DataViz2

September 23, 2024

@author : Samuel BIENVENU

@email : samuel.bienvenu@protonmail.com

Python provides a variety of libraries that offer different features for data visualization. All of these libraries offer different functionality and can support different types of charts.

In this tutorial, we will discuss four of these libraries. - Seaborn - Bokeh - Plotly - Altair

We will discuss these libraries one by one and will plot some most commonly used graphs.

[1]:
```
pip install streamlit
```

```
Requirement already satisfied: streamlit in
c:\users\welco\anaconda3\envs\data_viz\lib\site-packages (1.38.0)
Requirement already satisfied: altair<6,>=4.0 in
c:\users\welco\anaconda3\envs\data_viz\lib\site-packages (from streamlit)
(5.4.1)
Requirement already satisfied: blinker<2,>=1.0.0 in
c:\users\welco\anaconda3\envs\data_viz\lib\site-packages (from streamlit)
(1.8.2)
Requirement already satisfied: cachetools<6,>=4.0 in
c:\users\welco\anaconda3\envs\data_viz\lib\site-packages (from streamlit)
(5.5.0)
Requirement already satisfied: click<9,>=7.0 in
c:\users\welco\anaconda3\envs\data_viz\lib\site-packages (from streamlit)
(8.1.7)
Requirement already satisfied: numpy<3,>=1.20 in
c:\users\welco\anaconda3\envs\data_viz\lib\site-packages (from streamlit)
(2.1.1)
Requirement already satisfied: packaging<25,>=20 in
c:\users\welco\anaconda3\envs\data_viz\lib\site-packages (from streamlit) (24.1)
Requirement already satisfied: pandas<3,>=1.3.0 in
c:\users\welco\anaconda3\envs\data_viz\lib\site-packages (from streamlit)
(2.2.2)
Requirement already satisfied: pillow<11,>=7.1.0 in
c:\users\welco\anaconda3\envs\data_viz\lib\site-packages (from streamlit)
(10.4.0)
Requirement already satisfied: protobuf<6,>=3.20 in
c:\users\welco\anaconda3\envs\data_viz\lib\site-packages (from streamlit)
(5.28.1)
```

```
Requirement already satisfied: pyarrow>=7.0 in
c:\users\welco\anaconda3\envs\data_viz\lib\site-packages (from streamlit)
(17.0.0)
Requirement already satisfied: requests<3,>=2.27 in
c:\users\welco\anaconda3\envs\data_viz\lib\site-packages (from streamlit)
(2.32.3)
Requirement already satisfied: rich<14,>=10.14.0 in
c:\users\welco\anaconda3\envs\data_viz\lib\site-packages (from streamlit)
(13.8.1)
Requirement already satisfied: tenacity<9,>=8.1.0 in
c:\users\welco\anaconda3\envs\data_viz\lib\site-packages (from streamlit)
(8.5.0)
Requirement already satisfied: toml<2,>=0.10.1 in
c:\users\welco\anaconda3\envs\data_viz\lib\site-packages (from streamlit)
(0.10.2)
Requirement already satisfied: typing-extensions<5,>=4.3.0 in
c:\users\welco\anaconda3\envs\data_viz\lib\site-packages (from streamlit)
(4.12.2)
Requirement already satisfied: gitpython!=3.1.19,<4,>=3.0.7 in
c:\users\welco\anaconda3\envs\data_viz\lib\site-packages (from streamlit)
(3.1.43)
Requirement already satisfied: pydeck<1,>=0.8.0b4 in
c:\users\welco\anaconda3\envs\data_viz\lib\site-packages (from streamlit)
(0.9.1)
Requirement already satisfied: tornado<7,>=6.0.3 in
c:\users\welco\anaconda3\envs\data_viz\lib\site-packages (from streamlit)
(6.4.1)
Requirement already satisfied: watchdog<5,>=2.1.5 in
c:\users\welco\anaconda3\envs\data_viz\lib\site-packages (from streamlit)
(4.0.2)
Requirement already satisfied: jinja2 in
c:\users\welco\anaconda3\envs\data_viz\lib\site-packages (from
altair<6,>=4.0->streamlit) (3.1.4)
Requirement already satisfied: jsonschema>=3.0 in
c:\users\welco\anaconda3\envs\data_viz\lib\site-packages (from
altair<6,>=4.0->streamlit) (4.23.0)
Requirement already satisfied: narwhals>=1.5.2 in
c:\users\welco\anaconda3\envs\data_viz\lib\site-packages (from
altair<6,>=4.0->streamlit) (1.6.4)
Requirement already satisfied: colorama in
c:\users\welco\anaconda3\envs\data_viz\lib\site-packages (from
click<9,>=7.0->streamlit) (0.4.6)
Requirement already satisfied: gitdb<5,>=4.0.1 in
c:\users\welco\anaconda3\envs\data_viz\lib\site-packages (from
gitpython!=3.1.19,<4,>=3.0.7->streamlit) (4.0.11)
Requirement already satisfied: python-dateutil>=2.8.2 in
c:\users\welco\anaconda3\envs\data_viz\lib\site-packages (from
pandas<3,>=1.3.0->streamlit) (2.9.0)
```

```
Requirement already satisfied: pytz>=2020.1 in
c:\users\welco\anaconda3\envs\data_viz\lib\site-packages (from
pandas<3,>=1.3.0->streamlit) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in
c:\users\welco\anaconda3\envs\data_viz\lib\site-packages (from
pandas<3,>=1.3.0->streamlit) (2024.1)
Requirement already satisfied: charset-normalizer<4,>=2 in
c:\users\welco\anaconda3\envs\data_viz\lib\site-packages (from
requests<3,>=2.27->streamlit) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in
c:\users\welco\anaconda3\envs\data_viz\lib\site-packages (from
requests<3,>=2.27->streamlit) (3.8)
Requirement already satisfied: urllib3<3,>=1.21.1 in
c:\users\welco\anaconda3\envs\data_viz\lib\site-packages (from
requests<3,>=2.27->streamlit) (2.2.2)
Requirement already satisfied: certifi>=2017.4.17 in
c:\users\welco\anaconda3\envs\data_viz\lib\site-packages (from
requests<3,>=2.27->streamlit) (2024.8.30)
Requirement already satisfied: markdown-it-py>=2.2.0 in
c:\users\welco\anaconda3\envs\data_viz\lib\site-packages (from
rich<14,>=10.14.0->streamlit) (3.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in
c:\users\welco\anaconda3\envs\data_viz\lib\site-packages (from
rich<14,>=10.14.0->streamlit) (2.18.0)
Requirement already satisfied: smmap<6,>=3.0.1 in
c:\users\welco\anaconda3\envs\data_viz\lib\site-packages (from
gitdb<5,>=4.0.1->gitpython!=3.1.19,<4,>=3.0.7->streamlit) (5.0.1)
Requirement already satisfied: MarkupSafe>=2.0 in
c:\users\welco\anaconda3\envs\data_viz\lib\site-packages (from
jinja2->altair<6,>=4.0->streamlit) (2.1.3)
Requirement already satisfied: attrs>=22.2.0 in
c:\users\welco\anaconda3\envs\data_viz\lib\site-packages (from
jsonschema>=3.0->altair<6,>=4.0->streamlit) (24.2.0)
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in
c:\users\welco\anaconda3\envs\data_viz\lib\site-packages (from
jsonschema>=3.0->altair<6,>=4.0->streamlit) (2023.12.1)
Requirement already satisfied: referencing>=0.28.4 in
c:\users\welco\anaconda3\envs\data_viz\lib\site-packages (from
jsonschema>=3.0->altair<6,>=4.0->streamlit) (0.35.1)
Requirement already satisfied: rpds-py>=0.7.1 in
c:\users\welco\anaconda3\envs\data_viz\lib\site-packages (from
jsonschema>=3.0->altair<6,>=4.0->streamlit) (0.10.6)
Requirement already satisfied: mdurl~=0.1 in
c:\users\welco\anaconda3\envs\data_viz\lib\site-packages (from markdown-it-
py>=2.2.0->rich<14,>=10.14.0->streamlit) (0.1.2)
Requirement already satisfied: six>=1.5 in
c:\users\welco\anaconda3\envs\data_viz\lib\site-packages (from python-
dateutil>=2.8.2->pandas<3,>=1.3.0->streamlit) (1.16.0)
```

Note: you may need to restart the kernel to use updated packages.

[2]:
```
# Importing libaries and dataset

import pandas as pd # reading the database
data = pd.read_csv("tips.csv",delimiter =',')
```

[3]:
```
data.head()# printing the top 5 rows and bottom 5 rows
```

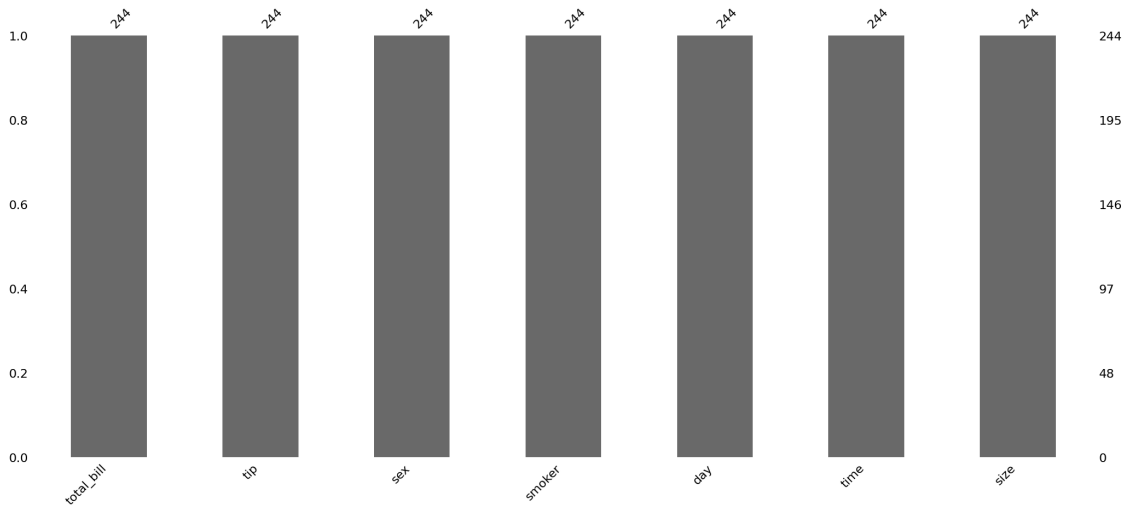
[3]:

	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

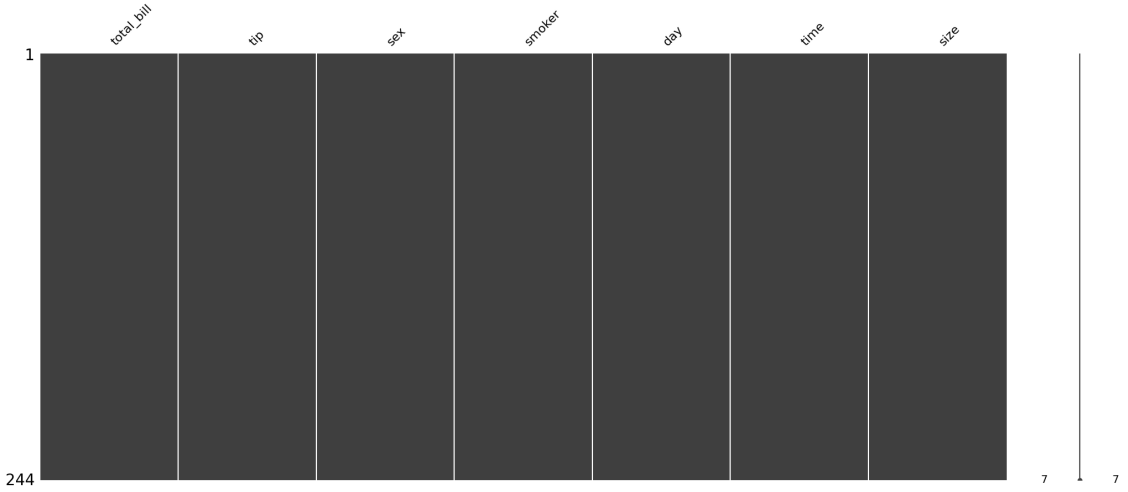
[4]:
```
# Checking missing information in the file

!pip install missingno
```

```
Unable to create process using 'C:\Users\Welco\anaconda3\python.exe
C:\Users\Welco\anaconda3\Scripts\pip-script.py install missingno'
```

[5]:
```
import missingno as msno
import matplotlib.pyplot as plt

# Visualize missing data as a bar chart
msno.bar(data)
plt.show()
```



[6]:
```
# Visualize missing data as a matrix
msno.matrix(data)
plt.show()
```



[7]:
```
# Visualize missing data as a heatmap
msno.heatmap(data)
plt.show()
```

```
c:\Users\Welco\anaconda3\envs\Data_Viz\Lib\site-packages\seaborn\matrix.py:309:
UserWarning: Attempting to set identical low and high xlims makes transformation
singular; automatically expanding.
  ax.set(xlim=(0, self.data.shape[1]), ylim=(0, self.data.shape[0]))
c:\Users\Welco\anaconda3\envs\Data_Viz\Lib\site-packages\seaborn\matrix.py:309:
UserWarning: Attempting to set identical low and high ylims makes transformation
singular; automatically expanding.
  ax.set(xlim=(0, self.data.shape[1]), ylim=(0, self.data.shape[0]))
```

[8]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 244 entries, 0 to 243
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   total_bill  244 non-null    float64
 1   tip         244 non-null    float64
 2   sex         244 non-null    object
 3   smoker      244 non-null    object
 4   day         244 non-null    object
 5   time        244 non-null    object
 6   size        244 non-null    int64
dtypes: float64(2), int64(1), object(4)
memory usage: 13.5+ KB
```

[9]:
```
data.describe()
```

[9]:

	total_bill	tip	size
count	244.000000	244.000000	244.000000
mean	19.785943	2.998279	2.569672
std	8.902412	1.383638	0.951100
min	3.070000	1.000000	1.000000

```
25%      13.347500    2.000000    2.000000
50%      17.795000    2.900000    2.000000
75%      24.127500    3.562500    3.000000
max      50.810000   10.000000    6.000000
```

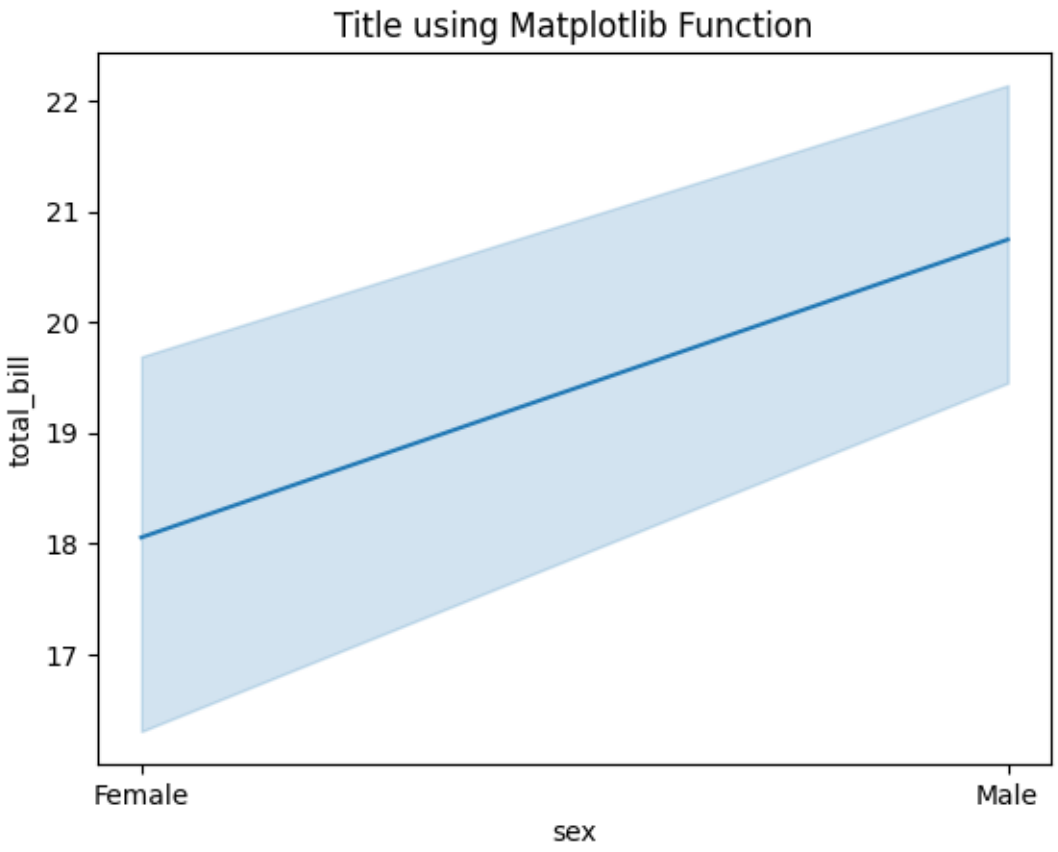
Seaborn

Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.

[10]:
```
!pip install seaborn
```

```
Unable to create process using 'C:\Users\Welco\anaconda3\python.exe
C:\Users\Welco\anaconda3\Scripts\pip-script.py install seaborn'
```

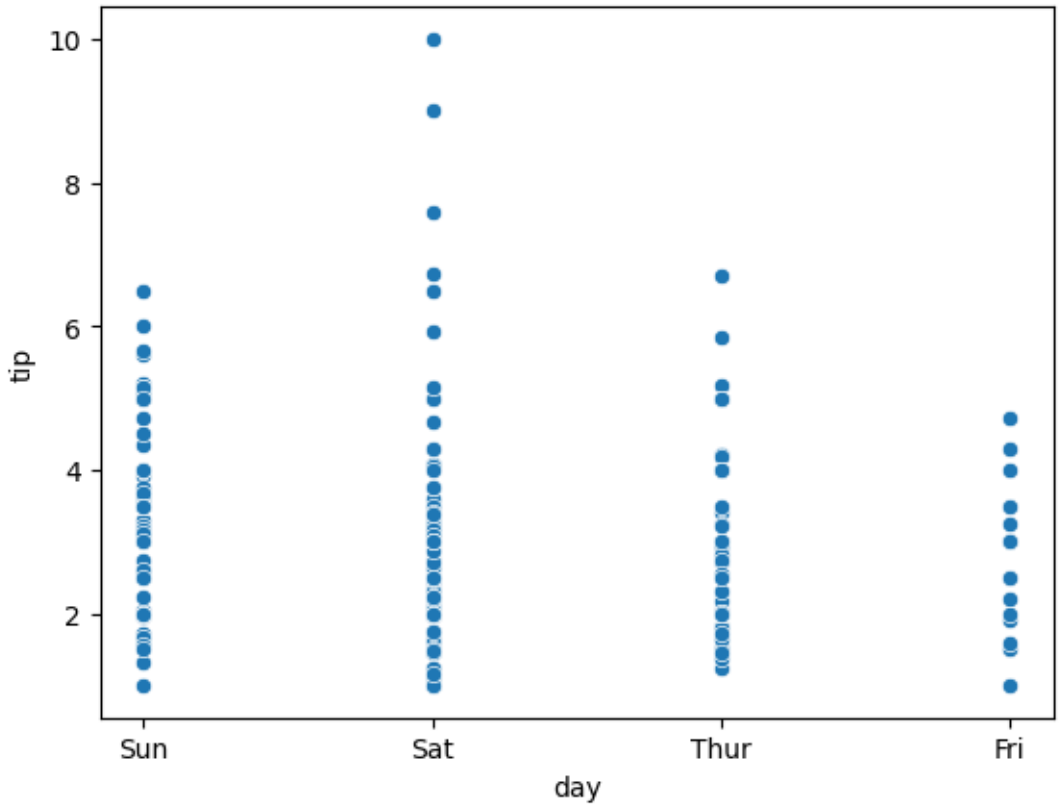
[11]:
```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

# draw lineplot
sns.lineplot(x="sex", y="total_bill", data=data)
# setting the title using Matplotlib
plt.title('Title using Matplotlib Function')
plt.show()
```



Scatter Plot

Scatter plot is plotted using the **scatterplot()** method. This is similar to Matplotlib, but additional argument data is required.

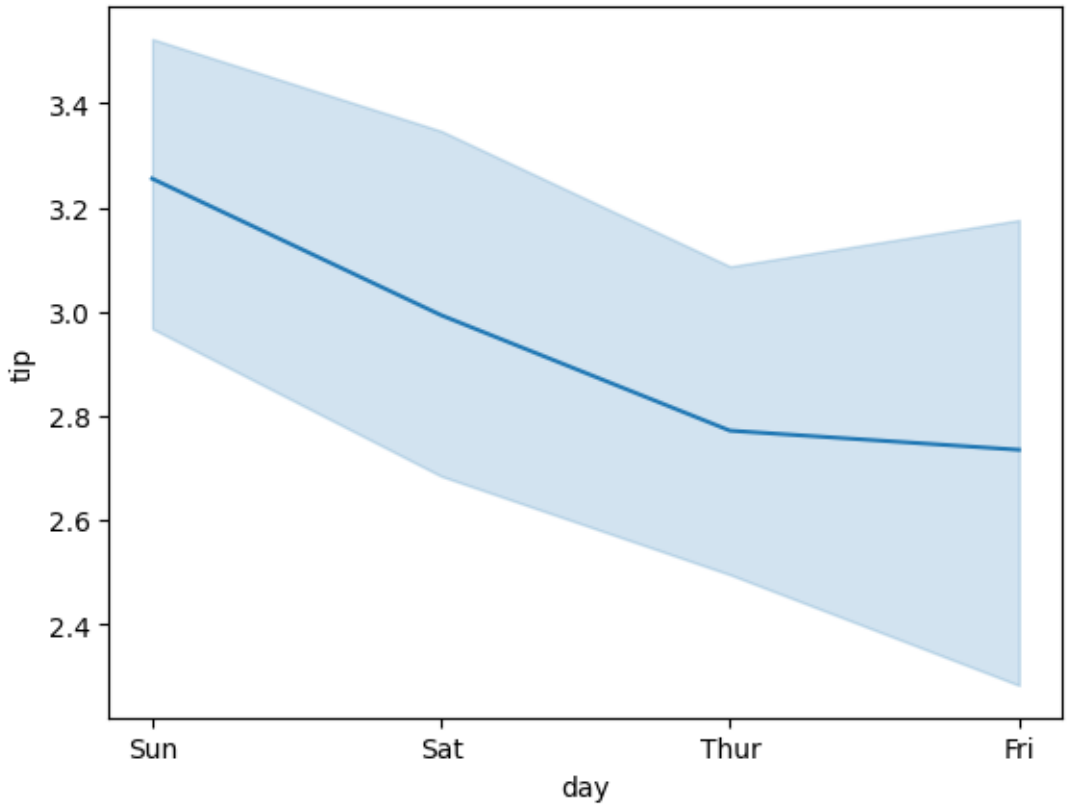
[12]:
```
sns.scatterplot(x='day', y='tip', data=data,)
plt.show()
```



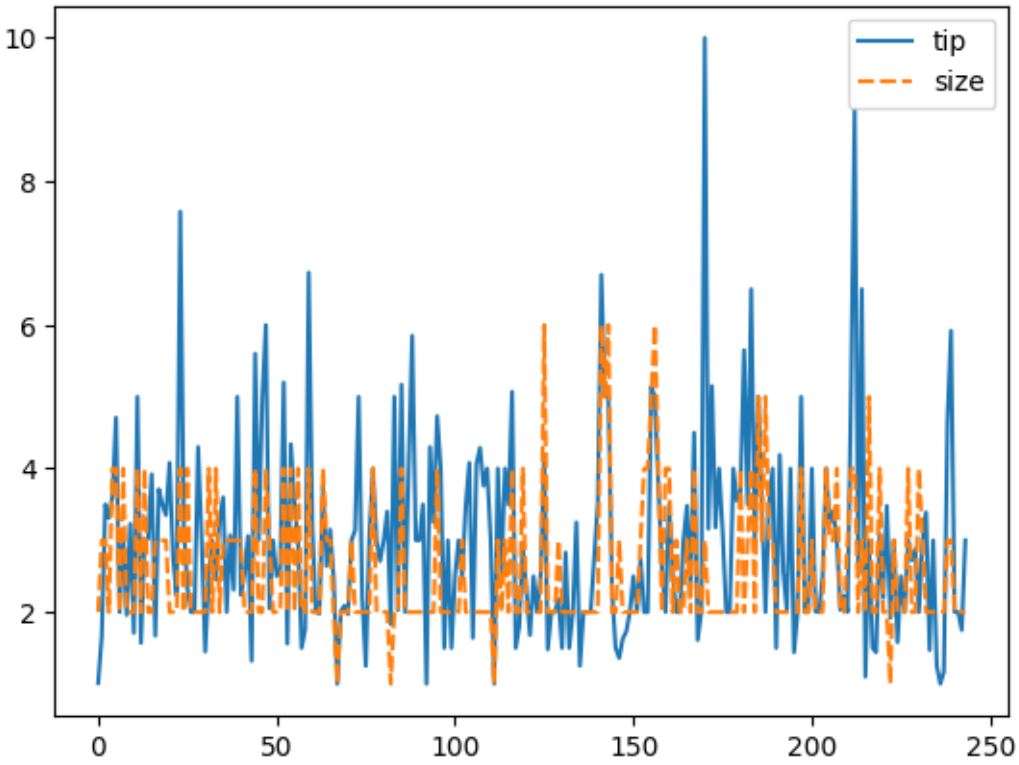
Line Plot

Line Plot in Seaborn plotted using the **lineplot()** method. In this, we can pass only the data argument also.

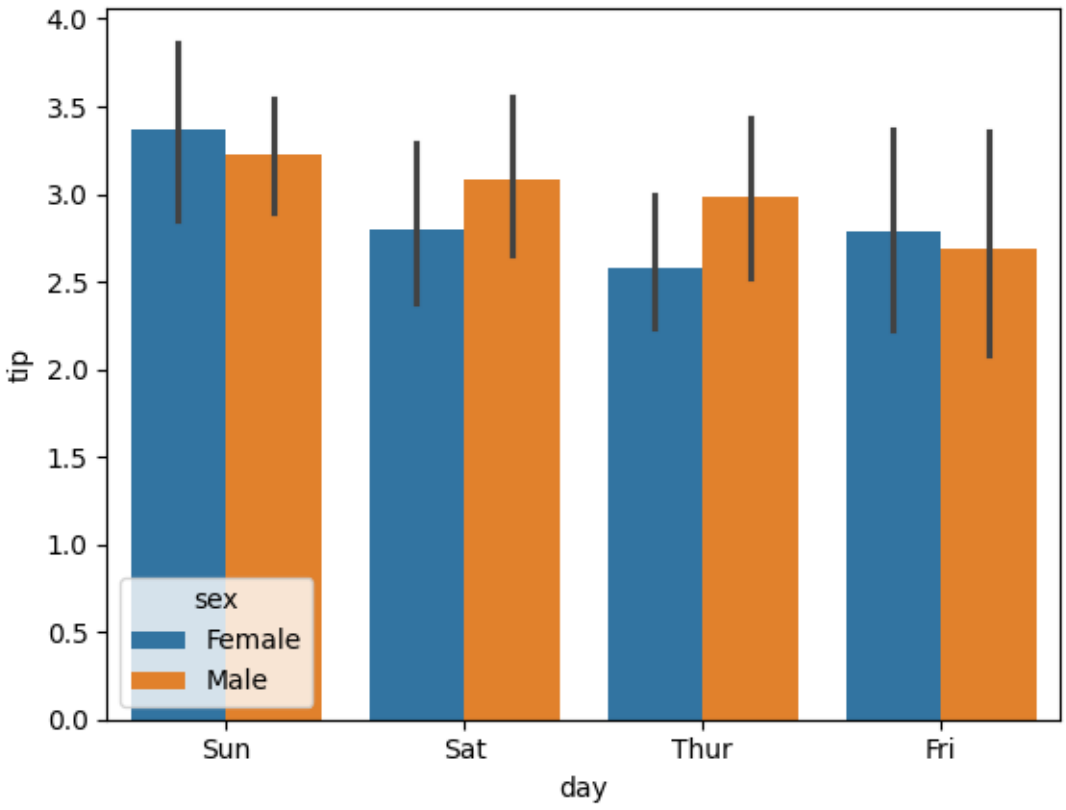
[13]:
```
sns.lineplot(x='day', y='tip', data=data)
plt.show()
```



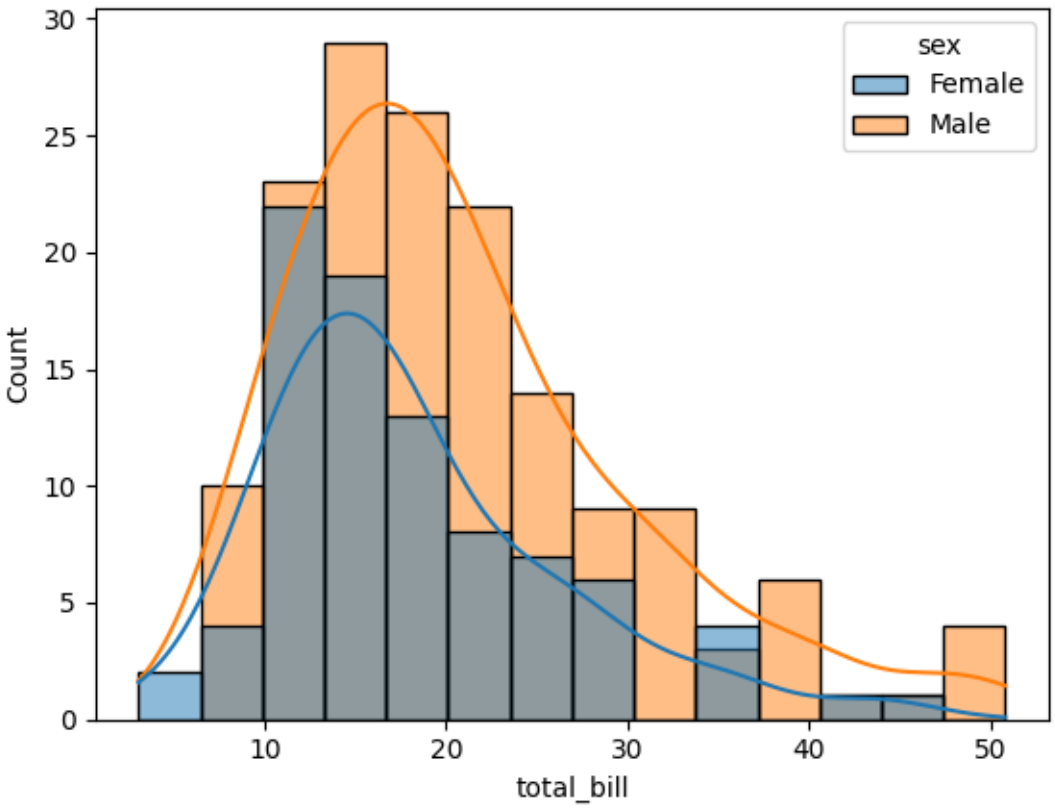
[14]:
```
# using only data attribute
sns.lineplot(data=data.drop(['total_bill'], axis=1))
plt.show()
```



[15]:
```
sns.barplot(x='day',y='tip', data=data, hue='sex')
plt.show()
```



[16]:
```
sns.histplot(x='total_bill', data=data, kde=True, hue='sex')
plt.show()
```



Bokeh

Let's move on to the third library of our list. Bokeh is mainly famous for its interactive charts visualization. Bokeh renders its plots using HTML and JavaScript that uses modern web browsers for presenting elegant, concise construction of novel graphics with high-level interactivity.

[17]:
```
!pip install bokeh
```

```
Unable to create process using 'C:\Users\Welco\anaconda3\python.exe
C:\Users\Welco\anaconda3\Scripts\pip-script.py install bokeh'
```

[18]:
```
# importing the modules
import bokeh
from bokeh.plotting import figure, output_file, show
from bokeh.palettes import magma
import pandas as pd

# instantiating the figure object
graph = figure(title = "Bokeh Scatter Graph")

# reading the database
data = pd.read_csv("tips.csv",delimiter =',')
```

```
# Reusing the data dataframe for this visualization
color = magma(256)

# plotting the graph
graph.scatter(data['total_bill'], data['tip'], color=color)

# displaying the model
show(graph)
```

```
BokehUserWarning: ColumnDataSource's columns must be of the same length. Current
lengths: ('line_color', 256), ('x', 244), ('y', 244)
BokehUserWarning: ColumnDataSource's columns must be of the same length. Current
lengths: ('hatch_color', 256), ('line_color', 256), ('x', 244), ('y', 244)
BokehUserWarning: ColumnDataSource's columns must be of the same length. Current
lengths: ('fill_color', 256), ('hatch_color', 256), ('line_color', 256), ('x',
244), ('y', 244)
```

Line Chart

A line plot can be created using the **line()** method of the plotting module.

[19]:
```
# importing the modules
from bokeh.plotting import figure, output_file, show
import pandas as pd

# instantiating the figure object
graph = figure(title = "Bokeh Bar Chart")

# reading the database
data = pd.read_csv("tips.csv") # Reusing the data dataframe for this␣
 ↪visualization

# Count of each unique value of # tip column
df = data['tip'].value_counts()

# plotting the graph
graph.line(df, data['tip'])

# displaying the model
show(graph)
```

```
BokehUserWarning: ColumnDataSource's columns must be of the same length. Current
lengths: ('x', 123), ('y', 244)
```

Bar Chart

Bar Chart can be of two types horizontal bars and vertical bars. Each can be created using the **hbar()** and **vbar()** functions of the plotting interface respectively.

[20]:
```
from bokeh.plotting import figure, output_file, show
import pandas as pd

# instantiating the figure object
graph = figure(title = "Bokeh Bar Chart")

# reading the database
data = pd.read_csv("tips.csv")

# plotting the graph
graph.vbar(data['total_bill'], top=data['tip'],
           legend_label = "Bill VS Tips", color='green')
graph.vbar(data['tip'], top=data['size'],
           legend_label = "Tips VS Size", color='red')
graph.legend.click_policy = "hide"

# displaying the model
show(graph)
```

Interactive Data Visualization

One of the key features of Bokeh is to add interaction to the plots. Let's see various interactions that can be added.

Interactive Legends

click_policy property makes the legend interactive. There are two types of interactivity – - Hiding: Hides the Glyphs. - Muting: Hiding the glyph makes it vanish completely, on the other hand, muting the glyph just de-emphasizes the glyph based on the parameters.

[21]:
```
# importing the modules
from bokeh.plotting import figure, output_file, show
import pandas as pd

# instantiating the figure object
graph = figure(title = "Bokeh Bar Chart")

# reading the database
data = pd.read_csv("tips.csv")

# plotting the graph
graph.vbar(data['total_bill'], top=data['tip'])

# displaying the model
show(graph)
```

[22]:
```
from bokeh.models import HoverTool

graph = figure(title = "Bokeh Bar Chart")
```

```
graph.vbar(data["total_bill"],
            top = data['tip'],
            legend_label = "Bill VS Tips",
            color = "green")

graph.vbar(data["tip"],
            top = data['size'],
            legend_label = "Tips VS Size",
            color = "red")

graph.legend.click_policy = "hide"
graph.add_tools(HoverTool())
show(graph)
```

[23]:
```
# Add tooltip (name, field) pairs to the tool. See below for a # description of␣
 ↪possible field values.
from bokeh.models import HoverTool

hover = HoverTool(tooltips=[("index", "$index"),
                            ("(x, y)", "($x, $y)"),
                            ("radius", "@radius"),
                            ("fill color", "@fill_color{hex, swatch}"),
                            ("foo", "@foo"),
                            ("bar", "@bar"),
                            ("baz", "@baz{safe}"),
                            ("total", "@total{$0,0.00}")
                            ])
```

Adding Widgets

Bokeh provides GUI features similar to HTML forms like buttons, sliders, checkboxes, etc. These provide an interactive interface to the plot that allows changing the parameters of the plot, modifying plot data, etc. Let's see how to use and add some commonly used widgets.

Buttons: This widget adds a simple button widget to the plot. We have to pass a custom JavaScript function to the CustomJS() method of the models class.

CheckboxGroup: Adds a standard check box to the plot. Similarly to buttons we have to pass the custom JavaScript function to the CustomJS() method of the models class.

RadioGroup: Adds a simple radio button and accepts a custom JavaScript function.

[24]:
```
from bokeh.io import show
from bokeh.models import Button, SetValue

button = Button(label="Foo",
                button_type="primary")

callback = SetValue(obj=button,
```

```
                    attr="label",
                    value="Bar")

button.js_on_event("button_click", callback)

show(button)
```

[25]:
```
#Use this for those whose SetValue did not work. It has been replaced by␣
 ↪CustomJS
from bokeh.io import show
from bokeh.models import Button, CustomJS

button = Button(label="Foo", button_type="primary")

# Define a CustomJS callback to change the button's label
callback = CustomJS(args=dict(button=button),
                    code="""  button.label = 'Bar'; """)

# Attach the callback to the button's'button_click' event
button.js_on_event('button_click', callback)
show(button)
```

[26]:
```
from bokeh.io import show
from bokeh.models import Button, CustomJS

button = Button(label="Foo", button_type="primary")
callback = CustomJS(code="""  // Get the button element by its id var button =␣
 ↪document.getElementById('%s');
                              // Change the label when the button is clicked␣
 ↪button.textContent = "Bar"; """ % button.id)

button.js_on_event("button_click", callback)
show(button)
```

[27]:
```
from bokeh.io import show
from bokeh.models import Button, CheckboxGroup, RadioGroup, CustomJS

# Create a button widget
button = Button(label="GFG")

# Add JavaScript callback on button click
button.js_on_click(CustomJS(code="console.log('button: click!', this.
 ↪toString())"))

# Labels for checkbox and radio buttons
L = ["First", "Second", "Third"]
```

```
# Create a CheckboxGroup widget
checkbox_group = CheckboxGroup(labels=L, active=[0, 2])

# Add JavaScript callback on CheckboxGroup change
checkbox_group.js_on_change('active', CustomJS(code="""
 ↪log('checkbox_group: active=' + this.active, this.toString()) """))

# Create a RadioGroup widget
radio_group = RadioGroup(labels=L, active=1)

# Add JavaScript callback on RadioGroup change
radio_group.js_on_change('active', CustomJS(code="""
 ↪active=' + this.active, this.toString()) """))

# Display the widgets
show(button)
show(checkbox_group)
show(radio_group)
```

[28]:
```
from bokeh.io import show
from bokeh.models import CustomJS, Slider

slider = Slider(start=1, end=20, value=1, step=2, title="Slider")
slider.js_on_change("value", CustomJS(code="""
 ↪this.value, this.toString()) """))

show(slider)
```

Plotly

This is the last library of our list and you might be wondering why plotly. Here's why – - Plotly has hover tool capabilities that allow us to detect any outliers or anomalies in numerous data points. - It allows more customization. - It makes the graph visually more attractive.

[29]:
```
!pip install plotly
```

```
Requirement already satisfied: plotly in
c:\users\welco\anaconda3\envs\data_viz\lib\site-packages (5.24.1)
Requirement already satisfied: tenacity>=6.2.0 in
c:\users\welco\anaconda3\envs\data_viz\lib\site-packages (from plotly) (8.5.0)
Requirement already satisfied: packaging in
c:\users\welco\anaconda3\envs\data_viz\lib\site-packages (from plotly) (24.1)
```

Scatter Plot

Scatter plot in Plotly can be created using the **scatter()** method of **plotly.express**. Like Seaborn, an extra data argument is also required here.

[30]:
```
import plotly.express as px
import pandas as pd

# reading the database
data = pd.read_csv("tips.csv")

# plotting the scatter chart
fig = px.scatter(data, x="day", y="tip", color='sex')

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

Line Chart

Line plot in Plotly is much accessible and illustrious annexation to plotly which manage a variety of types of data and assemble easy-to-style statistic. With **px.line** each data position is represented as a vertex

[31]:
```
import plotly.express as px
import pandas as pd

# reading the database
data = pd.read_csv("tips.csv")

# plotting the scatter chart
fig = px.line(data, y='tip', color='sex')

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

Bar Chart

Bar Chart in Plotly can be created using the **bar()** method of **plotly.express** class.

[32]:
```
# plotting the scatter chart
fig = px.bar(data, x='day', y='tip', color='sex')

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

Histogram

In plotly, histograms can be created using the **histogram()** function of the **plotly.express** class.

[33]:
```
# plotting the histogram chart
fig = px.histogram(data, x='total_bill', color='sex')

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

Adding interaction

Just like Bokeh, plotly also provides various interactions. Let's discuss a few of them. **Creating Dropdown Menu:** A drop-down menu is a part of the menu-button which is displayed on a screen all the time. Every menu button is associated with a Menu widget that can display the choices for that menu button when clicked on it. In plotly, there are 4 possible methods to modify the charts by using updatemenu method.

- **restyle:** modify data or data attributes
- **relayout:** modify layout attributes
- **update:** modify data and layout attributes
- **animate:** start or pause an animation

[34]:
```
import plotly.graph_objects as px
import pandas as pd

# reading the database
data = pd.read_csv("tips.csv")
plot = px.Figure(data=[px.Scatter( x=data['day'], y=data['tip'],␣
 ↪mode='markers',) ])

# Add dropdown
plot.update_layout( updatemenus=[ dict( buttons=list([ dict( args=["type",␣
 ↪"scatter"], label="Scatter Plot", method="restyle" ), dict( args=["type",␣
 ↪"bar"], label="Bar Chart", method="restyle" ) ]), direction="down", ), ] )

plot.show()
```

[35]:
```
import plotly.graph_objects as px
import pandas as pd

# reading the database
data = pd.read_csv("tips.csv")
plot = px.Figure(data=[px.Scatter( x=data['day'], y=data['tip'],␣
 ↪mode='markers',) ])

# Add dropdown
plot.update_layout( updatemenus=[ dict( type="buttons", direction="left",␣
 ↪buttons=list([ dict( args=["type", "scatter"], label="Scatter Plot",␣
 ↪method="restyle" ), dict( args=["type", "bar"], label="Bar Chart",␣
 ↪method="restyle" ) ]), ), ] )

plot.show()
```

Creating Sliders and Selectors:

In plotly, the range slider is a custom range-type input control. It allows selecting a value or a range of values between a specified minimum and maximum range. And the range selector is a tool for selecting ranges to display within the chart. It provides buttons to select pre-configured ranges in the chart. It also provides input boxes where the minimum and maximum dates can be manually input

[36]:
```
import plotly.graph_objects as px
import pandas as pd

# reading the database
data = pd.read_csv("tips.csv")
plot = px.Figure(data=[px.Scatter( y=data['tip'], mode='lines',) ])

plot.update_layout( xaxis=dict( rangeselector=dict( buttons=list([␣
 ↪dict(count=1, step="day", stepmode="backward"), ]) ), rangeslider=dict(␣
 ↪visible=True ), ) )

plot.show()
```

Vega-Altair

It is a declarative visualization library for Python. Its simple, friendly and consistent API, built on top of the powerful Vega-Lite grammar, empowers you to spend less time writing code and more time exploring your data.

[37]:
```
!pip install altair vega_datasets
#for importing vega datasets for downstream data analysis
```

```
Requirement already satisfied: altair in
c:\users\welco\anaconda3\envs\data_viz\lib\site-packages (5.4.1)
Requirement already satisfied: vega_datasets in
c:\users\welco\anaconda3\envs\data_viz\lib\site-packages (0.9.0)
Requirement already satisfied: jinja2 in
c:\users\welco\anaconda3\envs\data_viz\lib\site-packages (from altair) (3.1.4)
Requirement already satisfied: jsonschema>=3.0 in
c:\users\welco\anaconda3\envs\data_viz\lib\site-packages (from altair) (4.23.0)
Requirement already satisfied: narwhals>=1.5.2 in
c:\users\welco\anaconda3\envs\data_viz\lib\site-packages (from altair) (1.6.4)
Requirement already satisfied: packaging in
c:\users\welco\anaconda3\envs\data_viz\lib\site-packages (from altair) (24.1)
Requirement already satisfied: typing-extensions>=4.10.0 in
c:\users\welco\anaconda3\envs\data_viz\lib\site-packages (from altair) (4.12.2)
Requirement already satisfied: pandas in
c:\users\welco\anaconda3\envs\data_viz\lib\site-packages (from vega_datasets)
(2.2.2)
Requirement already satisfied: attrs>=22.2.0 in
c:\users\welco\anaconda3\envs\data_viz\lib\site-packages (from
jsonschema>=3.0->altair) (24.2.0)
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in
c:\users\welco\anaconda3\envs\data_viz\lib\site-packages (from
jsonschema>=3.0->altair) (2023.12.1)
Requirement already satisfied: referencing>=0.28.4 in
c:\users\welco\anaconda3\envs\data_viz\lib\site-packages (from
jsonschema>=3.0->altair) (0.35.1)
Requirement already satisfied: rpds-py>=0.7.1 in
```

```
c:\users\welco\anaconda3\envs\data_viz\lib\site-packages (from
jsonschema>=3.0->altair) (0.10.6)
Requirement already satisfied: MarkupSafe>=2.0 in
c:\users\welco\anaconda3\envs\data_viz\lib\site-packages (from jinja2->altair)
(2.1.3)
Requirement already satisfied: numpy>=1.26.0 in
c:\users\welco\anaconda3\envs\data_viz\lib\site-packages (from
pandas->vega_datasets) (2.1.1)
Requirement already satisfied: python-dateutil>=2.8.2 in
c:\users\welco\anaconda3\envs\data_viz\lib\site-packages (from
pandas->vega_datasets) (2.9.0)
Requirement already satisfied: pytz>=2020.1 in
c:\users\welco\anaconda3\envs\data_viz\lib\site-packages (from
pandas->vega_datasets) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in
c:\users\welco\anaconda3\envs\data_viz\lib\site-packages (from
pandas->vega_datasets) (2024.1)
Requirement already satisfied: six>=1.5 in
c:\users\welco\anaconda3\envs\data_viz\lib\site-packages (from python-
dateutil>=2.8.2->pandas->vega_datasets) (1.16.0)
```

[38]:
```
# Import the libraries

import altair as alt
import vega_datasets
```

[39]:
```
from vega_datasets import data
cars = data.cars()
cars
```

[39]:

	Name	Miles_per_Gallon	Cylinders	Displacement	Horsepower	Weight_in_lbs	Acceleration	Year	Origin
0	chevrolet chevelle malibu	18.0	8	307.0	130.0	3504	12.0	1970-01-01	USA
1	buick skylark 320	15.0	8	350.0	165.0	3693	11.5	1970-01-01	USA
2	plymouth satellite	18.0	8	318.0	150.0	3436	11.0	1970-01-01	USA
3	amc rebel sst	16.0	8	304.0					
4	ford torino	17.0	8	302.0					
..	…	…	…	…					
401	ford mustang gl	27.0	4	140.0					
402	vw pickup	44.0	4	97.0					
403	dodge rampage	32.0	4	135.0					
404	ford ranger	28.0	4	120.0					
405	chevy s-10	31.0	4	119.0					

```
3          150.0          3433          12.0 1970-01-01     USA
4          140.0          3449          10.5 1970-01-01     USA
..           …             …             …         …      …
401         86.0          2790          15.6 1982-01-01     USA
402         52.0          2130          24.6 1982-01-01  Europe
403         84.0          2295          11.6 1982-01-01     USA
404         79.0          2625          18.6 1982-01-01     USA
405         82.0          2720          19.4 1982-01-01     USA

[406 rows x 9 columns]
```

Variable parameters allow for a value to be defined once and then reused throughout the rest of the chart. Here is a simple scatter-plot created from the cars dataset:

[40]:
```
import altair as alt
from vega_datasets import data

cars = data.cars.url
alt.Chart(cars).mark_circle().encode( x='Horsepower:Q', y='Miles_per_Gallon:Q',␣
 ↪color='Origin:N' )
```

[40]:
```
alt.Chart(…)
```

Zero, One, and Two-dimensional Charts

Using Altair, we can being to explore this data. The most basic chart contains the dataset, along with a mark to represent each row: This is a pretty silly chart, because it consists of 406 points, all laid-out on top of each other. To make it more interesting, we need to encode columns of the data into visual features of the plot (e.g. x position, y position, size, color, etc.) Let's encode miles per gallon on the x-axis using the **encode()** method:

This is a bit better, but the **point** mark is probably not the best for a 1D chart like this. Let's try the **tick** mark instead:

[41]:
```
import altair as alt
import warnings
warnings.filterwarnings("ignore", category=FutureWarning, module="altair.utils.
 ↪core")
from vega_datasets import data

# Load the example 'cars' dataset
cars = data.cars()

# Define the 'brush' selection
brush = alt.selection_interval(encodings=['x'])

# Create the first chart
```

```
chart1 = alt.Chart(cars).mark_point().encode( y='Horsepower:Q', color=alt.
 ↪condition(brush, 'Origin:N', alt.value('lightgray')) ).properties(␣
 ↪width=250, height=250 ).add_params()

# Create the second chart
chart2 = alt.Chart(cars).mark_point().encode( y='Miles_per_Gallon:Q', color=alt.
 ↪condition(brush, 'Origin:N', alt.value('lightgray')) ).properties(␣
 ↪width=250, height=250 ).add_params()

# Arrange the charts side by side
(chart1.encode(x='Acceleration:Q') | chart2.encode(x='Miles_per_Gallon:Q'))
```

[41]: alt.HConcatChart(…)

Simple Interactions

One of the nicest features of Altair is the grammar of interaction that it provides. The simplest kind of interaction is the ability to pan and zoom along charts; Altair contains a shortcut to enable this via the **interactive()** method:

This lets you click and drag, as well as use your computer's scroll/zoom behavior to zoom in and out on the chart.

[42]:
```
alt.Chart(cars).mark_point().encode( x='Miles_per_Gallon', y='Horsepower' ).
 ↪interactive()
```

[42]: alt.Chart(…)

A Third Dimension: Color

A 2D plot allows us to encode two dimensions of the data. Let's look at using color to encode a third:

[43]:
```
alt.Chart(cars).mark_point().encode( x='Miles_per_Gallon', y='Horsepower',␣
 ↪color='Origin' ).interactive()
```

[43]: alt.Chart(…)

Notice that when we use a categorical value for color, it chooses an appropriate color map for categorical data. Let's see what happens when we use a continuous color value:

[44]:
```
alt.Chart(cars).mark_point().encode( x='Miles_per_Gallon', y='Horsepower',␣
 ↪color='Acceleration' )
```

[44]: alt.Chart(…)

A continuous color results in a color scale that is appropriate for continuous data.

What about the in-between case: ordered categories, like number of cylinders?

[45]:
```
alt.Chart(cars).mark_point().encode( x='Miles_per_Gallon', y='Horsepower',␣
↪color='Cylinders' )
```

[45]: alt.Chart(…)

Altair still chooses a continuous value because the number of Cylinders is numerical. We can improve this by specifying that the data should be treated as a discrete ordered value; we can do this by adding **":O"** ("O" for "ordinal" or "ordered categories") after the encoding:

[46]:
```
alt.Chart(cars).mark_point().encode( x='Miles_per_Gallon', y='Horsepower',␣
↪color='Cylinders:O' )
```

[46]: alt.Chart(…)

Now we get a discrete legend with an ordered color mapping.

Binning and aggregation

Let's return quickly to our 1D chart of miles per gallon:

[47]:
```
alt.Chart(cars).mark_tick().encode( x='Miles_per_Gallon', )
```

[47]: alt.Chart(…)

Another way we might represent this data is to creat a histogram: to bin the x data and show the count on the y axis. In many plotting libraries this is done with a special method like **hist()**. In Altair, such binning and aggregation is part of the declarative API.

To move beyond a simple field name, we use **alt.X()** for the x encoding, and we use **'count()'** for the y encoding:

[48]:
```
alt.Chart(cars).mark_bar().encode( x=alt.X('Miles_per_Gallon', bin=True),␣
↪y='count()' )
```

[48]: alt.Chart(…)

If we want more control over the bins, we can use **alt.Bin** to adjust bin parameters

[49]:
```
alt.Chart(cars).mark_bar().encode( x=alt.X('Miles_per_Gallon', bin=alt.
↪Bin(maxbins=30)), y='count()' )
```

[49]: alt.Chart(…)

If we apply another encoding (such as color), the data will be automatically grouped within each bin:

[50]:
```
alt.Chart(cars).mark_bar().encode( x=alt.X('Miles_per_Gallon', bin=alt.
↪Bin(maxbins=30)), y='count()', color='Origin' )
```

[50]: alt.Chart(…)

If you prefer a separate plot for each category, the column encoding can help:

[51]:
```
alt.Chart(cars).mark_bar().encode( x=alt.X('Miles_per_Gallon', bin=alt.
↪Bin(maxbins=30)), y='count()', color='Origin', column='Origin' ).
↪interactive()
```

[51]: alt.Chart(…)

[52]:
```
alt.Chart(cars).mark_rect().encode( x=alt.X('Miles_per_Gallon', bin=True),␣
↪y=alt.Y('Horsepower', bin=True), color='count()' )
```

[52]: alt.Chart(…)

[53]:
```
alt.Chart(cars).mark_rect().encode( x=alt.X('Miles_per_Gallon', bin=True),␣
↪y=alt.Y('Horsepower', bin=True), color='mean(Weight_in_lbs)' )
```

[53]: alt.Chart(…)

Time-Series & Layering

So far we've been ignoring the date column, but it's interesting to see the trends with time of, for example, miles per gallon:

[54]:
```
alt.Chart(cars).mark_point().encode( x='Year', y='Miles_per_Gallon' )
```

[54]: alt.Chart(…)

Each year has a number of cars, and a lot of overlap in the data. We can clean this up a bit by plotting the mean at each x value:

[55]:
```
alt.Chart(cars).mark_line().encode( x='Year', y='mean(Miles_per_Gallon)', )
```

[55]: alt.Chart(…)

[56]:
```
alt.Chart(cars).mark_area().encode( x='Year', y='ci0(Miles_per_Gallon)',␣
↪y2='ci1(Miles_per_Gallon)' )
```

[56]: alt.Chart(…)

Alternatively, we can change the mark to area and use the ci0 and ci1 mark to plot the confidence interval of the estimate of the mean:

Let's adjust this chart a bit: add some opacity, color by the country of origin, and make the width a bit wider, and add a cleaner axis title:

[57]:
```
alt.Chart(cars).mark_area(opacity=0.3).encode( x=alt.X('Year',␣
↪timeUnit='year'), y=alt.Y('ci0(Miles_per_Gallon)', axis=alt.
↪Axis(title='Miles per Gallon')), y2='ci1(Miles_per_Gallon)', color='Origin'␣
↪).properties( width=800 )
```

[57]: alt.Chart(…)

[58]:
```
spread = alt.Chart(cars).mark_area(opacity=0.3).encode( x=alt.X('Year',␣
↪timeUnit='year'), y=alt.Y('ci0(Miles_per_Gallon)', axis=alt.
↪Axis(title='Miles per Gallon')), y2='ci1(Miles_per_Gallon)', color='Origin'␣
↪).properties( width=800 )
lines = alt.Chart(cars).mark_line().encode( x=alt.X('Year', timeUnit='year'),␣
↪y='mean(Miles_per_Gallon)', color='Origin' ).properties( width=800 )
spread + lines
```

[58]:
```
alt.LayerChart(…)
```

The nice thing about this selection API is that it automatically applies across any compound charts; for example, here we can horizontally concatenate two charts, and since they both have the same selection they both respond appropriately:

[59]:
```
interval = alt.selection_interval()
base = alt.Chart(cars).mark_point().encode( y='Horsepower', color=alt.
↪condition(interval, 'Origin', alt.value('lightgray')), tooltip='Name' ).
↪add_params( interval )

base.encode(x='Miles_per_Gallon') | base.encode(x='Acceleration')
```

[59]:
```
alt.HConcatChart(…)
```

We can do even more sophisticated things with selections as well. For example, let's make a histogram of the number of cars by Origin, and stack it on our scatterplot:

[60]:
```
interval = alt.selection_interval()
base = alt.Chart(cars).mark_point().encode( y='Horsepower', color=alt.
↪condition(interval, 'Origin', alt.value('lightgray')), tooltip='Name' ).
↪add_params( interval )

hist = alt.Chart(cars).mark_bar().encode( x='count()', y='Origin',␣
↪color='Origin' ).properties( width=800, height=80 ).transform_filter(␣
↪interval )

scatter = base.encode(x='Miles_per_Gallon') | base.encode(x='Acceleration')

scatter & hist
```

[60]:
```
alt.VConcatChart(…)
```

1 Exploring another dataset named stocks

[61]:
```
from vega_datasets import data
stocks = data.stocks()
stocks.head()
```

```
alt.Chart(stocks).mark_line().encode( x='date:T', y='price:Q', color='symbol:N'␣
↪)
```

[61]: alt.Chart(…)

[62]:
```
# here is the same plot with a circle mark:
alt.Chart(stocks).mark_circle().encode( x='date:T', y='price:Q', color='symbol:
↪N' )
```

[62]: alt.Chart(…)

[63]:
```
lines = alt.Chart(stocks).mark_line().encode( x='date:T', y='price:Q',␣
↪color='symbol:N' )
points = alt.Chart(stocks).mark_circle().encode( x='date:T', y='price:Q',␣
↪color='symbol:N' )
lines + points
```

[63]: alt.LayerChart(…)

[64]:
```
base.mark_line() | base.mark_circle()
```

[64]: alt.HConcatChart(…)

[65]:
```
alt.hconcat(base.mark_line(),base.mark_circle())
```

[65]: alt.HConcatChart(…)