010- A Conceptual Framework for Data Visualization

Dependency modeling

- The fundamental principle of modeling data to determine the nature and structure of the representation
- This process searches for relationships between the data elements
- Example: A department store might gather data on the purchasing habits of its customers. This process helps the department store deduce the information about frequent purchases

Clustering

- The task of discovering groups in the data that have, in some way or another, a “similar pattern”, without using known structures in the data

Organising data

- Database management systems allow users to store data in a structured format
- Ways to structure the data:
  - Storing large data in disks in a structured format like tables, trees, or graphs
  - Storing data in memory using data structure formats for faster Access

Where does visualization fit in?

- Visualization is useful in both

organizing and making sense out of data

• Visualization can be used wherever we see increased knowledge or value of data

Importance of data visualization

• Complex data can be easily understood

• A simple visual representation of outliers, target audiences, and future markets can be created

• Storytelling can be done using dashboards and animations

• Data can be explored through interactive visualizations

Data Wrangling

• To draw conclusions from visualized data, we need to handle our data and transform it into the best possible representation

• It is the discipline of augmenting, transforming, and enriching data in a way that allows it to be displayed and understood by machine learning algorithms

020- Visualization Plots

Comparison plots

• Include charts that are well-suited for comparing multiple variables or variables over time

• Examples: **Line chart, bar chart, radar chart**

Line chart

- Display quantitative values over a continuous time period and show information as a series
- A line chart is ideal for a time series, which is connected by straight-line segments.
- The value is placed on the y-axis, while the x-axis is the timescale.

Line chart: Uses

- For comparing multiple variables and visualizing trends for both single as well as multiple variables, especially if your dataset has many time periods (roughly more than ten)
- For smaller time periods, vertical bar charts might be the better Choice

Bar chart

- The bar length encodes the value
- Two variants of bar charts:
  - Vertical bar charts
  - Horizontal bar charts

Bar chart: Uses

- While they are both used to compare numerical values across categories, vertical bar charts are sometimes used to show a single variable over time

Radar chart

- Also known as spider, or web charts
- Visualize multiple variables with each variable plotted on its own axis, resulting in a polygon
- All axes are arranged radially, starting at the center with equal distances between one another and have the same scale

Radar chart: Uses

- Radar charts are great for comparing multiple quantitative variables for a single group or multiple groups
- Useful to show which variables score high or low within a dataset, making them ideal to visualize performance

Composition plots

- Ideal if you think about something as a part of a whole
- Examples: **Pie chart, stacked bar chart, stacked area chart, Venn diagrams**

Pie chart

- Illustrate numerical proportion by dividing a circle into slices
- Each arc length represents a proportion of a category

- The full circle equals to 100%

Pie chart: Uses

- Compare items that are part of a whole

Stacked bar chart

- To show how a category is divided into sub-categories and the proportion of the sub-category, in comparison to the overall category
- Can either compare total amounts across each bar

Stacked bar chart: Uses

- Compare variables that can be divided into sub-variables

Stacked area chart

- To show trends for part-of-a-whole relations
- The values of several groups are illustrated on top of one another
- Helps to analyze both individual and overall trend information

Stacked area chart: Uses

- Show trends for time series that are part of a whole

Distribution plots

- Give a deep insight into how your data is distributed
- Examples: **Histogram, violin plot, box plot**

Histogram

- Visualizes the distribution of a single numerical variable
- Each bar represents the frequency for a certain interval
- Histograms help get an estimate of statistical measures
- Allows to see where values are concentrated, hence easily detecting outliers
- Can either plot a histogram with absolute frequency values or alternatively normalize the histogram
- Different colour bars can be used if comparison of distribution of multiple variables is needed

Histogram: Uses

- Get insights into the underlying distribution for a dataset

Density plot

- Shows the distribution of a numerical variable
- A variation of a histogram that uses **kernel smoothing**, allowing for smoother distributions
- An advantage they have over histograms is that density plots are better at

determining the distribution shape, since the distribution shape for histograms heavily depends on the number of bins (data intervals)

Density plot: Uses

• Compare the distribution of several variables by plotting the density on the same axis and using different colors

Box plot

• Shows multiple statistical measurements

• The box extends from the lower to the upper quartile values of the data, thus allowing us to visualize the interquartile range

• The horizontal line within the box denotes the median

• The **whiskers** extending from the box show the range of the data

• It is also an option to show data **outliers**, usually as circles or diamonds, past the end of the whiskers

Box plot: Uses

• Shows statistical measures

Violin plot

• A combination of box plots and density plots

• Both the statistical measures and the distribution are visualized

• The thick black bar in the center represents the interquartile range, the thin black line shows the 95% confidence interval, and the white dot shows the median

• On both sides of the center-line, the density is visualized

Violin plot: Uses

• Compare statistical measures for multiple variables or groups

Relation plots

• Perfectly suited to show relationships among variables.

• Examples: **Scatter plot, bubble plot, correlogram, heatmaps**

Scatter plot

• Show data points for two numerical variables, displaying a variable on both axes

Scatter plot: Uses

• Can detect whether a correlation (relationship) exists between two variables

• Allow you to plot the relationship for multiple groups or categories using different colors

Bubble plot

- Extends a scatter plot by introducing a third numerical variable
- The value of the variable is represented by the size of the dots
- The area of the dots is proportional to the value
- A legend is used to link the size of the dot to an actual numerical value

Bubble plot: Uses

- To show a correlation between three variables

Correlogram

- A combination of scatter plots and histograms
- A correlogram or correlation matrix visualizes the relationship between each pair of numerical variables using a scatter plot
- The diagonals of the correlation matrix represent the distribution of each variable in the form of a histogram
- For exploratory data analysis to get a feeling for your data, especially the correlation between variable pairs

Heatmap

- A visualization where values contained in a matrix are represented as colors or color saturation
- For visualizing multivariate data, where categorical variables are placed in the rows and columns and a numerical or categorical variable is

represented as colors or color saturation

Heatmap: Uses

- Visualization of multivariate data
- For finding patterns in your data

Geological plots

- Visualize geospatial data
- Examples: **Dot maps, choropleth maps, connection maps**

Dot maps

- Each dot represents a certain number of observations
- Each dot has the same size and value (the number of observations each dot represents).

‣ The dots are not meant to be counted—they are only intended to give an impression of magnitude

‣ The size and value are important factors for the effectiveness and impression of the visualization

‣ Can use different colors or symbols for the dots to show multiple categories or groups

Dot maps: Uses

- For the visualization of geospatial data

Choropleth map

- Each tile is colored to encode a variable
- A tile represents a geographic region for, for example, counties and countries.
- Shows how a variable varies across a geographic area
- One thing to keep in mind for choropleth maps is that the human eye naturally gives more prominence to larger areas, so you might want to normalize your data by dividing the map area-wise.

Choropleth map: Uses

- For the visualization of geospatial data grouped into geological regions, for example, states, or countries

Connection map

- Each line represents a certain number of connections between two locations
- The link between the locations can be drawn with a straight or rounded line representing the shortest distance between them

‣ Each line has the same thickness and value (number of connections each line represents)

‣ The lines are not meant to be counted; they are only intended to give an impression of magnitude.

‣ The size and value of a connection line are important factors for the effectiveness and impression of the visualization.

‣ Can use different colors for the lines to show multiple categories or groups, or you can use a colormap to encode the length of the connection

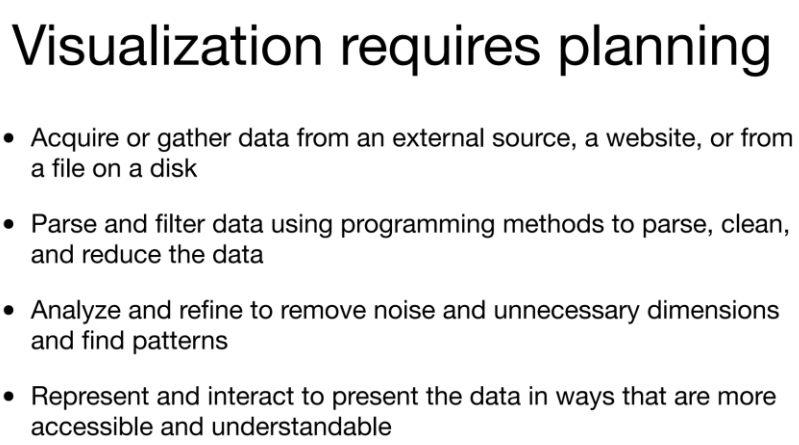
Connection map: Uses

- For the visualization of connections

Word Cloud

- Also known as tag cloud or word art
- A simple visualization of data
- Words are shown in varying sizes depends on the frequency of appearance in the data

030-VisualizationBasic



Creating interesting stories with data

- Data visualization regularly promotes its ability to reveal stories with data
- Understand that data would be understood and remembered better if presented in the right way
- Reader-driven narratives and author-driven narratives

Author-driven narratives

- Has data and visualization that are chosen by the author and presented to the public reader
- Has a strict linear path through the visualization, relies on messaging, and has no interactivity

Reader-driven narratives

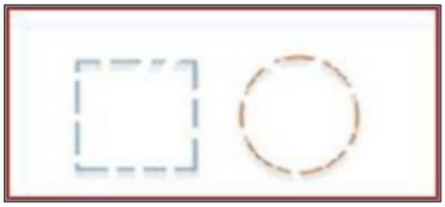
- Provides tools and methods for the reader to play with the data, which gives the reader more flexibility and choices to analyze and understand the visualization
- Has no prescribed ordering of images, no messaging, and has a high degree of interactivity

The Gestalt principles of perception

- Closure
- Continuity
- Figure / ground
- Proximity
- Similarity
- Enclosure
- Connection
- Common fate

Closure

- Even if a part of the border of a shape is missing, we still tend to see the shape as completely enclosed by the border and ignore the gaps



Continuity

- Some things are important as a whole, which means if there are interruptions, then it changes the perceptive reading
- We perceive the two crossed lines instead of four lines meeting at the center:

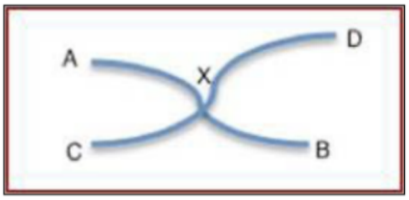
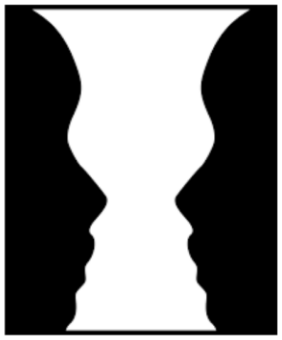


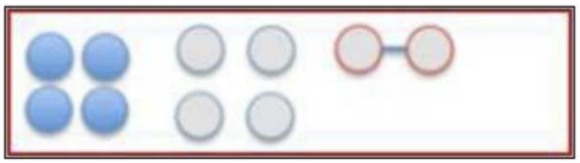
Figure / Ground

- People instinctively perceive objects as either being in the foreground or the background.
- Either stand out prominently in the front (the figure) or recede into the back (the ground).



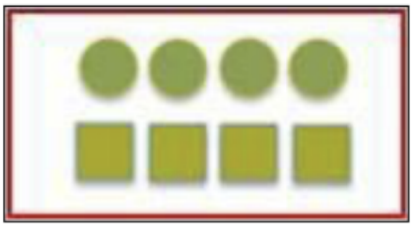
Proximity

- Objects that are close together or connected to each other are perceived as a group, reducing the need to process smaller objects separately



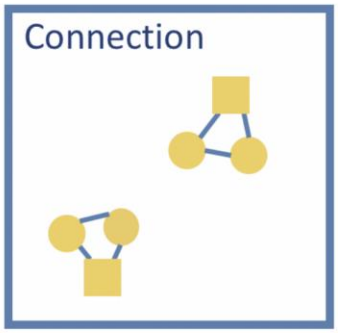
Similarity

- Objects that share similar attributes, color, or shape are perceived as a group



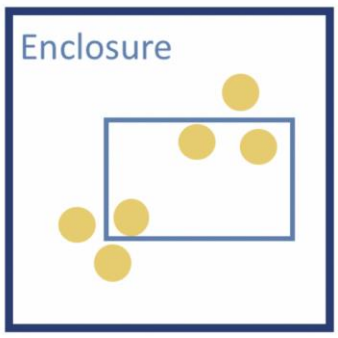
Connection

- Our tendency to perceive objects that are physically connected are part of the same group, whether or not that is true or relevant to the data at hand.
- For example, our eyes connect the group of three shapes in the lower left together and the three shapes in the upper right together, before connecting the dots and squares together.
- The connection principle often is stronger than proximity or similarity principles, but not as strong as enclosure.



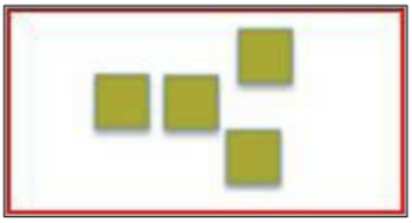
Enclosure

- Reflects our tendency to perceive objects physically enclosed together in a ways that seems to create a boundary around that which is related or belonging to part of the same group, whether or not that is true or relevant to the data at hand.
- For example, our eyes see the yellow dots within the blue rectangle as having something in common with one another as they are all “in the box” together.
- We attribute meaning to the “box” whether that is accurate or not given what we are looking at.



Common fate

- When both the principles of proximity and similarity are in place, a movement takes place. Then they appear to change grouping

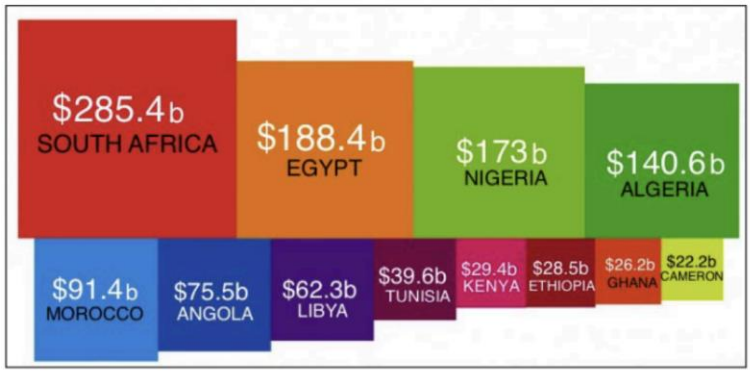


Best practices for visualization

- Comparison and ranking
- Correlation
- Distribution
- Location-specific or geodata
- Part-to-whole relationships
- Trends over time

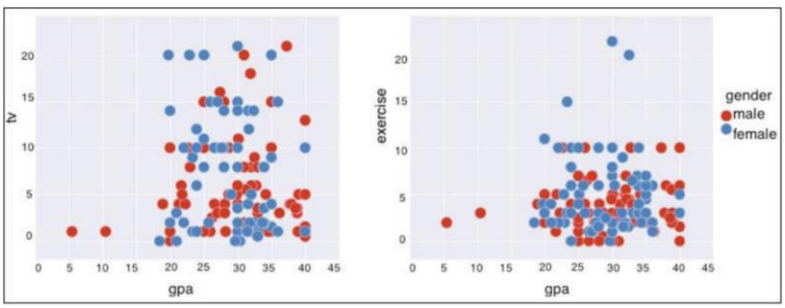
Comparison and ranking

- Typically using bar charts



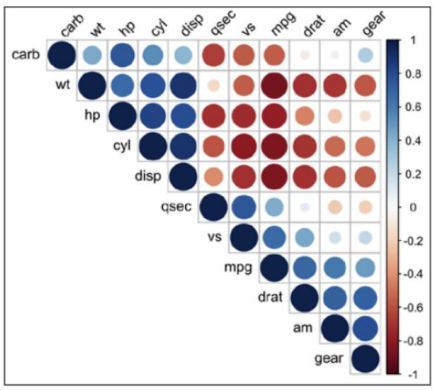
Correlation

- Typically using scatter plot to detect the correlations between two factors



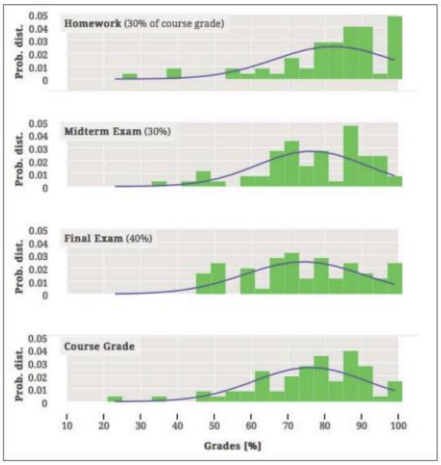
Correlation

- Use of correlation matrix to investigate the dependence between multiple variables at the same time



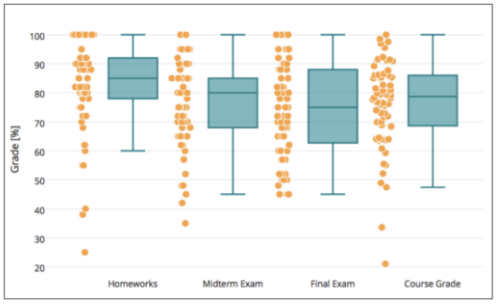
Distribution

- A distribution analysis shows how the quantitative values are distributed across their range
- Histogram, box plot, or box-and-whisker plot



Distribution

- The box-and-whisker plots are excellent for displaying multiple distributions
- Can easily identify the low values, the 25th-percentile values, the medians, the 75th-percentiles, and the maximum values across all categories—all at the same time

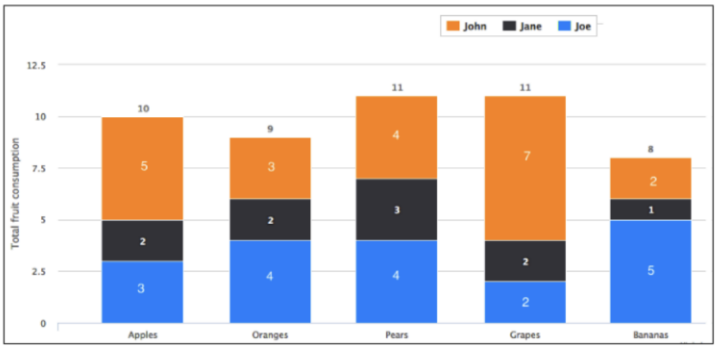


Location-specific or geodata

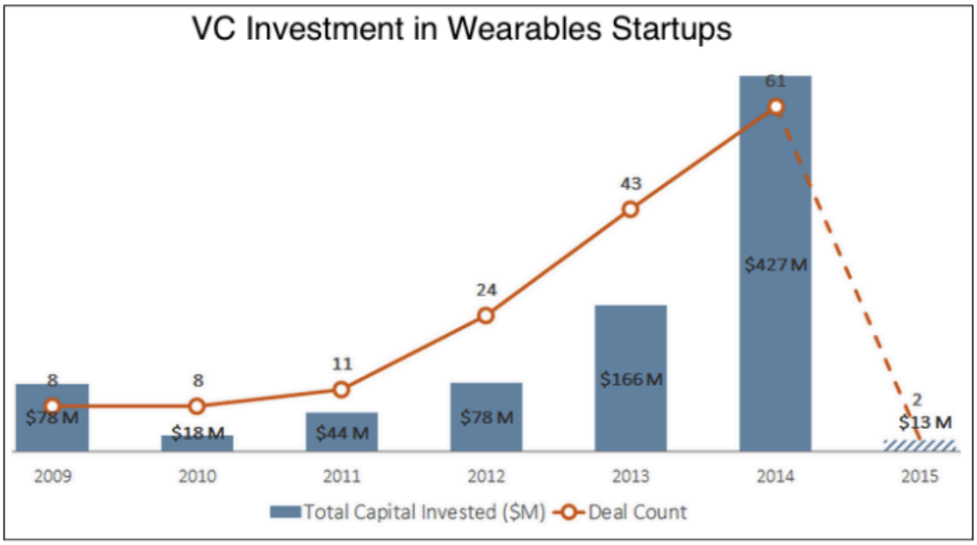
- Paired with another chart that details what the map is displaying

Part-to-whole relationships

- Pie chart, grouped bar charts or stacked column charts



Trends over time



Interactive visualization

- For a visualization to be considered interactive, it must satisfy two criteria:
  - Human input: The control of some aspect of the visual representation of information must be available to humans
  - Response time: The changes made by humans must be incorporated into the visualization in a timely manner

Advantages and Disadvantages

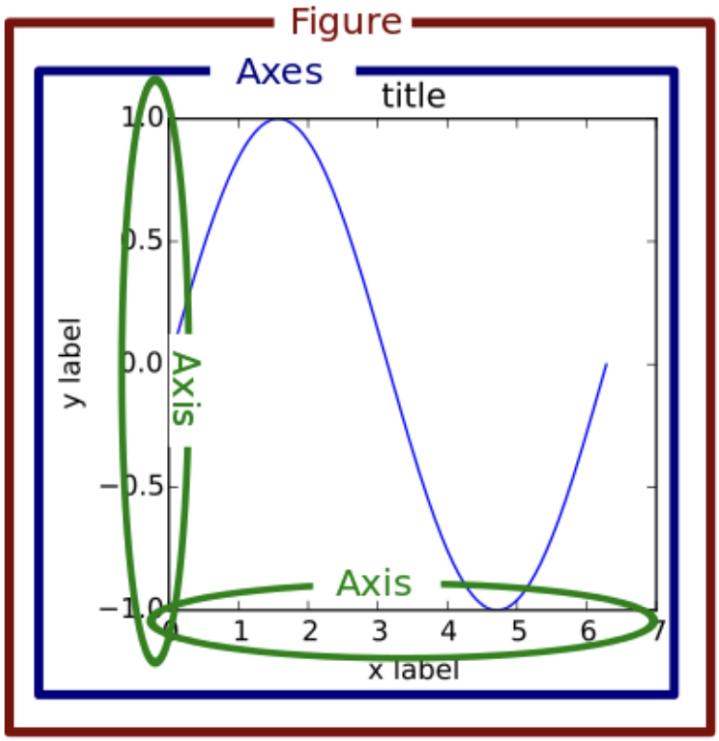
- Advantage: People can explore a larger information space in a shorter time, which can be understood through one platform
- Disadvantage: It requires a lot of time to exhaustively check every possibility to test the visualization system

040-Matplotlib

Components of Plots

- The two main components of a plot are as follows:
  - **Figure:**
    - An outermost container and is used as a canvas to draw on
    - Can draw multiple plots within it
    - Holds any number of Axes object
    - Can configure the Title
  - **Axes**:
    - An actual plot, or subplot, depending on whether you want to plot single or multiple visualizations
    - Its sub-objects include the x and y axis, spines, and legends

Parts of a `Figure`



Axes

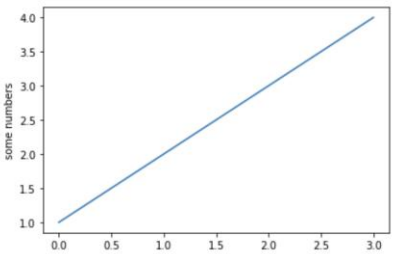
- This is what you think of as ‘a plot’
- A given figure can contain many Axes, but a given `Axes` object can only be in one `Figure`
- The Axes contains two (or three in the case of 3D) `Axis` objects (be aware of the difference between **Axes** and **Axis**) which take care of the data limits (the data limits can also be controlled via the `set_xlim()` and `set_ylim()` `Axes` methods)
- Each `Axes` has a title (set via `set_title()`), an x-label (set via `set_xlabel()`), and a y-label set via `set_ylabel()`)

Plotting data points

```
import matplotlib.pyplot as plt
```

- `plt.plot([x], y, [fmt])`
  - Plot data points as lines and/or markers
- `plt.show()` is used to display a Figure or multiple Figures

```
# Plot data points as lines (default)
plt.plot([1,2,3,4])
plt.ylabel("some numbers")
plt.show()
```



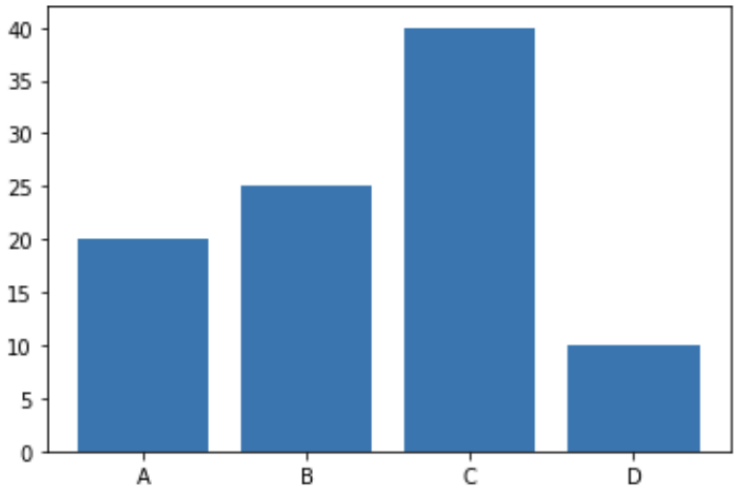
Bar chart

- `plt.bar(x, height, [width])` creates a vertical bar plot. For horizontal bars, use the `plt.barh()` function
- Important parameters:
  - `x`: Specifies the x coordinates of the bars
  - `height`: Specifies the height of the bars
  - `width` (optional): Specifies the width of all bars; the default is 0.8

Bar chart

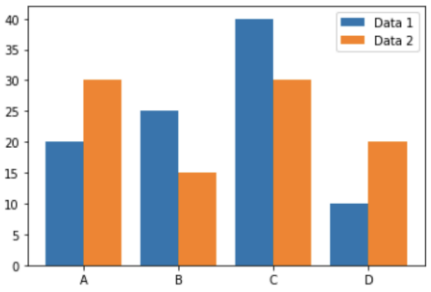
```
plt.bar(['A', 'B', 'C', 'D'], [20, 25, 40, 10])
```

```
<BarContainer object of 4 artists>
```



Bar chart

```
import numpy as np

labels = ['A', 'B', 'C', 'D']
x = np.arange(len(labels))
width = 0.4

# Manually position the bars
plt.bar(x-width / 2, [20, 25, 40, 10], width=width, label='Data 1')
plt.bar(x+width / 2, [30, 15, 30, 20], width=width, label='Data 2')

# Get the current Axes
ax = plt.gca()

ax.legend()

# Set the xticks to be x
ax.set_xticks(x)
# Set the xticklabels
ax.set_xticklabels(labels)

plt.show()
```

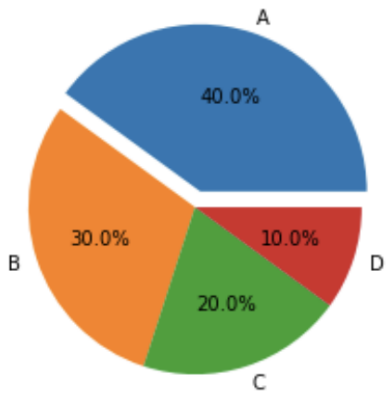


Pie chart

- `plt.pie(x, [explode], [labels], [autopct])` function creates a pie chart
- Important parameters:
  - `x`: Specifies the slice sizes
  - `explode` (optional): Specifies the fraction of the radius offset for each slice. The
    explode-array must have the same length as the x-array
  - `labels` (optional): Specifies the labels for each slice
  - `autopct` (optional): Shows percentages inside the slices according to the specified format string (e.g.,`'%1.1f%%'`)

Pie chart

```
plt.pie([0.4, 0.3, 0.2, 0.1], explode=(0.1, 0, 0, 0),
        labels=['A', 'B', 'C', 'D'], autopct='%1.1f%%')
plt.show()
```

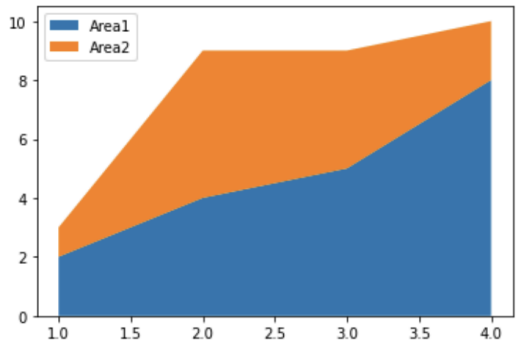


Stacked area chart

- `plt.stackplot(x, y)` creates a stacked area plot
- Important parameters:
  - `x`: Specifies the x-values of the data series
  - `y`: Specifies the y-values of the data series. For multiple series, either as a 2D array, or any number of 1D arrays, use `plt.stackplot(x, y1, y2, y3, …)`
  - `labels` (Optional): Specifies the labels as a list or tuple for each data series

Stacked area chart

```
# plt.stackplot(x, y1, y2)
plt.stackplot([1, 2, 3, 4], [2, 4, 5, 8], [1, 5, 4, 2], labels=['Area1', 'Area2'])
plt.legend(loc='upper left')
plt.show()
```

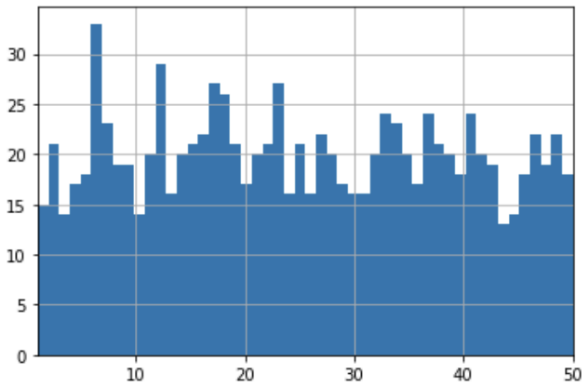


Histogram

- `plt.hist(x)` creates a histogram
- Important parameters:
  - `x`: Specifies the input values
  - `bins`: (optional): Either specifies the number of bins as an integer or specifies the bin edges as a list
  - `range`: (optional): Specifies the lower and upper range of the bins as a tuple
  - `density`: (optional): If true, the histogram represents a probability density

Histogram

```
import random

# generate a list of 100 random numbers with range from 1 to 50 (inclusive)
x = [random.randint(1,50) for x in range(1000)]

plt.hist(x, bins=50, density=False)
plt.xlim(1,50)
plt.grid()
plt.show()
```

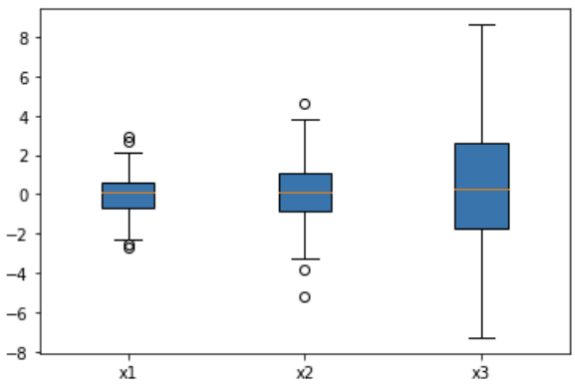


Box plot

- `plt.boxplot(x)` creates a box plot
- Important parameters:
  - `x`: Specifies the input data. It specifies either a 1D array for a single box or a sequence of arrays for multiple boxes.
  - `notch` (Optional): If true, notches will be added to the plot to indicate the confidence interval around the median
  - `labels`: Optional: Specifies the labels as a sequence
  - `showfliers`: Optional: By default, it is true, and outliers are plotted beyond the caps
  - `showmeans`: Optional: If true, arithmetic means are shown

Box plot

```
import random
import numpy as np

# generate some random test data
all_data = [np.random.normal(0, std, size=100) for std in range(1, 4)]
labels = ['x1', 'x2', 'x3']

plt.boxplot(all_data, vert=True,  # vertical box alignment
        patch_artist=True,        # fill with color
        labels=labels)            # will be used to label x-ticks
plt.show()
```

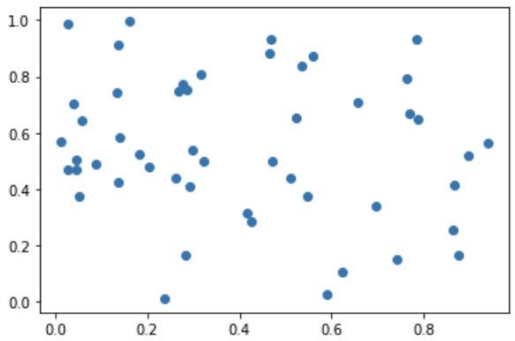


Scatter plot

- `plt.scatter(x, y)` creates a scatter plot of y versus x with optionally varying marker size and/or color
- Important parameters:
  - `x`, `y`: Specifies the data positions
  - `s` (Optional): Specifies the marker size in points squared
  - `c` (Optional): Specifies the marker color. If a sequence of numbers is specified, the numbers will be mapped to colors of the color map

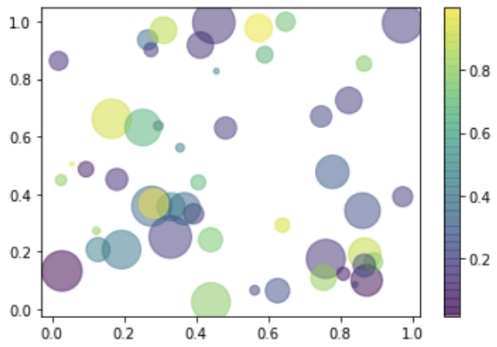
Scatter plot

```
import numpy as np

N = 50

# np.random.rand(N): Return a sample (or samples) from the "standard normal" distribution
x = np.random.rand(N)
y = np.random.rand(N)

plt.scatter(x, y)
plt.show()
```



Bubble plot

```
N = 50

# np.random.rand(N): Return a sample (or samples) from the "standard normal" distribution
x = np.random.rand(N)
y = np.random.rand(N)
colors = np.random.rand(N)
area = (30 * np.random.rand(N))**2  # 0 to 15 point radii

plt.scatter(x, y, s=area, c=colors, alpha=0.5)
plt.colorbar()
plt.show()
```



050-Seaborn

Default data set

- Seaborn comes with a number of example datasets
- The dataset are also available at: https://github.com/mwaskom/seaborn-data

```
sns.get_dataset_names()
```

```
['anagrams',
 'anscombe',
 'attention',
 'brain_networks',
 'car_crashes',
 'diamonds',
 'dots',
 'exercise',
 'flights',
 'fmri',
 'gammas',
 'geyser',
 'iris',
 'mpg',
 'penguins',
 'planets',
 'tips',
 'titanic']
```

Loading default data set

- The data set can be loaded as `pandas DataFrame`

```
# Load an example dataset
tips = sns.load_dataset("tips")
type(tips)
```

```
pandas.core.frame.DataFrame
```

```
tips
```

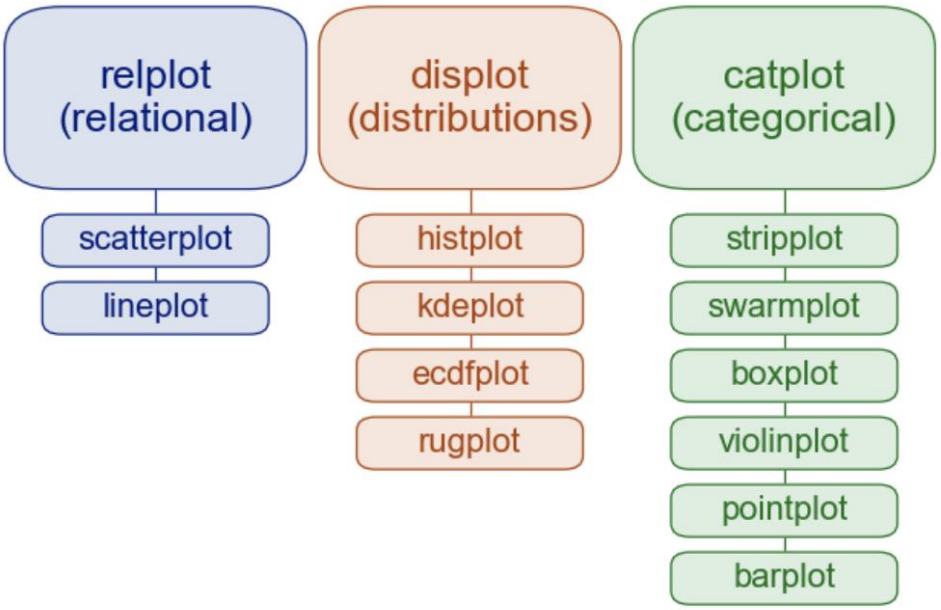
	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4
...	...	...	...	...	...	...	...
239	29.03	5.92	Male	No	Sat	Dinner	3
240	27.18	2.00	Female	Yes	Sat	Dinner	2
241	22.67	2.00	Male	Yes	Sat	Dinner	2
242	17.82	1.75	Male	No	Sat	Dinner	2
243	18.78	3.00	Female	No	Thur	Dinner	2

244 rows × 7 columns

Figure styles

- There are five preset seaborn themes: `darkgrid` (default), `whitegrid`, `dark`, `white`, and `ticks`
- Can use the method `set_theme()` with the parameter `style` to change the theme

Plots



Visualizing statistical relationships

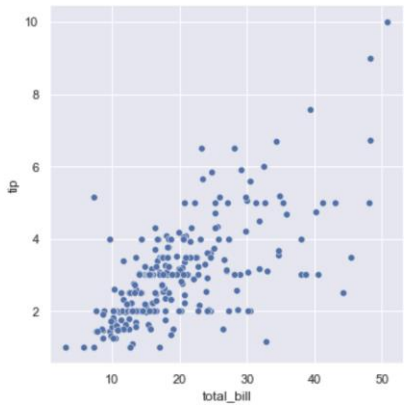
- Statistical analysis is a process of understanding how variables in a dataset relate to each other and how those relationships depend on other variables
- Visualization can be a core component of this process because, when data are visualized properly, the human visual system can see trends and patterns that indicate a relationship

`relplot()`

- For visualizing statistical relationships using two common approaches:
  - Scatter plots (with **`kind="scatter"`**; default)
    - Draw a scatter plot with possibility of several semantic groupings
  - Line plots (with **`kind="line"`**)
    - Draw a line plot with possibility of several semantic groupings

Two dimensional data

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

tips = sns.load_dataset("tips")

sns.set_theme(style="darkgrid")

# data: pandas.DataFrame
sns.relplot(x="total_bill", y="tip", data=tips)
```



Themes

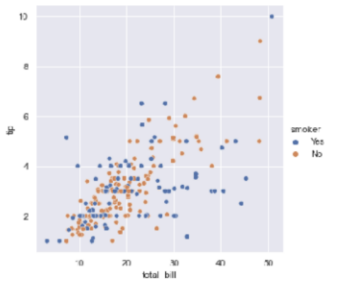
- There are five preset seaborn themes: `darkgrid`, `whitegrid`, `dark`, `white`, and `ticks`.
- The default theme is `darkgrid`

Adding one more dimension using `hue`

- "hue semantic"
- While the points are plotted in two dimensions, another dimension can be added to the plot by coloring the points according to a third variable

```
# add 1 more dimension - whether that is a smoker, represents by different color
sns.relplot(x="total_bill", y="tip", hue="smoker", data=tips)
```

```
<seaborn.axisgrid.FacetGrid at 0x7fb6b4636610>
```



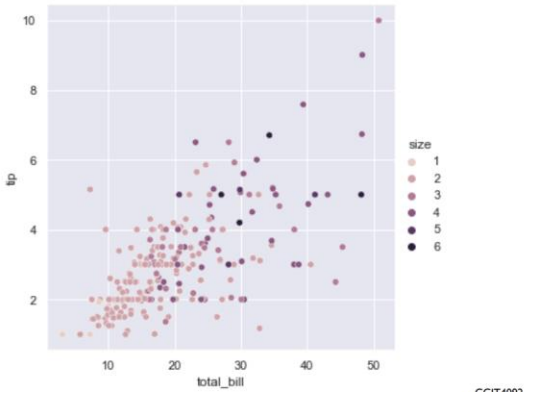
Color palette

- Qualitative palettes, good for representing categorical data
- Sequential palettes, good for representing numeric data
- Diverging palettes, good for representing numeric data with a categorical boundary

Representing numeric values using `hue`

- Sequential color palettes are appropriate when the data ranges from relatively low or uninteresting values to relatively high or interesting values

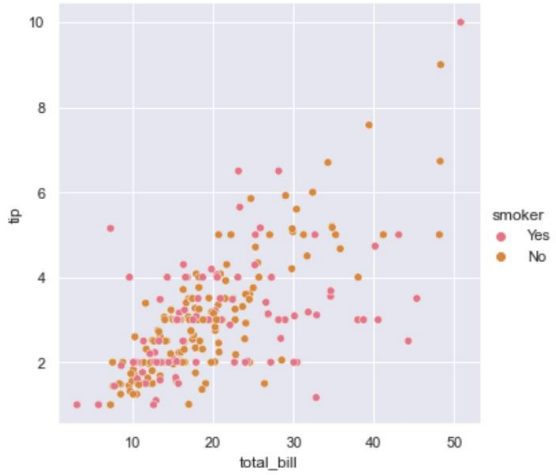
```
# Use color to represent numerical values
# sequential palette is used
sns.relplot(x="total_bill", y="tip", hue="size", data=tips);
```



Using custom palette (1)

```
# Using custom palette
cp=sns.set_palette(sns.color_palette("husl", 12))
sns.relplot(x="total_bill", y="tip", hue="smoker", data=tips, palette=cp)
```

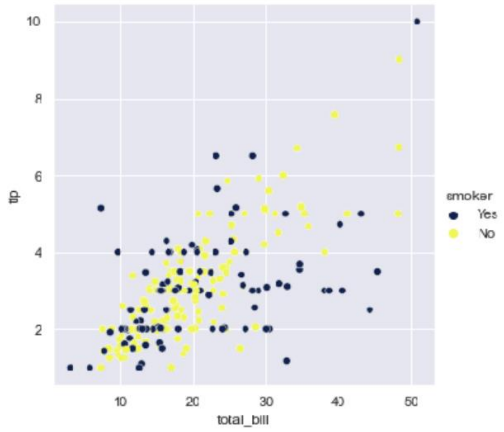
```
<seaborn.axisgrid.FacetGrid at 0x7fe6ea7a10a0>
```



Using custom palette (2)

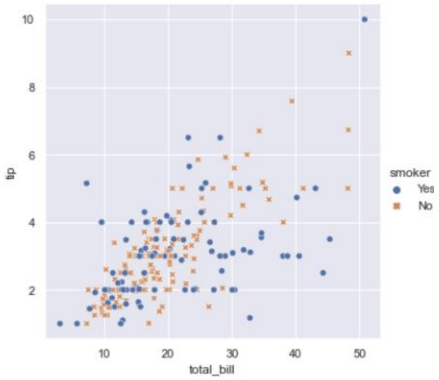
```
# Using custom palette

# Create an array with the colors
colors = ["#112452", "#F0F71A"]
# Set your custom color palette
cp = sns.set_palette(sns.color_palette(colors))

sns.relplot(x="total_bill", y="tip", hue="smoker", data=tips, palette=cp)
```

```
<seaborn.axisgrid.FacetGrid at 0x7fe5eb843df0>
```



Using `style`

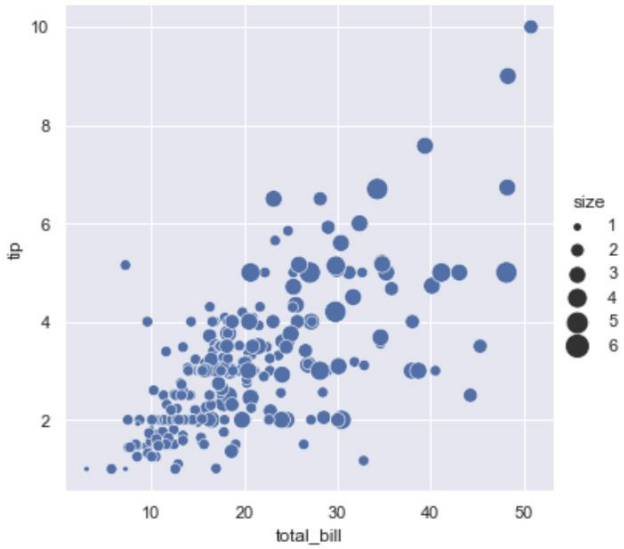
- To emphasize the difference between the classes, and to improve accessibility
- Use a different marker style for each class
- Note that eye is much less sensitive to shape than to color




```
# change the marker style according to the catagories
sns.relplot(x="total_bill", y="tip", hue="smoker", style="smoker", data=tips)
```

Using `size`

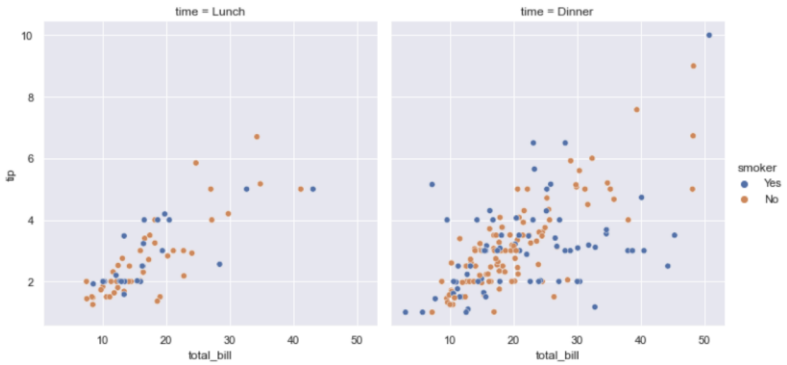
```
# Use size of dots to represent numerical values
# sizes is used to customize the size of the dots
sns.relplot(x="total_bill", y="tip", size="size", sizes=(15, 200), data=tips)
```



Showing multiple relationships

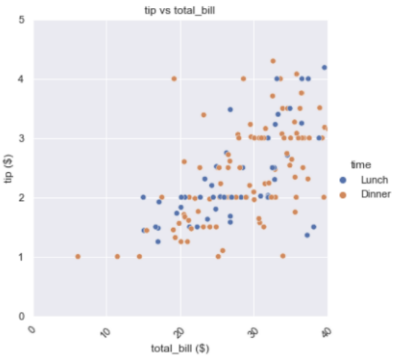
- Make more than one plots
- Using `col`

```
sns.relplot(x="total_bill", y="tip", hue="smoker",
            col="time", data=tips);
```



Customize the plot

```
g = sns.relplot(x="total_bill", y="tip", hue="time", data=tips)
g.set(xlim=(0,20), ylim=(0,5), xticks=[0,5,10,15,20], yticks=[0,1,2,3,4,5])
g.set_ylabels('tip ($)')
g.set_xlabels('total_bill ($)')
g.set_xticklabels(rotation=45)
plt.title('tip vs total_bill')
plt.show()
```



Distribution model in seaborn

- Axes level functions:
  - `histplot()`
  - `kdeplot()`
  - `ecdfplot()`
  - `rugplot()`
- Grouped together at figure level:
  - `displot()`
  - `jointplot()`
  - `pairplot()`

`displot()`

- For visualizing distribution using the following approaches
  - Histogram (with **`kind="hist"`**; default)
  - KDE plot (with `kind="kde"`)
  - ECDF Plot (with `kind="ecdf"`)

Histogram

- A histogram is a classic visualization tool that represents the distribution of one or more variables by counting the number of observations that fall within discrete bins
  - A bar plot where the axis representing the data variable is divided into a set of discrete bins and the count of observations falling within each bin is shown using the height of the corresponding bar

KDE Plot

- A kernel density estimate (KDE) plot is a method for visualizing the distribution of observations in a dataset, analogous to a histogram
- KDE represents the data using a continuous probability density curve in one or more dimensions

displot()

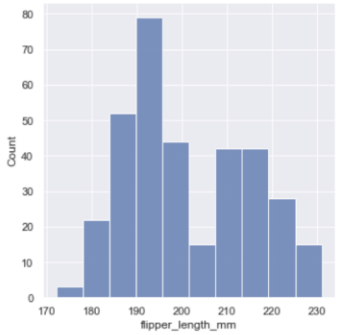
```
penguins = sns.load_dataset("penguins")
penguins
```

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
0	Adelie	Torgersen	39.1	18.7	181.0	3750.0	Male
1	Adelie	Torgersen	39.5	17.4	186.0	3800.0	Female
2	Adelie	Torgersen	40.3	18.0	195.0	3250.0	Female
3	Adelie	Torgersen	NaN	NaN	NaN	NaN	NaN
4	Adelie	Torgersen	36.7	19.3	193.0	3450.0	Female
...	...	...	...	...	...	...	...
339	Gentoo	Biscoe	NaN	NaN	NaN	NaN	NaN
340	Gentoo	Biscoe	46.8	14.3	215.0	4850.0	Female
341	Gentoo	Biscoe	50.4	15.7	222.0	5750.0	Male
342	Gentoo	Biscoe	45.2	14.8	212.0	5200.0	Female
343	Gentoo	Biscoe	49.9	16.1	213.0	5400.0	Male

344 rows × 7 columns

```
sns.displot(penguins, x="flipper_length_mm")
```

```
<seaborn.axisgrid.FacetGrid at 0x7fb6b5f3d9a0>
```

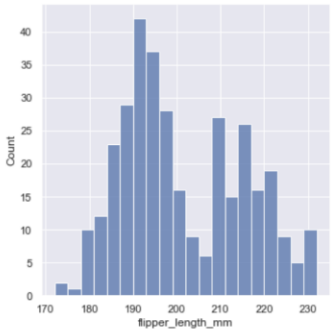


Choosing the bin size - binwidth

- The size of the bins is an important parameter, and using the wrong bin size can mislead by obscuring important features of the data or by creating apparent features out of random variability
- By default, default bin size is chosen based on the variance of the data and the number of observations

```
# Choosing the bin size
sns.displot(penguins, x="flipper_length_mm", binwidth=3)
```

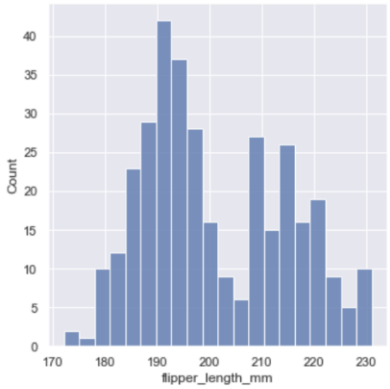
```
<seaborn.axisgrid.FacetGrid at 0x7fb6b5bc1970>
```



Choosing the number of bins - `bins`

```
# Choosing the number of bins
sns.displot(penguins, x="flipper_length_mm", bins=20)
```

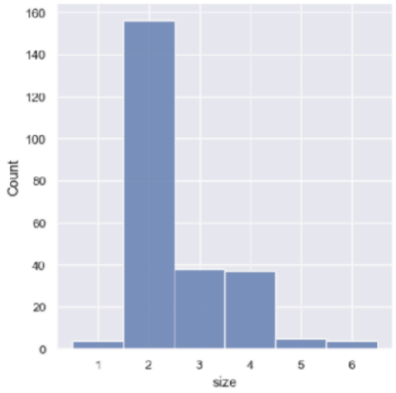
```
<seaborn.axisgrid.FacetGrid at 0x7fb6b5f44550>
```



`discrete=True`

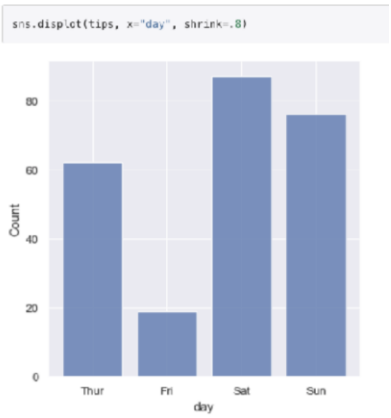
- Chooses bin breaks that represent the unique values in a dataset with bars that are centered on their corresponding value.

```
sns.displot(tips, x="size", discrete=True)
```



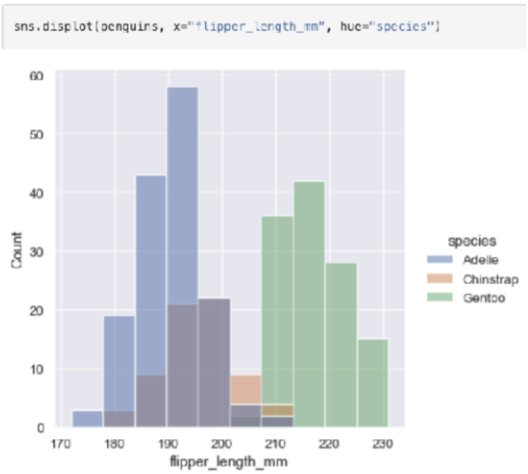
Visualize the distribution of a categorical variable

- Discrete bins are automatically set for categorical variables



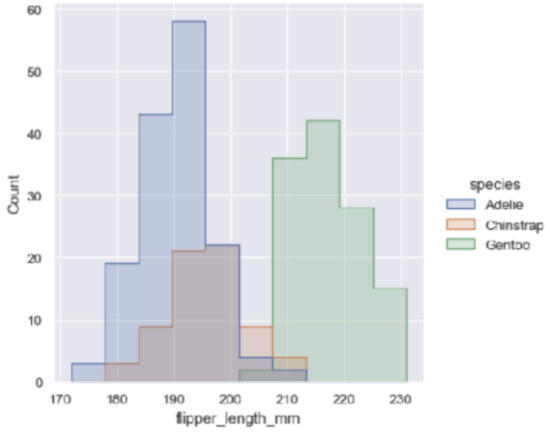
Conditioning on other variables

- Check whether features of that distribution differ across other variables in the dataset
- By default, the different histograms are “layered” on top of each other
- Conditioning via the `hue` parameter



Changing visual representation - `element = 'step'`

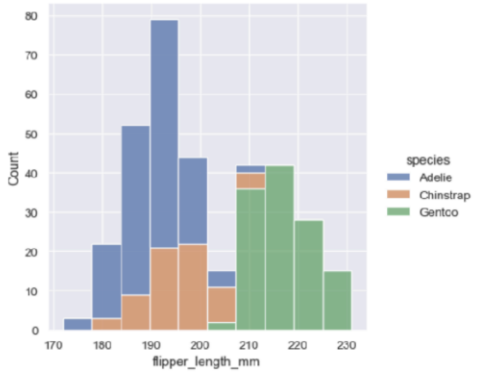
```
sns.displot(penguins, x="flipper_length_mm", hue="species", element="step")
```



Stacked histogram — `multiple='stack'`

```
sns.displot(penguins, x="flipper_length_mm", hue="species", multiple="stack")
```

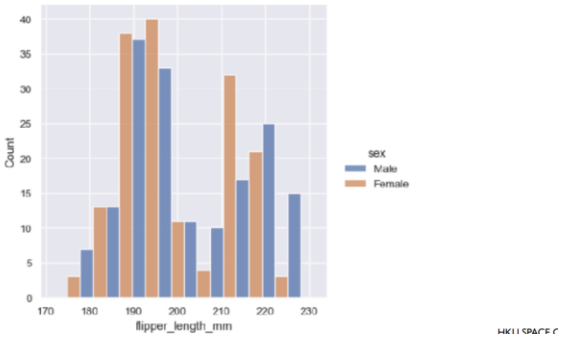
- Emphasizes the part-whole relationship between the variables



Changing visual representation - `multiple = 'dodge'`

- The option "dodge" moves the bars horizontally and reduces their width to ensure no overlaps and the bars remain comparable in terms of height
- Only works well when the categorical variable has a small number of levels

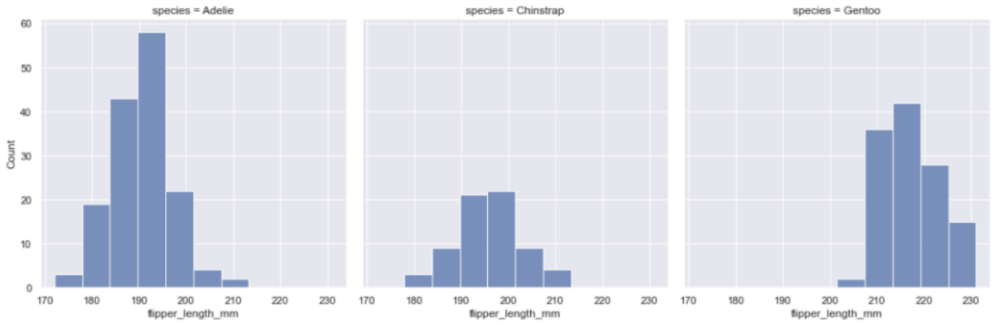
```
sns.displot(penguins, x="flipper_length_mm", hue="sex", multiple="dodge")
```



Conditioning on other variables

- Use of `FacetGrid` — `col` or `row` parameter

```
sns.displot(penguins, x="flipper_length_mm", col="species")
<seaborn.axisgrid.FacetGrid at 0x7fb6b0124790>
```



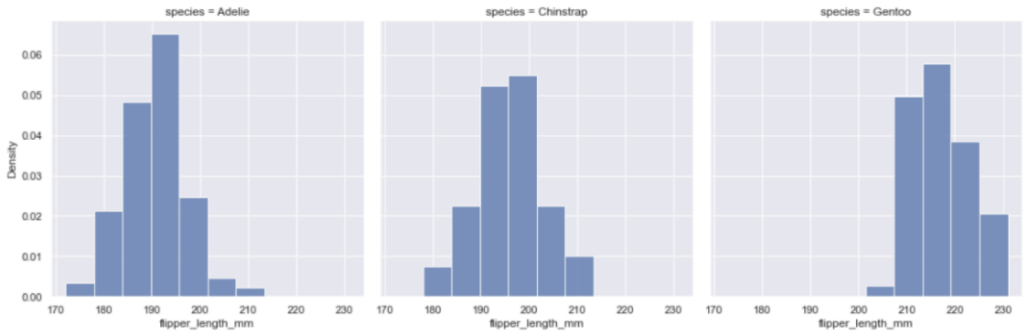
Normalized histogram statistics

- When the subsets have unequal numbers of observations, comparing their distributions in terms of counts may not be ideal
  - One solution is to normalize the counts using the `stat` parameter
  - Density normalization scales the bars so that their areas sum to 1
  - Probability normalisation normalize the bars to that their heights sum to 1

Normalize the counts — `stat="density"`

```
# common_norm=False, each subset will be normalized independently
# Density normalization scales the bars so that their areas sum to 1
sns.displot(penguins, x="flipper_length_mm", col="species", stat="density", common_norm=False)
```

```
<seaborn.axisgrid.FacetGrid at 0x7fb6b52a9f70>
```



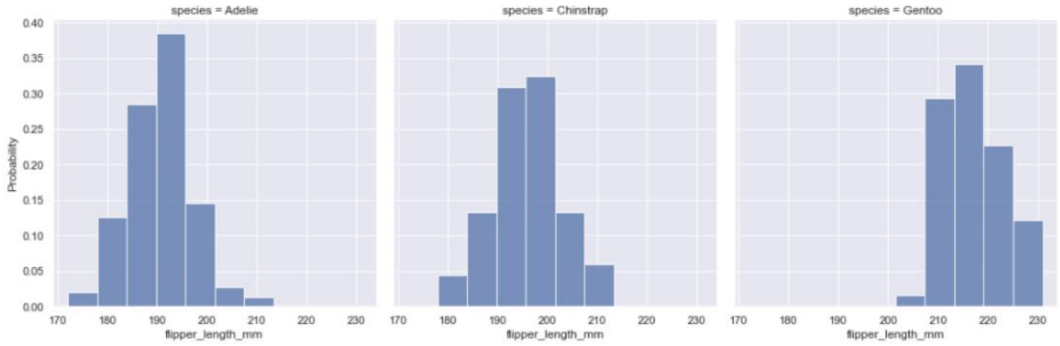
Normalization on entire distribution?

- The normalization is applied to the entire distribution by rescales the height of the bars
- To normalise each of the subsets independently, set `common_norm=False`

`stat="probability"`

```
sns.displot(penguins, x="flipper_length_mm", col="species", stat="probability", common_norm=False)
```

```
<seaborn.axisgrid.FacetGrid at 0x7fb6b528ad30>
```



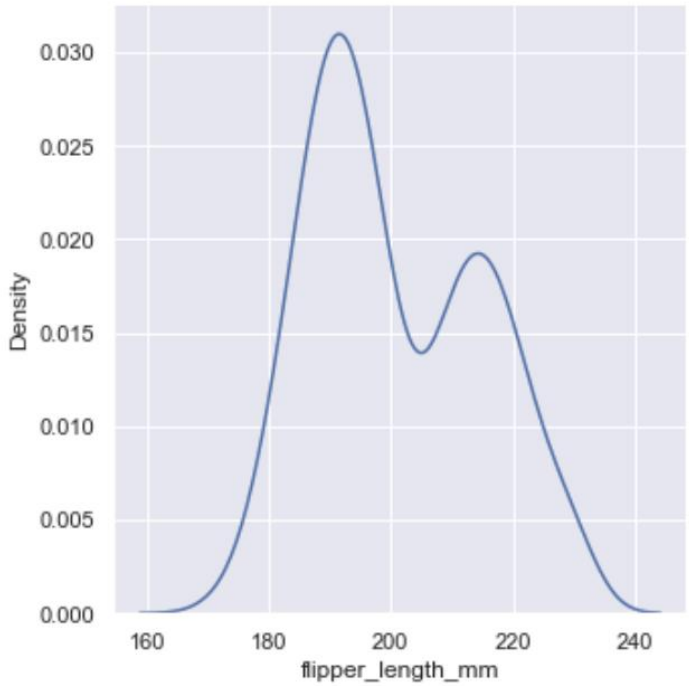
Histogram vs KDE

- A histogram aims to approximate the underlying probability density function that generated the data by binning and counting observations
- Kernel density estimation (KDE) presents a different solution to the same problem. Rather than using discrete bins, a KDE plot smooths the observations with a Gaussian kernel, producing a continuous density estimate

kind="kde"

```
sns.displot(penguins, x="flipper_length_mm", kind="kde")
```

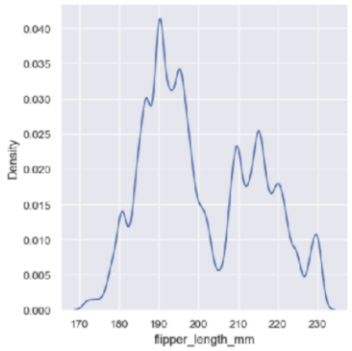
```
<seaborn.axisgrid.FacetGrid at 0x7fb6b79dcca0>
```



Choosing the smoothing bandwidth

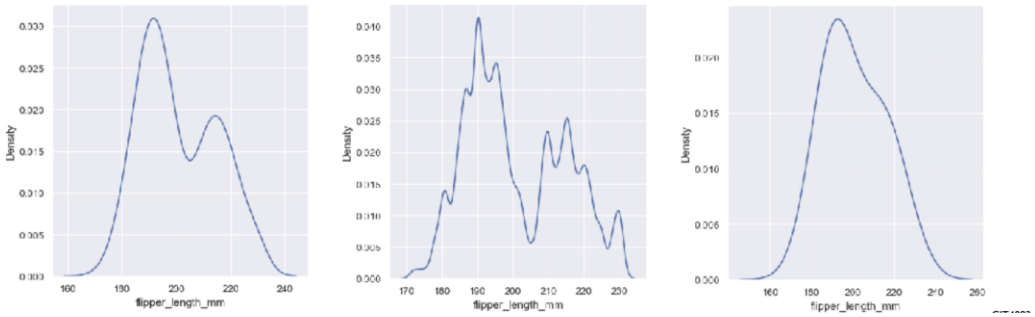
- The ability of the KDE to accurately represent the data depends on the choice of smoothing bandwidth
- An over-smoothed estimate might erase meaningful features, but an under-smoothed estimate can obscure the true shape within random noise
- The easiest way to check the robustness of the estimate is to adjust the default bandwidth — `bw_adjust`

```
sns.displot(penguins, x="flipper_length_mm", kind="kde", bw_adjust=.25)
```



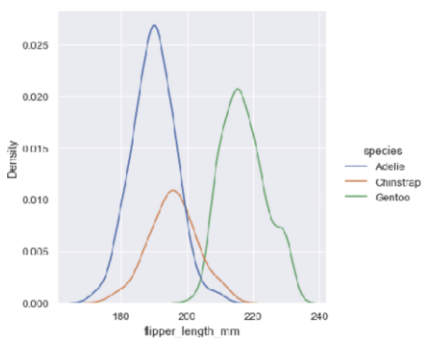
Choosing the bandwidth

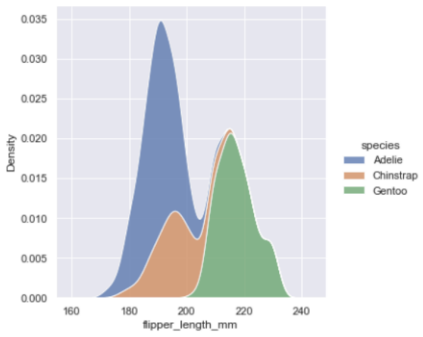
- Narrow bandwidth makes the bimodality much more apparent, but the curve is much less smooth
- Larger bandwidth obscures the bimodality almost completely



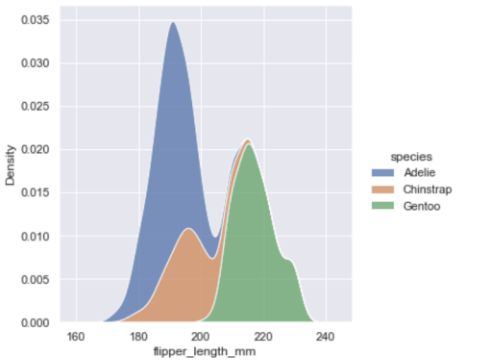
Layered Histogram vs Layered KDE

```
sns.displot(penguins, x="flipper_length_mm", hue="species", multiple="stack", kind="kde")
```

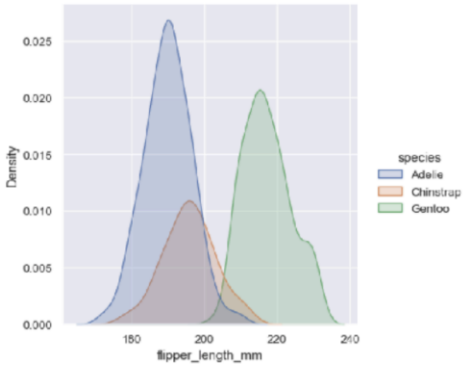




Easier interpretation with alpha




```
sns.displot(penguins, x="flipper_length_mm", hue="species", kind="kde", fill=True)
```



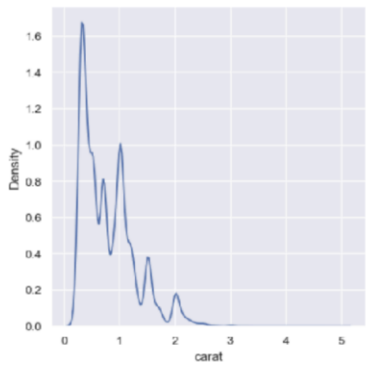
KDE pitfall (1)

- Important features of the data are easy to discern (central tendency, bimodality, skew), and they afford easy comparisons between subsets
- The logic of KDE assumes that the underlying distribution is smooth and unbounded
  - When the assumption is invalid under a certain scenario, KDE poorly represents the underlying data

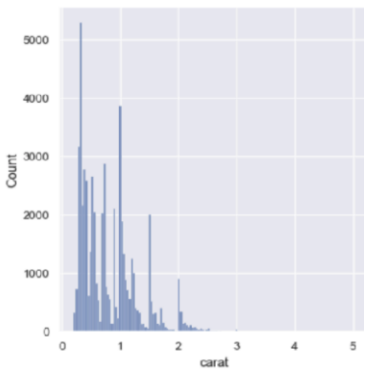
KDE pitfall (2)

- The KDE approach also fails for discrete data or when data are naturally continuous but specific values are over-represented
- When the data themselves is not smooth, it still show you a smooth curve

```
diamonds = sns.load_dataset("diamonds")
sns.displot(diamonds, x="carat", kind="kde")
```

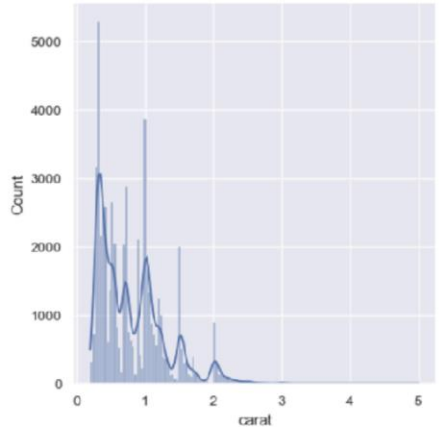



```
sns.displot(diamonds, x="carat")
```



Combining KDE and histogram

```
sns.displot(diamonds, x="carat", kde=True)
```

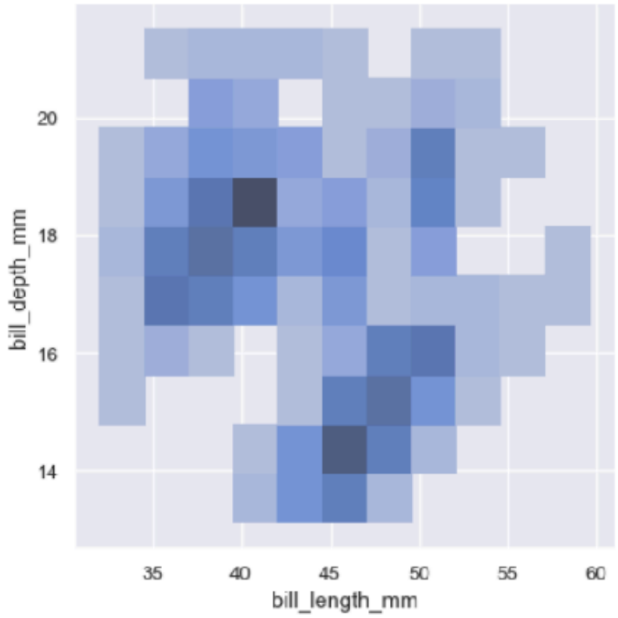


Visualizing bivariate distributions

- A bivariate histogram bins the data within rectangles that tile the plot and then shows the count of observations within each rectangle with the fill color
- A bivariate KDE plot smoothes the (x, y) observations with a 2D Gaussian with the default representation using contours of 2D density

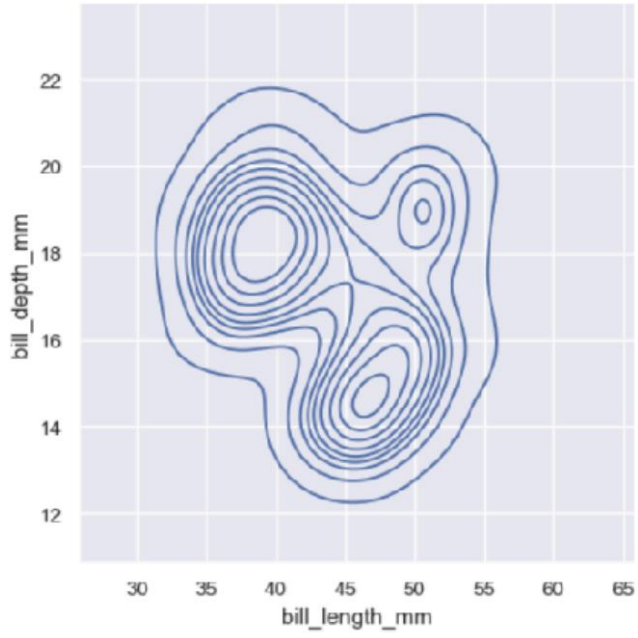
A bivariate histogram

```
sns.displot(penguins, x="bill_length_mm", y="bill_depth_mm")
```



A bivariate KDE plot

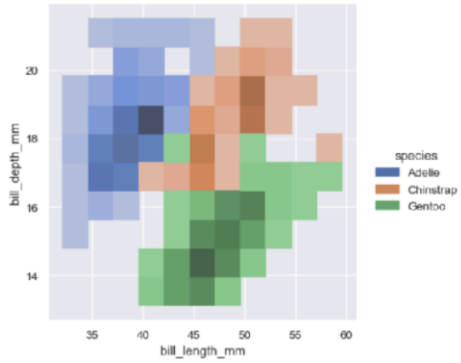
```
sns.displot(penguins, x="bill_length_mm", y="bill_depth_mm", kind="kde")
```



A bivariate plot with multiple categories — `hue`

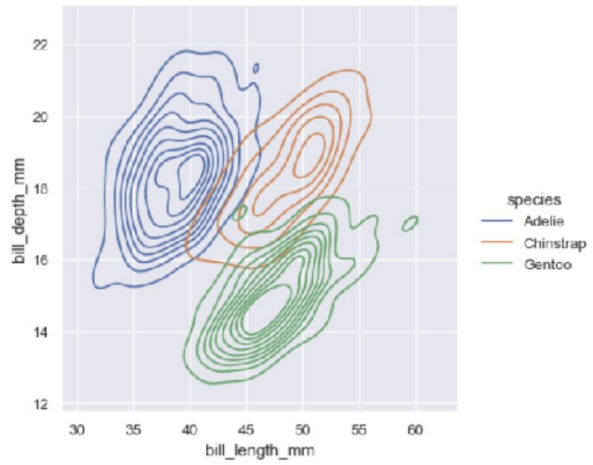
- Only work well if there is minimal overlap between the conditional distributions

```
sns.displot(penguins, x="bill_length_mm", y="bill_depth_mm", hue="species")
```



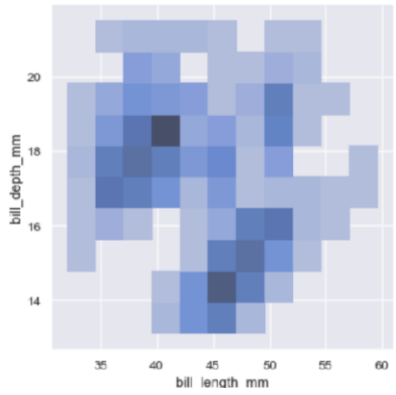
A bivariate plot with multiple categories — `hue`

```
sns.displot(penguins, x="bill_length_mm", y="bill_depth_mm", hue="species", kind="kde")
```

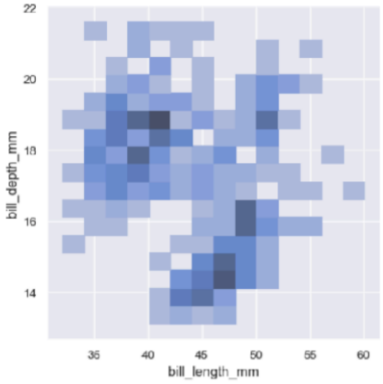


Choosing the binsize

```
sns.displot(penguins, x="bill_length_mm", y="bill_depth_mm")
```

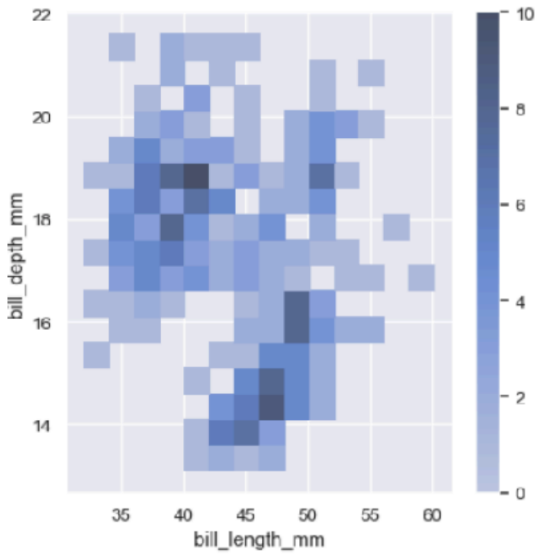



```
sns.displot(penguins, x="bill_length_mm", y="bill_depth_mm", binwidth=(2, .5))
```



Adding the color bar

```
sns.displot(penguins, x="bill_length_mm", y="bill_depth_mm", binwidth=(2, .5), cbar=True)
```



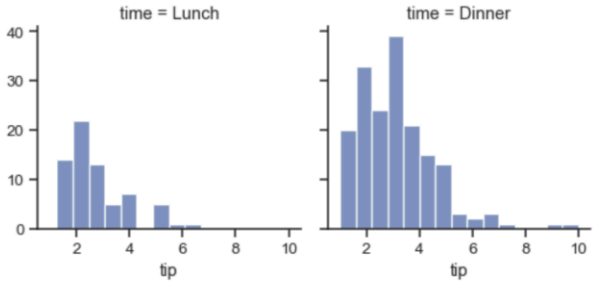
FacetGrid

- Useful when you want to visualize the distribution of a variable or the relationship between multiple variables separately within subsets of your dataset
- Can be drawn with up to three dimensions: row, col and hue
  - These variables should be categorical or discrete, and then the data at each level of the variable will be used for a facet along that axis

`FacetGrid.map()`

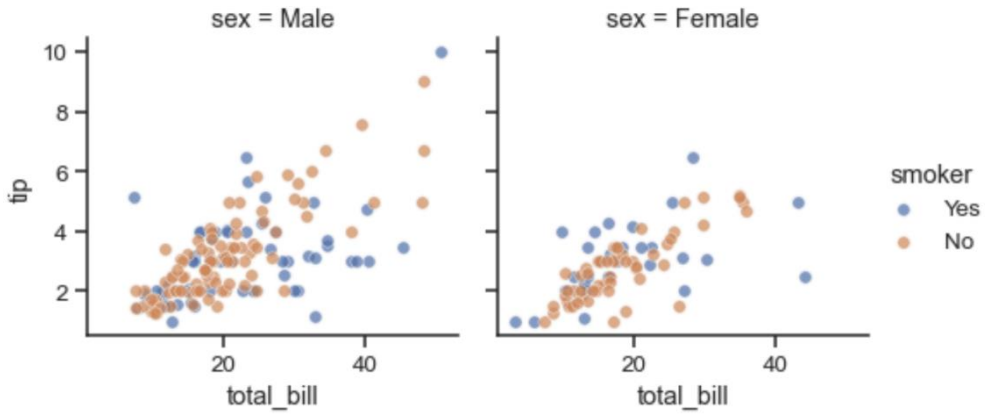
- Provide it with a plotting function and the name(s) of variable(s) in the dataframe to plot

```
g = sns.FacetGrid(tips, col="time")
g.map(sns.histplot, "tip")
```



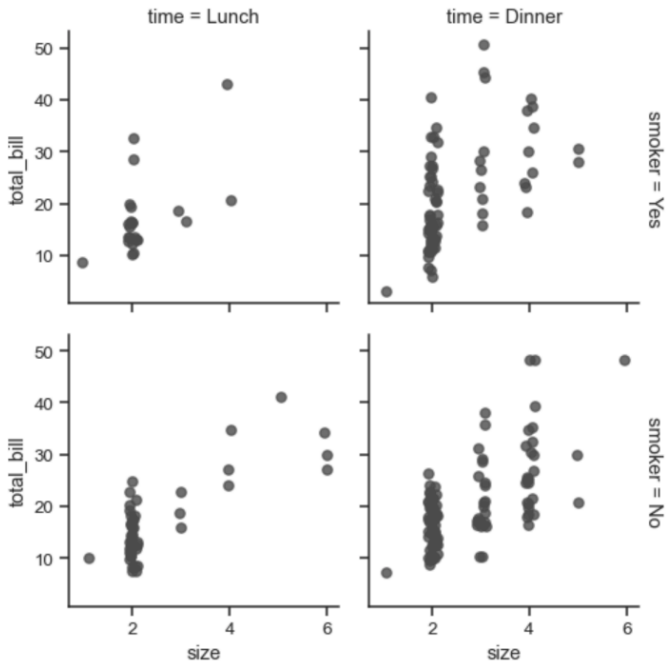
FacetGrid.map()

```
g = sns.FacetGrid(tips, col="sex", hue="smoker")
g.map(sns.scatterplot, "total_bill", "tip", alpha=.7)
g.add_legend()
```



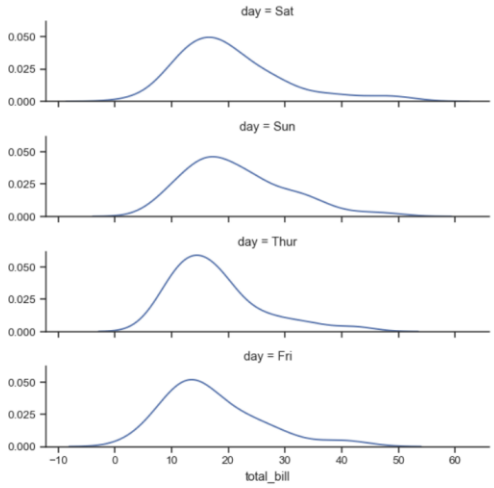
FacetGrid.map()


```
g = sns.FacetGrid(tips, row="smoker", col="time", margin_titles=True)
g.map(sns.regplot, "size", "total_bill", color=".3", fit_reg=False, x_jitter=.1)
```



FacetGrid.map()

```
ordered_days = tips.day.value_counts().index
g = sns.FacetGrid(tips, row="day", row_order=ordered_days,
                  height=1.7, aspect=4,)
g.map(sns.kdeplot, "total_bill")
```

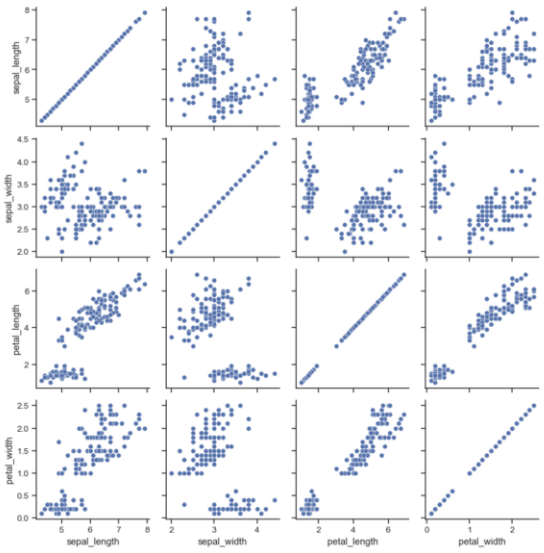


Plotting pairwise relationship

- Uses a “small-multiple” approach to visualize the univariate distribution of all variables in a dataset along with all of their pairwise relationship — `pairplot()`

pairplot()

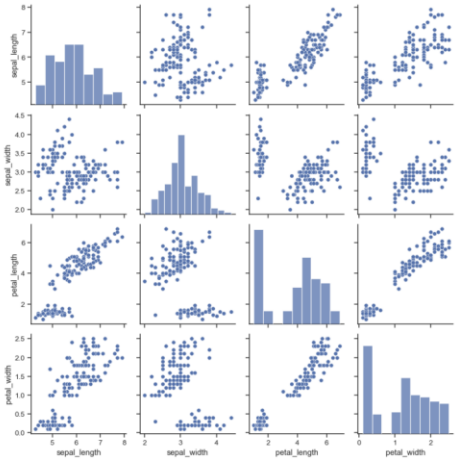
```
iris = sns.load_dataset("iris")
g = sns.PairGrid(iris)
g.map(sns.scatterplot)
```



Make use of the diagonal

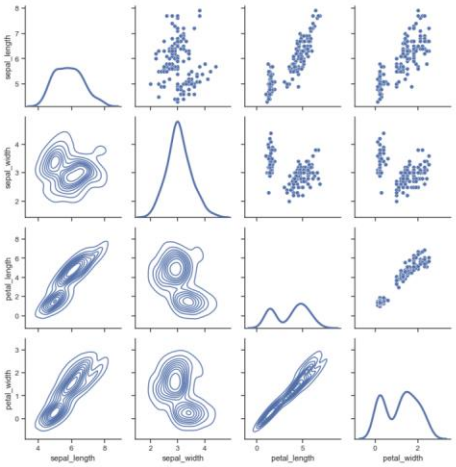
- Plot a different function on the diagonal to show the univariate distribution of the variable in each column

```
g = sns.PairGrid(iris)
g.map_diag(sns.histplot)
g.map_offdiag(sns.scatterplot)
```



Upper and lower triangles

```
g = sns.PairGrid(iris)
g.map_upper(sns.scatterplot)
g.map_lower(sns.kdeplot)
g.map_diag(sns.kdeplot, lw=3, legend=False)
```



Plotting with categorical data

- If one of the main variables in the data set is “categorical” (divided into discrete groups), it may be helpful to use a more specialized approach to visualization

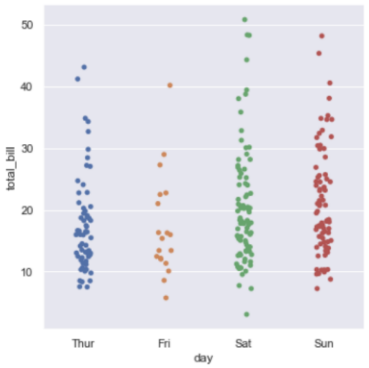
`catplot()`

- For drawing categorical plots using the following approaches
  - Categorical scatter plots (with **`kind="strip"`**, the default; `kind="swarm"`)
  - Categorical distribution plots (with `kind="box"`, `kind="violin"`, `kind="boxen"`)
  - Categorical estimate plots (with `kind="point"`, `kind="bar"`, `kind="count"`)

`catplot()`

- Default representation is a scatterplot
- All of the points belonging to one category would fall on the same position along the axis corresponding to the categorical variable

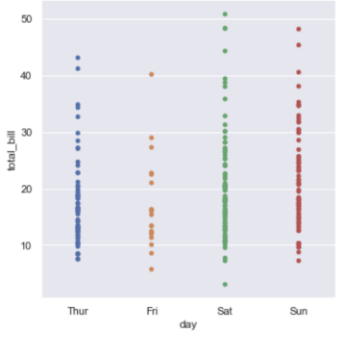
```
tips = sns.load_dataset("tips")
sns.catplot(x="day", y="total_bill", data=tips)
```



`jitter`

- The `jitter` parameter controls the magnitude of jitter
- Disable the jitter by set it to `False`

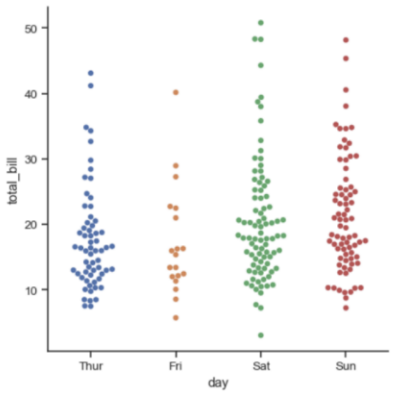
```
sns.catplot(x="day", y="total_bill", jitter=False, data=tips)
<seaborn.axisgrid.FacetGrid at 0x7fb6b50880a0>
```



kind="swarm"

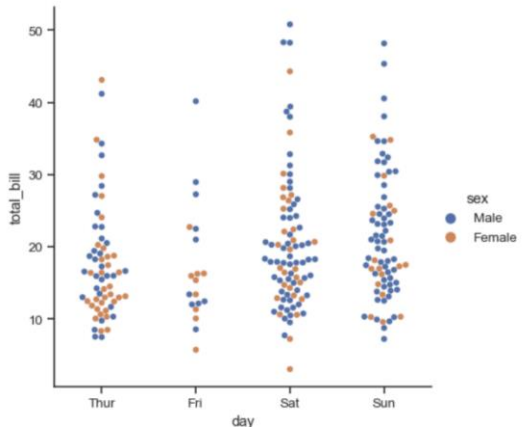
- Adjusts the points along the categorical axis using an algorithm that prevents them from overlapping
- Give a better representation of the distribution of observations,
- Only works well for relatively small datasets

```
sns.catplot(x="day", y="total_bill", kind="swarm", data=tips)
```



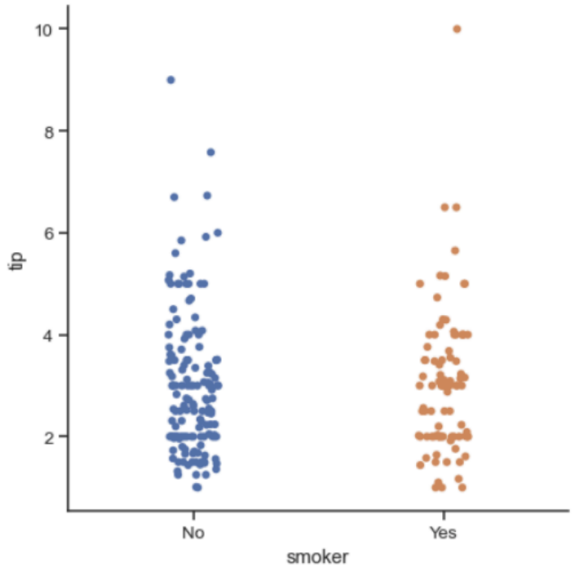
Adding dimension using 'hue'

```
sns.catplot(x="day", y="total_bill", hue="sex", kind="swarm", data=tips)
```



Setting default ordering

```
sns.catplot(x="smoker", y="tip", order=["No", "Yes"], data=tips)
```



Distributions of observations within categories

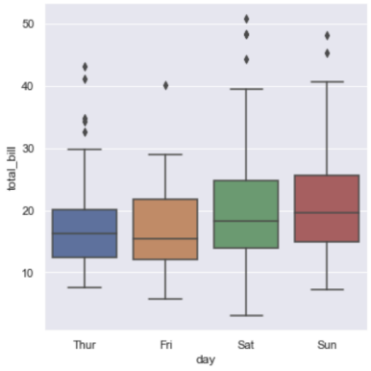
- As the size of the dataset grows, categorical scatter plots become limited in the information they can provide about the distribution of values within each category
- Can consider using `kind='box'` or `kind='violin'`

`kind="box"`

- Each category is represented by a box plot

```
sns.catplot(x="day", y="total_bill", kind="box", data=tips)
```

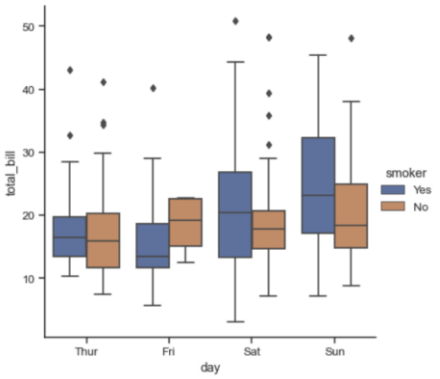
```
<seaborn.axisgrid.FacetGrid at 0x7fb6b7b950d0>
```



Adding one more variable with hue

- The box for each level of the semantic variable is moved along the categorical axis with no overlap
- The box can set to be overlap by setting the parameter `dodge` to `False`

```
sns.catplot(x="day", y="total_bill", hue="smoker", kind="box", data=tips)
```

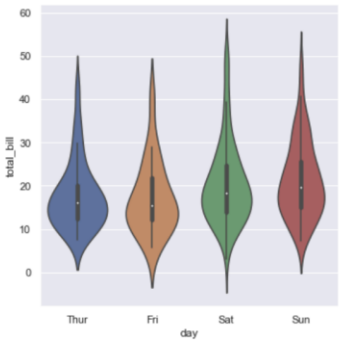


`kind="violin"`

- Each category is represented by a violin plot

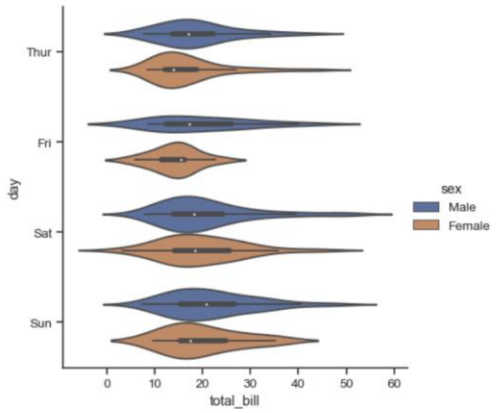
```
sns.catplot(x="day", y="total_bill", kind="violin", data=tips)
```

```
<seaborn.axisgrid.FacetGrid at 0x7fb6b8367a60>
```



Adding one more variable with hue

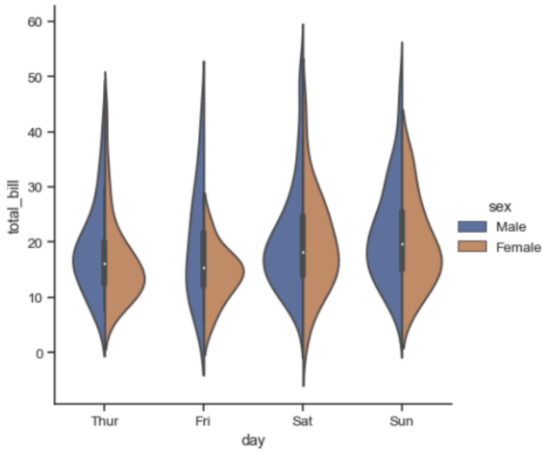
```
sns.catplot(x="total_bill", y="day", hue="sex",
            kind="violin", data=tips)
```



Split the violins

- Possible to “split” the violins when the hue parameter has only two levels, which can allow for a more efficient use of space

```
sns.catplot(x="day", y="total_bill", hue="sex",
            kind="violin", split=True, data=tips)
```



Data for generating heatmap

- A pivot table takes simple column-wise data as input, and groups the entries into a two-dimensional table that provides a multidimensional summarization of the data
  - Can use the function `pivot()` in `pandas` for converting the DataFrame into a pivot table
- A heatmap takes a 2D dataset that can be coerced into an ndarray. If a Pandas DataFrame is provided, the index/column information will be used to label the columns and rows

pivot()

- Reshape data (produce a “pivot” table) based on column values
- Uses unique values from specified *index* / *columns* to form axes of the resulting DataFrame
  - `DataFrame.pivot(index=None, columns=None, values=None)`

heatmap()

- Plot rectangular data as a color-encoded matrix

```
flights = sns.load_dataset('flights')
flights = flights.pivot("month", "year", "passengers")
f, ax = plt.subplots(figsize=(9, 6))
sns.heatmap(flights, annot=True, fmt="d", linewidth=0.5)
```

```
<AxesSubplot:xlabel='year', ylabel='month'>
```

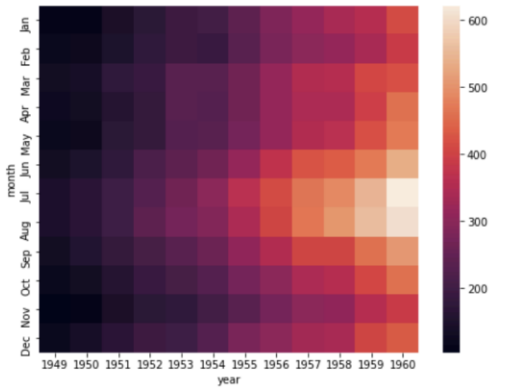
Parameters

- Some parameters such as `vmin`, `vmax`, `cmap`, `annot`, `fmt`, `linewidths`, `linecolor`, `cbar`, are useful for customising the heatmap

Controlling the figure size

- Control the figure size using `figure()` in `matplotlib`

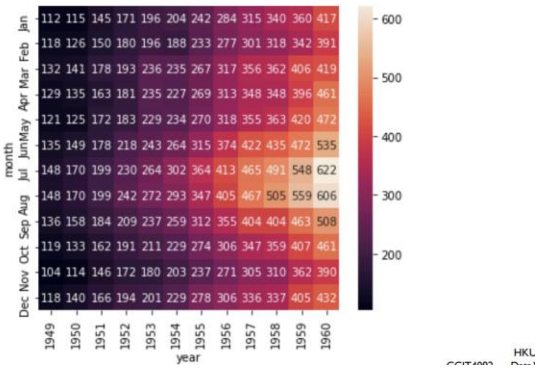
```
# Controlling the figure size
figure = plt.figure(figsize=(8,6))
ax = plt.subplot(1,1,1)
hm = sns.heatmap(ax=ax, data=flights)
plt.show()
```



Annotate

- Show the numbers on the heatmap with `annot` and `fmt`
- `fmt` is used to specify the format of the number

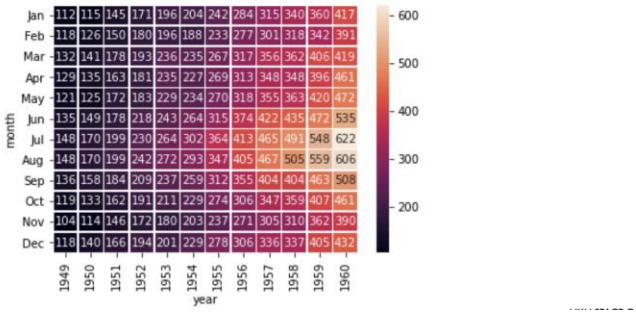
```
# With annotation
figure = plt.figure(figsize=(6,5))
ax = plt.subplot(1,1,1)
hm = sns.heatmap(data=flights, ax=ax, annot=True, fmt="d")
plt.show()
```



Add lines between each cell

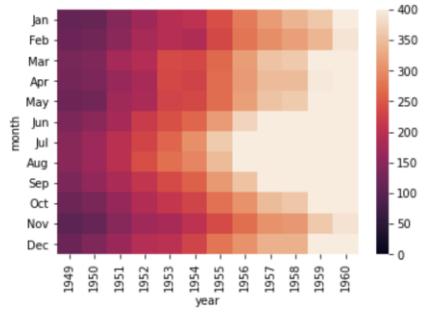
- Control the line between each cell using the parameter `linewidth`

```
# Add lines between each cell
figure = plt.figure(figsize=(6,4))
ax = plt.subplot(1,1,1)
hm = sns.heatmap(data=flights, annot=True, fmt="d", linewidth=0.5)
plt.show()
```

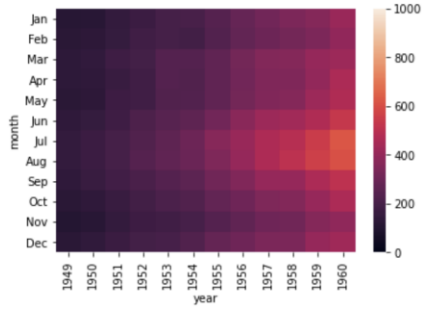


Adjusting the color range

```
# Adjusting the color range
figure = plt.figure(figsize=(6,4))
ax = plt.subplot(1,1,1)
hm = sns.heatmap(data=flights, vmin=0, vmax=400)
plt.show()
```

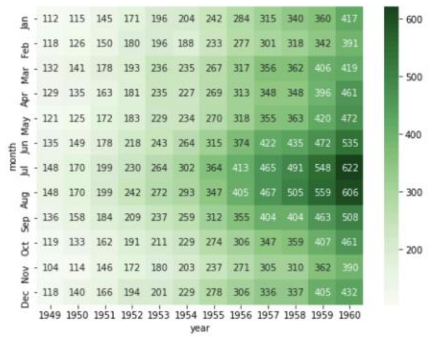



```
# Adjusting the color range
figure = plt.figure(figsize=(6,4))
ax = plt.subplot(1,1,1)
hm = sns.heatmap(data=flights, vmin=0, vmax=1000)
plt.show()
```

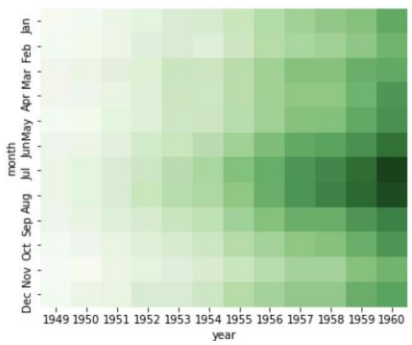


Changing the color map

```
# Changing the color map
# https://matplotlib.org/stable/gallery/color/colormap_reference.html
figure = plt.figure(figsize=(8,6))
ax = plt.subplot(1,1,1)
hm = sns.heatmap(data=flights, annot=True, fmt="d", cmap='Greens')
plt.show()
```

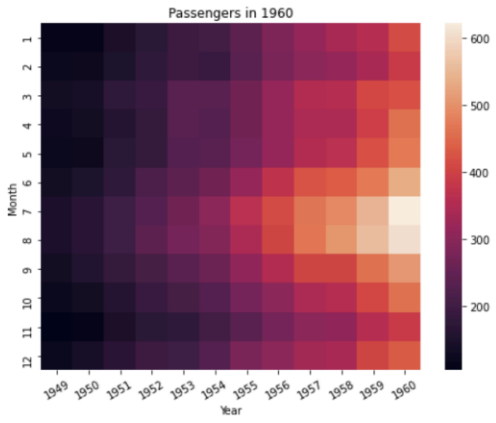



```
# Don't draw a colorbar
figure = plt.figure(figsize=(6,5))
ax = plt.subplot(1,1,1)
hm = sns.heatmap(data=flights, cmap='Greens', cbar=False)
plt.show()
```



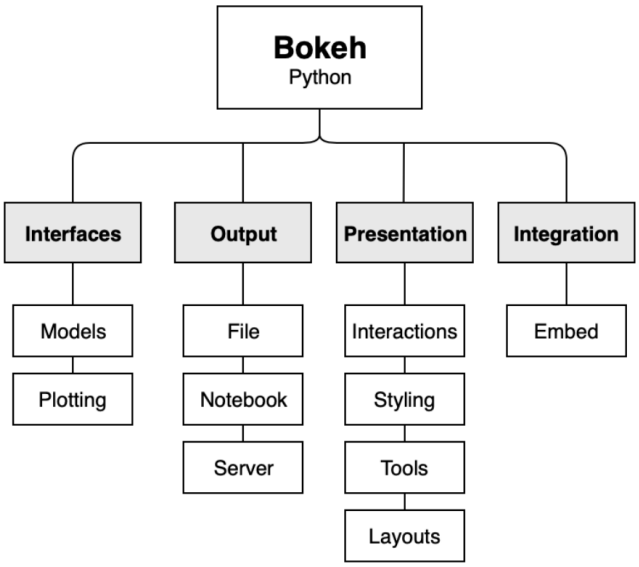
Customize the labels

```
# Customize the labels
figure = plt.figure(figsize=(8,6))
ax = plt.subplot(1,1,1)
hm = sns.heatmap(data=flights)
hm.set_xticklabels(hm.get_xticklabels(), rotation=30)
hm.set_yticklabels(labels=range(1,13), rotation=90)
hm.set_xlabel('Year')
hm.set_ylabel('Month')
plt.title('Passengers in 1960')
plt.show()
```



060-Bokeh

Concepts



Interfaces

- The interface-based approach provides different levels of complexity for users that either simply want to create some basic plots with very few customizable parameters or the users who want full control over their visualizations and want to customize every single element of their plots.
- This layered approach is divided into two levels:
  - **Plotting**: `bokeh.plotting`
  - **Models interface**: `bokeh.models`

`bokeh.plotting`

- A mid-level interface that has a comparable API to matplotlib
- The workflow is to create a figure and then enrich this figure with different glyphs that render data points in the figure
- Comparable to Matplotlib, the composition of sub-elements such as axes, grids, and the **Inspector** (they provide basic ways of exploring your data through zooming, panning, and hovering) is done without additional configuration

`bokeh.models`

- A low-level interface that is composed of two libraries: the JavaScript library called **BokehJS**, which gets used for displaying the charts in the browser and the Python library, which provides the developer interface

Glyph

- Glyphs are the basic visual building blocks of Bokeh plots
- API objects that draw vectorized graphics to represent data
  - Includes elements such as lines, rectangles, squares, wedges, or the circles of a scatter plot

Import relevant libraries

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

Tell Bokeh where to generate output

- Using `output_file()` to write to a HTML file, or `output_notebook()` for use in Jupyter notebooks

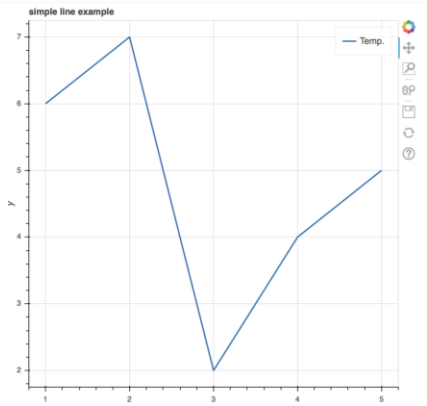
```
# make bokeh display figures inside the notebook
from bokeh.io import output_notebook

output_notebook()
```

BokehJS 2.2.3 successfully loaded.

```
from bokeh.io import output_file
```

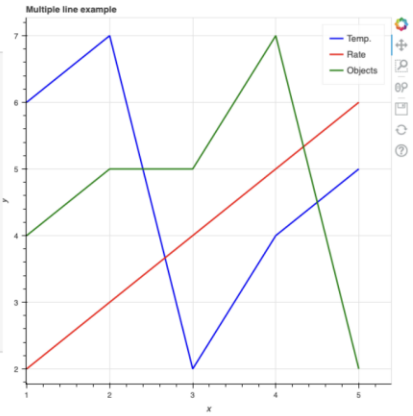
Example

```
# prepare some data
x = [1, 2, 3, 4, 5]
y = [6, 7, 2, 4, 5]

# output to static HTML file
output_file("lines.html")

# create a new plot with a title and axis labels
p = figure(title="simple line example",
           x_axis_label='x', y_axis_label='y')

# add a line renderer with legend and line thickness
p.line(x, y, legend_label="Temp.", line_width=2)

# show the results
show(p)
```



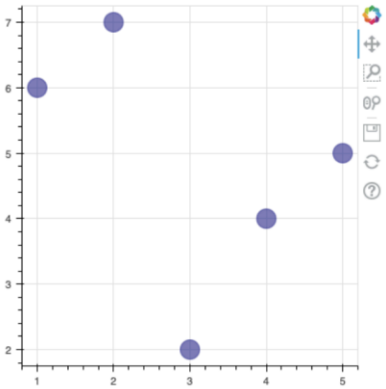
Graph with multiple lines

```
# multi-line plot

# prepare some data
x = [1, 2, 3, 4, 5]
y1 = [6, 7, 2, 4, 5]
y2 = [2, 3, 4, 5, 6]
y3 = [4, 5, 5, 7, 2]

# create a new plot with a title and axis labels
p = figure(title="Multiple line example", x_axis_label="x", y_axis_label="y")

# add multiple renderers
p.line(x, y1, legend_label="Temp.", line_color="blue", line_width=2)
p.line(x, y2, legend_label="Rate", line_color="red", line_width=2)
p.line(x, y3, legend_label="Objects", line_color="green", line_width=2)

# show the results
show(p)
```



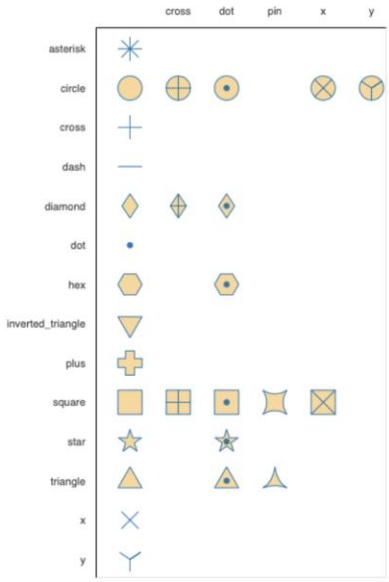
Scatter plots

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

# output to static HTML file
output_file("line.html")

p = figure(plot_width=400, plot_height=400)

# add a circle renderer with a size, color, and alpha
p.circle([1, 2, 3, 4, 5], [6, 7, 2, 4, 5], size=20, color="navy", alpha=0.5)

# show the results
show(p)
```



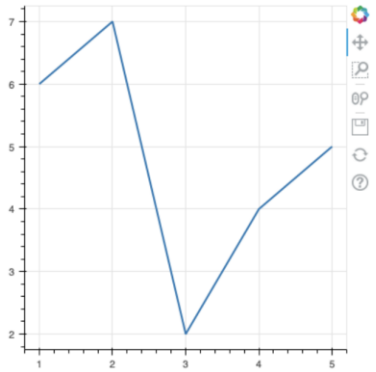
Scatter markers

- asterisk()
- circle()
- circle_cross()
- circle_dot()
- circle_x()
- circle_y()
- cross()
- dash()
- dot()
- diamond()
- diamond_cross()
- diamond_dot()
- hex()
- hex_dot()
- inverted_triangle()
- plus()
- square()
- square_cross()
- square_dot()
- square_pin()
- square_x()
- star()
- star_dot()
- triangle()
- triangle_dot()
- triangle_pin()
- x()
- y()



Line glyphs: Single lines

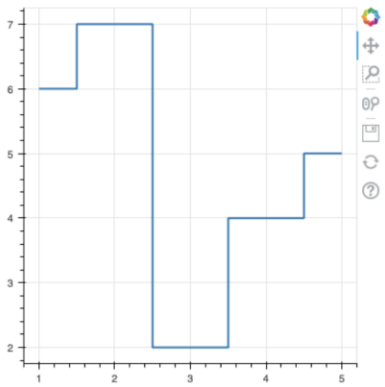
```
from bokeh.plotting import figure, output_file, show

output_file("line.html")

p = figure(plot_width=400, plot_height=400)

# add a line renderer
p.line([1, 2, 3, 4, 5], [6, 7, 2, 4, 5], line_width=2)

show(p)
```



Line glyphs: Step lines

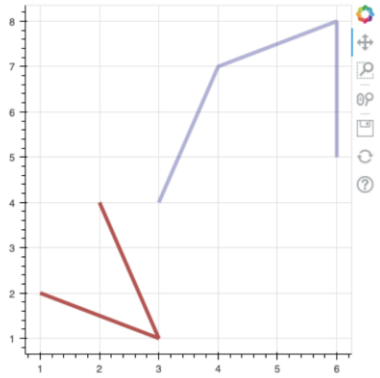
```
from bokeh.plotting import figure, output_file, show

output_file("line.html")

p = figure(plot_width=400, plot_height=400)

# add a steps renderer
p.step([1, 2, 3, 4, 5], [6, 7, 2, 4, 5], line_width=2, mode="center")

show(p)
```

- Adjust the mode parameter to draw step levels with the x-coordinates before, after, or in the middle of each step



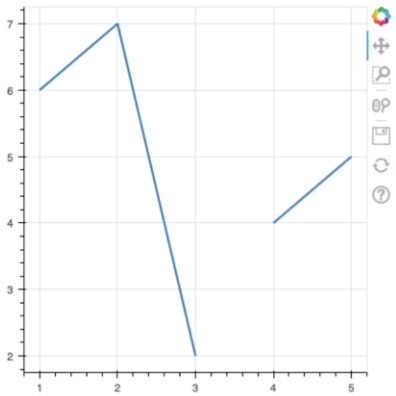
Line glyphs: Multiple lines

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

output_file("patch.html")

p = figure(plot_width=400, plot_height=400)

p.multi_line([[1, 3, 2], [3, 4, 6, 6]], [[2, 1, 4], [4, 7, 8, 5]],
             color=["firebrick", "navy"], alpha=[0.8, 0.3], line_width=4)

show(p)
```



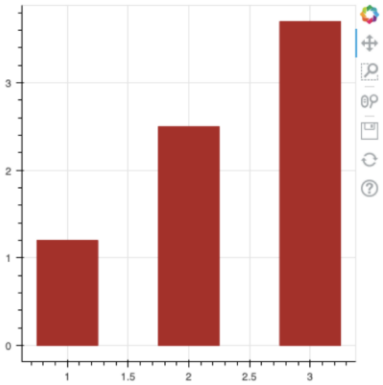
Handling missing points

- If there are NaN values, Bokeh will produces disjointed lines with gaps



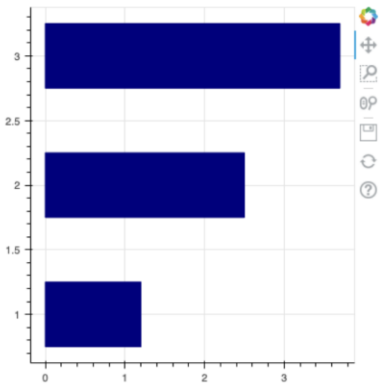
Bars: `vbar()`

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

output_file('vbar.html')

p = figure(plot_width=400, plot_height=400)
p.vbar(x=[1, 2, 3], width=0.5, bottom=0,
       top=[1.2, 2.5, 3.7], color="firebrick")

show(p)
```



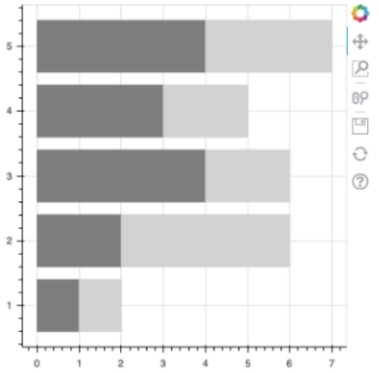
Bars: `hbar()`

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

output_file('hbar.html')

p = figure(plot_width=400, plot_height=400)
p.hbar(y=[1, 2, 3], height=0.5, left=0,
       right=[1.2, 2.5, 3.7], color="navy")

show(p)
```



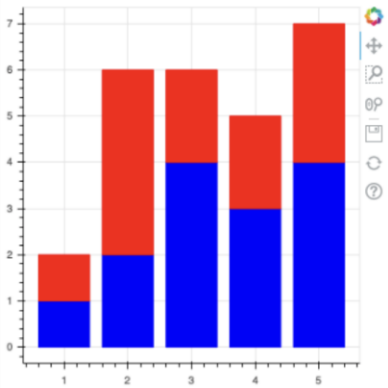
Stacked bars: `hbar_stack()`

```
from bokeh.models import ColumnDataSource

source = ColumnDataSource(data=dict(
    y=[1, 2, 3, 4, 5],
    x1=[1, 2, 4, 3, 4],
    x2=[1, 4, 2, 2, 3],
))
p = figure(plot_width=400, plot_height=400)

p.hbar_stack(['x1', 'x2'], y='y', height=0.8, color=("grey", "lightgrey"), source=source)

show(p)
```



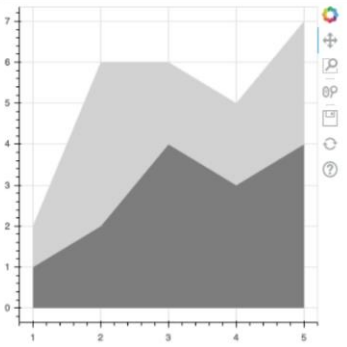
Stacked bars: `vbar_stack()`

```
from bokeh.models import ColumnDataSource

source = ColumnDataSource(data=dict(
    x=[1, 2, 3, 4, 5],
    y1=[1, 2, 4, 3, 4],
    y2=[1, 4, 2, 2, 3],
))
p = figure(plot_width=400, plot_height=400)

p.vbar_stack(['y1', 'y2'], x='x', width=0.8, color=("blue", "red"), source=source)

show(p)
```

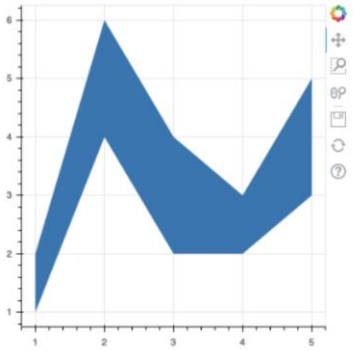


Directed areas — Filled regions

```
# Stacked areas
source = ColumnDataSource(data=dict(
    x=[1, 2, 3, 4, 5],
    y1=[1, 2, 4, 3, 4],
    y2=[1, 4, 2, 2, 3],
))
p = figure(plot_width=400, plot_height=400)

p.varea_stack(['y1', 'y2'], x='x', color=("grey", "lightgrey"), source=source)

show(p)
```

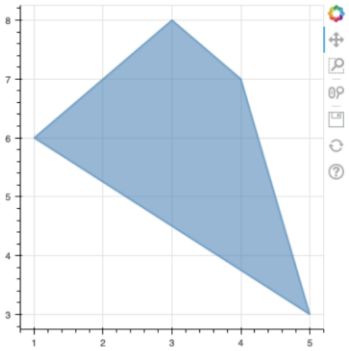


```
# Single Areas
p = figure(plot_width=400, plot_height=400)

p.varea(x=[1, 2, 3, 4, 5],
        y1=[2, 6, 4, 3, 5],
        y2=[1, 4, 2, 2, 3])

show(p)
```

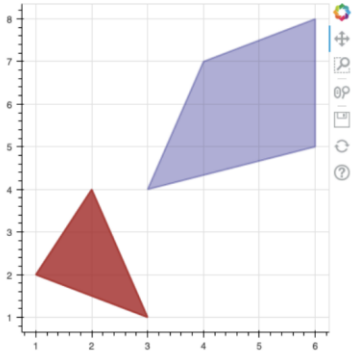


Patches

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

output_file("patch.html")

p = figure(plot_width=400, plot_height=400)

# add a patch renderer with an alpha and line width
p.patch([1, 2, 3, 4, 5], [6, 7, 8, 7, 3], alpha=0.5, line_width=2)

show(p)
```

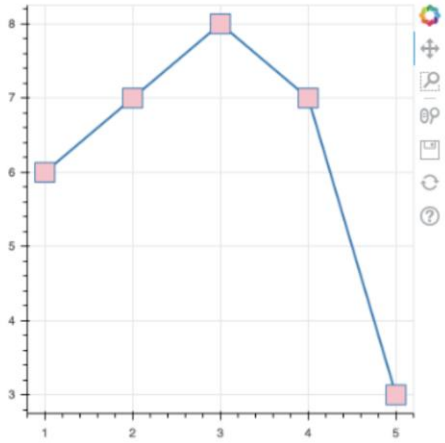


```
from bokeh.plotting import figure, output_file, show

output_file("patch.html")

p = figure(plot_width=400, plot_height=400)

p.patches([[1, 3, 2], [3, 4, 6, 6]], [[2, 1, 4], [4, 7, 8, 5]],
          color=["firebrick", "navy"], alpha=[0.8, 0.3], line_width=2)

show(p)
```



Combining multiple glyphs

```
# Combining multiple glyphs

x = [1, 2, 3, 4, 5]
y = [6, 7, 8, 7, 3]

output_file("multiple.html")

p = figure(plot_width=400, plot_height=400)

# add both a line and circles on the same plot
p.line(x, y, line_width=2)
p.square(x, y, fill_color="pink", size=18)

show(p)
```



ColumnDataSource

- ColumnDataSource is an object that can be used for the renders
- Can be created from python `Dictionary`, or Pandas `DataFrame`

```
# ColumnDataSource using python Dictionary

data = {'x_values': [1, 2, 3, 4, 5],
        'y_values': [6, 7, 2, 3, 6]}

source = ColumnDataSource(data=data)

p = figure (plot_width=400, plot_height=400,
            title = "data with ColumnDataSource", x_axis_label = 'x', y_axis_label = 'y')
p.circle_dot(x='x_values', y='y_values', source=source, size=20)
show(p)
```

Categories

- **Gestures**
  - Pan/Drag tools
  - Click/Tap tools
  - Scroll/Pinch tools
- **Actions**
  - e.g., ResetTool
- **Inspectors**
  - e.g., HoverTool
- **Edit tools**
  - Can add, delete or modify glyphs on a plot

070-ChoroplethMap

Choropleth map

- A map of region (coloured polygons) with different divisions coloured to indicate the value of a specific feature in that division
- The division can be a country, state, district, or any other well-documented area
- Represent spatial variations of a quantity

Loading geojson

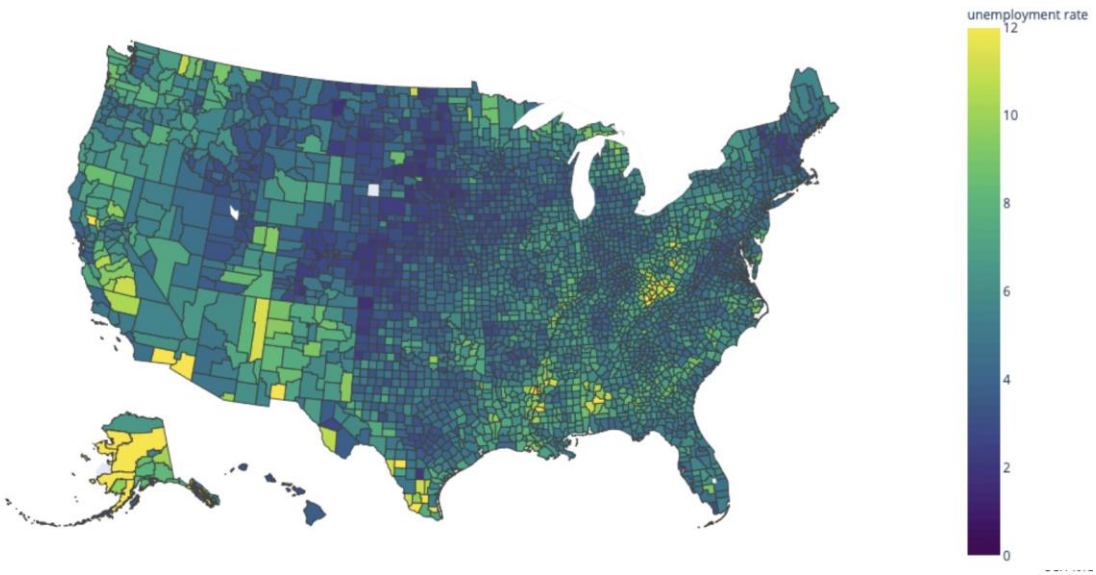
```
from urllib.request import urlopen
import json
with urlopen('https://raw.githubusercontent.com/plotly/datasets/master/geojson-counties-fips.json') as response:
    counties = json.load(response)
```


```
{'type': 'FeatureCollection',
 'features': [{'type': 'Feature',
   'properties': {'GEO_ID': '0500000US01001',
    'STATE': '01',
    'COUNTY': '001',
    'NAME': 'Autauga',
    'LSAD': 'County',
    'CENSUSAREA': 594.436},
   'geometry': {'type': 'Polygon',
    'coordinates': [[[-86.496774, 32.344437],
      [-86.717897, 32.402814],
      [-86.814912, 32.340803],
      [-86.890581, 32.502974],
      [-86.917595, 32.664169],
      [-86.71339, 32.661732],
      [-86.714219, 32.705694],
      [-86.413116, 32.707386],
      [-86.411172, 32.409937],
      [-86.496774, 32.344437]]]},
```

Plotting (1)

```
import pandas as pd
df = pd.read_csv("https://raw.githubusercontent.com/plotly/datasets/master/fips-unemp-16.csv",
                 dtype={"fips": str})

import plotly.express as px

fig = px.choropleth(df, geojson=counties, locations='fips', color='unemp',
                           color_continuous_scale="Viridis",
                           range_color=(0, 12),
                           scope="usa",
                           labels={'unemp':'unemployment rate'}
                          )
fig.update_layout(margin={"r":0,"t":0,"l":0,"b":0})
fig.show()
```

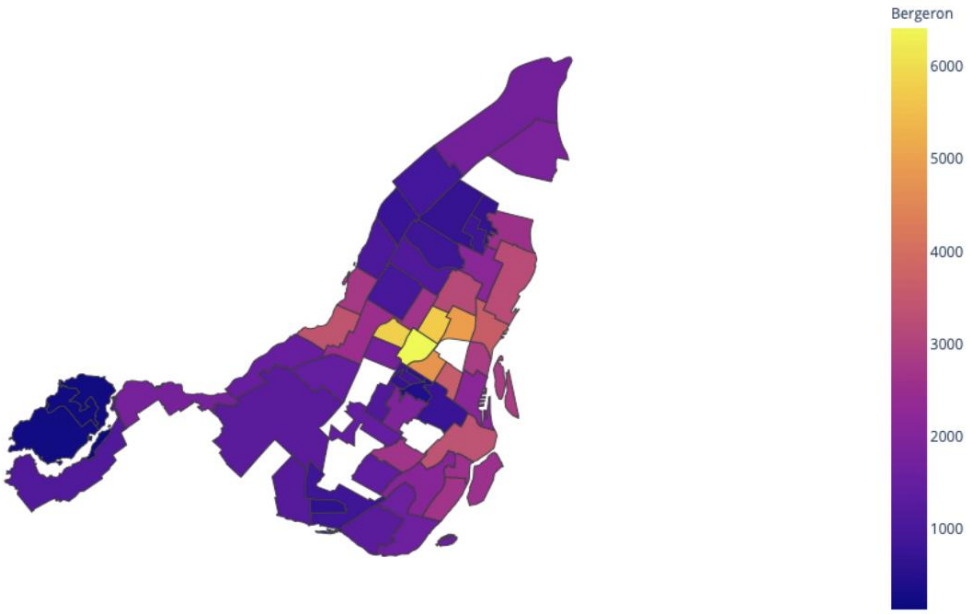
Plotting (1): Result



Plotting (2)

```
import plotly.express as px

df = px.data.election()
geojson = px.data.election_geojson()

fig = px.choropleth(df, geojson=geojson, color="Bergeron",
                    locations="district", featureidkey="properties.district",
                    projection="mercator"
                   )
fig.update_geos(fitbounds="locations", visible=False)
fig.update_layout(margin={"r":0,"t":0,"l":0,"b":0})
fig.show()
```

Plotting (2): Result



Creating a graph

- Create an empty graph with no nodes and no edges.

```
>>> import networkx as nx
>>> G=nx.Graph()
```

Adding a node

- Add a node at a time

```
>>> G.add_node(1)
```

- Add a list of nodes

```
>>> G.add_nodes_from([2,3])
```

Adding an edge

- Add an edge at a time

```
>>> G.add_edge(1,2)
>>> e=(2,3)
>>> G.add_edge(*e) # unpack edge tuple*
```

- Add a list of nodes

```
>>> G.add_edges_from([(1,2),(1,3)])
```

Remove

- `remove_node()`
- `remove_nodes_from()`
- `remove_edge()`
- `remove_edges_from()`

Some useful functions

- Clear all nodes and edges

```
>>> G.clear()
```

- Get the number of nodes

```
>>> G.number_of_nodes()
```

- Get the number of edges

```
>>> G.number_of_edges()
```

Some useful functions (1)

- Examine all the nodes

```
>>> G.nodes()
```

- Examine all the edges

```
>>> G.edges()
```

- Examine the neighbours

```
>>> G.neighbors(1)
```

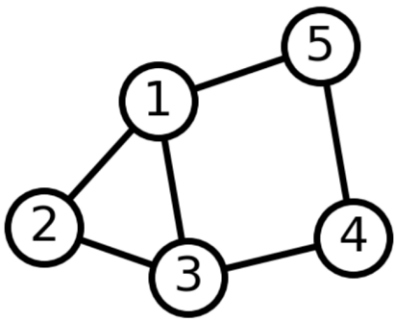
To draw the graph

- To draw the graph, it is necessary to use the matplotlib's plot interface

```
>>> nx.draw(G)
>>> plt.show()
```

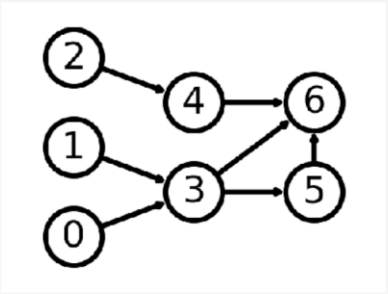
Example: Simple graph

```
# https://networkx.org/documentation/stable/auto_examples/basic/plot_simple_graph.html

G = nx.Graph()
G.add_edge(1, 2)
G.add_edge(1, 3)
G.add_edge(1, 5)
G.add_edge(2, 3)
G.add_edge(3, 4)
G.add_edge(4, 5)

options = {
    "font_size": 36,
    "node_size": 3000,
    "node_color": "white",
    "edgecolors": "black",
    "linewidths": 5,
    "width": 5,
}
nx.draw_networkx(G, **options)

ax = plt.gca()
ax.margins(0.20)
plt.axis("off")
plt.show()
```



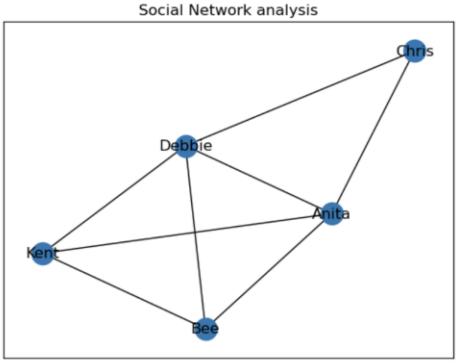
Example: Directed graph

```
G = nx.DiGraph([(0, 3), (1, 3), (2, 4), (3, 5), (3, 6), (4, 6), (5, 6)])

# group nodes by column
left_nodes = [0, 1, 2]
middle_nodes = [3, 4]
right_nodes = [5, 6]

# set the position according to column (x-coord)
pos = {n: (0, i) for i, n in enumerate(left_nodes)}
pos.update({n: (1, i + 0.5) for i, n in enumerate(middle_nodes)})
pos.update({n: (2, i + 0.5) for i, n in enumerate(right_nodes)})

nx.draw_networkx(G, pos, **options)

# Set margins for the axes so that nodes aren't clipped
ax = plt.gca()
ax.margins(0.20)
plt.axis("off")
plt.show()
```



Analysis of social network: Example

```
import networkx as nx
G_symmetric = nx.Graph()
G_symmetric.add_edge('Anita','Debbie')
G_symmetric.add_edge('Anita','Kent')
G_symmetric.add_edge('Anita','Chris')
G_symmetric.add_edge('Anita','Bee')
G_symmetric.add_edge('Debbie','Kent')
G_symmetric.add_edge('Debbie','Chris')
G_symmetric.add_edge('Debbie','Bee')
G_symmetric.add_edge('Bee','Kent')

nx.draw_networkx(G_symmetric)
plt.title("Social Network analysis")
plt.show()
```



Stochastic block models

- For a network partition
- Characterized by a scalar *n,* which represents the number of groups or the number of clusters and a matrix that shows the nodes and their connections

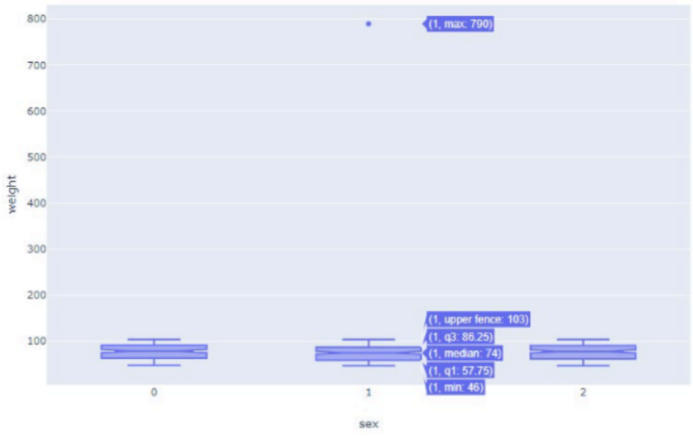
090-Common pitfalls in visualization

Outliners

- Data containing inaccurate values or instances that are significantly different from the rest of the data in a dataset
  - *Genuine* — they seem incorrect but are actually not
  - Mistakes that are made while collecting or storing the data

Outliners: Example

	age	weight	sex
0	29	88	2
1	45	96	1
2	35	91	0
3	37	790	1
4	27	62	0



Dealing with outliners — Deletion

- Remove those instances / features from the dataset
- Not always a good idea

Dealing with outliners — Imputation

- Impute the outliers with the mean, median or mode of the column
- Linear interpolation
- Linear regression model

Dealing with outliners — Transformation

- Transforming the outliners by building up the column of data wherein the outliner lies

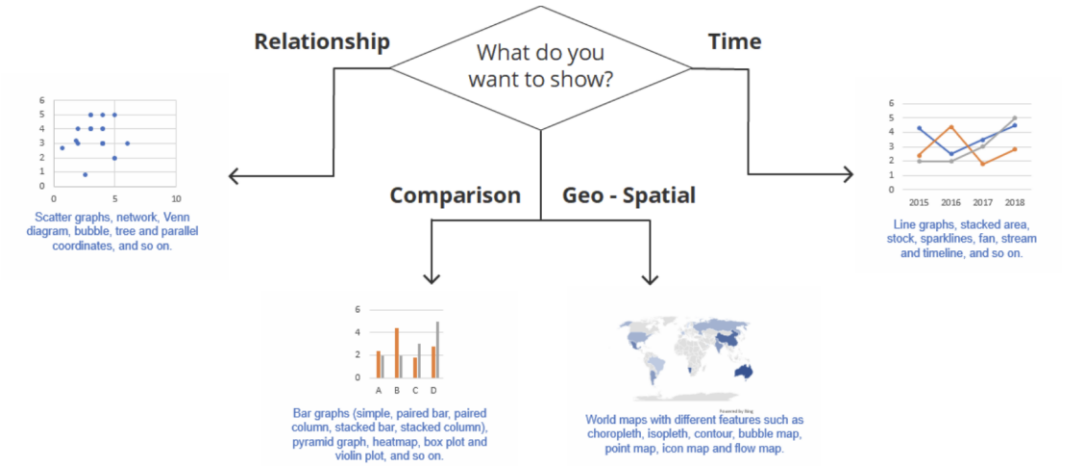
Missing data

- Missing data is as its name states – values that are blank (NaN, -, 0 when they shouldn’t be 0, and so on)
- Missing values in visualizations may display a trend that doesn't actually exist or fail to portray a relationship between two variables that, in reality, is significant

Duplicate instances and/or features

- Unnecessary elements in the dataset and if they are not removed, they can impact the trends and insights that are displayed by a visualization

Choosing visualization



Relationship

- Used when you want to show a link between two or more variables
- Examples: scatter plots, bubble plots, network graphs

Comparison

- Use when you want to show the differences or similarities between two or more variables
- Examples: bar chart, heat maps, box plots, violin plots

Geo-spatial

- Specific to data that is geographical in nature
- Location is a feature that must exist in the data
- Example: Choropleth maps

Time

- Temporal data
- When data consists of dates and/or times, these visualizations are used to track the necessary changes
- Examples: Line graphs, stacked area charts, timeline charts

Common pitfalls

- Visualizing too much information
- Inconsistent scales
- Mislabeling elements

Visualizing too much information

- Too much information basically means incorporating more than four or five features in your visualization, thereby introducing more than 5 colors and having too many words
- Convey too much information results in them becoming too complicated

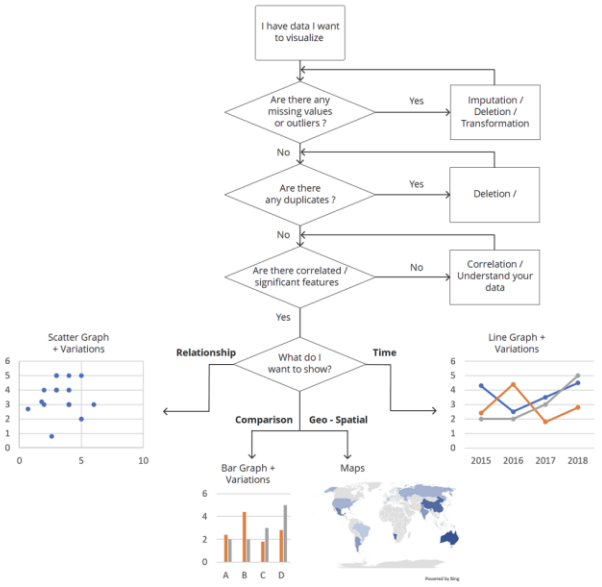
Inconsistent scales

- Each feature has its own range within which all its data falls
  - Numerical — Fall within this range
  - Categorical feature — A discrete set of classes
- Each feature has its own scale
  - Not considering the scale of each feature often leads to confusing visualizations that show trends where there are none
  - Inconsistent scales also often force relationships that do not exist
  - Mislead viewers

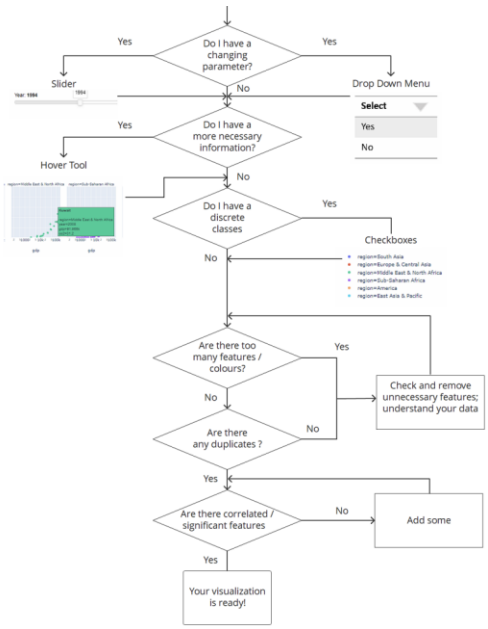
Mislabeling elements

- Labels are often overlooked and considered as trivial elements of a visualization
- Visualizations without labels become very confusing as the viewer doesn't know what they're seeing

Visualization process



Visualization process



Lab Answers:

CCIT4092 Data Visualization

03 Lab matplotlib (Suggested Answer)

Your Name:

Your Student ID:

Given the file 'Table210-06101_en-UnemploymentRate.xlsx', visualize the statistics according to the question specification using `matplotlib`.

[16]:
```
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_excel('Table210-06101_en-UnemploymentRate.xlsx')
display(df)
```

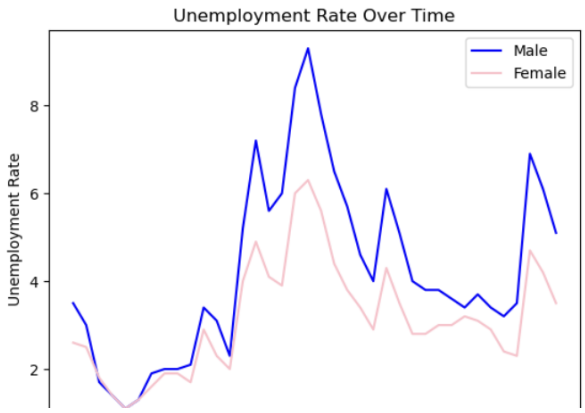
	Year	Male	Female	Both sexes
0	1985	3.5	2.6	3.2
1	1986	3.0	2.5	2.8
2	1987	1.7	1.8	1.7
3	1988	1.4	1.4	1.4
4	1989	1.1	1.1	1.1
5	1990	1.3	1.3	1.3
6	1991	1.9	1.6	1.8
7	1992	2.0	1.9	2.0
8	1993	2.0	1.9	2.0

Task 1

visualizes the unemployment rate over time using a line graph, with each gender represented by a separate line according to the following specification:

- x-axis: year
- y-axis: unemployment rate
- title: Unemployment Rate Over Time
- x-ticks: 3 years per tick
- x-ticks labels: rotated at an angle of 45 degrees
- legend: yes
- male: blue color
- female: pink color

```
### Your code here

plt.plot(df['Year'], df['Male'], label='Male', color='blue')
plt.plot(df['Year'], df['Female'], label='Female', color='pink')
plt.xlabel('Year')
plt.ylabel('Unemployment Rate')
plt.title('Unemployment Rate Over Time')
plt.legend()

num_years = len(df['Year'])
plt.xticks(range(0, num_years, 3), rotation=45)

plt.show()
```

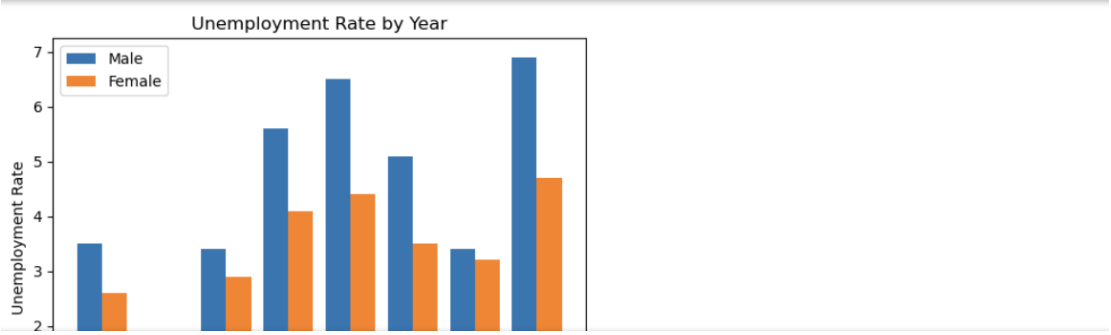


Task 2

visualizes the unemployment rate over time using a bar chart, with each gender represented by a separate bar according to the following specification:

- x-axis: year
- y-axis: unemployment rate
- title: Unemployment Rate Over Time
- x-ticks: 5 years per tick
- x-ticks labels: rotated at an angle of 45 degrees
- legend: yes
- male: blue color
- female: pink color
- bar width: 2

```
import numpy as np
import matplotlib.pyplot as plt

bar_width = 2
index = np.arange(0, len(df['Year']), 5)

plt.bar(index, df['Male'][::5], bar_width, label='Male')
plt.bar(index + bar_width, df['Female'][::5], bar_width, label='Female')

plt.xlabel('Year')
plt.ylabel('Unemployment Rate')
plt.title('Unemployment Rate by Year')
plt.legend()

plt.xticks(index + bar_width / 2, df['Year'][::5], rotation=45)

plt.show()
```

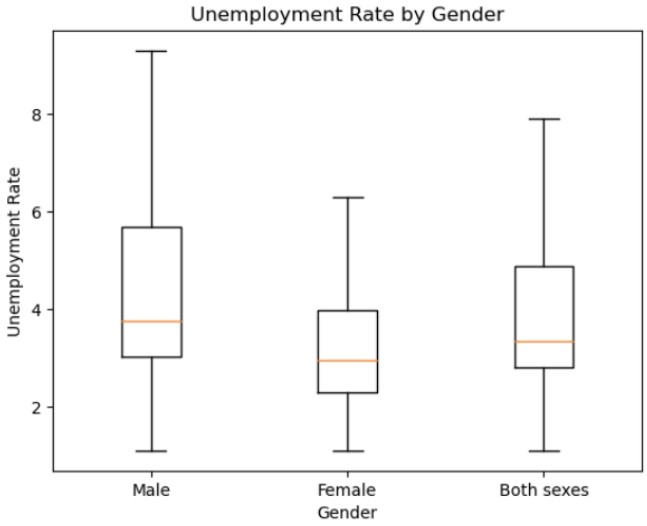


Task 3

visualizes display the distribution of the unemployment rate for different genders using a box plot according to the following specification:

- The data for males, females, and both sexes should be included in the box plot.
- x-axis: gender
- y-axis: unemployment rate
- title: Unemployment Rate by Gender

```
import matplotlib.pyplot as plt

plt.boxplot([df['Male'], df['Female'], df['Both sexes']])
plt.xlabel('Gender')
plt.ylabel('Unemployment Rate')
plt.title('Unemployment Rate by Gender')
plt.xticks([1, 2, 3], ['Male', 'Female', 'Both sexes'])
plt.show()
```



CCIT4092 Data Visualization

04 Lab seaborn (Suggested Answer)

Your Name:

Your Student ID:

Task 1

You are required to show distribution plot using the 'mpg' data.

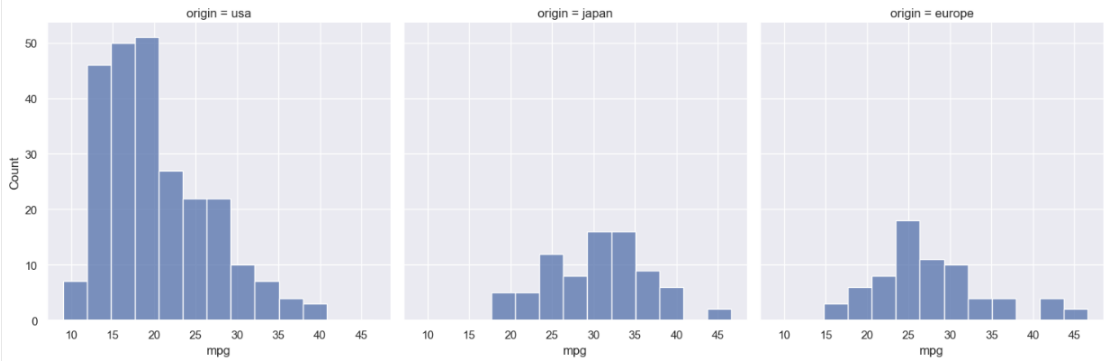
1. Load the default dataset 'mpg' from seaborn.
2. Visualize the distribution of 'mpg' using three separate distribution graph according to the 'origin'.
3. Visualize the distribution of 'mpg' of different origin using a single stack graph.
4. Repeat step (3) with a stacked KDE plot.
5. Repeat step (4) by change the color palette to 'yellow', 'blue' and 'magenta'.
6. Create a bivariate histogram using the 'acceleration' and 'mpg'.
7. Repeat step (6) with an additional variable 'origin' using 'hue'.

1]:
```
import seaborn as sns

data = sns.load_dataset("mpg", data_home='./data/')
display(data)

sns.set_theme(style="darkgrid")
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	name
0	18.0	8	307.0	130.0	3504	12.0	70	usa	chevrolet chevelle malibu
1	15.0	8	350.0	165.0	3693	11.5	70	usa	buick skylark 320
2	18.0	8	318.0	150.0	3436	11.0	70	usa	plymouth satellite
3	16.0	8	304.0	150.0	3433	12.0	70	usa	amc rebel sst
4	17.0	8	302.0	140.0	3449	10.5	70	usa	ford torino
...	...	...	...	...	...	...	...	...	...
393	27.0	4	140.0	86.0	2790	15.6	82	usa	ford mustang gl
394	44.0	4	97.0	52.0	2130	24.6	82	europe	vw pickup
395	32.0	4	135.0	84.0	2295	11.6	82	usa	dodge rampage

```
# Visualize the distribution of 'mpg' using three separate distribution graph according to the 'origin'.
sns.displot(data=data, x='mpg', col='origin')
```

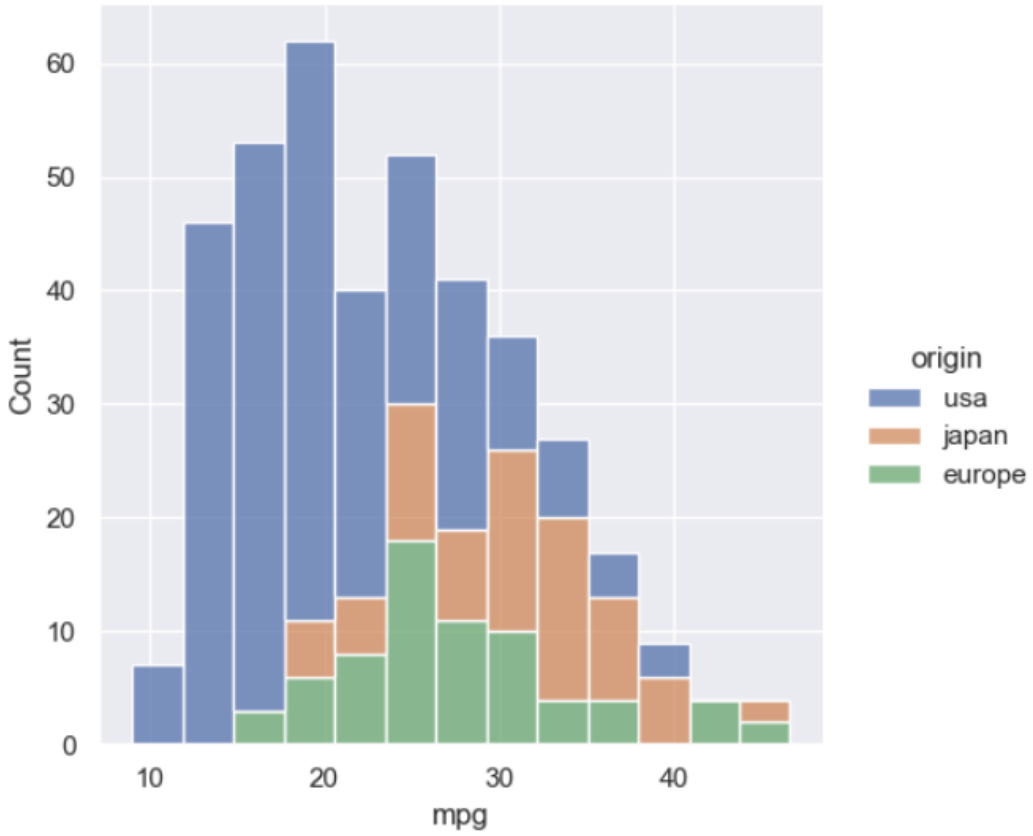
```
D:\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed in a future version. Conv
ert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
<seaborn.axisgrid.FacetGrid at 0x1c9329dff90>
```



```
# Visualize the distribution of 'mpg' of different origin using a single stack graph.
sns.displot(data, x="mpg", hue="origin", multiple="stack")
```

```
D:\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na c
ert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

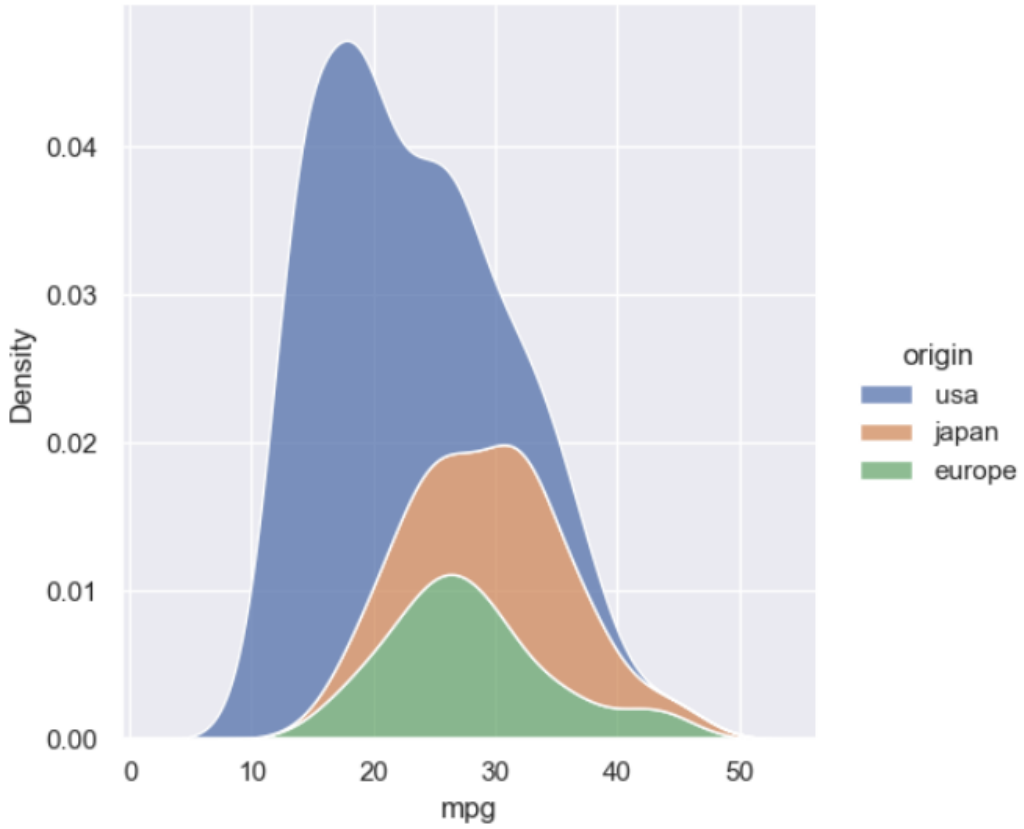
```
<seaborn.axisgrid.FacetGrid at 0x1c9387d49d0>
```



```
# Repeat step (3) with a stacked KDE plot
sns.displot(data, x="mpg", hue="origin", multiple="stack", kind='kde')
```

```
D:\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na
ert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

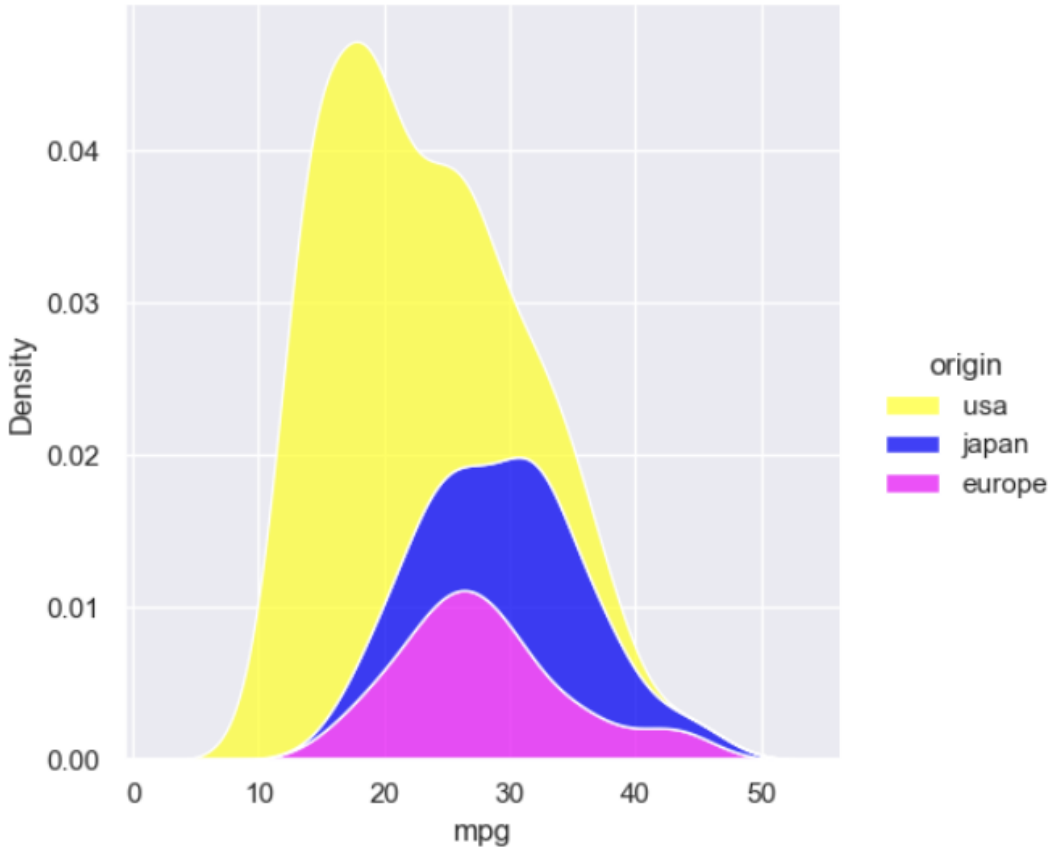
```
<seaborn.axisgrid.FacetGrid at 0x1c938856290>
```



```
# Repeat Step (4) by change the color palette to 'yellow', 'blue', 'magenta'
cp = sns.color_palette(['yellow', 'blue', 'magenta'])
sns.displot(data, x="mpg", hue="origin", multiple="stack", kind='kde', palette=cp)
```

```
D:\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na
ert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

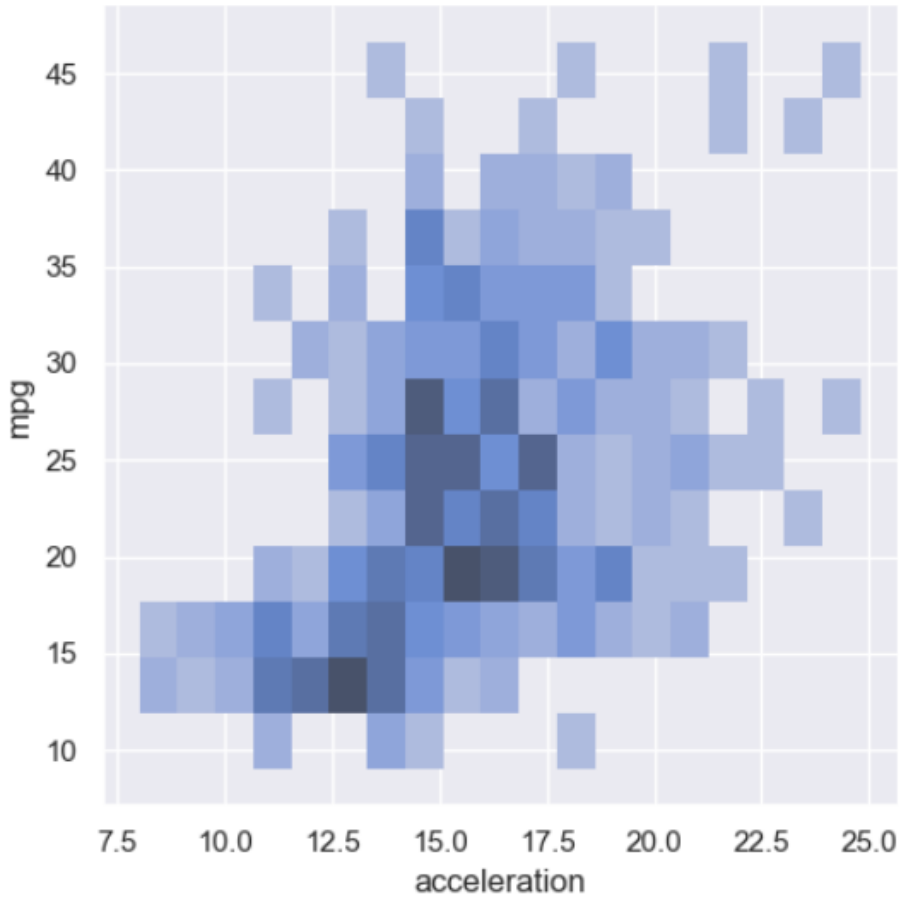
```
<seaborn.axisgrid.FacetGrid at 0x1c9383bf8d0>
```



```
# Create a bivariate histogram using the 'acceleration' and 'mpg'.
sns.displot(data, x="acceleration", y="mpg")
```

```
D:\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_
ert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
D:\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_
ert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

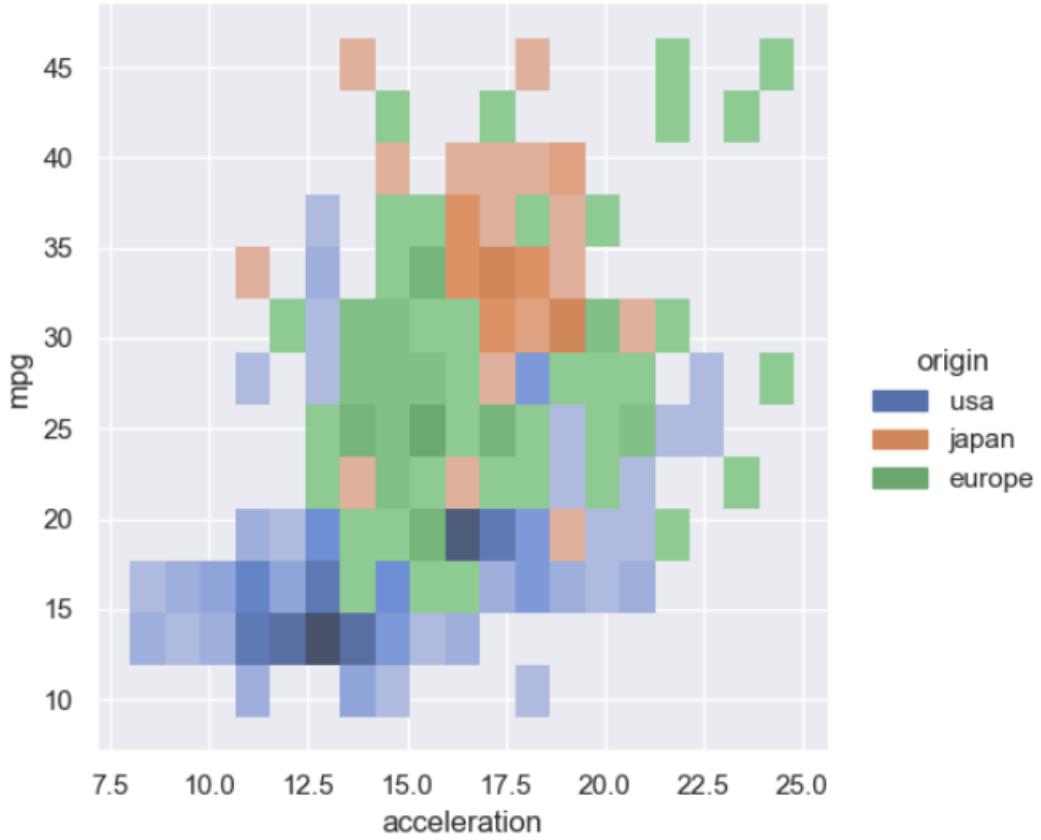
```
<seaborn.axisgrid.FacetGrid at 0x1c938796710>
```



```
# (7) Repeat step (6) with an additional variable 'origin' using 'hue'
sns.displot(data, x="acceleration", y="mpg", hue='origin')
```

```
D:\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na
ert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
D:\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na
ert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

```
<seaborn.axisgrid.FacetGrid at 0x1c93991ec90>
```

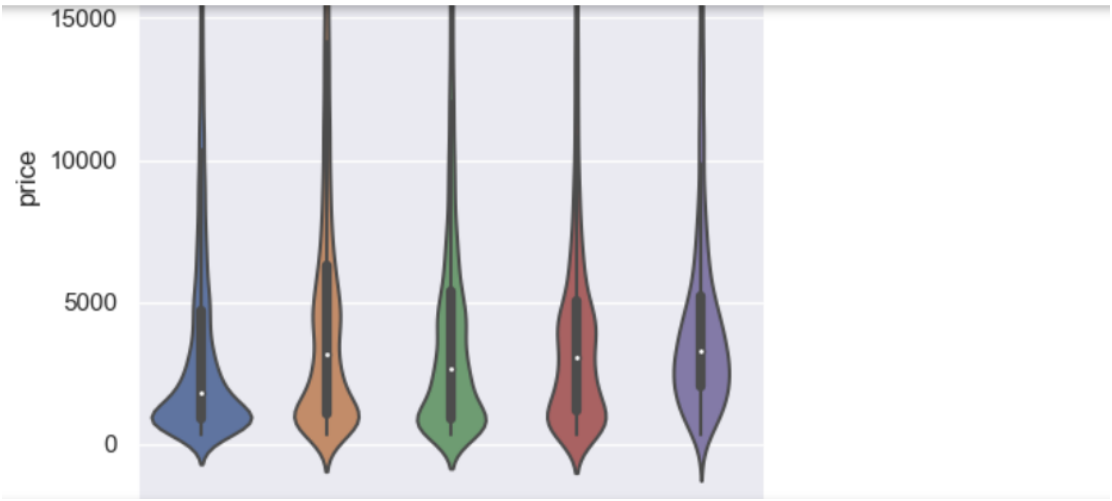


Task 2

You are going to use a violin plot to visualize the relationship between the cut and price of diamonds.

1. Load the default dataset 'diamonds' from seaborn.
2. Take a look at the data.
3. Visualize the 'price' values of different 'cut' of exercise.

```
## Your code here

import seaborn as sns

data = sns.load_dataset("diamonds", data_home='./data/')
sns.catplot(x='cut', y='price', data=data, kind='violin')
```



CCIT4092 Data Visualization

Lab 05 seaborn ¶

Your Name:

Your Student ID:

You are required to show your work **step by step**. Use at least one Jupyter Notebook cell for each part of your answer.

Task 1

You are going to use a heatmap plot to visualize the number of customers in a restaurant from Mon to Sun, Hours 00 to Hours 24.

1. Load the CSV file 'restaurant.csv' to a variable named `df`.
2. Take a look at the data. You should find that the index (df.index) of the DataFrame is unnamed with an automatically assigned number.
3. Convert the table to a two-dimensional dataframe using the function `pivot()` of pandas. The row should be the 'hours' while the column should be the 'day'.
4. After step (3), you should find that the DataFrame header is now sorted by the day in alphabetical order. To fix this re-index the column names to ['mon','tue','wed','thu','fri','sat','sun'].
5. Replace all missing data (NaN) to 0.
6. Create a figure with size 8x8. Use a subplot to visualize the data using heatmap. The color of the heatmap should shows the number of customers in that hour. Use the cmap `viridis`. You have to show the color bar, and show the numbers on the heatmap.
7. Create another figure with size 8x10. Use a subplot to visualize the data using heatmap. The color of the heatmap should shows the number of customers in that hour. The heatmap should have the following features:
   A. x-axis: title='Hour' - labels = 0 to 23, rotation=0
   B. y-axis: title='Day' - labels = ['M','T','W','T','F','S','S']
   C. color map: 'coolwarm'
   D. a horizontal color bar, with label 'number of customers'
   E. customer numbers on the heatmap cell
   F. linewidth between the cells = 0.2
   G. title: 'customer in a week'

   Hint: You may have to preprocess 'df' in order to plot the required heatmap

```
## 1, 2.
## Your code here

import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

df = pd.read_csv('data/restaurant.csv')
df
```

```
## 3.
## Your code here
df = df.pivot(index='hours', columns='day', values='customers')
df
```

day	fri	mon	sat	sun	thu	tue	wed
hours							
0	0.0	0.0	0.0	NaN	0.0	0.0	0.0
1	0.0	0.0	0.0	NaN	0.0	0.0	0.0
2	0.0	0.0	0.0	NaN	0.0	0.0	0.0
3	0.0	0.0	0.0	NaN	0.0	0.0	0.0
4	0.0	0.0	0.0	NaN	0.0	0.0	0.0

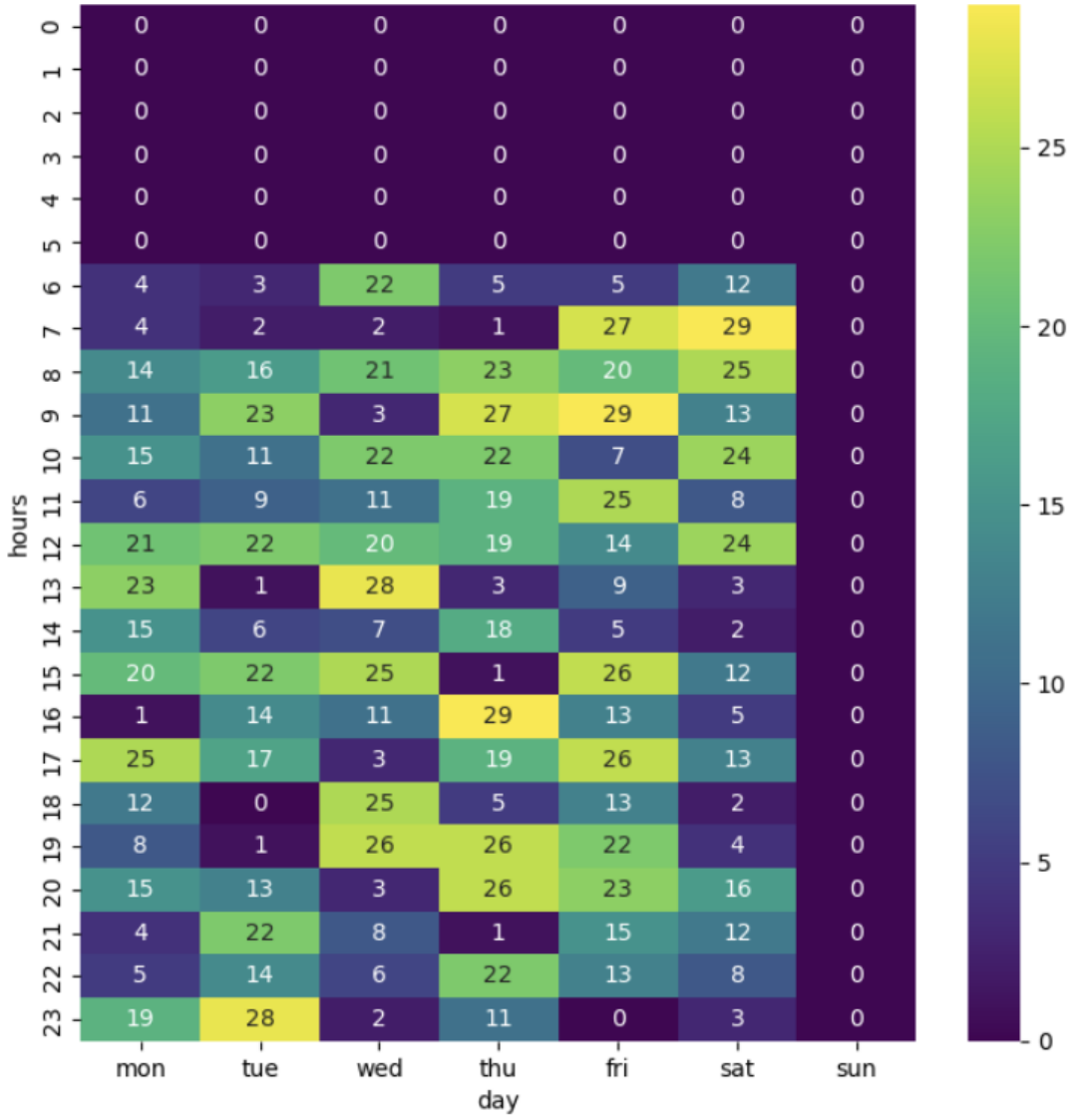
```
# 4.
## Your code here

df = df.reindex(columns= ['mon','tue','wed','thu','fri','sat','sun'] )
df
```

day hours	mon	tue	wed	thu	fri	sat	sun
0	0.0	0.0	0.0	0.0	0.0	0.0	NaN
1	0.0	0.0	0.0	0.0	0.0	0.0	NaN
2	0.0	0.0	0.0	0.0	0.0	0.0	NaN
3	0.0	0.0	0.0	0.0	0.0	0.0	NaN
4	0.0	0.0	0.0	0.0	0.0	0.0	NaN
5	0.0	0.0	0.0	0.0	0.0	0.0	NaN
6	4.0	3.0	22.0	5.0	5.0	12.0	NaN
7	4.0	2.0	2.0	1.0	27.0	29.0	NaN

```
# 5.
## Your code here
df = df.fillna(0)
df
```

day hours	mon	tue	wed	thu	fri	sat	sun
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6	4.0	3.0	22.0	5.0	5.0	12.0	0.0

```
# 6.
## Your code here
fig= plt.figure(figsize=(8, 8))
ax=plt.subplot(1,1,1)
hm = sns.heatmap(df, annot=True, cmap="viridis", ax=ax)
plt.show()
```

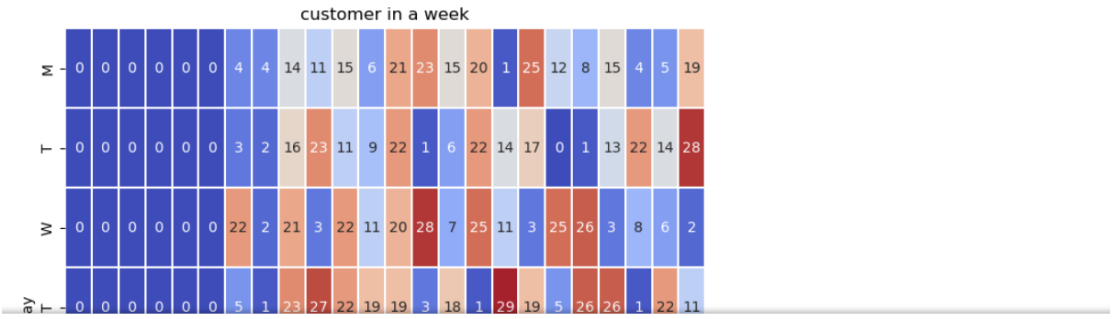


```
# 7.
## Your code here
df = df.transpose()
df
```

hours	0	1	2	3	4	5	6	7	8	9	...	14	15	16	17	18	19	20	21	22	23
day																					
mon	0.0	0.0	0.0	0.0	0.0	0.0	4.0	4.0	14.0	11.0	...	15.0	20.0	1.0	25.0	12.0	8.0	15.0	4.0	5.0	19.0
tue	0.0	0.0	0.0	0.0	0.0	0.0	3.0	2.0	16.0	23.0	...	6.0	22.0	14.0	17.0	0.0	1.0	13.0	22.0	14.0	28.0
wed	0.0	0.0	0.0	0.0	0.0	0.0	22.0	2.0	21.0	3.0	...	7.0	25.0	11.0	3.0	25.0	26.0	3.0	8.0	6.0	2.0
thu	0.0	0.0	0.0	0.0	0.0	0.0	5.0	1.0	23.0	27.0	...	18.0	1.0	29.0	19.0	5.0	26.0	26.0	1.0	22.0	11.0
fri	0.0	0.0	0.0	0.0	0.0	0.0	5.0	27.0	20.0	29.0	...	5.0	26.0	13.0	26.0	13.0	22.0	23.0	15.0	13.0	0.0
sat	0.0	0.0	0.0	0.0	0.0	0.0	12.0	29.0	25.0	13.0	...	2.0	12.0	5.0	13.0	2.0	4.0	16.0	12.0	8.0	3.0
sun	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

7 rows × 24 columns

```
fig= plt.figure(figsize=(8, 10))
ax=plt.subplot(1,1,1)
hm = sns.heatmap(data=df, annot=True, linewidth=0.2, cmap="coolwarm", ax=ax, cbar_kws={'label': 'number of customers', 'orientation': 'horizontal'})
hm.set_yticklabels(['M','T','W','T','F','S','S'])
hm.set_xticklabels(range(0,24), rotation=0)
hm.set_ylabel('Day')
hm.set_xlabel('Hour')
hm.set_title('customer in a week')
plt.show()
```



CCIT4092 Data Visualization

Lab 06 Bokeh (Suggested Solution)

Your Name:

Your Student ID:

You are required to show your work **step by step**. Use at least one Jupyter Notebook cell for each part of your answer.

Task 1

You are going to use `Bokeh` to plot a graph that shows the Open Price of Amazon and Apple.

1. Configure the Jupyter Notebook so that `Bokeh` can display the graph in Jupyter Notebook.
2. Load the file 'AMZN_data.csv' and 'AAPL_data.csv' using pandas. Stores the DataFrame as 'amazon_data' and 'apple_data' respesctively.
3. Generate two lists, amazon_x and amazon_y, where amazon_x is the column 'date' from the data, amazon_y is the column 'open' from the amazon_data.
4. Generate two lists, apple_x and apple_y, where apple_x is the column 'date' from the data, apple_y is the column 'open' from the apple_data.
5. Shorten the four lists. Only the last 50 data samples will be shown in the graph.
6. Visualize Apple and Amazon Open Prices using Bokeh Line Plot. Set the title, x_label and y_label appropriately. The x_label should be the date with 45 degree orientation.

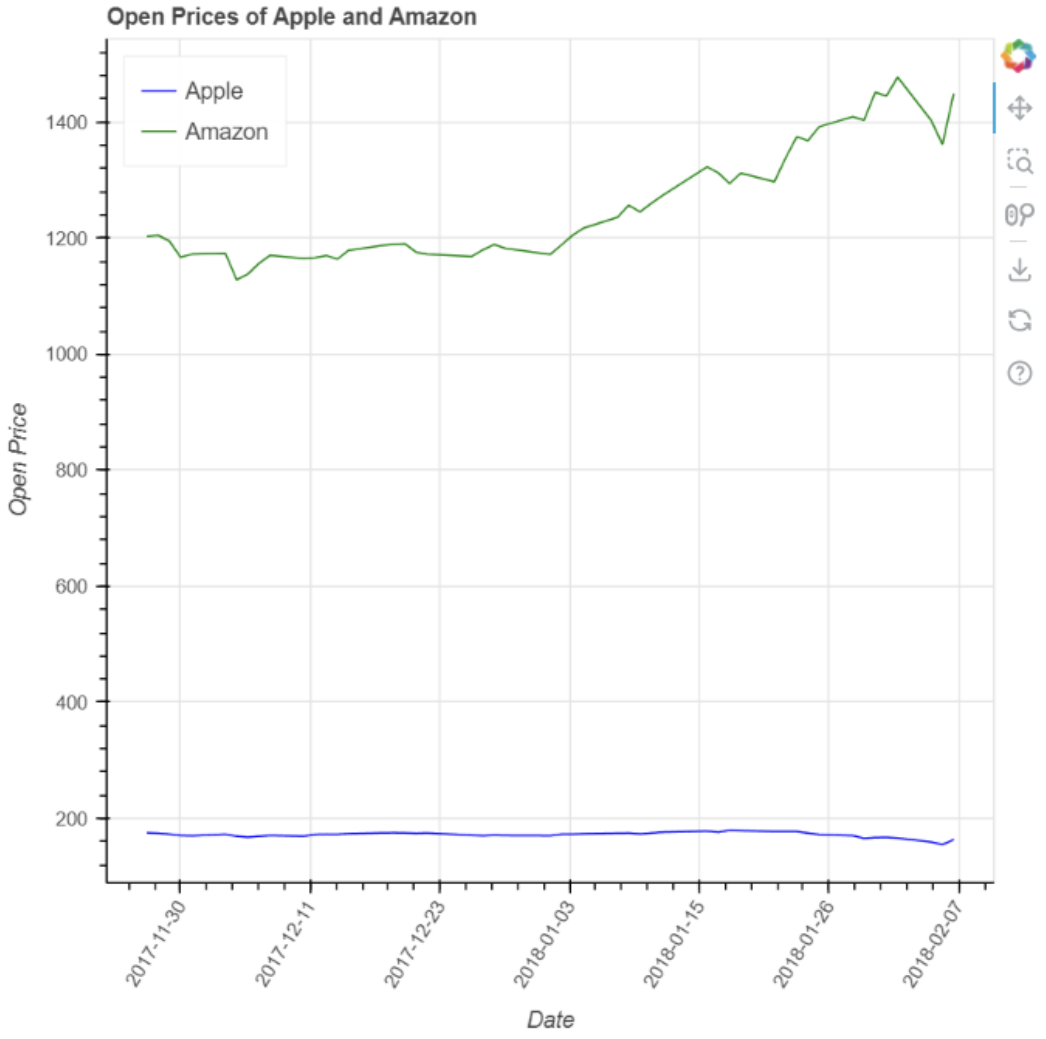
```
## Your code here

from bokeh.plotting import figure, show
from bokeh.io import output_notebook

output_notebook()

import pandas as pd

apple_data = pd.read_csv('AAPL_data.csv', parse_dates=['date'])
amazon_data = pd.read_csv('AMZN_data.csv', parse_dates=['date'])
```

BokehJS 3.3.4 successfully loaded.

```
apple_x = apple_data['date'].tolist()
apple_y = apple_data['open'].tolist()

amazon_x = amazon_data['date'].tolist()
amazon_y = amazon_data['open'].tolist()
```

```
apple_x = apple_x[-50:]
apple_y = apple_y[-50:]

amazon_x = amazon_x[-50:]
amazon_y = amazon_y[-50:]
```

```
from bokeh.plotting import figure, show
from bokeh.models import DatetimeTickFormatter

# Create a figure
p = figure(title='Open Prices of Apple and Amazon', x_axis_label='Date', y_axis_label='Open Price')

p.xaxis.formatter = DatetimeTickFormatter(days="%Y-%m-%d")
p.xaxis.major_label_orientation = 45

# Plot the lines for Apple and Amazon
p.line(apple_x, apple_y, legend_label='Apple', line_color='blue')
p.line(amazon_x, amazon_y, legend_label='Amazon', line_color='green')

# Add a legend
p.legend.location = 'top_left'

# Show the plot
show(p)
```


Open Prices of Apple and Amazon
Apple
Amazon
1400
1200
1000
800
600
400
200
Open Price
2017-11-30
2017-12-11
2017-12-23
2018-01-03
2018-01-15
2018-01-26
2018-02-07
Date

The correct answer is:

Show relationships among variables → heatmaps,

Shows the statistics of the dataset → Violin plot,

Check how the data are distributed. → Histogram,

Visualize geospatial data → Choropleth map,

Comparison of single variable → Bar chart,

Comparison of multiple variables → Radar chart,

Represent something as a part of a whole → Pie chart

The correct answer is:

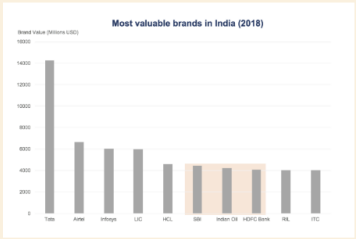
Our tendency to perceive objects that are physically connected are part of the same group, whether or not that is true or relevant to the data at hand. → Connection,

Reflects our tendency to perceive objects physically enclosed together in a ways that seems to create a boundary around that which is related or belonging to part of the same group, whether or not that is true or relevant to the data at hand.

→ Enclosure,

Perceive objects close to each other as belonging to a group. → Proximity,

How people perceive the orange box in the figure?

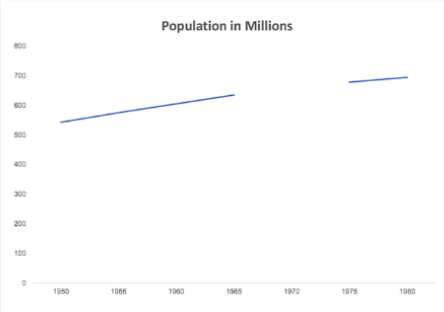


→ Enclosure,

How people perceive this figure?

→ Figure and Ground,

How human perceive the line in the picture?



→ Closure,

How people perceive the moving number in the figure?

→ Common Fate,

Seek similarities and differences in objects and link similar objects as belonging to a group. → Similarity