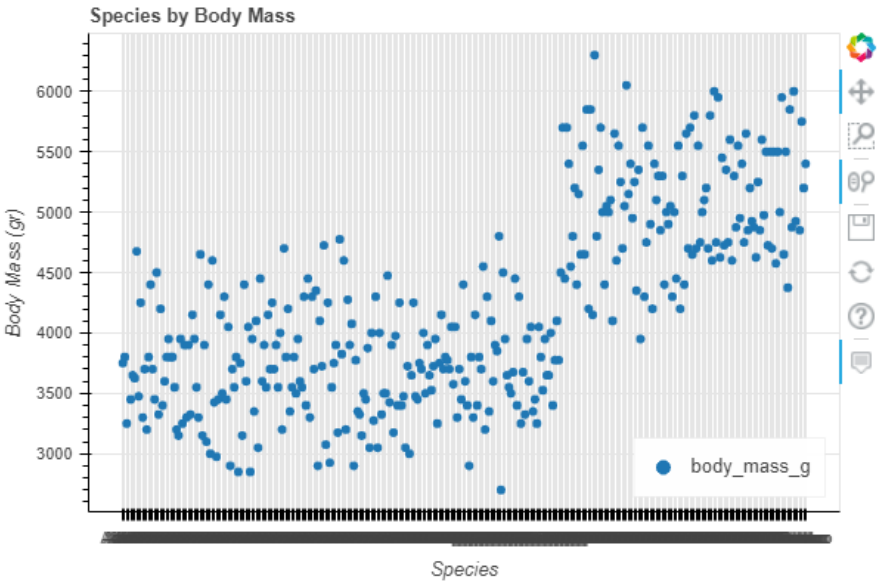
However, the legend of the figure dos NOT look like at the right location. Let us arrange its location.

```
from bokeh.plotting import figure, output_file, show

p = penguins.plot_bokeh.scatter(x='body_mass_g',
                                y='flipper_length_mm', category='species',
                                legend = "top_left",
                                title='The Correlation Between Flipper Length & Body Mass')
```



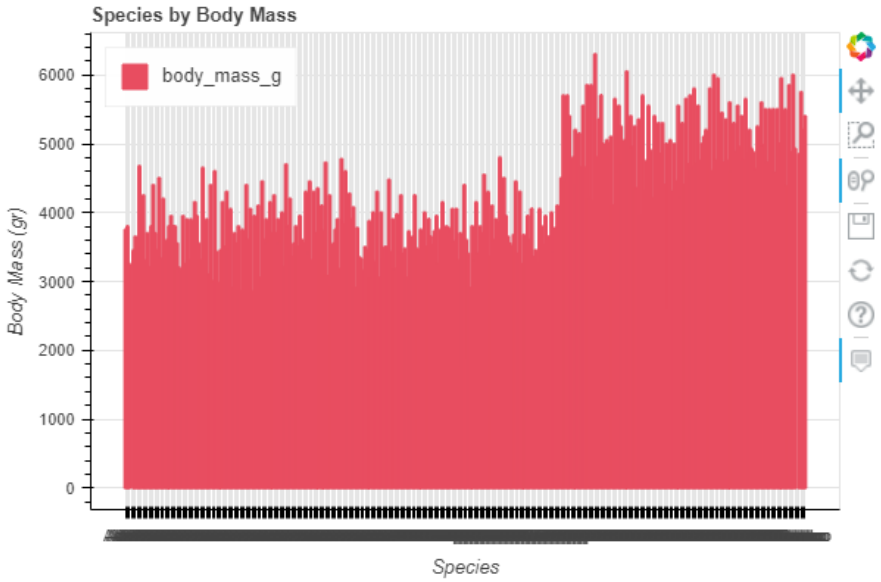
Let's create a Point Chart by focusing on more parameters at Bokeh library:

```
penguins.plot_bokeh(kind='point',
                    x='species',
                    y="body_mass_g",
                    xlabel='Species',
                    ylabel='Body Mass (gr)',
                    title='Species by Body Mass',
                    legend = "bottom_right")
```

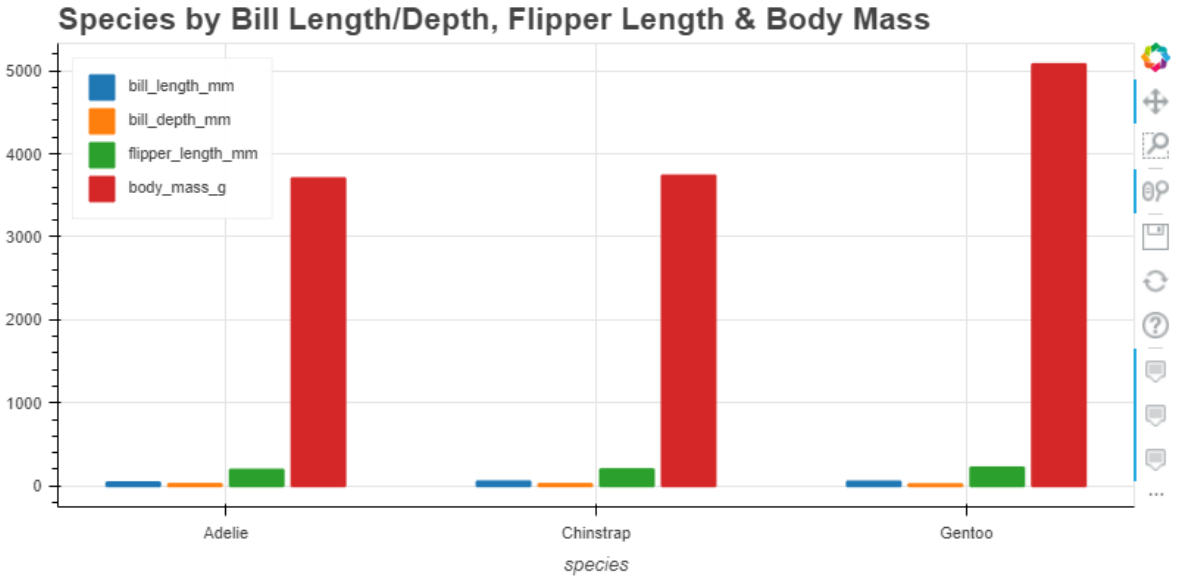


Let's create a Bar Chart using Bokeh library:

```
penguins.plot_bokeh(kind='bar',
                    x='species',
                    y="body_mass_g",
                    color="#e84d60",
                    xlabel='Species',
                    ylabel='Body Mass (gr)',
                    legend = "top_left",
                    title='Species by Body Mass')
```

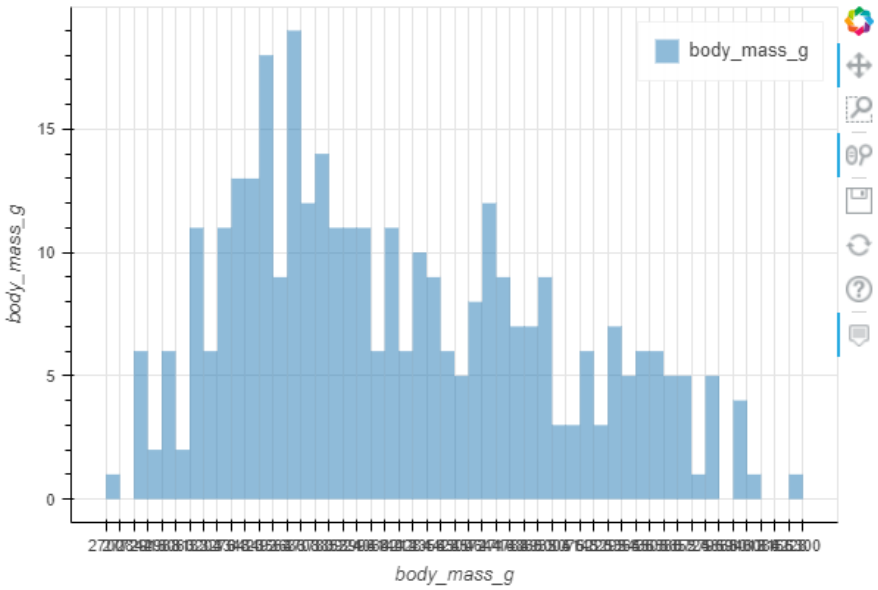



```
result = penguins.groupby("species").mean()
result.index = result.index.astype(str)

ax= result.plot_bokeh.bar(figsize=(800, 400),
                          title="Species by Bill Length/Depth, Flipper Length & Body Mass",
                          fontsize_title=16,
                          legend = "top_left",
                          fontsize_legend=8)
```



Let's create a Histogram Chart using Bokeh library:

```
penguins["body_mass_g"].plot_bokeh(kind='hist',
                                   bins=50)
```



PYGAL LIBRARY

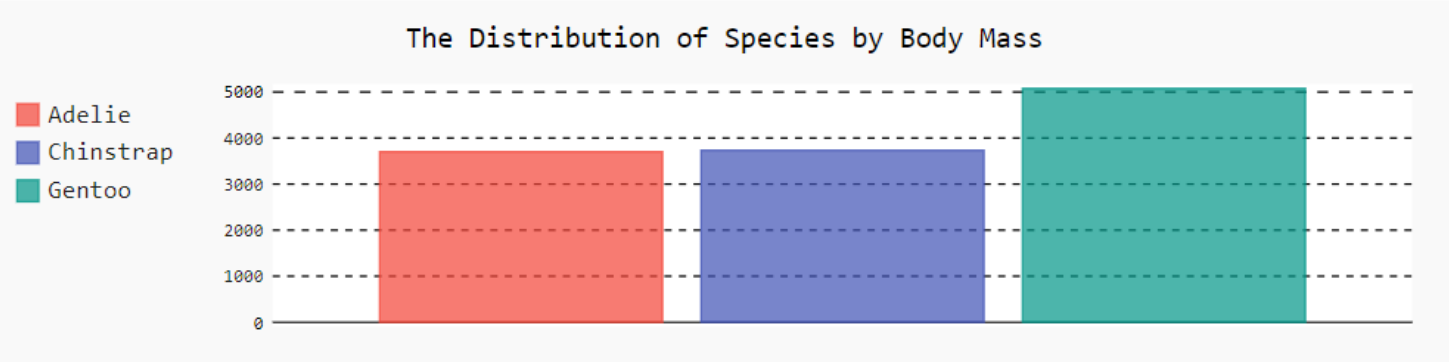
What is PYGAL?

Pygal is an open-source python library that not only creates highly interactive plots but also creates SVG images of the graphs/plots so that we can use it and customize it accordingly. Pygal is highly customizable and creates graphs with a few lines of code.

Pygal creates a variety of plots like a bar, area, histogram, line, etc and due to the high scalability of the images of the plot downloaded as SVG, we will have good quality images that can be embedded into different projects, websites, etc. SOURCE. Therefore, not only does the pygal library offer multiple charting options beyond what you consider standard charts suc as bar charts, line charts, and pie graphs, but it also includes a world map, funnel charts, radar charts, and box plots, to name just a few.

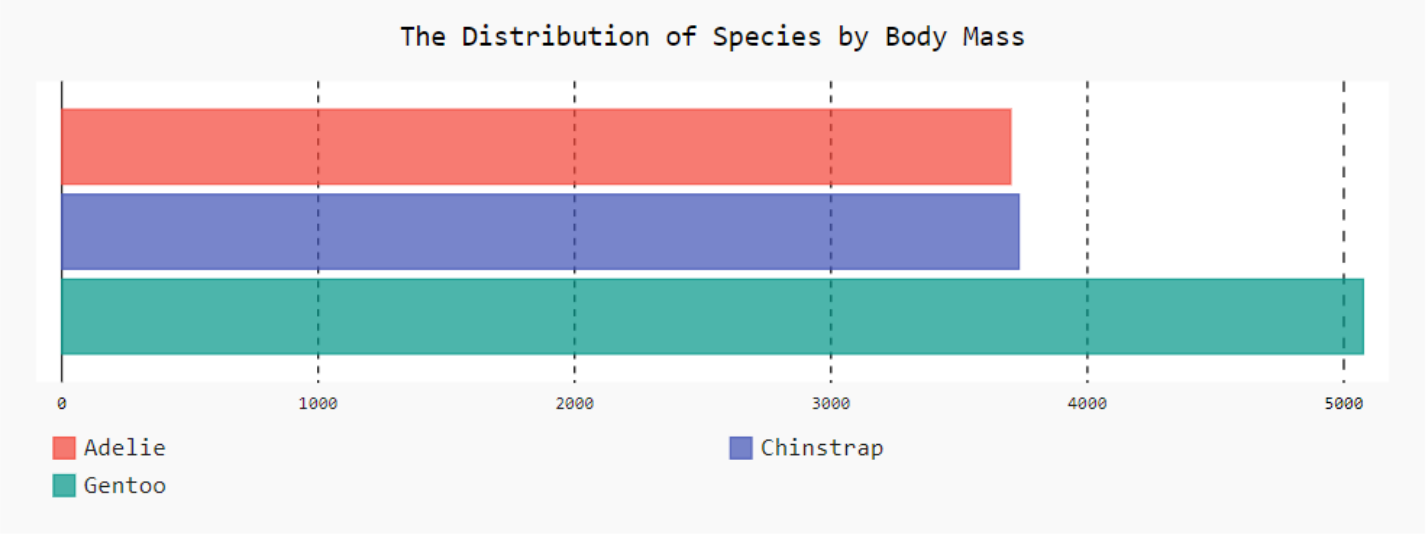
Let's create a basic Bar Chart using Pygal library:

```
# creating  bar chart object
barChart = pygal.Bar(height=200)

# naming the title
barChart.title = 'The Distribution of Species by Body Mass'

[barChart.add(x[0], x[1]) for x in mean_per_body.items()]

display(HTML(base_html.format(rendered_chart=barChart.render(is_unicode=True))))
```



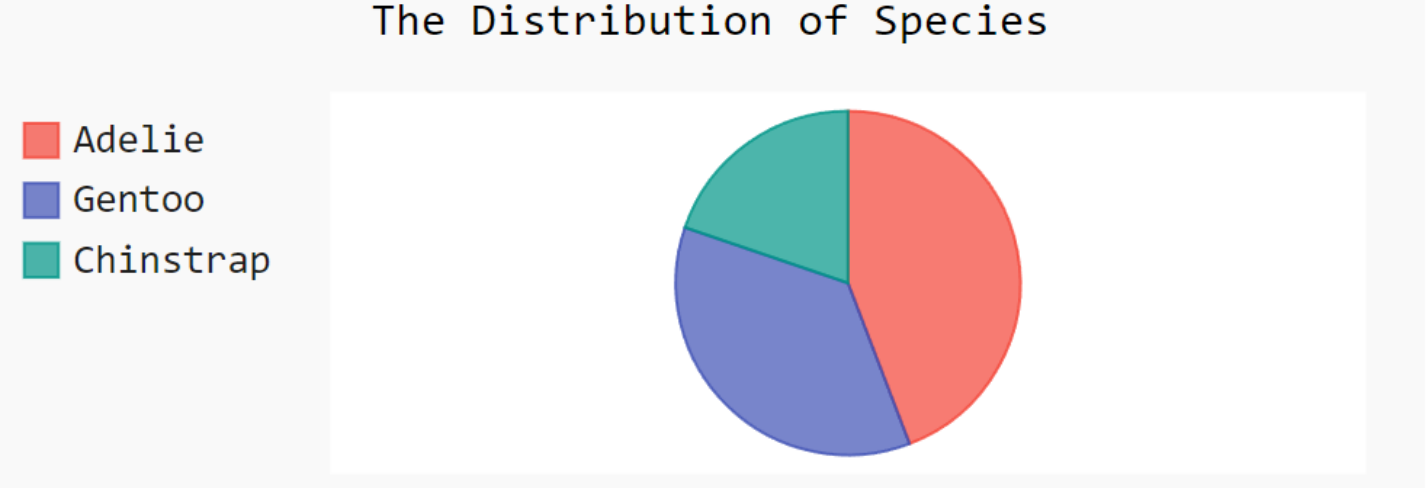
Let's horizontally create our Bar Chart using Pygal library:

```
import pygal
from IPython.display import SVG, display

# creating horizontal bar chart object
bar_chart = pygal.HorizontalBar(height=300, legend_at_bottom=True)

# naming the title
bar_chart.title = 'The Distribution of Species by Body Mass'

# we are adding the data from our dataset by iterating over the data set using itterrow()
for x in mean_per_body.items():
    bar_chart.add(x[0], x[1])

# rendering the file
display(SVG(bar_chart.render(is_unicode=True)))
# bar_chart.render_to_file("bar chart.svg")
```

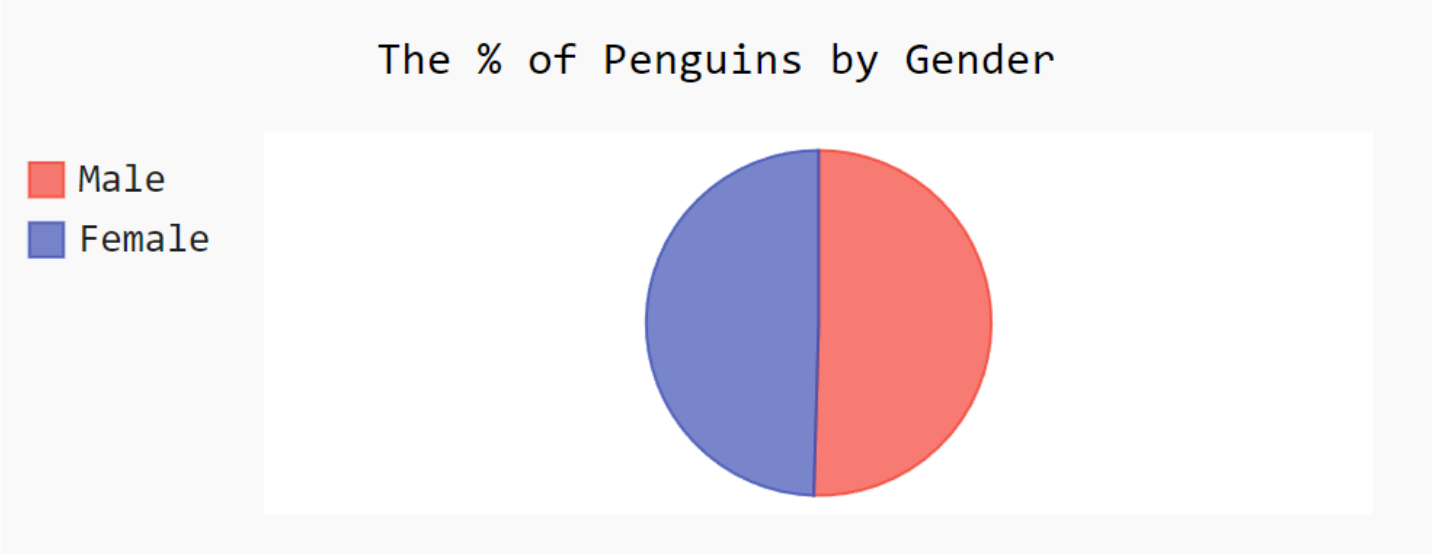


Now, let's create a basic Bar Chart using Pygal library:

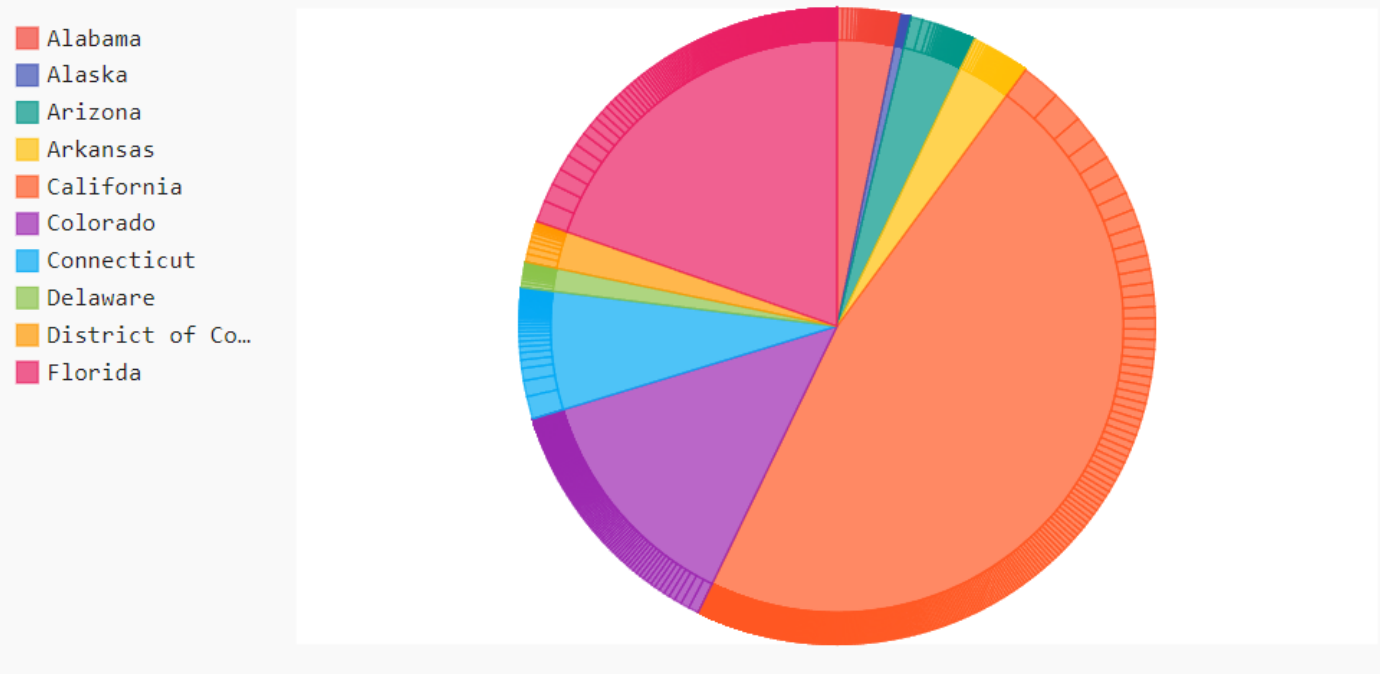
```
series = penguins.species.value_counts()

# Creates a Pygal chart
pie_chart = pygal.Pie(width=500, height=200)

# Adds a title (heading)
pie_chart.title = "The Distribution of Species"

# Add the data to be displayed
for idx, name in enumerate(series.index.tolist()):
    pie_chart.add(name, series.values[idx])
```



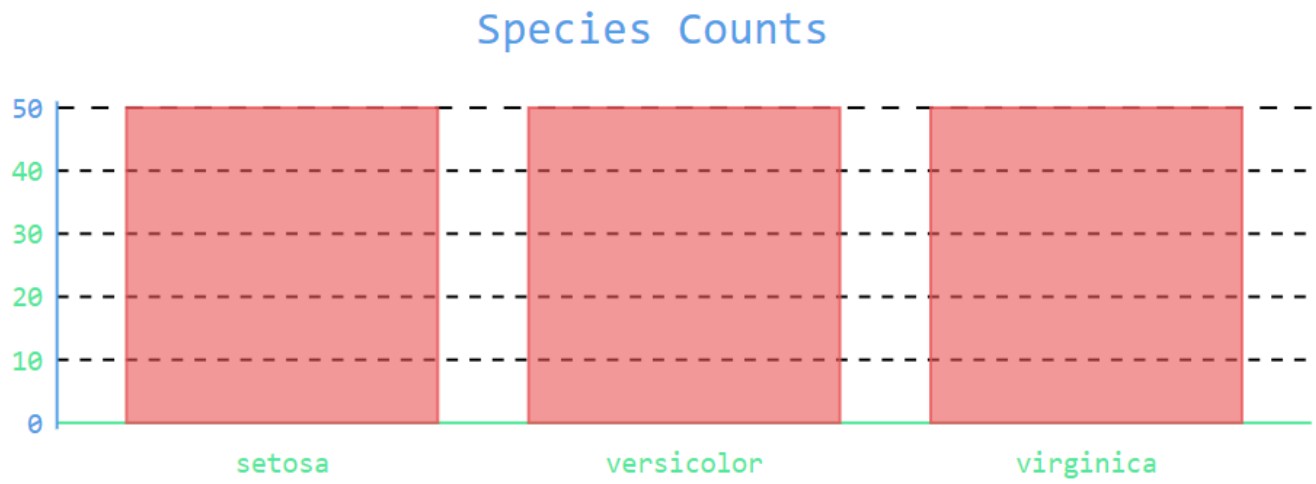
```
series1 = penguins.sex.value_counts(normalize=True)*100

pie_chart = pygal.Pie(width=500, height=200)
pie_chart.title = 'The % of Penguins by Gender'

for idx, name in enumerate(series1.index.tolist()):
    pie_chart.add(name, series1.values[idx])

pie_chart.value_formatter = lambda x: "%.2f%%" % x
display(HTML(base_html.format(rendered_chart=pie_chart.render(is_unicode=True))))
```




```
#Draw the bar chart
pie_chart = pygal.Pie(height=400)

#Get the top 10 states
first10 = list(sort_by_cases.items())[:10]
[pie_chart.add(x[0], x[1]) for x in first10]

display(HTML(base_html.format(rendered_chart=pie_chart.render(is_unicode=True))))
```

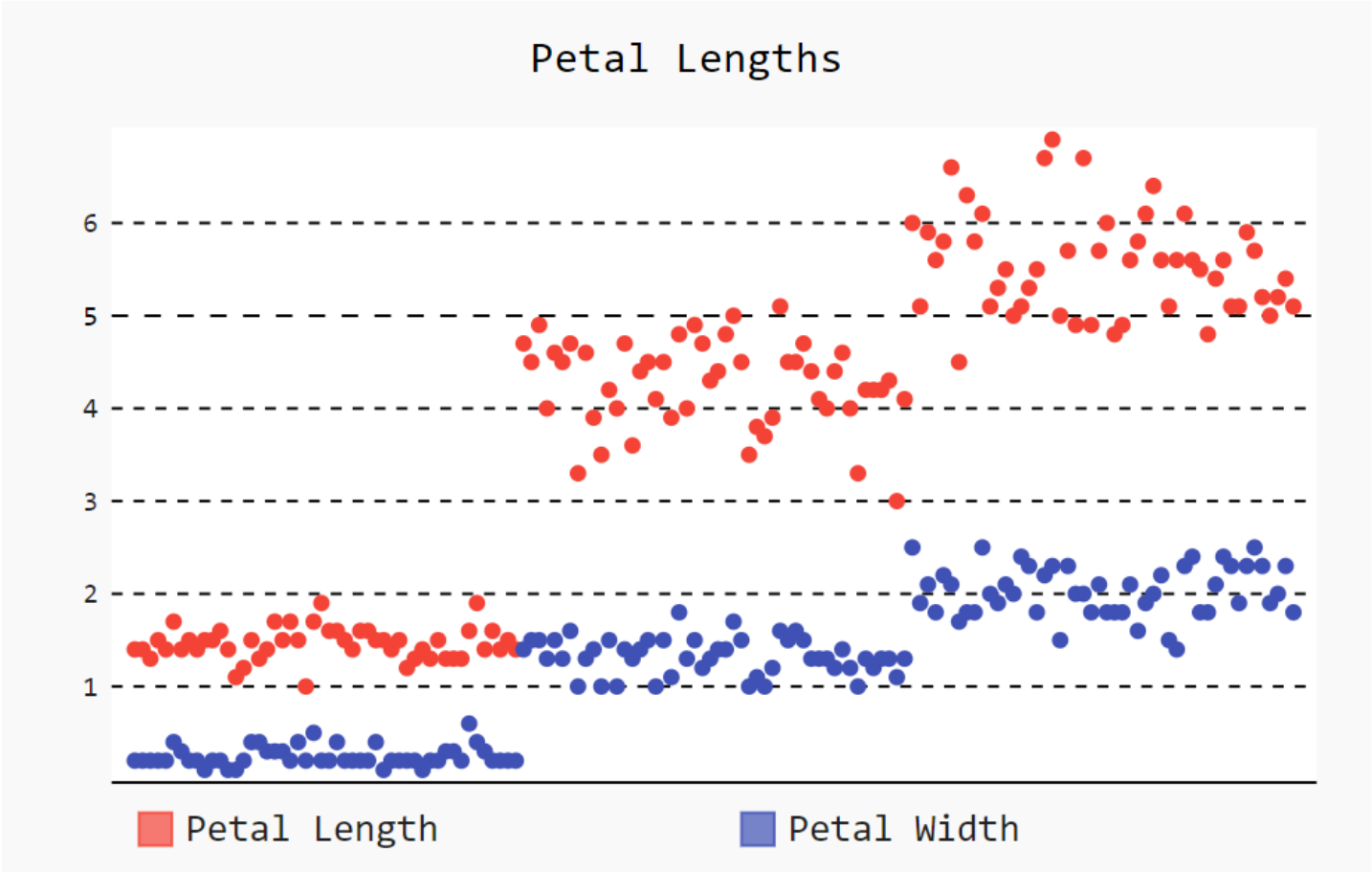


Now it's time to create a Bar Chart the number of each species by custom style:



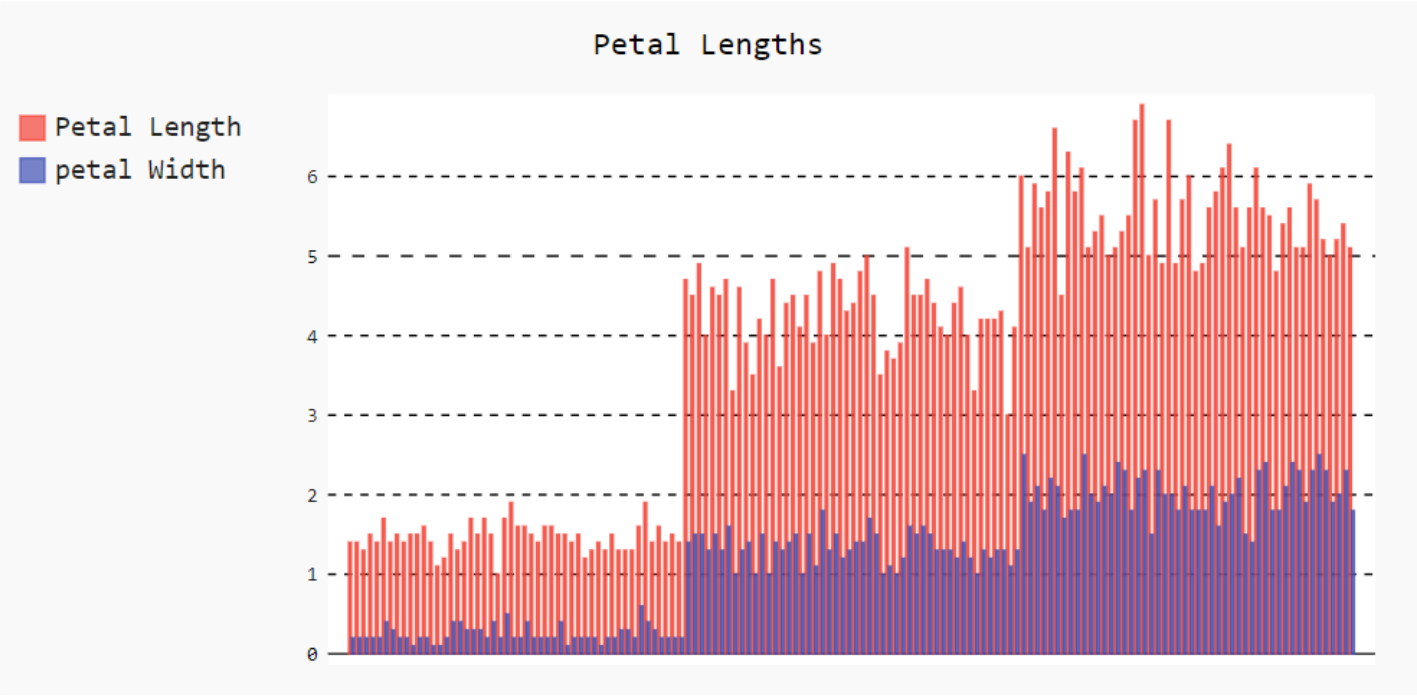
Let us examine the relationship between Petal Length & Petal Width by plotting a Scatter Chart.

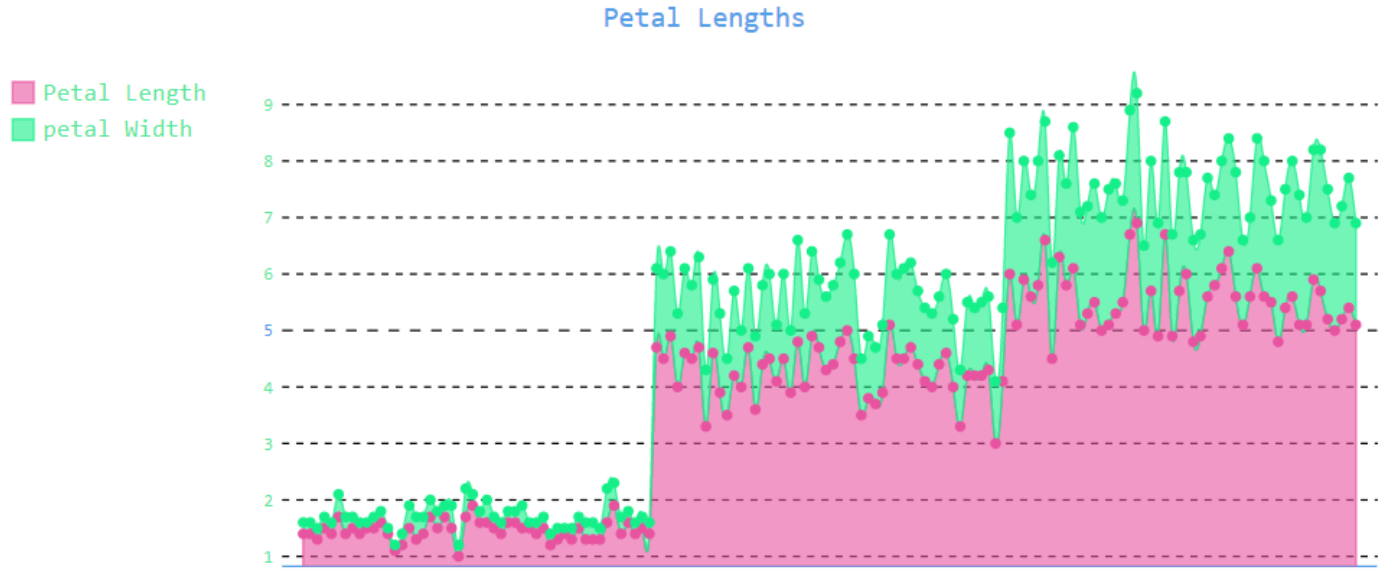
```
# we can create a scatter plot looking at Petal Lengths

line_chart = pygal.Line(width=500,
                        height=350,
                        stroke = False,legend_at_bottom=True)
line_chart.title = "Petal Lengths"
line_chart.add("Petal Length", iris['petal_length'])
line_chart.add("Petal Width", iris['petal_width'])

display(HTML(base_html.format(rendered_chart=line_chart.render(is_unicode=True))))
# line_chart.render_to_file('IrisScatterChart.svg')
```



Let's go back to iris dataset to see the distribution of both Petal Length & Petal Width at the same Histogram Chart.

```
# We can also create a histogram comparing petal length and width

hist_chart = pygal.Bar(width=700, height=350)
hist_chart.title = "Petal Lengths"
hist_chart.add("Petal Length", iris["petal_length"])
hist_chart.add("petal Width", iris["petal_width"])

display(HTML(base_html.format(rendered_chart=hist_chart.render(is_unicode=True))))
# hist_chart.render_to_file('hist_chart.svg')
```





ALTAIR LIBRARY

What is Altair library?

Altair is a declarative statistical visualization library for Python, based on Vega and Vega-Lite, and the source is available on GitHub. With Altair, you can spend more time understanding your data and its meaning.

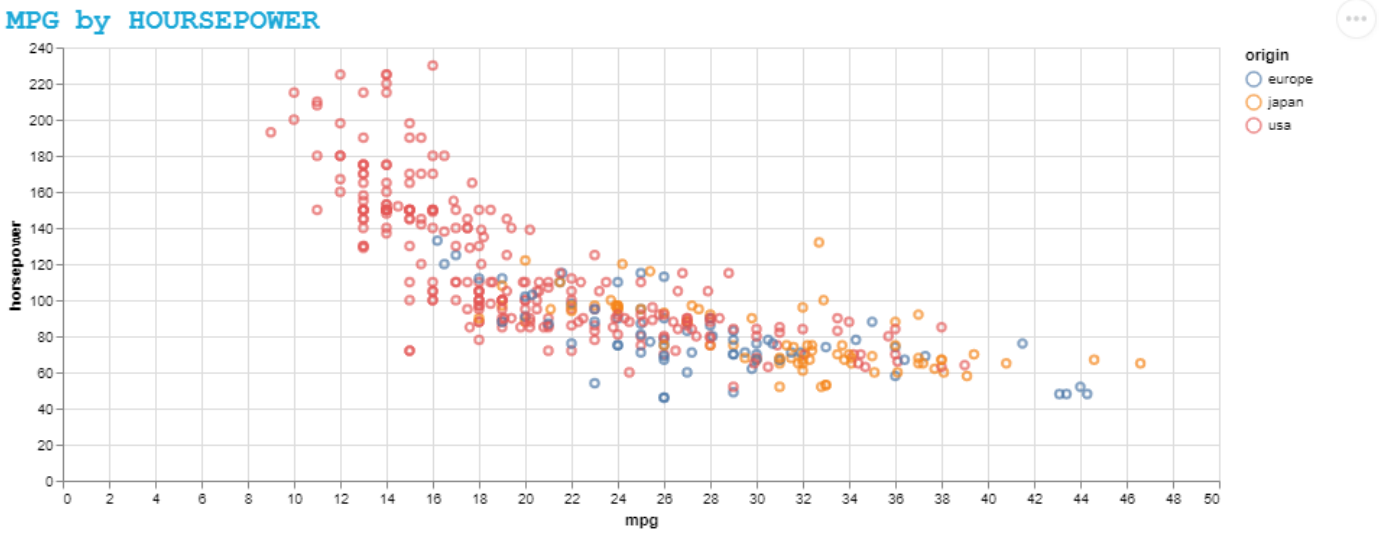
What is Altair Data Visualization?

Altair offers a comprehensive suite of data visualization software suitable for enterprise deployment. Business users, engineers, and analysts can connect to virtually any data source and build data monitoring, analysis, and reporting applications without writing a single line of code.

What is Altair chart?

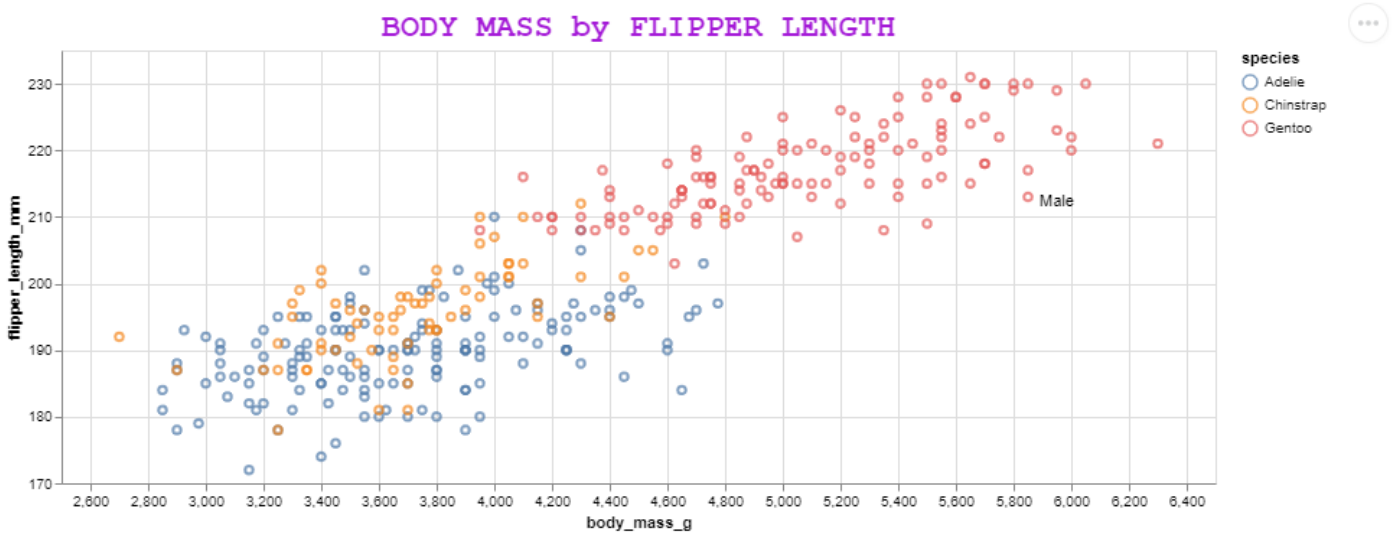
Image result for altair library This chart is created with Python Data Visualization library Altair. Altair is a declarative statistical visualization library for Python, based on Vega and Vega-Lite. Altair offers a powerful and concise visualization grammar that enables you to build a wide range of statistical visualizations quickly

```
# the 'empty' setting makes all text hidden before any mouseover occurs.
pointer = alt.selection_single(on='mouseover', nearest=True, empty='none')

base = alt.Chart(title="MPG by HOURSEPOWER").encode(x='mpg', y='horsepower')

chart = alt.layer(base.mark_point()
            .properties(selection=pointer, width=800, height=300)
            .encode(color='origin'),
        base.mark_text(dx=8, dy=3, align='left')
            .encode(text=alt.condition(pointer, 'name', alt.value(''))), data=mpg)

chart = chart.configure_title(fontSize=20,
                              font='Courier',
                              anchor='start',
                              color='#1BA8D8')
chart
```



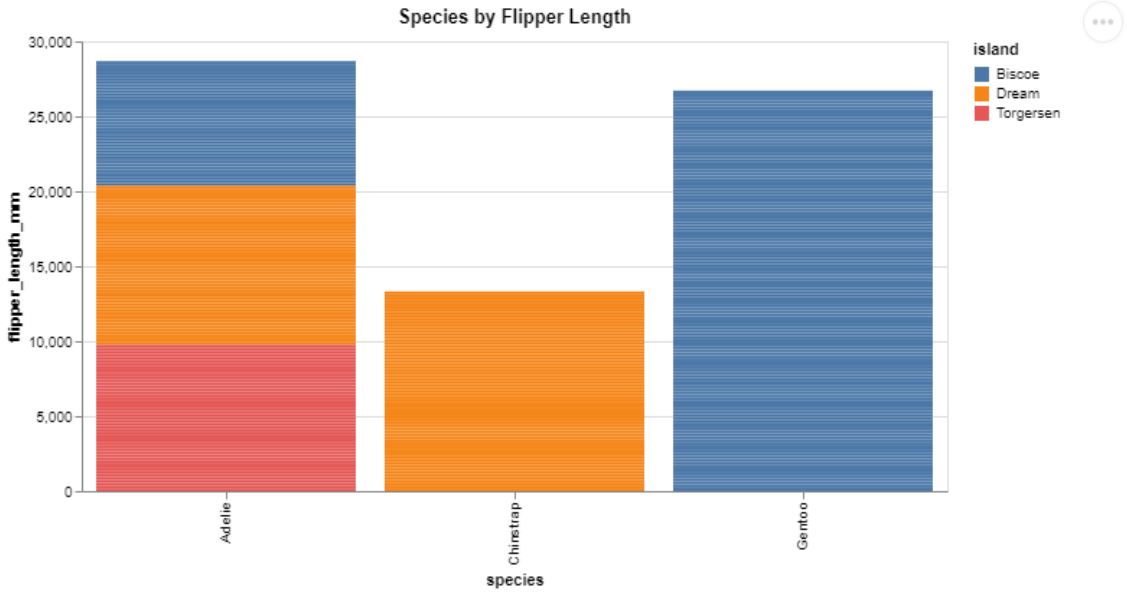
How about the relationship between 'body_mass_g' & 'flipper_length_mm'?

```
# the 'empty' setting makes all text hidden before any mouseover occurs.
pointer = alt.selection_single(on='mouseover', nearest=True, empty='none')

base = alt.Chart(title="BODY MASS by FLIPPER LENGTH").encode(alt.X('body_mass_g', scale=alt.Scale(zero=False)),
                                                            alt.Y('flipper_length_mm', scale=alt.Scale(zero=False)))

chart = alt.layer(base.mark_point()
          .properties(selection=pointer, width=800, height=300)
          .encode(color='species'), base.mark_text(dx=8, dy=3, align='left')
          .encode(text=alt.condition(pointer, 'sex', alt.value(''))), data=penguins)

chart = chart.configure_title(fontSize=22,
                              font='Courier',
                              anchor='middle',
                              color='#A51BD8')

chart
```



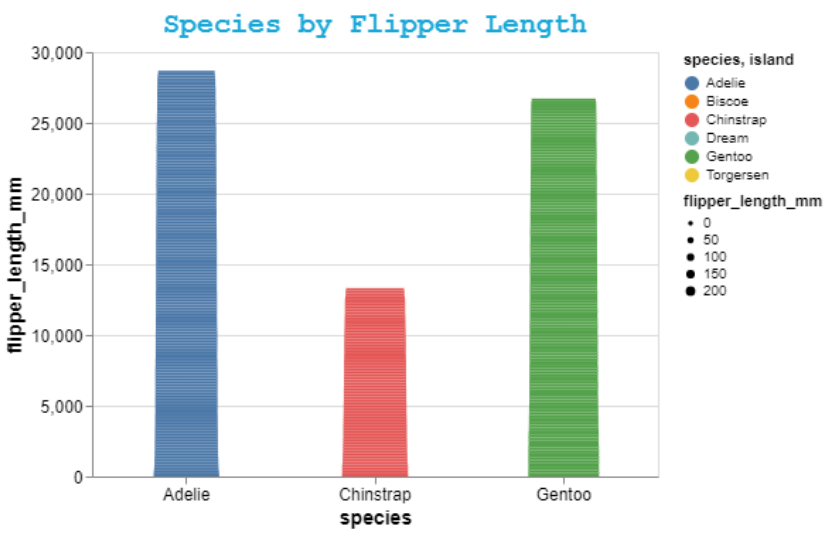
First we will create a static Bar Chart using Altair library.

```
alt.Chart(penguins, title="Species by Flipper Length").mark_bar().encode(x='species',
                                    y='flipper_length_mm',
                                    color="island")\
                                .properties(width=600, height=300)
```



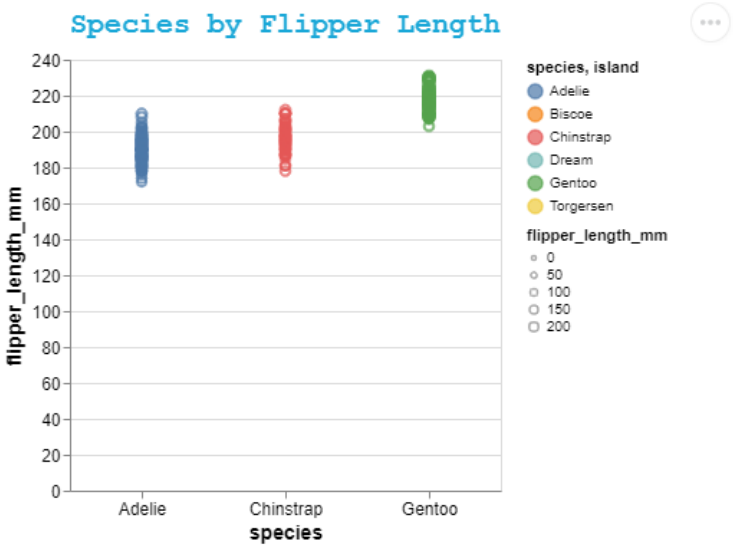
Now it's time to convert this static Bar Chart into interactive one and let's see what's happening.

```
# the 'empty' setting makes all text hidden before any mouseover occurs.
pointer = alt.selection_single(on='mouseover', nearest=True, empty='none')

base = alt.Chart(penguins, title="Species by Flipper Length").mark_bar().encode(x='species',
                                        y='flipper_length_mm',
                                        color="island",
                                        size=alt.Size('flipper_length_mm', scale=alt.Scale(range=[10, 50])))

chart = alt.layer(base.mark_bar()
           .properties(selection=pointer, width=400, height=300)
           .encode(color='species'), base.mark_text(dx=8, dy=3, align='left')
           .encode(text=alt.condition(pointer, 'island', alt.value('')), data=penguins)\
           .configure_axis(labelFontSize=12, titleFontSize=14, labelAngle=0)

chart = chart.configure_title(fontSize=20,
                              font='Courier',
                              anchor='middle',
                              color='#1BA8D8')

chart
```



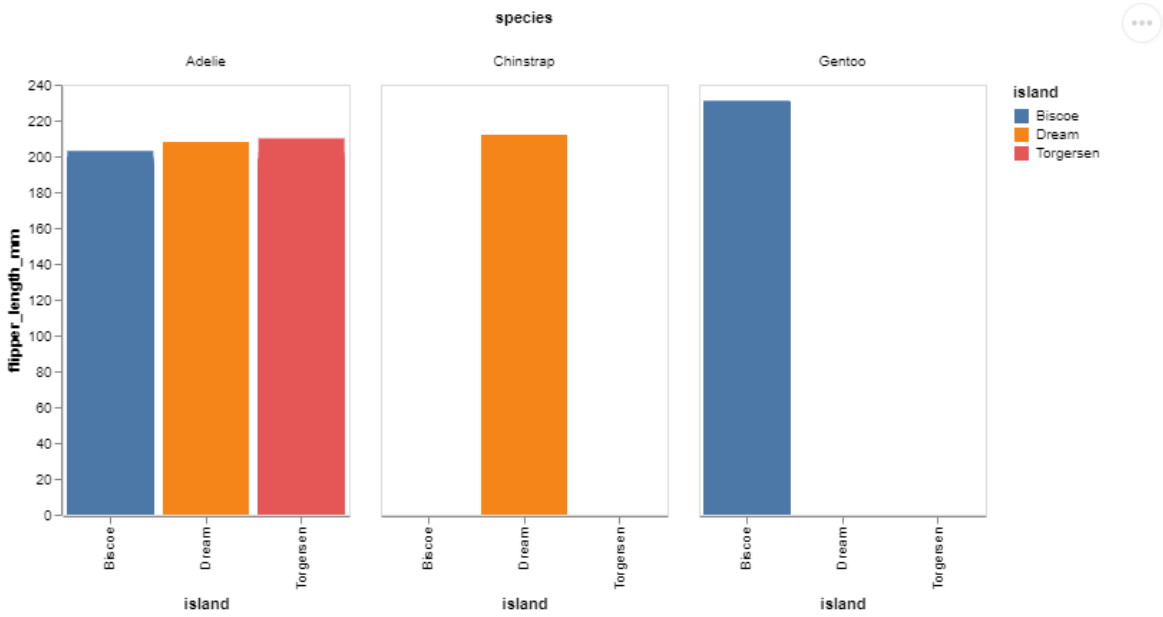
Altair library also provides us to create Point Chart.

```
# the 'empty' setting makes all text hidden before any mouseover occurs.
pointer = alt.selection_single(on='mouseover', nearest=True, empty='none')

base = alt.Chart(penguins, title="Species by Flipper Length").mark_bar().encode(x='species',
                                          y='flipper_length_mm',
                                          color="island",
                                          size=alt.Size('flipper_length_mm', scale=alt.Scale(range=[10, 50])))

chart = alt.layer(base.mark_point()
          .properties(selection=pointer, width=300, height=300)
          .encode(color='species'), base.mark_text(dx=8, dy=3, align='left')
          .encode(text=alt.condition(pointer, 'island', alt.value('')), data=penguins)\
          .configure_axis(labelFontSize=12, titleFontSize=14, labelAngle=0)

chart = chart.configure_title(fontSize=20,
                              font='Courier',
                              anchor='middle',
                              color='#1BA8D8')

chart
```



NOTE : However, in Altair Library, Faceted charts cannot be layered. They remain static. What a shame!

```
# Faceted charts cannot be layered

group_chart = alt.Chart(penguins).mark_bar()\
                            .encode(alt.Column('species'),
                                    alt.X('island'),
                                    alt.Y('flipper_length_mm', axis=alt.Axis(grid=False)), alt.Color('island'))\
                            .properties(width=200, height=300)


group_chart.display()
```



HOLOVIEWS LIBRARY

HoloViews is an open-source Python library designed to make data analysis and visualization seamless and simple. With HoloViews, you can usually express what you want to do in very few lines of code, letting you focus on what you are trying to explore and convey, not on the process of plotting SOURCE.

image.png

What is HoloViews?

HoloViews allows you to collect and annotate your data in a way that reveals it naturally, with a minimum of effort needed for you to see your data as it actually is. HoloViews is not a plotting library -- it connects your data to plotting code implemented in other packages, such as matplotlib or Bokeh. HoloViews is also not primarily a mass storage or archival data format like HDF5 -- it is designed to package your data to make it maximally visualizable and viewable interactively.

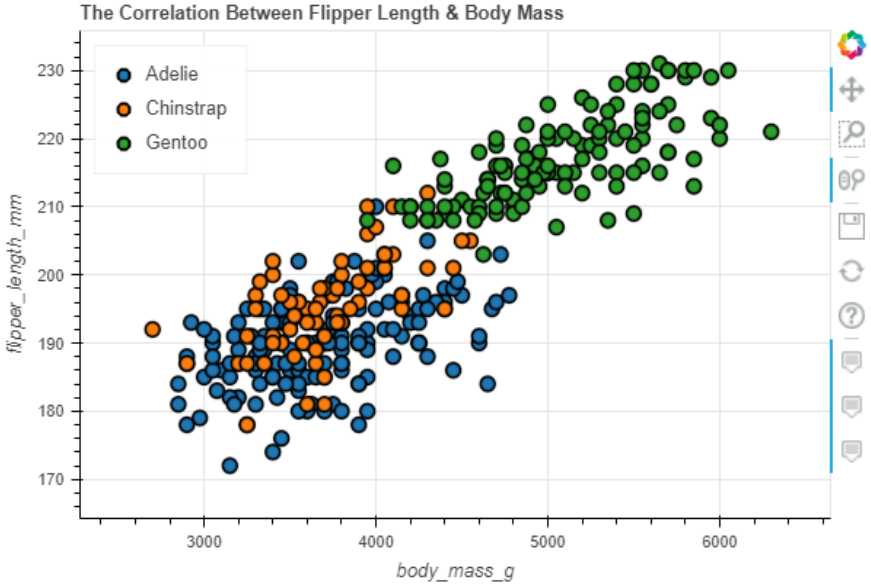
If you supply just enough additional information to the data of interest, HoloViews allows you to store, index, slice, analyze, reduce, compose, display, and animate your data as naturally as possible. HoloViews makes your numerical data come alive, revealing itself easily and without extensive coding SOURCE.

Important Information

Holoviews maintains metadata about your plot and it does not do any plotting by itself. It just organizes metadata about how to plot data and uses it's underlying back-end library like bokeh, matplotlib or plotly to actually plot data.

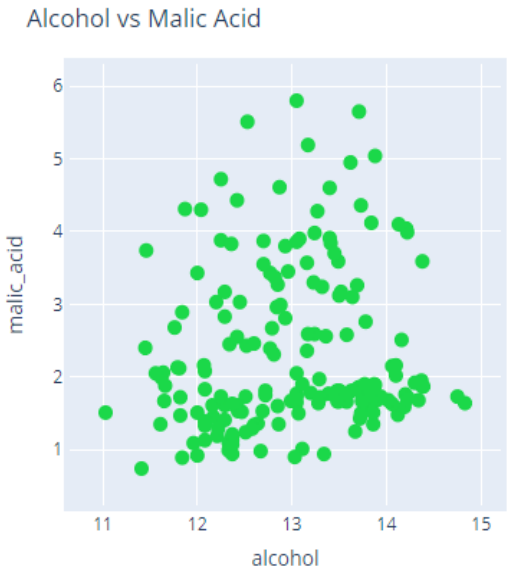
Let us first remember what Bokeh gives us.

```
p = penguins.plot_bokeh.scatter(x='body_mass_g',
                                y='flipper_length_mm', category='species',
                                legend = "top_left",
                                title='The Correlation Between Flipper Length & Body Mass')
```



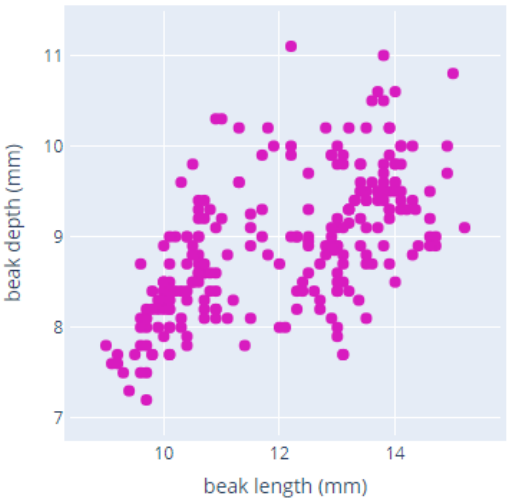
We are plotting the scatter plot below to show the relationship between alcohol and malic_acid values in wine. We can see that just one line of code is enough to create a simple interactive graph. We need to pass the first argument as a dataframe that maintains data and then kdims and vdims to represent x and y of a graph.

```
hv.extension("plotly")

scatter = hv.Points(data=wine_df,
                    kdims=["alcohol", "malic_acid"],
                    vdims="Target",
                    label="Alcohol vs Malic Acid")

scatter.opts(color='#1BD849', size=10, marker='circle')
```

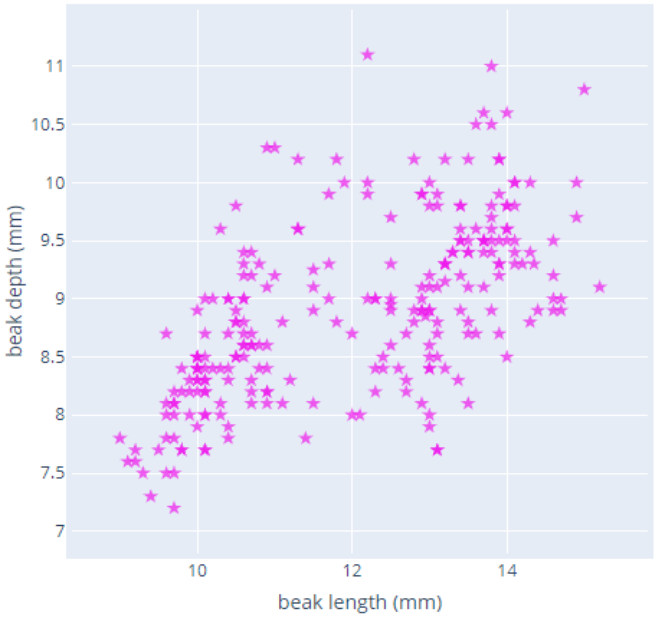


Let us create our first basic chart by playing with fundamental parameters.

```
hv.extension("plotly")

df_2012 = df.loc[df['year']==2012, :].copy()

hv.Points(data=df_2012,
          kdims=['beak length (mm)', 'beak depth (mm)'],
          vdims=['species']).opts(color='#D81BBF', size=8, marker='circle')
```

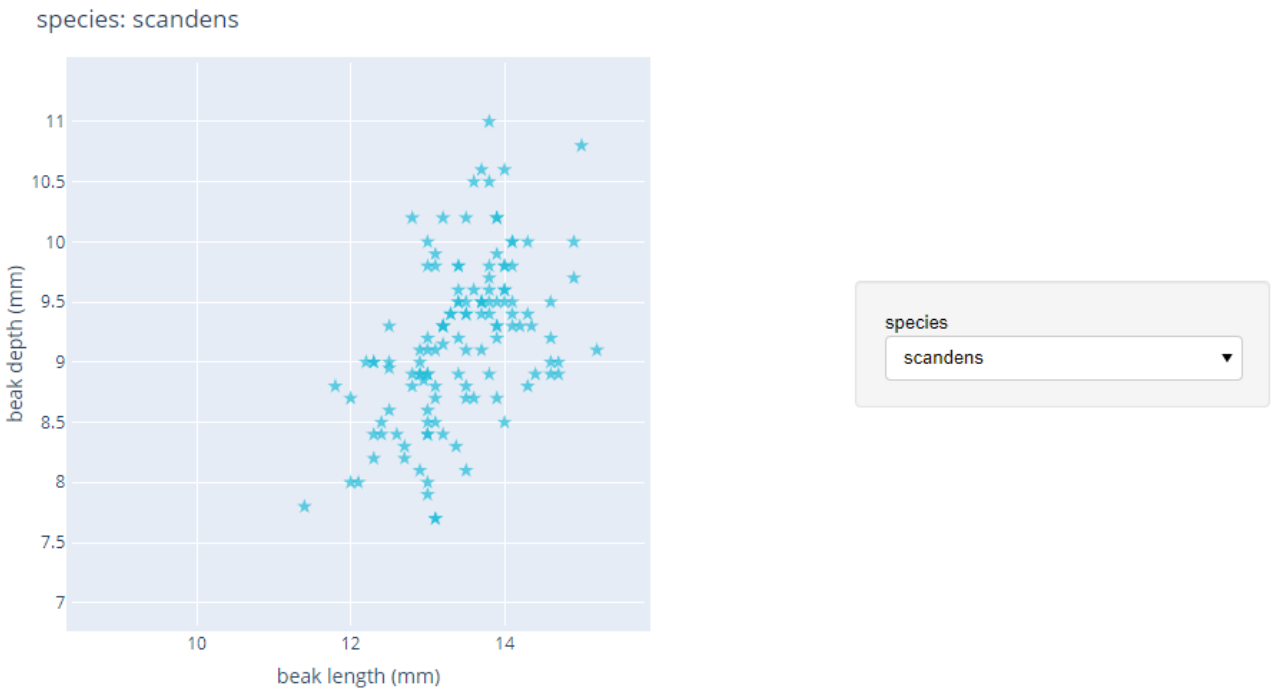



```
hv.extension("plotly")

hv.Points(data=df_2012,
          kdims=['beak length (mm)', 'beak depth (mm)'],
          vdims=['species']).opts(alpha=0.7,
                                  color='#EF15EC',
                                  size=8,
                                  marker='star',
                                  height=500,
                                  width=500,
                                  show_grid=True)
```



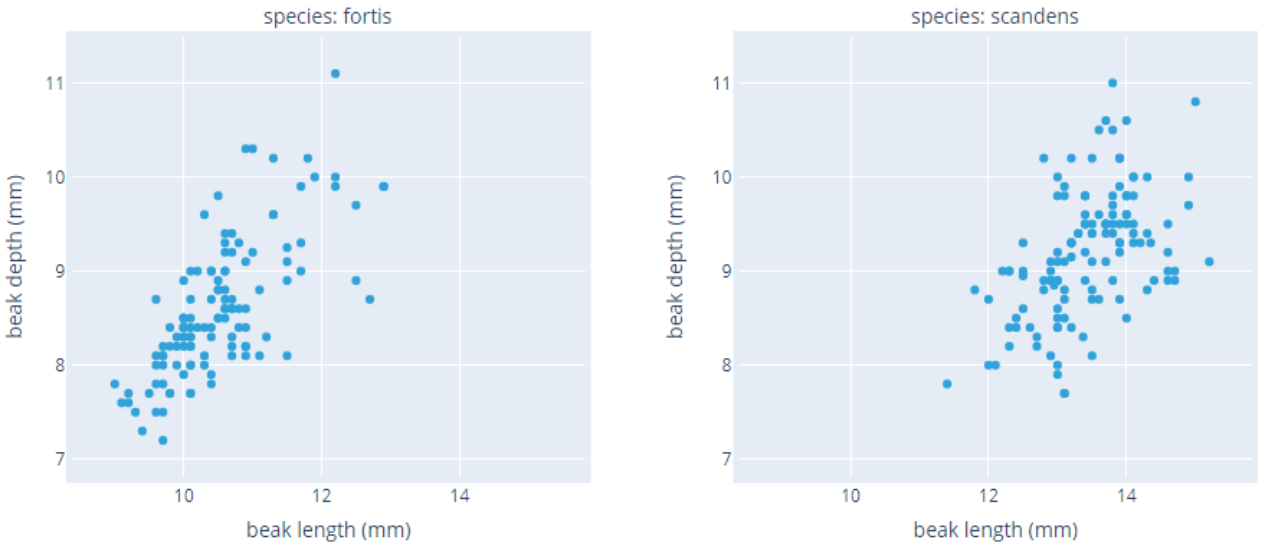
Sometimes we need to separate out the glyphs by species. To do this, we can do a groupby operation on the Element. That's right, we can do groupby operations on graphical elements!

```
hv.extension("plotly")

hv.Points(data=df_2012,
          kdims=['beak length (mm)', 'beak depth (mm)'],
          vdims=['species']).groupby('species').opts(alpha=0.7,
                                                     color='#1BBCD8',
                                                     size=8,
                                                     marker='star',
                                                     height=500,
                                                     width=500,
                                                     show_grid=True)
```



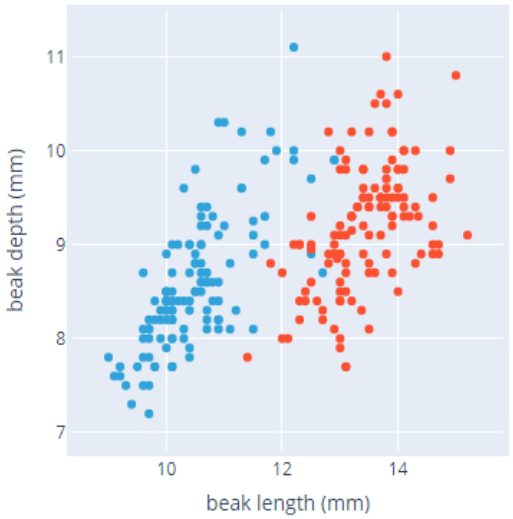
Sometimes we may be interested with grouping species and laying the plots out next to each other, creating a layout. We can use the layout() method do to this.

```
hv.extension("plotly")

hv.Points(data=df_2012,
          kdims=['beak length (mm)', 'beak depth (mm)'],
          vdims=['species']).groupby('species').opts(height=400, width=400).layout()
```



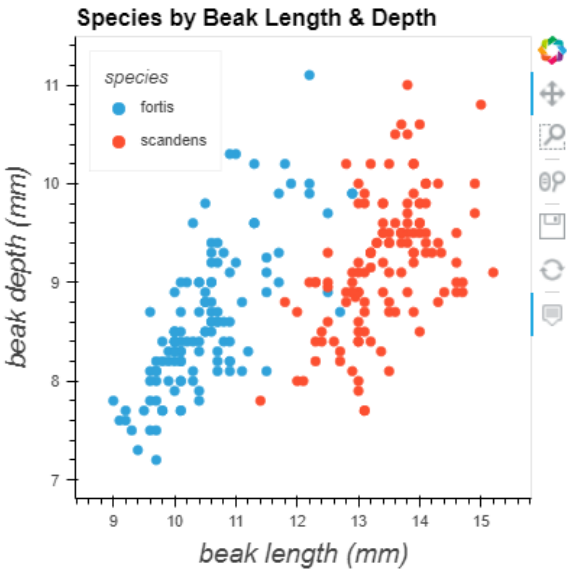
If we may wish to split each species at the same chart, we need to use overlay.

```
hv.extension("plotly")

hv.Points(data=df_2012,
          kdims=['beak length (mm)', 'beak depth (mm)'],
          vdims=['species']).groupby('species').opts(height=400, width=400).overlay()
```

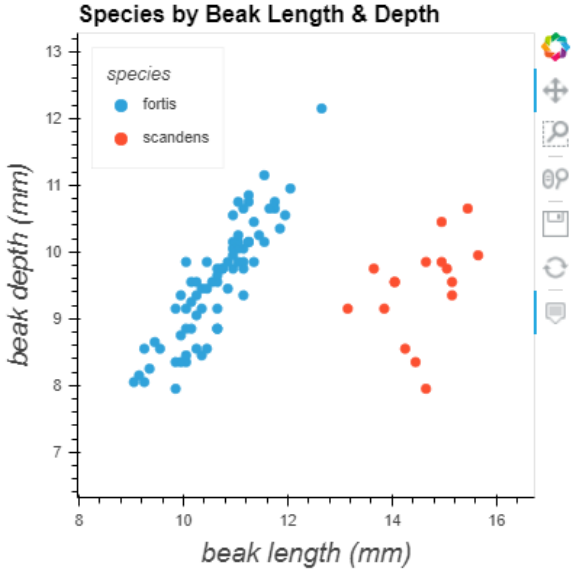


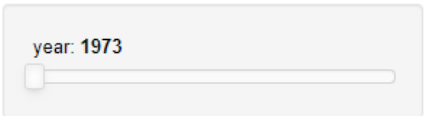
We can use .opts() to add tooltips where we can hover and get additional information from the vdims.

```
hv.extension("bokeh")

hv.Points(data=df_2012,
          kdims=['beak length (mm)', 'beak depth (mm)'],
          vdims=['species']).groupby('species').opts(tools=['hover'],
                                                     size=6,
                                                     legend_position='top_left',
                                                     fontsize={'title': 12,'labels': 14,'xticks': 8,'yticks': 8, "legend":8},
                                                     height=400,
                                                     width=400,
                                                     title='Species by Beak Length & Depth').overlay()
```



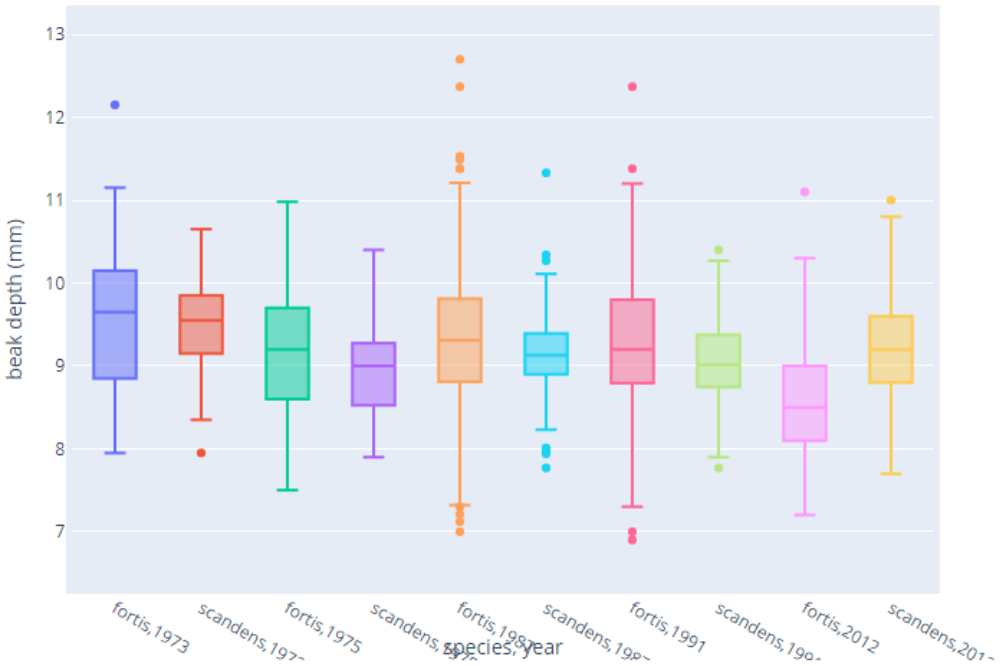

```
hv.extension("bokeh")

hv.Points(data=df,
          kdims=['beak length (mm)', 'beak depth (mm)'],
          vdims=['species', 'year']).groupby(['species', 'year']).opts(tools=['hover'],
                                                     size=6,
                                                     legend_position='top_left',
                                                     fontsize={'title': 12,'labels': 14,'xticks': 8,'yticks': 8, "legend":8},
                                                     height=400,
                                                     width=400,
                                                     title='Species by Beak Length & Depth').overlay('species')
```





Box plots are made using hv.BoxWhisker elements. If multiple key dimensions are specified, nested categorical axes are automatically set up.

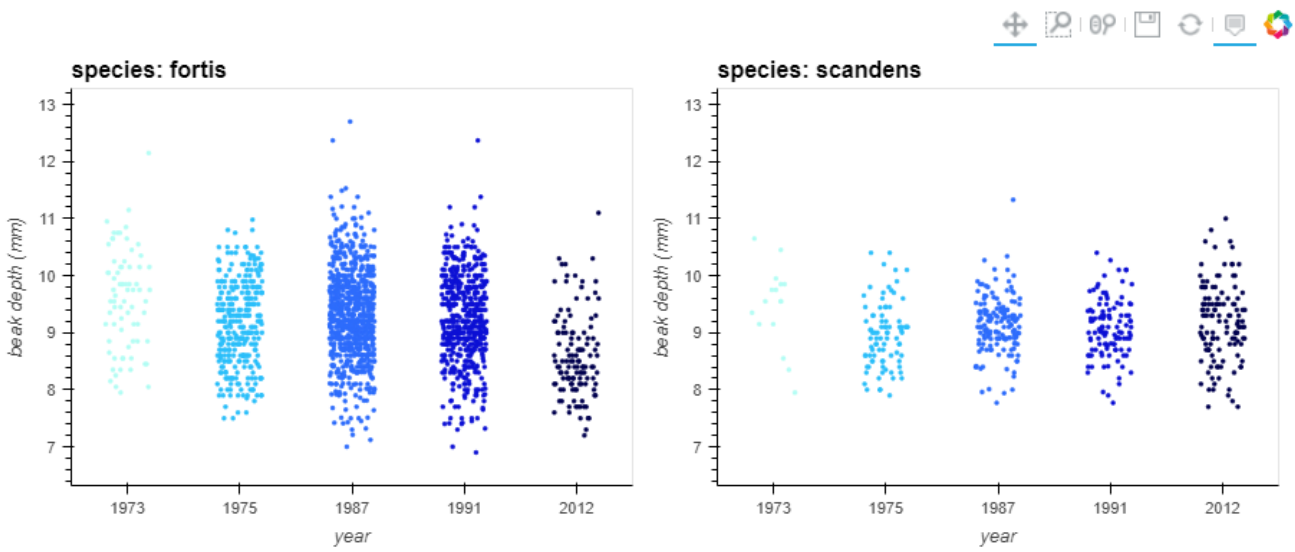
```
hv.extension("plotly")

hv.BoxWhisker(data=df,
              kdims=['species', 'year'],
              vdims=['beak depth (mm)']).opts(height=500, width=700)
```



We need to use hv.Scatter() to generate strip plots. When we specify the jitter kwargs, we specify the width of the jitter.

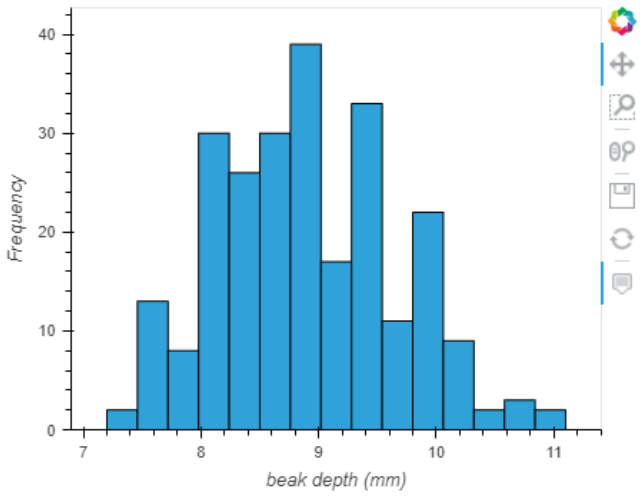
Note that nested categorical axes are currently (as of June 10, 2021) only supported for box, violin, and bar plots, as per the docs, but will eventually be supported for many more plot types, including Scatter, which are used to generate strip plots.

```
hv.extension("bokeh")

# Make the year column a string to be able to use as categorical

df['year_str'] = df['year'].astype(str)

hv.Scatter(data=df,
           kdims=[('year_str', 'year')],
           vdims=['beak depth (mm)', 'species']).groupby('species').opts(color='year',
                                                                         tools=['hover'],
                                                                         jitter=0.4,
                                                                         show_legend=False,
                                                                         width=450,
                                                                         height=350).layout()
```

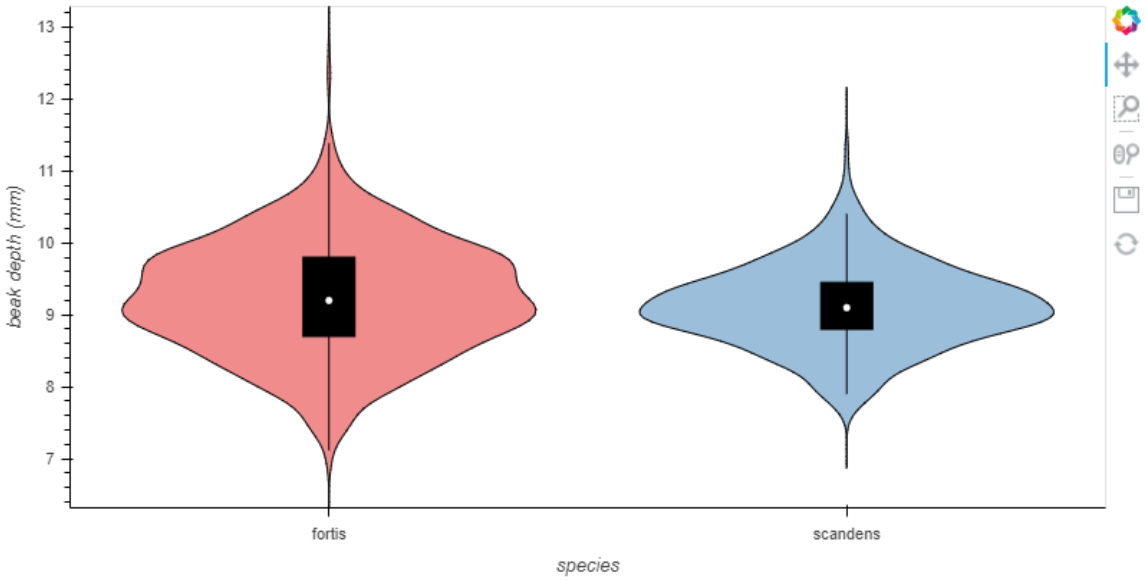


When making a histogram, the values of the bin edges and counts must be computed beforehand using np.histogram().

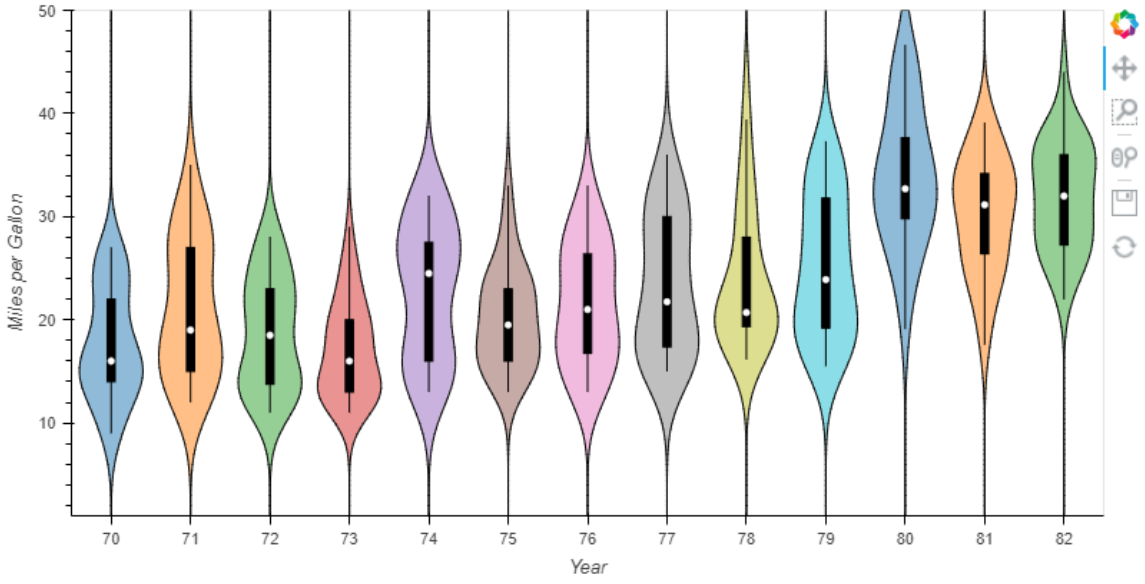
```
hv.extension("bokeh")

edges, counts = np.histogram(df_2012['beak depth (mm)'], bins=int(np.sqrt(len(df_2012))))

hv.Histogram(data=(edges, counts),
             kdims='beak depth (mm)').opts(tools=['hover'],
                                           show_legend=False,
                                           width=450,
                                           height=350)
```



```
violin = hv.Violin(df, "species", "beak depth (mm)")
violin.opts(height=400, width=800, violin_fill_color=dim('species'), cmap='Set1')
```



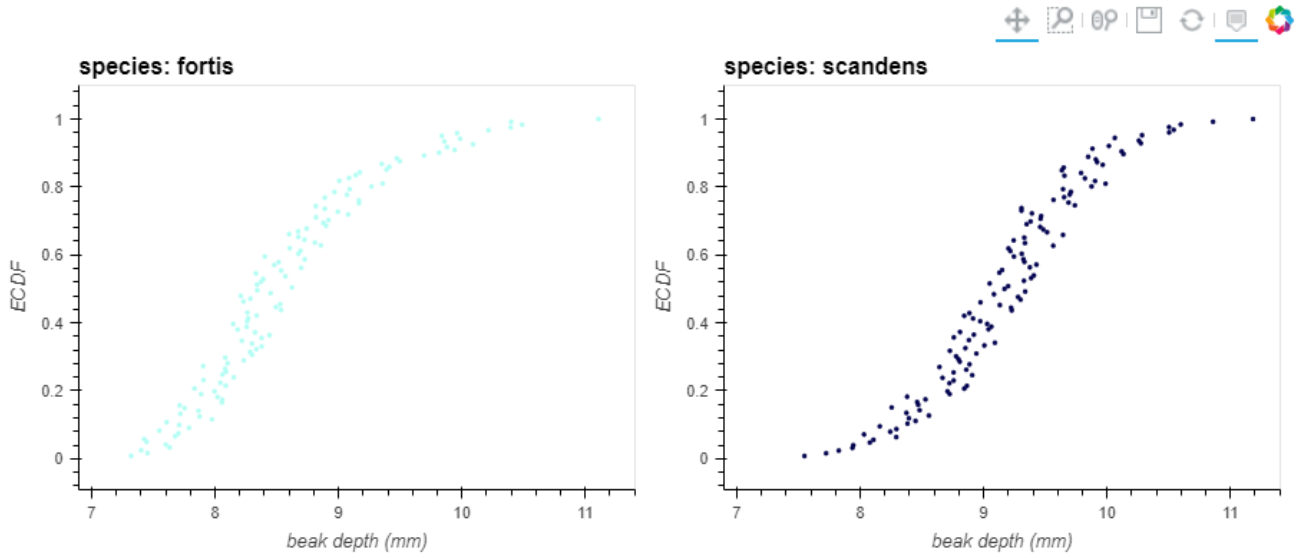

```
violin = hv.Violin(autompg, ('yr', 'Year'), ('mpg', 'Miles per Gallon')).redim.range(mpg=(1, 50))
violin.opts(height=400, width=800, violin_fill_color=dim('Year').str(), cmap='tab10')
```



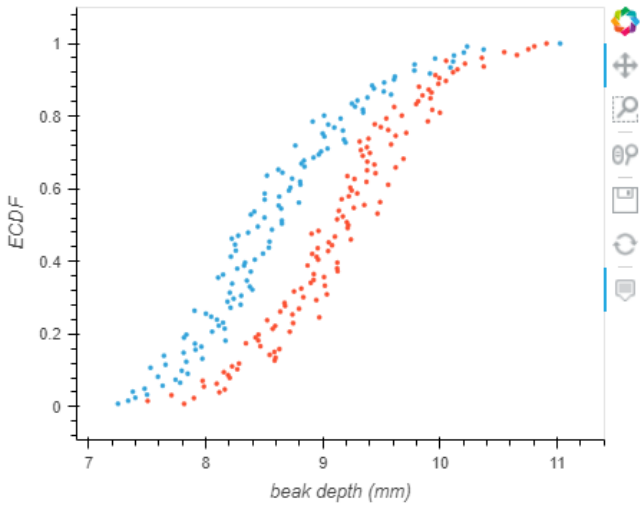
Empirical Cumulative Distribution Functions (ECDFs) are very important plots for visualizing how data are distributed.

```
def ecdf_transform(data):
    return data.rank(method="first") / len(data)

df_2012["beak depth ECDF"] = df_2012.groupby("species")["beak depth (mm)"].transform(ecdf_transform).values
```

```
hv.Scatter(data=df_2012,
           kdims='beak depth (mm)',
           vdims=[('beak depth ECDF', 'ECDF'), 'species']).groupby('species').opts(color='species',
                                                                                   tools=['hover'],
                                                                                   jitter=0.4,
                                                                                   show_legend=False,
                                                                                   width=450,
                                                                                   height=350).layout()
```




```
hv.Scatter(data=df_2012,
           kdims='beak depth (mm)',
           vdims=[('beak depth ECDF', 'ECDF'), 'species']).groupby('species').opts(tools=['hover'],
                                                                                   jitter=0.4,
                                                                                   show_legend=False,
                                                                                   width=450,
                                                                                   height=350).overlay('species')
```



PLOTLY LIBRARY

Plotly Express is a new Python visualization library that acts as a wrapper for Plotly, exposing a simple syntax for complex charts. It was inspired by Seaborn and ggplot2 and was specifically designed to have a terse, consistent, and easy-to-learn API: with a single import, you can make richly interactive plots with faceting, maps, animations, and trendlines in a single function call.

Plotly Express includes over 30 functions for creating various types of figures. The API for these functions was carefully designed to be as consistent and easy to learn as possible, allowing you to easily switch from a scatter plot to a bar chart to a histogram to a sunburst chart during a data exploration session.

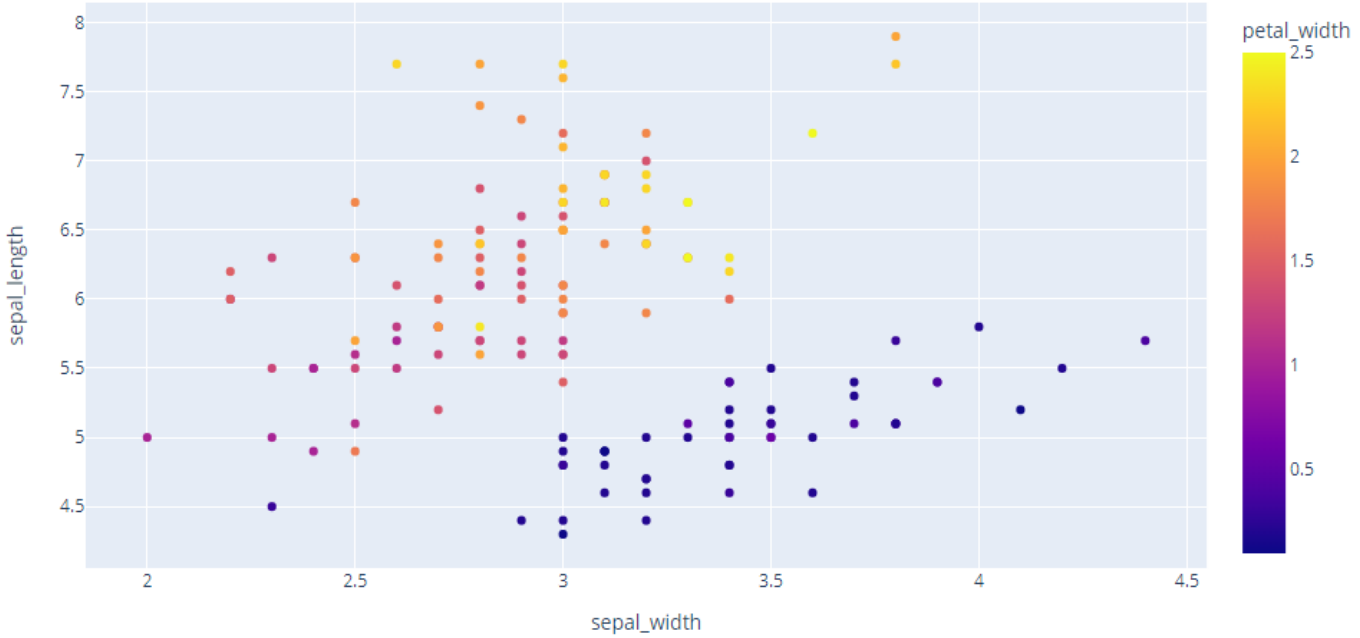
Why do we use Plotly Express?

Plotly Express simply **plots the points in the order they appear in the dataframe**. There is no sort parameter in the Plotly Express line() function, so your best bet would be to sort the data by the variable of interest and then plot the data accordingly.

How is Plotly used?

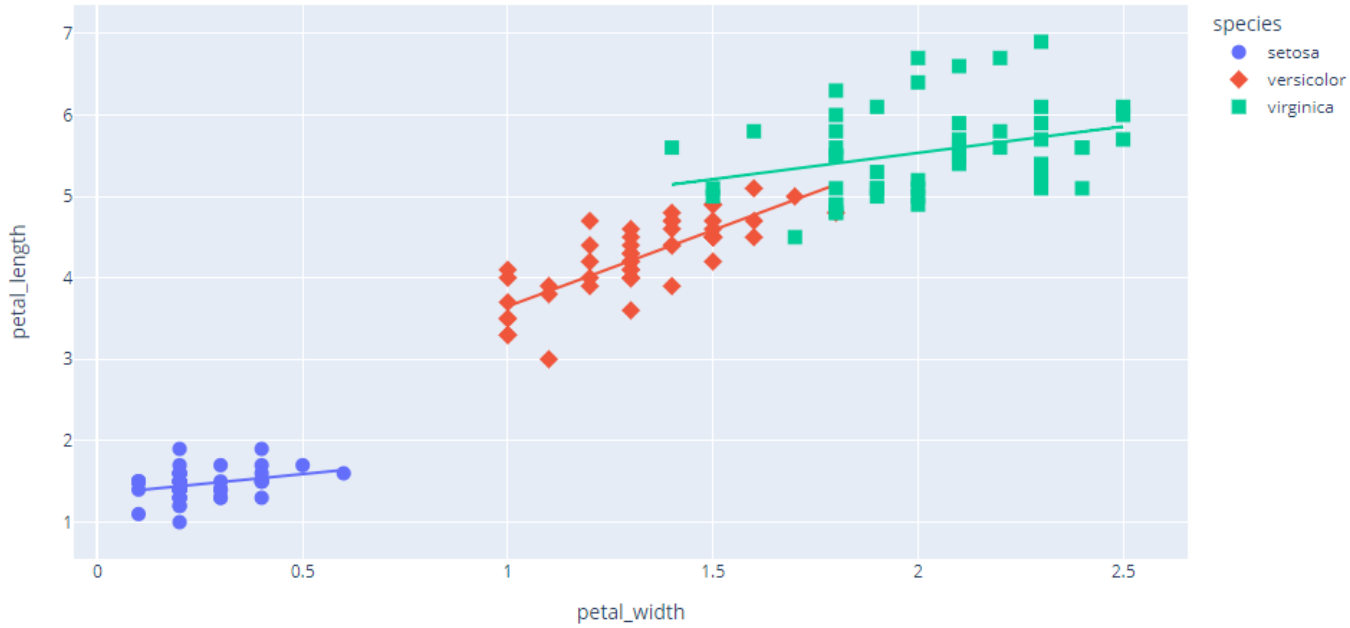
Plotly **provides online graphing, analytics, and statistics tools for individuals and collaboration**, as well as scientific graphing libraries for Python, R, MATLAB, Perl, Julia, Arduino, and REST.

```
fig = px.scatter(iris,
                 x="sepal_width",
                 y="sepal_length",
                 color='petal_width')
fig.show()
```

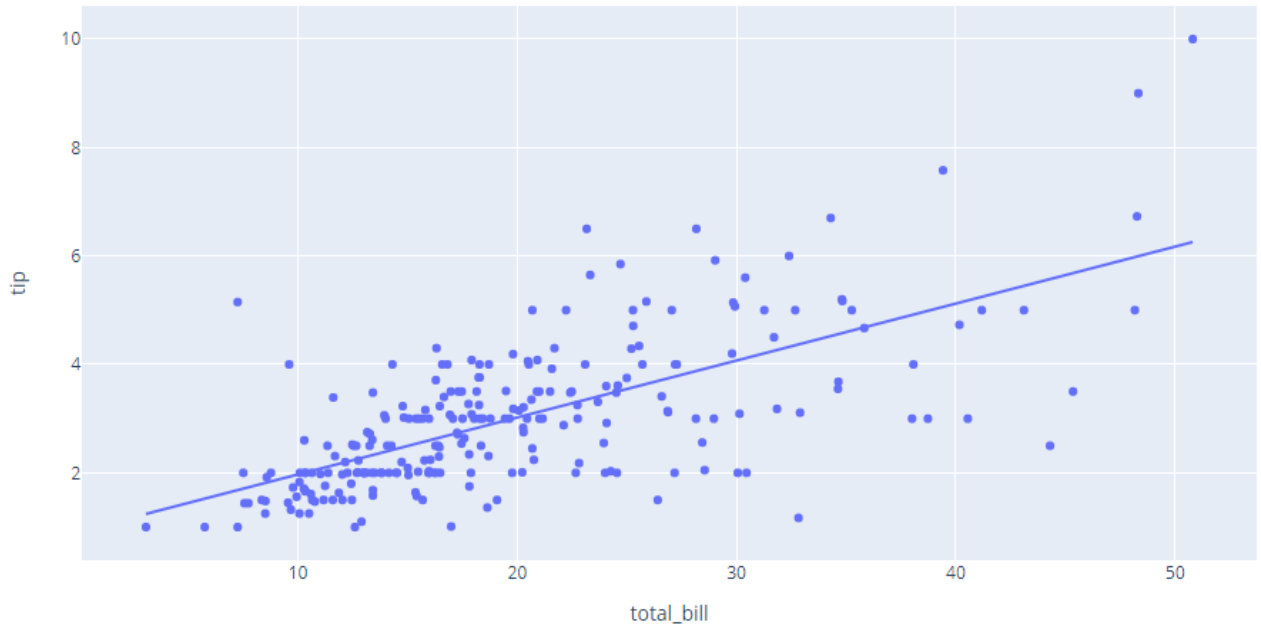


Let us add some styles to the plots.

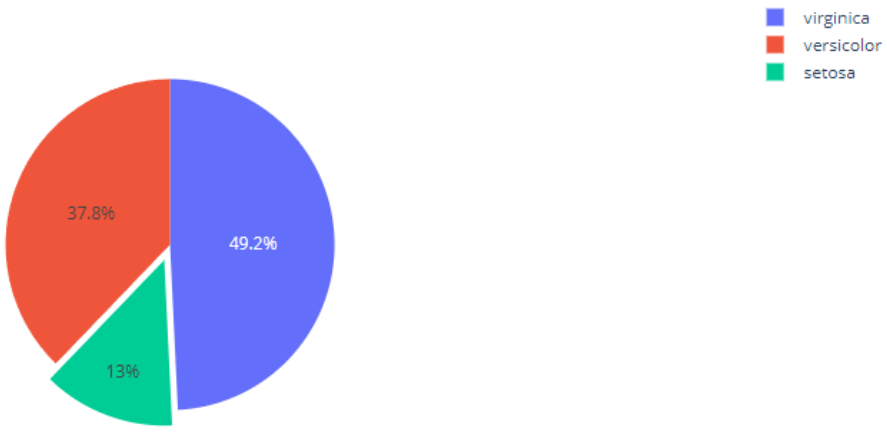
```
fig = px.scatter(iris,
                 y="petal_length",
                 x="petal_width",
                 color="species",
                 symbol="species",
                 trendline="ols")

fig.update_traces(marker_size=10)
```



```
fig = px.scatter(tips, x="total_bill", y="tip", trendline="ols")
fig.show()
```

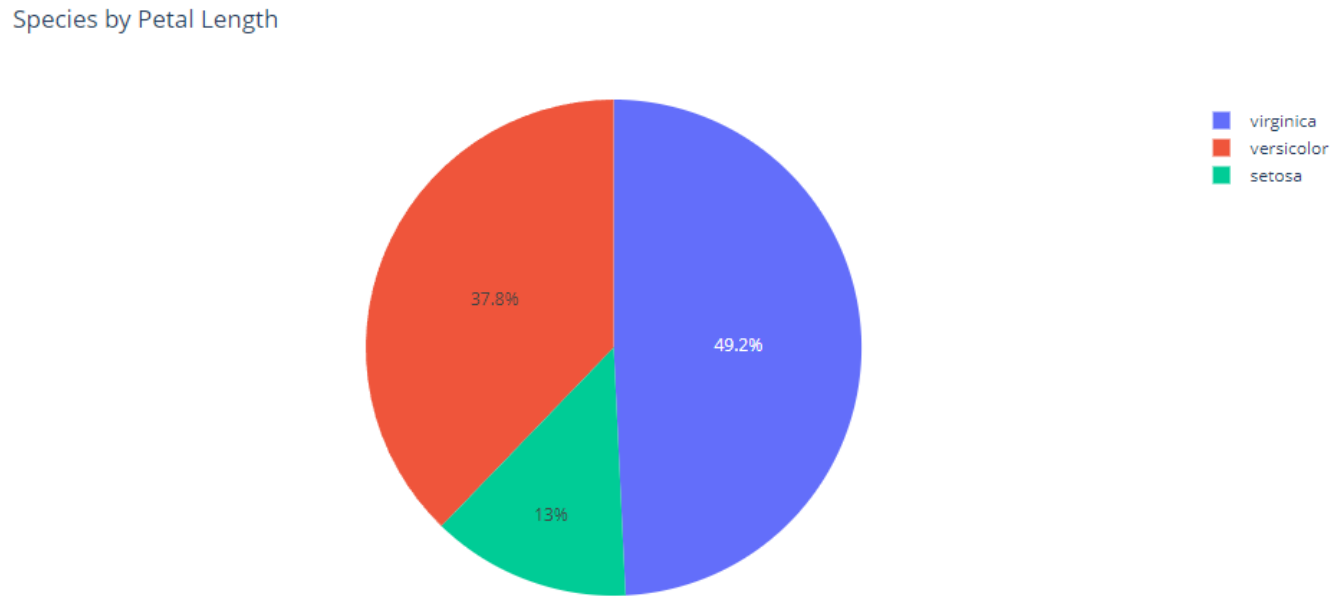



```
import plotly.graph_objects as go

fig = go.Figure(data=[go.Pie(labels=iris['species'], values=iris['petal_length'], pull=[0.1, 0.2, 0.5])])
fig.show()
```

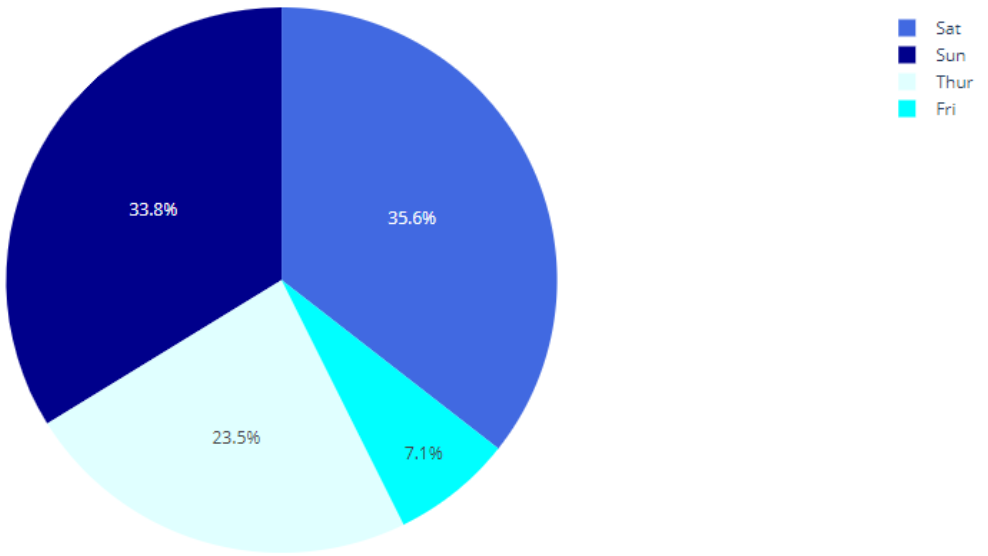


```python
a = iris.groupby("species")['petal_length'].sum().reset_index().T
print(a.to_string(header=False))

fig = px.pie(iris,
             values='petal_length',
             names='species',
             title='Species by Petal Length')
fig.show()
```

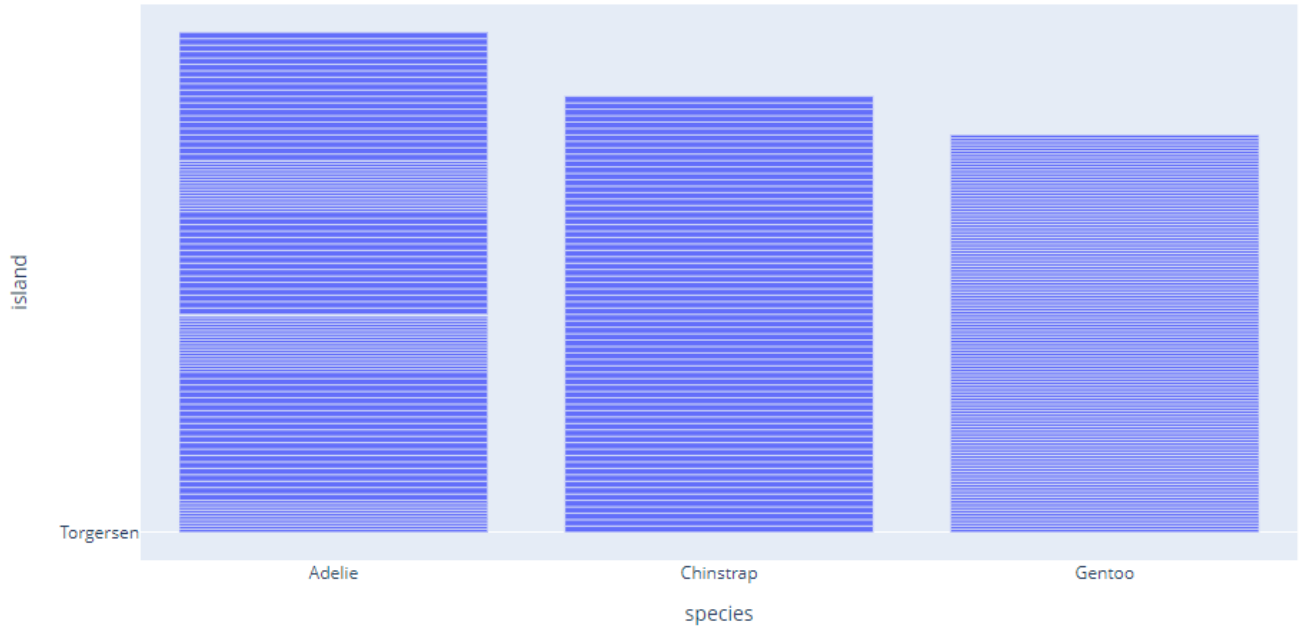
```
species       setosa  versicolor  virginica
petal_length    73.2       213.0      277.6
```




```python
fig = px.pie(tips,
             values='tip',
             names='day',
             color='day',
             color_discrete_map={'Thur':'lightcyan',
                                 'Fri':'cyan',
                                 'Sat':'royalblue',
                                 'Sun':'darkblue'})
fig.show()
```

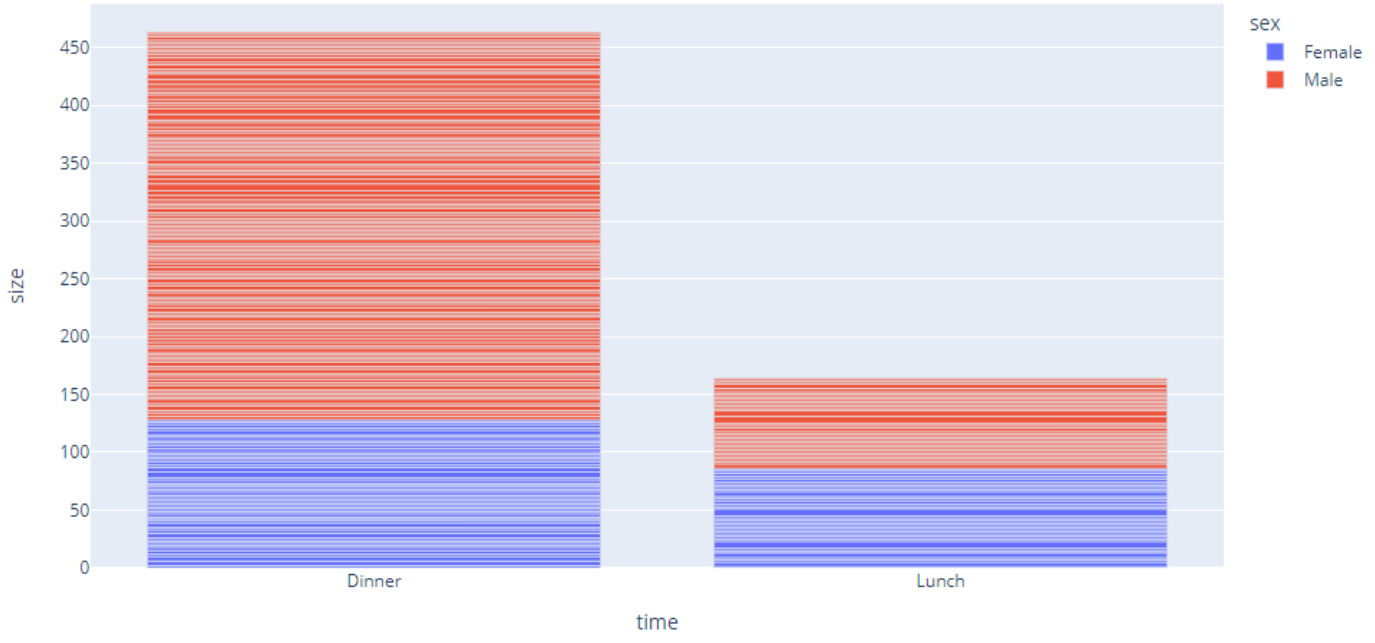


```
fig = px.bar(penguins, x='species', y='island')
fig.show()
```



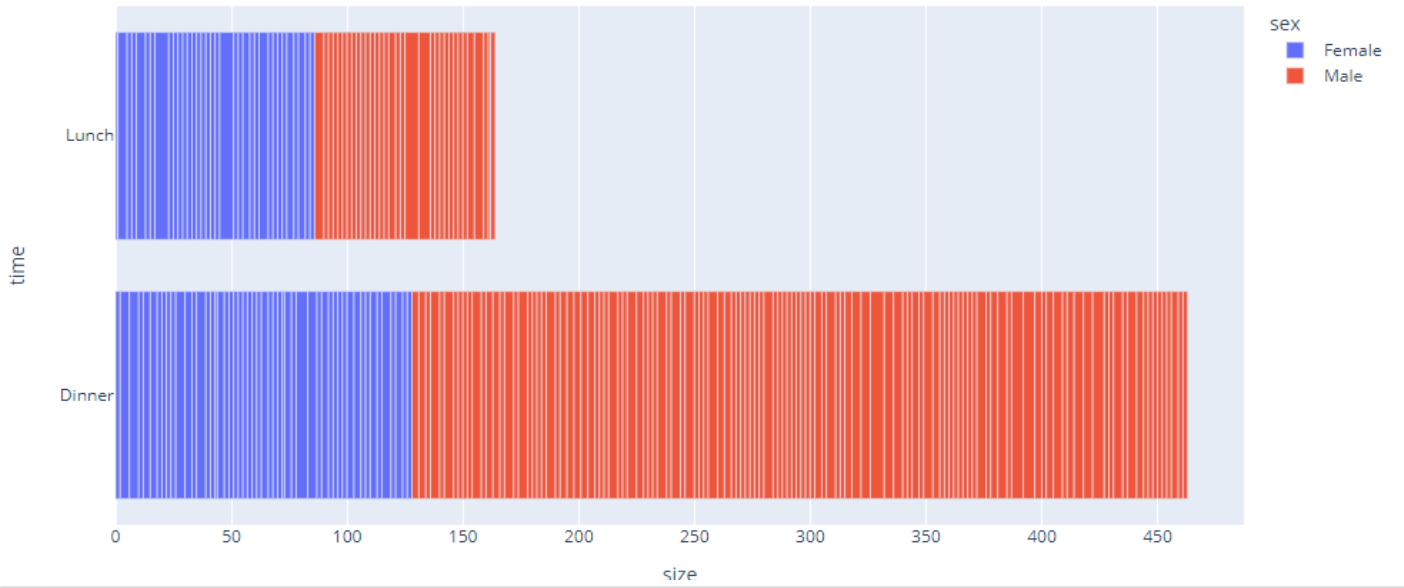

```
b = tips.groupby("time")['size'].sum().reset_index().T
print(b.to_string(header=False))

fig = px.bar(tips, x='time', y='size', color="sex")
fig.show()
```

```
time  Dinner  Lunch
size     463    164
```



```python
c = tips.groupby(["time", "sex"])['size'].sum().reset_index().T
print(c.to_string(header=False))

fig = px.bar(tips,
             y='time',
             x='size',
             color="sex",
             orientation='h')
fig.show()
```

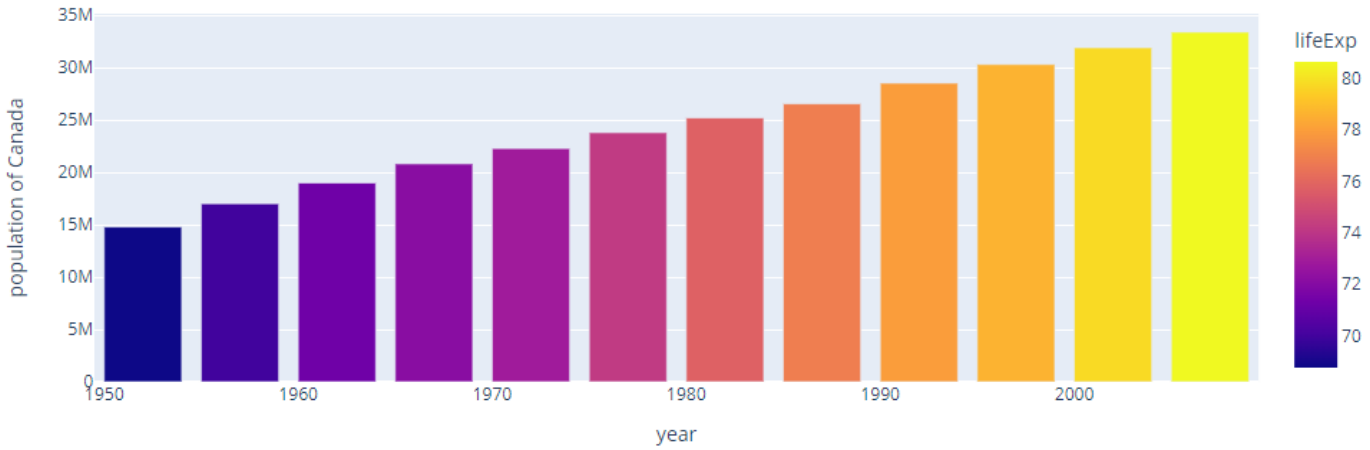
```
time  Dinner  Dinner   Lunch  Lunch
sex   Female    Male Female   Male
size     128     335      86     78
```



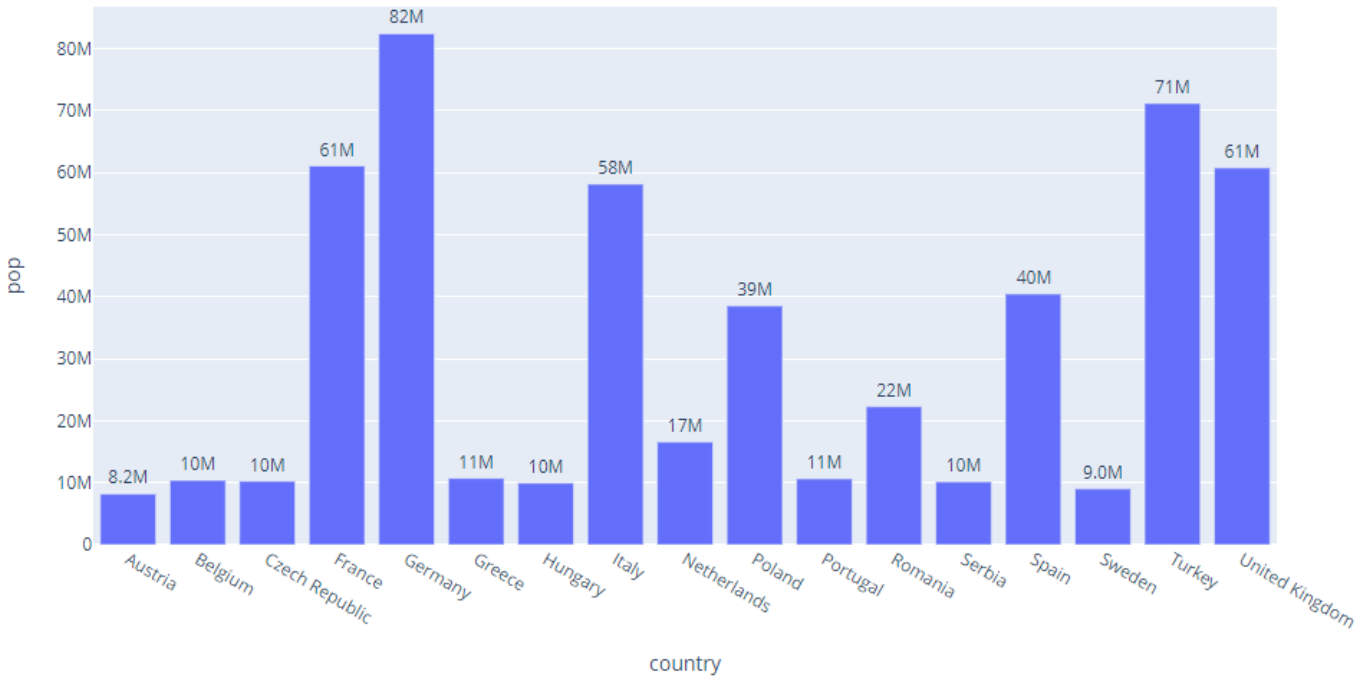

```python
d = tips.groupby(["time", "sex", "smoker"])['size'].sum().reset_index().T
print(d.to_string(header=False))

fig = px.bar(tips,
             x='time',
             y='size',
             color="sex",
             pattern_shape="smoker",
             pattern_shape_sequence=["x", "+"])
fig.show()
```

```
time    Dinner  Dinner  Dinner  Dinner   Lunch   Lunch  Lunch  Lunch
sex     Female  Female    Male    Male  Female  Female   Male   Male
smoker      No     Yes      No     Yes      No     Yes     No    Yes
size        77      51     213     122      63      23     50     28
```

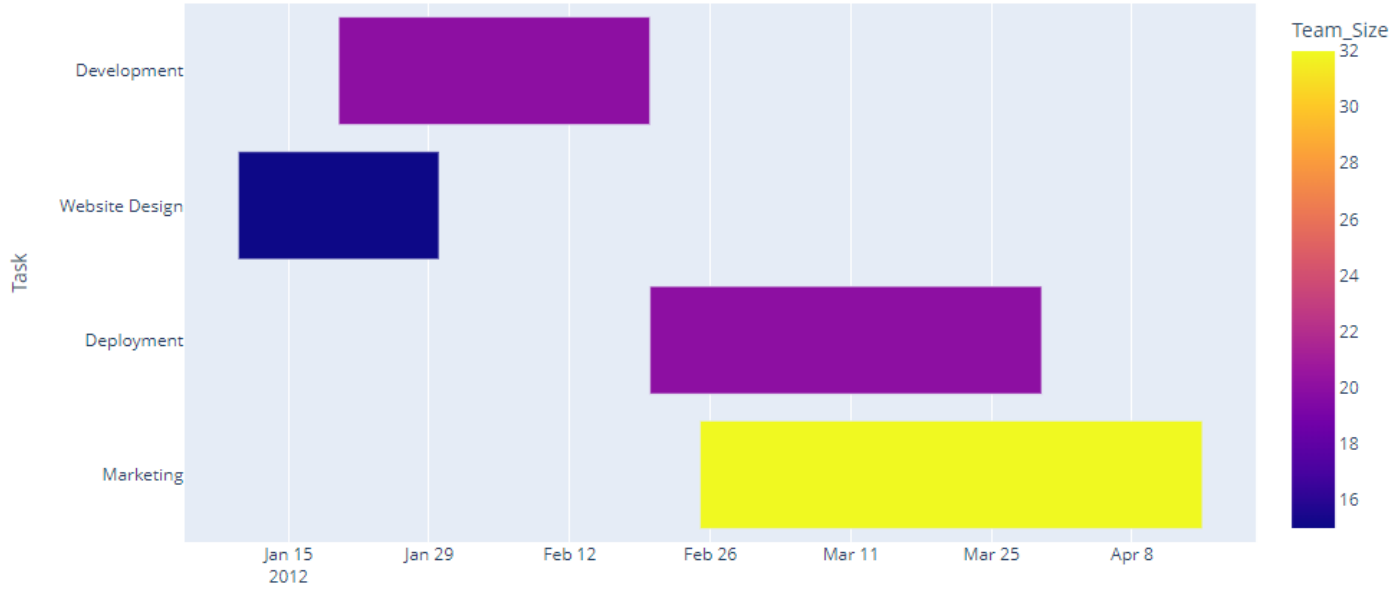
```
data = px.data.gapminder()
data_canada = data[data.country == 'Canada']

fig = px.bar(data_canada,
             x='year',
             y='pop',
             hover_data=['lifeExp', 'gdpPercap'],
             color='lifeExp',
             labels={'pop':'population of Canada'},
             height=400)
fig.show()
```



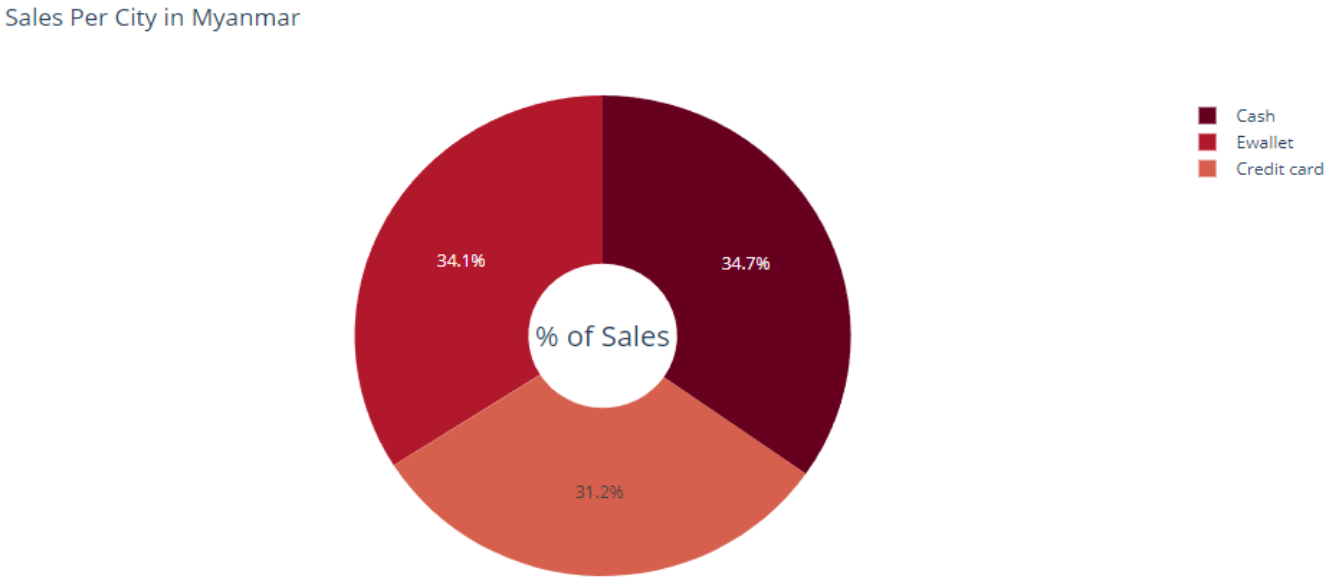

```
eu_pop2007 = px.data.gapminder().query("continent == 'Europe' and year == 2007 and pop > 8000000")

fig = px.bar(eu_pop2007, y='pop', x='country', text='pop')
fig.update_traces(texttemplate='%{text:.2s}', textposition='outside')
fig.show()
```



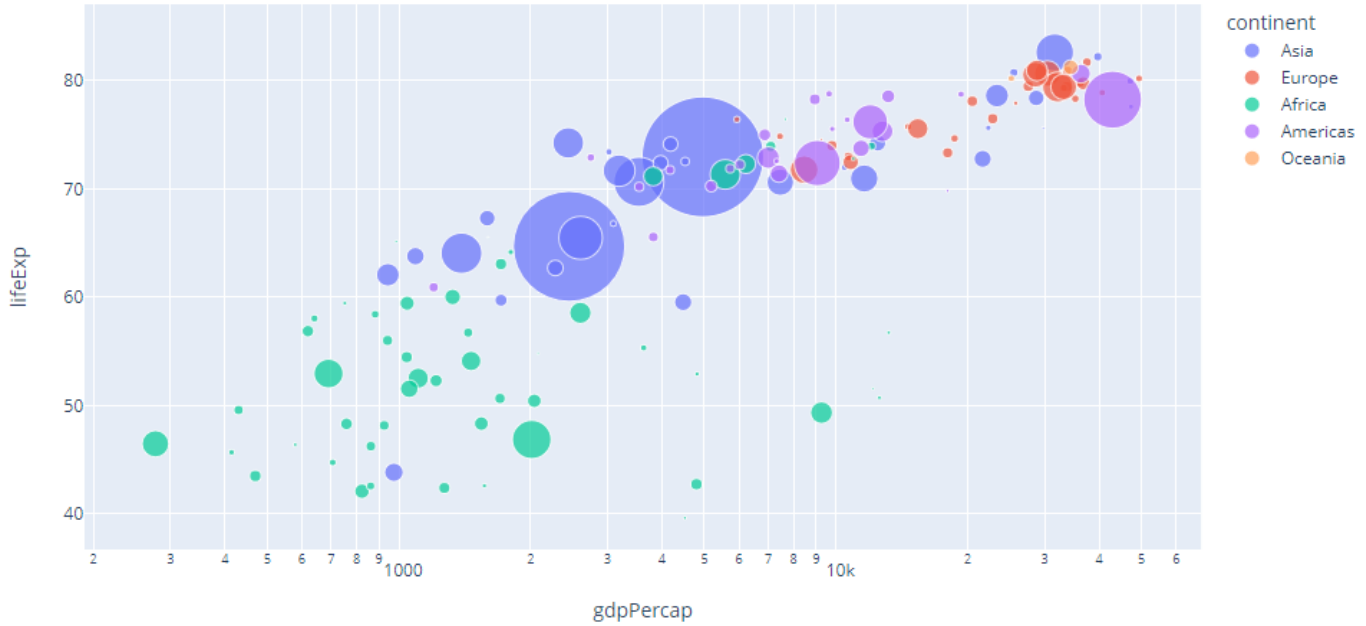
```python
df = pd.DataFrame([dict(Task="Development", Start='2012-01-20', Finish='2012-02-20', Team="Team A",Team_Size=20),
                   dict(Task="Website Design", Start='2012-01-10', Finish='2012-01-30', Team="Team B",Team_Size=15),
                   dict(Task="Deployment", Start='2012-02-20', Finish='2012-03-30', Team="Team A",Team_Size=20),
                   dict(Task="Marketing", Start='2012-02-25', Finish='2012-04-15', Team="Team C",Team_Size=32)])

fig = px.timeline(df, x_start="Start", x_end="Finish", y="Task",color="Team_Size")
fig.update_yaxes(autorange="reversed")
fig.show()
```



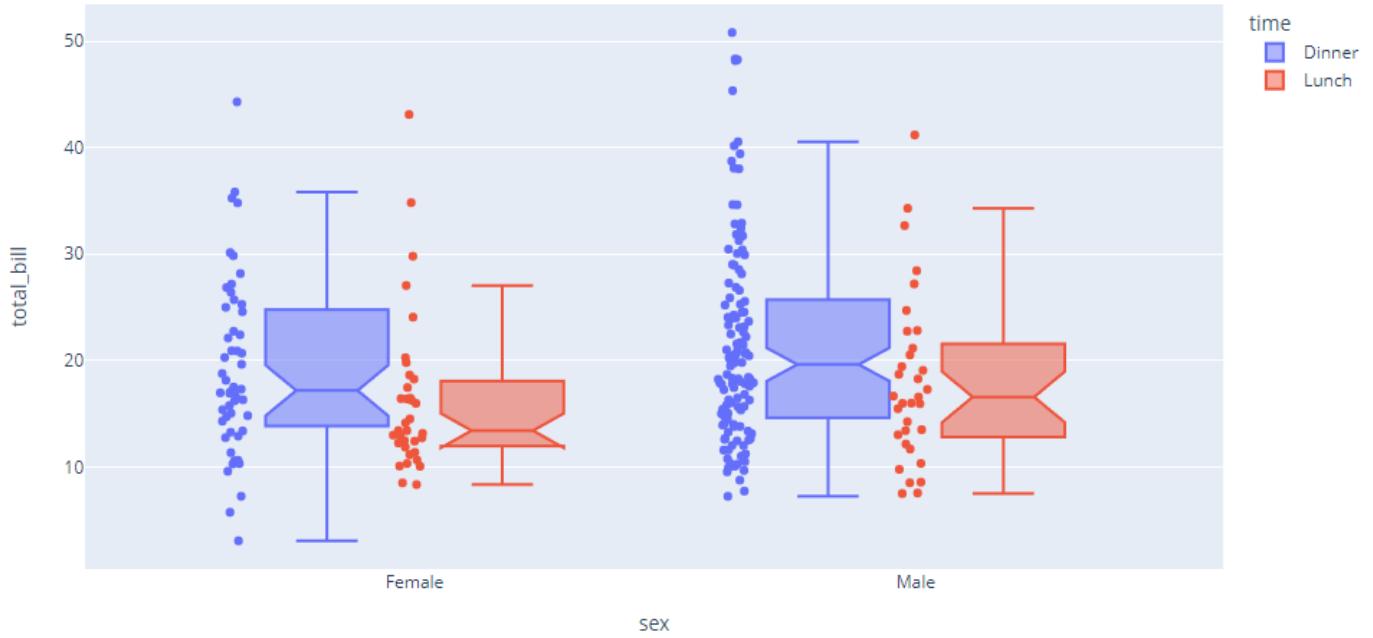

```python
fig = px.pie(sales,
             values='Total',
             names='Payment',
             color_discrete_sequence=px.colors.sequential.RdBu,
             hole=.3,
             title='Sales Per City in Myanmar')

fig.update_layout(
                # Add annotations in the center of the donut pies.
                annotations=[dict(text='% of Sales', x=0.5, y=0.5, font_size=20, showarrow=False)])

fig.show()

# fig = px.bar(sales, x="City", y="Total", color='Gender')
# fig.show()
```



```python
df = px.data.gapminder()

fig = px.scatter(df.query("year==2007"),
                 x="gdpPercap",
                 y="lifeExp",
                 size="pop",
                 color="continent",
                 hover_name="country",
                 log_x=True,
                 size_max=60)
fig.show()
```




```python
fig = px.box(tips,
             x='sex',
             y="total_bill",
             color="time",
             notched=True,
             points="all")
fig.show()
```



```python
fig = px.histogram(tips,
                   x="total_bill",
                   y="tip",
                   color="sex",
                   marginal="box",
                   hover_data=tips.columns)
fig.show()
```