EPL448: Data Mining on the Web – Lab 5

University of Cyprus
Department of Computer Science

Παύλος Αντωνίου

Γραφείο: B109, ΘΕΕ01

Why data visualization?

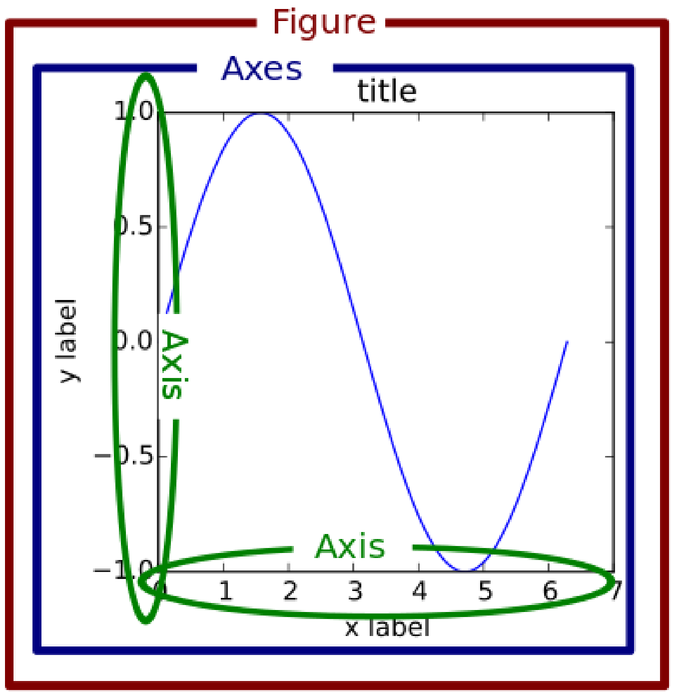
- By inspecting the raw data stored in DB tables or files we cannot infer any direct associations between variables or properties or trends
- Data visualization translates information into a visual context
  - make data easier for the human brain to understand and pull insights from
  - allows to reveal/identify patterns, trends and outliers
- Data visualization is of utmost importance in Exploratory Data Analysis (EDA)
  - EDA refers to the critical process of performing **initial investigations** on data so as to **discover patterns**, to spot anomalies, to test hypothesis and to **check assumptions** using summary statistics and graphical representations
- This initial analysis can help to easily rule out the models that won’t be suitable for such a data – then we will only use suitable models, without wasting our valuable time and computational resources

Data Visualization Libraries in Python

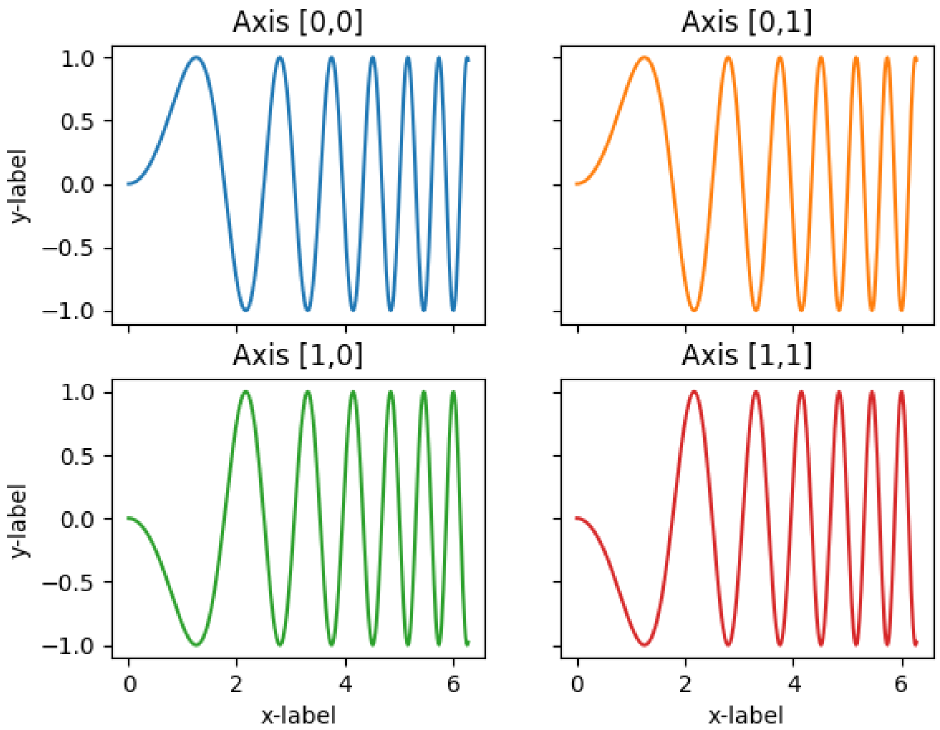
- Matplotlib
  - Most prominent plotting lib
  - import it to a notebook using: `import matplotlib.pyplot as plt`
- Seaborn
  - Visually appealing plots
  - Based on Matplotlib
  - import it to a notebook using: `import seaborn as sns`
- Plotly
  - Interactive charts and maps
  - Not pre-installed in Anaconda. Run: `conda install -c plotly plotly` on Anaconda prompt

Matplotlib Figure

- Matplotlib plots the data on **Figures** each of which can contain one or more **Axes**
- An Axes is attached to a Figure and contains a region for plotting data and sets the coordinate system



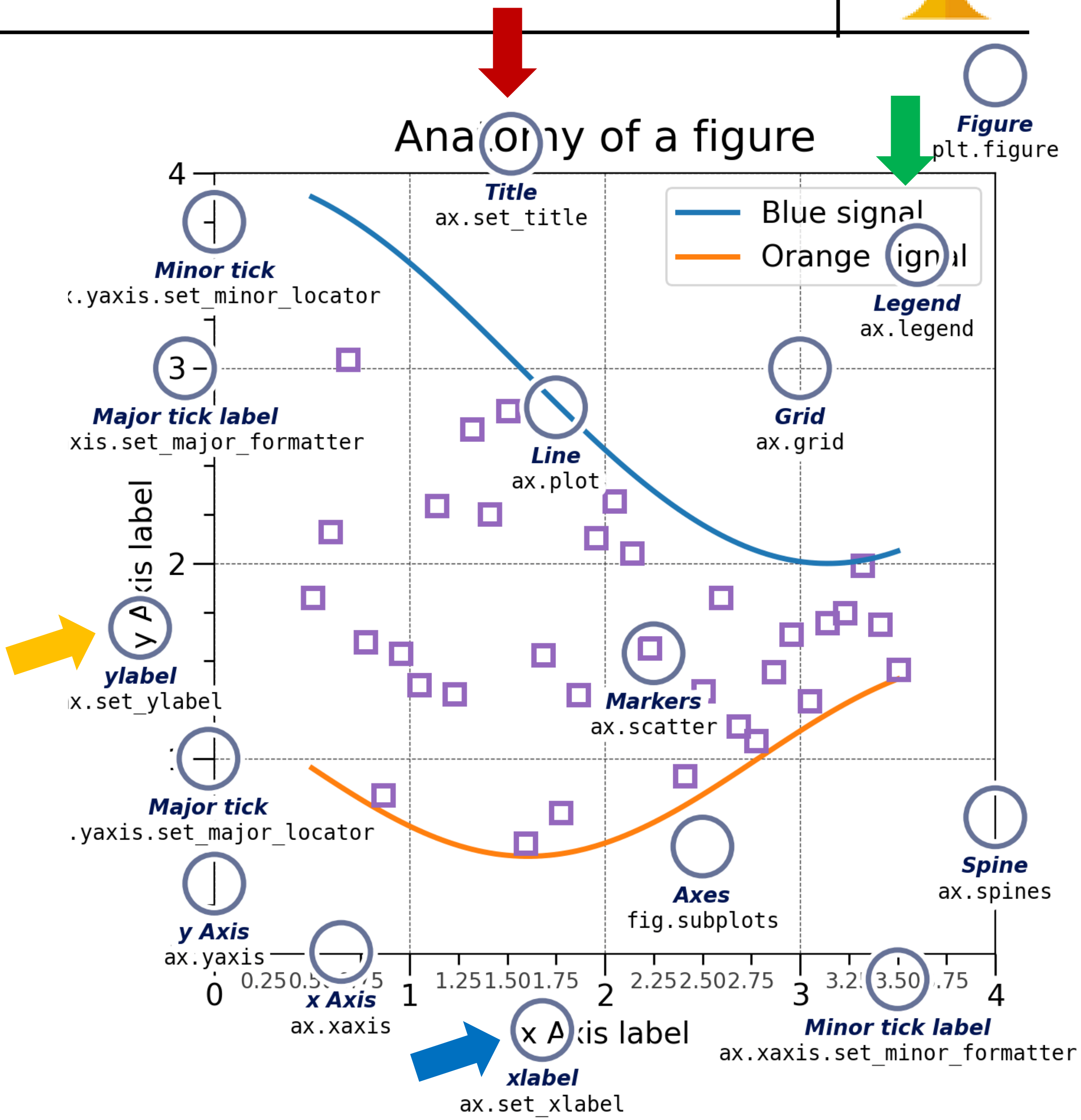
1 Figure with 1 Axes



1 Figure with 4 (2x2) Axes

Figure anatomy

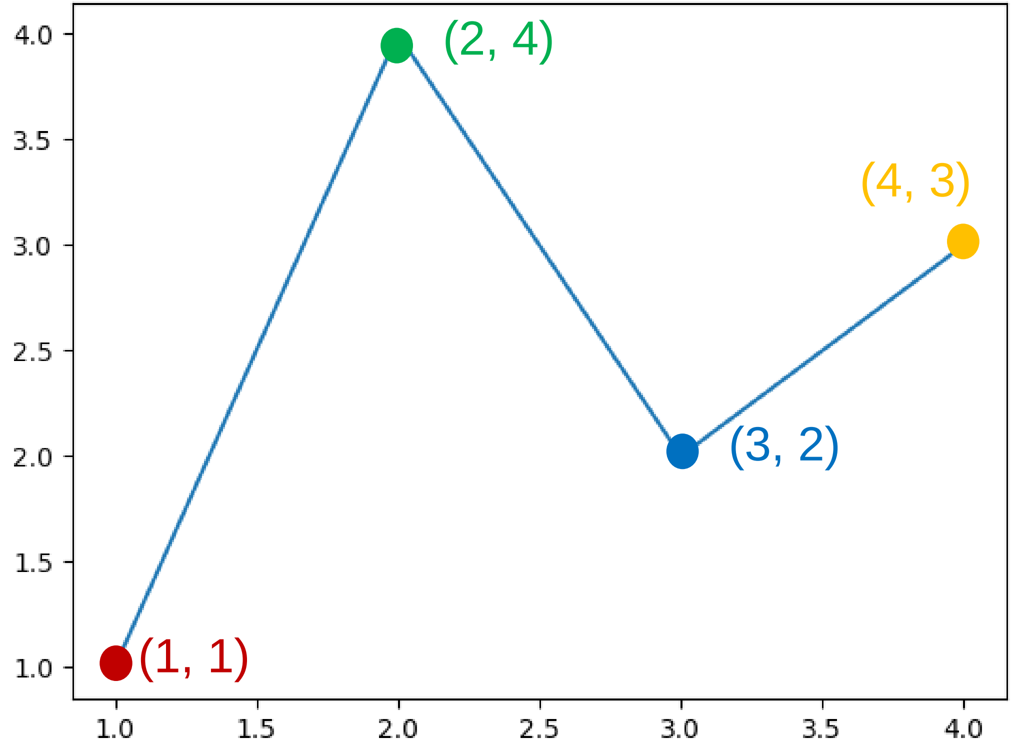
- The Figure keeps track of all the child Axes, and of the group of 'special' objects (titles, figure legends, xlabel, ylabel, etc) belonging to each Axes



Simple example

- The simplest way of creating a Figure is by using `plt.subplots` which returns an Axes object. We can then use `plot` function on the Axes object to draw some data:

```
# Create a figure containing a single axes
fig, ax = plt.subplots()
# Plot some data on the axes
ax.plot([1, 2, 3, 4], [1, 4, 2, 3])
```



- Alternatively, we can create a figure with no axes object. We can then use `plt.plot` to draw data

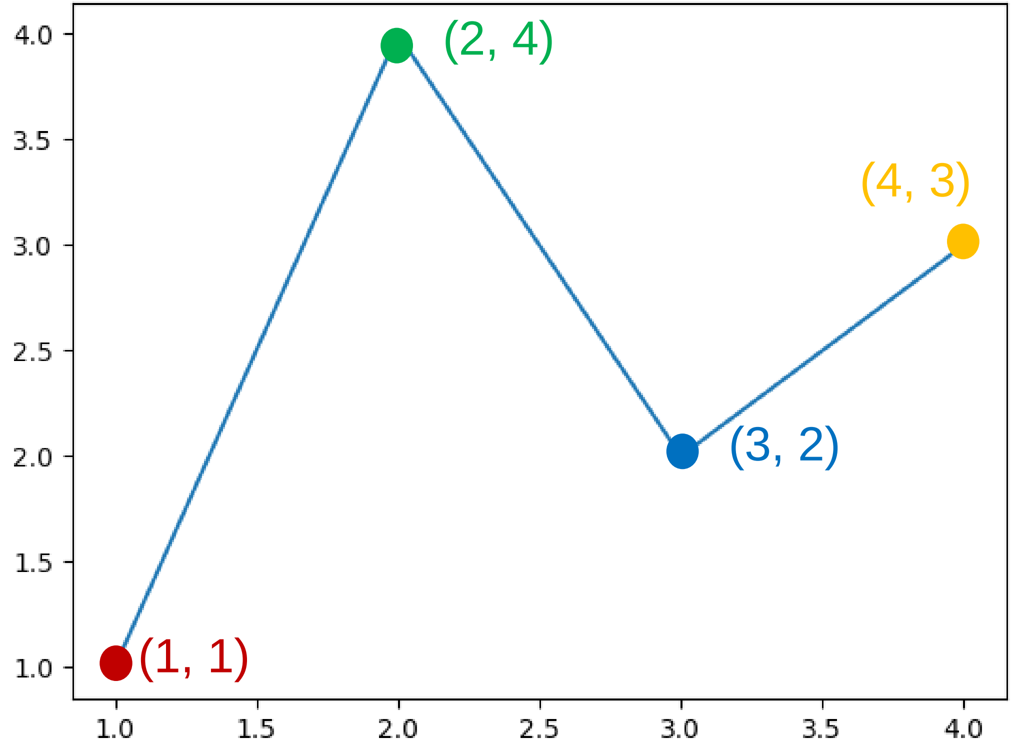
```
# Create a figure with no axes
plt.figure()
# Plot some data on the current figure
# an axes will be created automatically
plt.plot([1, 2, 3, 4], [1, 4, 2, 3])
```

Simple example

- The simplest way of creating a Figure is by using `plt.subplots` which returns an Axes object. We can then use `plot` function on the Axes object to draw some data:

```
# Create a figure con    ing a single axes
fig, ax = plt.sub        ()
# Plot some da      the axes
ax.plot([1      3, 4], [1, 4, 2, 3]);
```

Axes-level plotting



- Alternatively, we can create a figure with no axes object. We can then use `plt.plot` to draw data

```
# Create a figure with      xes
plt.figure()
# Plot some dat     the current figure
# an axes wi       created automatically
plt.plot([1  2, 3, 4], [1, 4, 2, 3]);
```

Figure-level plotting

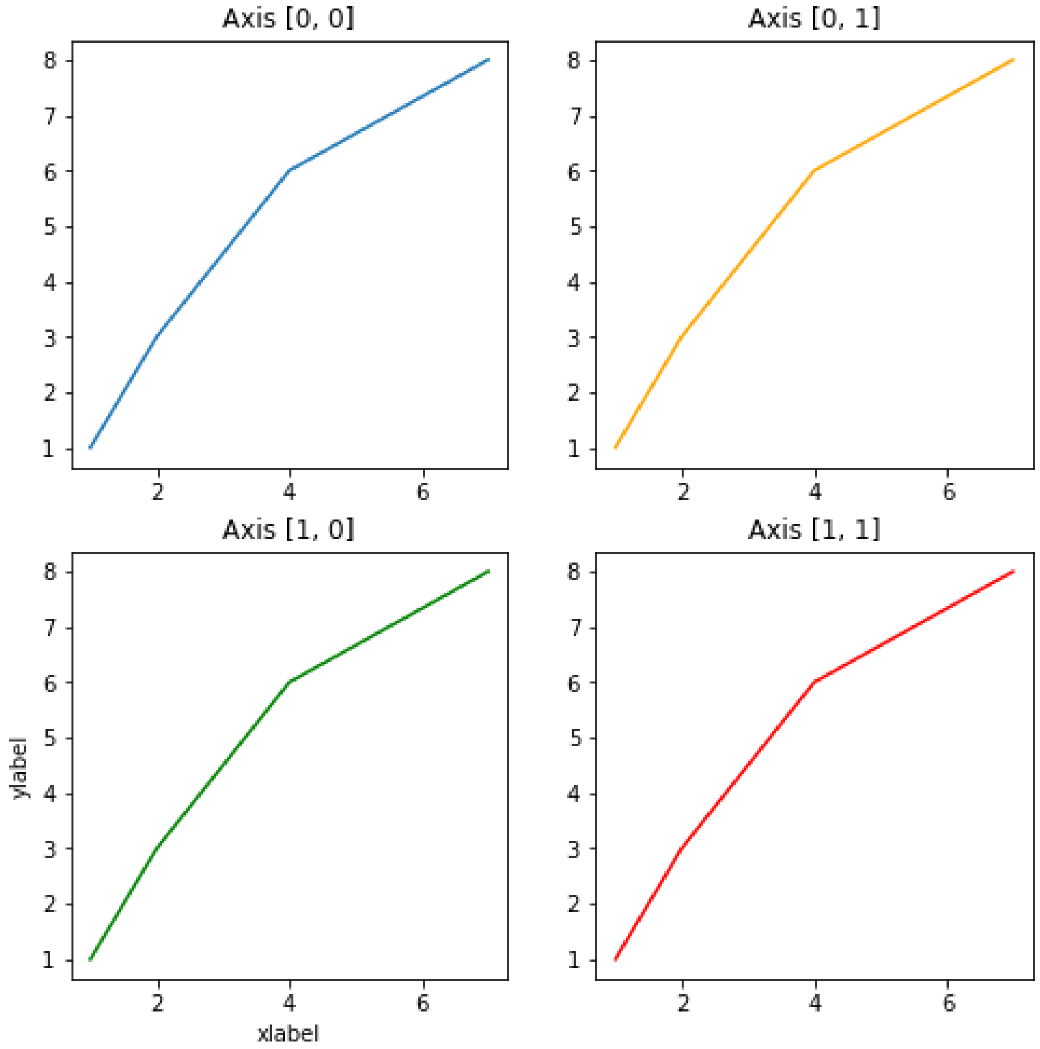
plt.subplots(*nrows*, *ncols*)

- Used when we want 2 or more axes plots in a single figure

- One of the most useful features when we need to compare two or more plots hand to hand instead of having them separately

Subplots – Option 1: Axes-level plotting

```
import matplotlib.pyplot as plt
import numpy as np
# Some example data to display
x = [1, 2, 4, 7] # x values
y = [1, 3, 6, 8] # y values
# Create a figure containing 2x2 axes
fig, axs = plt.subplots(2,2, figsize=(8,8))
# Plot some data on the axes
axs[0, 0].plot(x, y)
axs[0, 0].set_title('Axis [0, 0]')
axs[0, 1].plot(x, y, color='orange')
axs[0, 1].set_title('Axis [0, 1]')
axs[1, 0].plot(x, y, color='green')
axs[1, 0].set_title('Axis [1, 0]')
axs[1, 0].set_xlabel('xlabel')
axs[1, 0].set_ylabel('ylabel')
axs[1, 1].plot(x, y, color='red')
axs[1, 1].set_title('Axis [1, 1]')
```

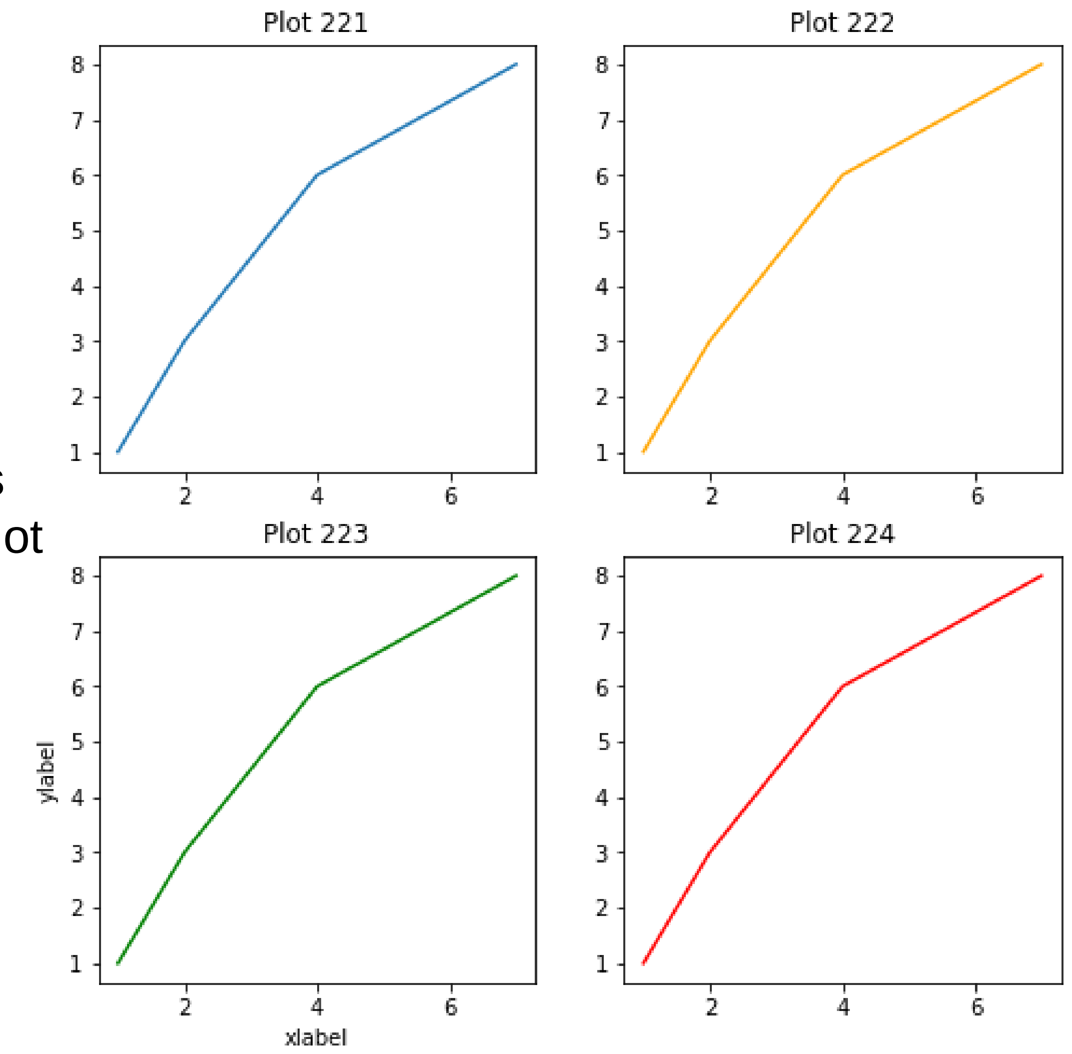


Specifies the width and height of the figure in unit inches.
By default, the figure has the dimensions as (6.4, 4.8)

Subplots – Option 2: Figure-level plotting

```
import matplotlib.pyplot as plt
import numpy as np
# Some example data to display
x = [1, 2, 4, 7]
y = [1, 3, 6, 8]
# Create a figure with no axes
plt.figure(figsize=(8, 8))
# Plot some data
plt.subplot(221)
plt.plot(x, y)
plt.title('Plot 221')
plt.subplot(222)
plt.plot(x, y, color='orange')
plt.title('Plot 222')
plt.subplot(223)
plt.plot(x, y, color='green')
plt.xlabel('xlabel')
plt.ylabel('ylabel')
plt.title('Plot 223')
plt.subplot(224)
plt.plot(x, y, color='red')
plt.title('Plot 224')
```

3-digit integer where:
1st num = No of rows
2nd num = No of columns
3rd num = index of that plot

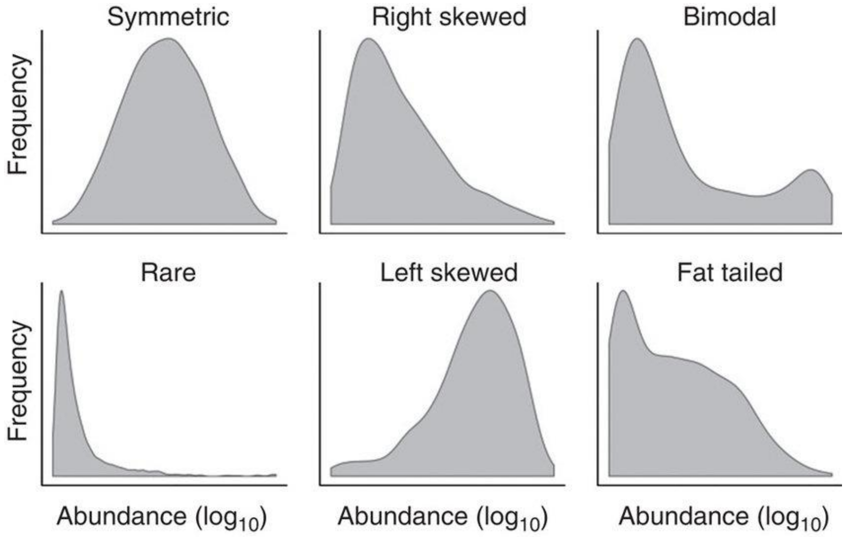


Different types of analysis

- There are different types of graphical analysis as mentioned below.
  - **Univariate**: In univariate analysis we will be using a **single feature** (variable) to analyze almost of its properties
  - **Bivariate**: When we compare the data between exactly **2 features** then it's called bivariate analysis
  - **Multivariate**: Comparing **more than 2 features** is called as Multivariate analysis
- Plot types discussed in the next slides will be marked as (U),(B) & (M) to represent them as Univariate, Bivariate and Multivariate plots correspondingly.

Distribution plots (U/B)

- An early step in any effort to analyze or model data should be to understand how the variables are distributed
- Techniques for distribution visualization can provide quick answers to many important questions:
  - What range do the observations cover?
  - What is their central tendency?
  - Are they heavily skewed in one direction?
  - Is there evidence for bimodality?
  - Are there significant outliers?
  - Do the answers to these questions vary across subsets defined by other variables?



Datasets for plotting

- In order to better explain the usage of each plot we introduce two popular datasets:
  - Haberman dataset
  - Iris dataset

Haberman dataset

- Dataset contains 306 cases (rows or observations) from a study that was conducted between 1958 and 1970 at the University of Chicago's Billings Hospital on the survival of patients who had undergone surgery for breast cancer
- Features of each patient:
  – Age of patient at time of operation (numerical)
  – Patient's year of operation (year - 1900, numerical)
  – Number of positive axillary nodes detected (numerical)
- Target value
  – Survival status (class attribute)
    - 1 = the patient survived 5 years or longer
    - 2 = the patient died within 5 year

Haberman dataset

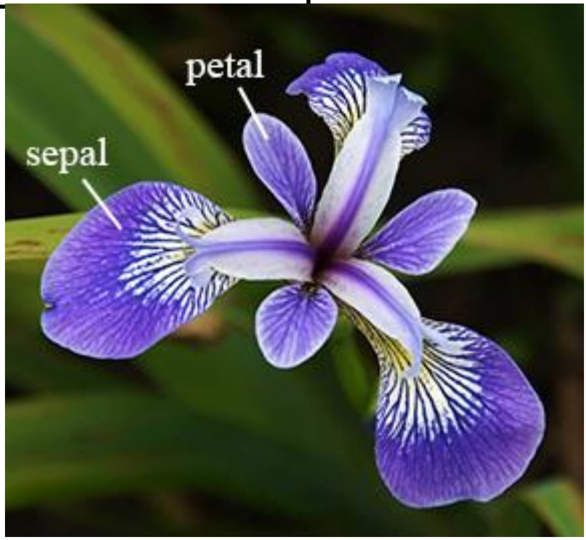
```
df1 = pd.read_csv('haberman.csv', names=['age', 'op_year', 'axil_nodes', 'surv_status'])
```

- DataFrame structure

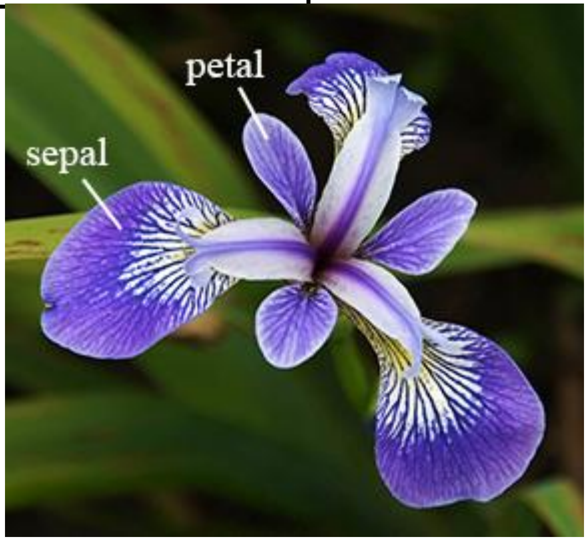
```
     age  op_year  axil_nodes  surv_status
0     30       64           1            1
1     30       62           3            1
2     30       65           0            1
3     31       59           2            1
4     31       65           4            1
..   ...      ...         ...          ...
301   75       62           1            1
302   76       67           0            1
303   77       65           3            1
304   78       65           1            2
305   83       58           2            2
```

Iris dataset

- Small dataset with 150 observations of iris flowers
  - each observation (row) has 4 columns of measurements (or variables or features) of the flowers (in centimeters)
  - target column (the 5th column) is the species (class) of the flower observed
    - all observed flowers belong to one of three species (setosa, versicolor, virginica)
- More info: https://en.wikipedia.org/wiki/Iris_flower_data_set



Dataset Overview



- Features:
  - sepal length in cm
  - sepal width in cm
  - petal length in cm
  - petal width in cm

- Target:
  - target column (class attribute)
    - Iris Setosa : 0
    - Iris Versicolour: 1
    - Iris Virginica: 2

Iris dataset

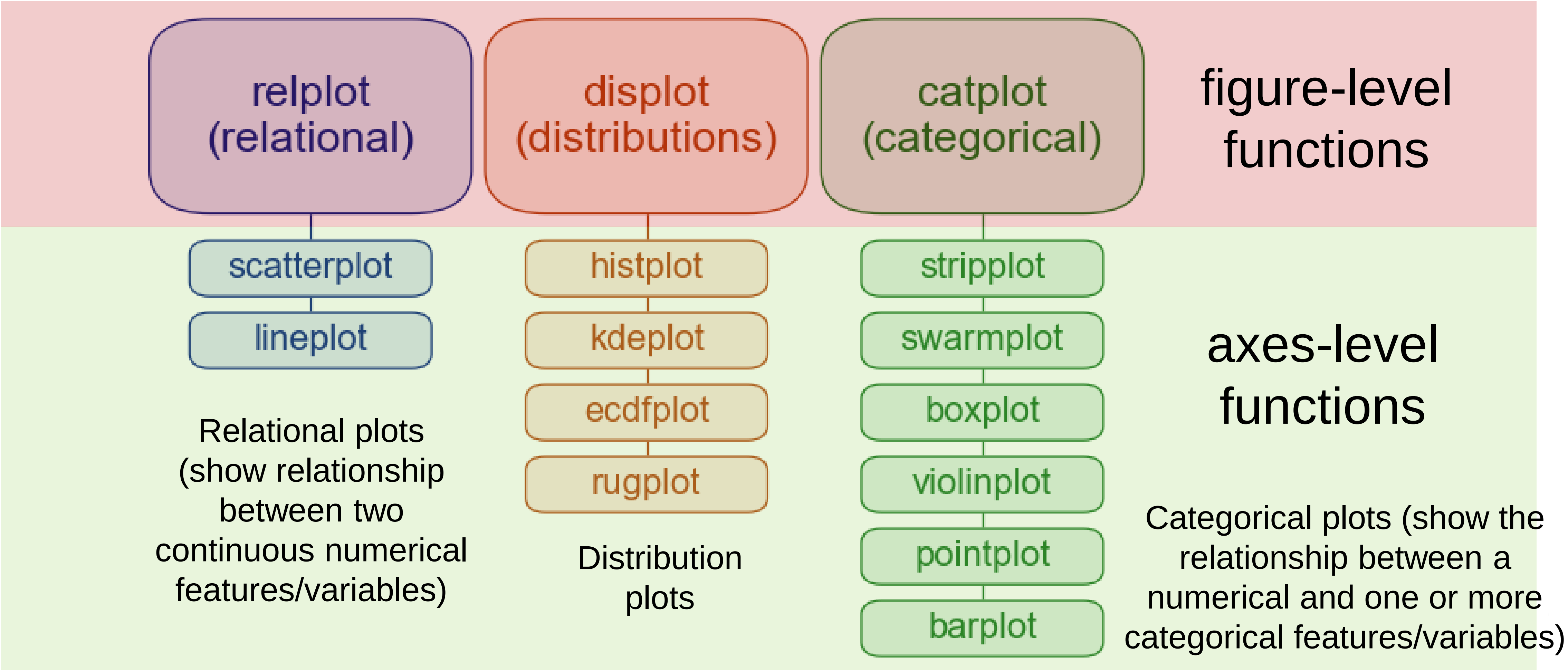
```
df1 = pd.read_csv('iris.csv')
```

- DataFrame structure

```
     sepal_length_(cm)  sepal_width_(cm)  petal_length_(cm)  petal_width_(cm)  target
0                  5.1               3.5                1.4               0.2       0
1                  4.9               3.0                1.4               0.2       0
2                  4.7               3.2                1.3               0.2       0
3                  4.6               3.1                1.5               0.2       0
4                  5.0               3.6                1.4               0.2       0
..                 ...               ...                ...               ...     ...
145                6.7               3.0                5.2               2.3       2
146                6.3               2.5                5.0               1.9       2
147                6.5               3.0                5.2               2.0       2
148                6.2               3.4                5.4               2.3       2
149                5.9               3.0                5.1               1.8       2
```

Seaborn plotting functions

- Each seaborn module has a single figure-level function, which offers a unitary interface to its various axes-level functions



Continuous (numerical) vs categorical data

- Continuous numerical variable/feature: contains data that can take on any value within a **defined range** and is often measured on a continuous scale, such as weight, height, or temperature

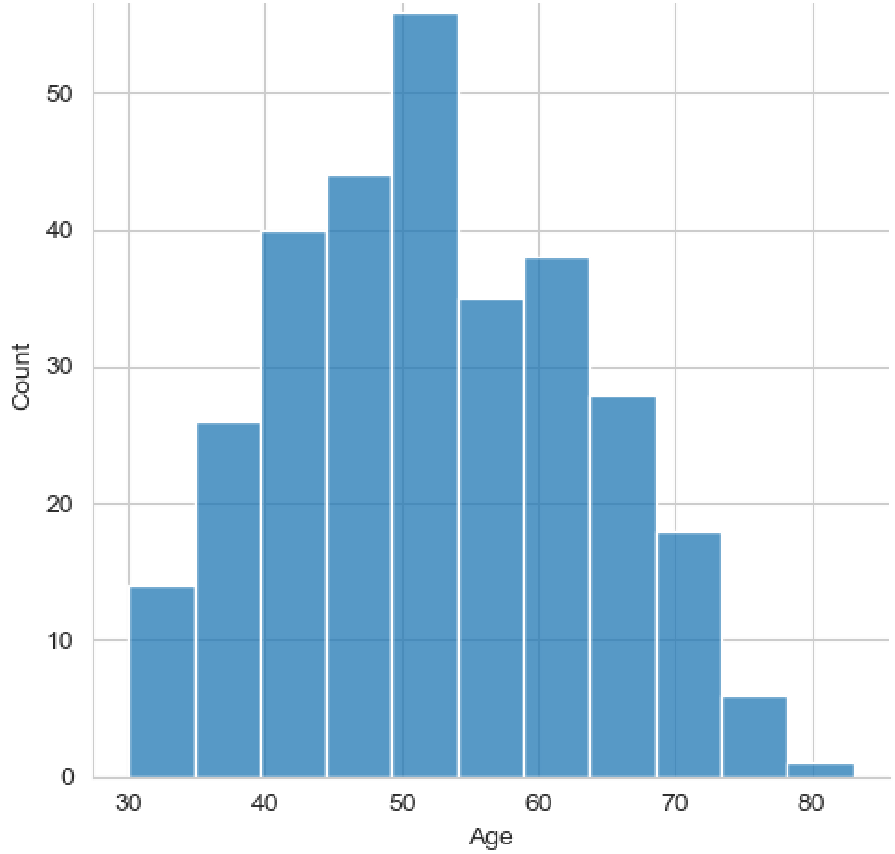
- Categorical variable/feature: contains data consisting of **discrete values** that fall into distinct categories or groups, such as gender, ethnicity, or product types. The values can be either strings or limited-range integer numbers
  - Example:
    - product type: electronics, food, furniture
    - product type: 0, 1, 2

Distribution plots (U/B)

- Understand data distribution ➔ tailor-made Machine Learning models to best fit our case study
- Machine Learning models are designed to work best under some distribution assumption
  - ML models such as LDA, Gaussian Naive Bayes, Logistic Regression and Linear Regression require all variables (features) to be bivariate or multivariate normal
- Knowing with which distributions we are working with, can help us to identify which models are best to use or if we are in need of transforming data before applying any machine learning model

Histogram plot (U) – displot() – Seaborn

- By default, displot() creates histogram (histplot() can be used instead)
  - A histogram aims to approximate the underlying probability density function that generated the data by binning (grouping) and counting observations




```
plt.figure()
sns.displot(data=df1, x='age')
plt.xlabel('Age')
```

OR

```
fig,ax = plt.subplots()
sns.histplot(data=df1, x='age')
ax.set_xlabel('Age')
```

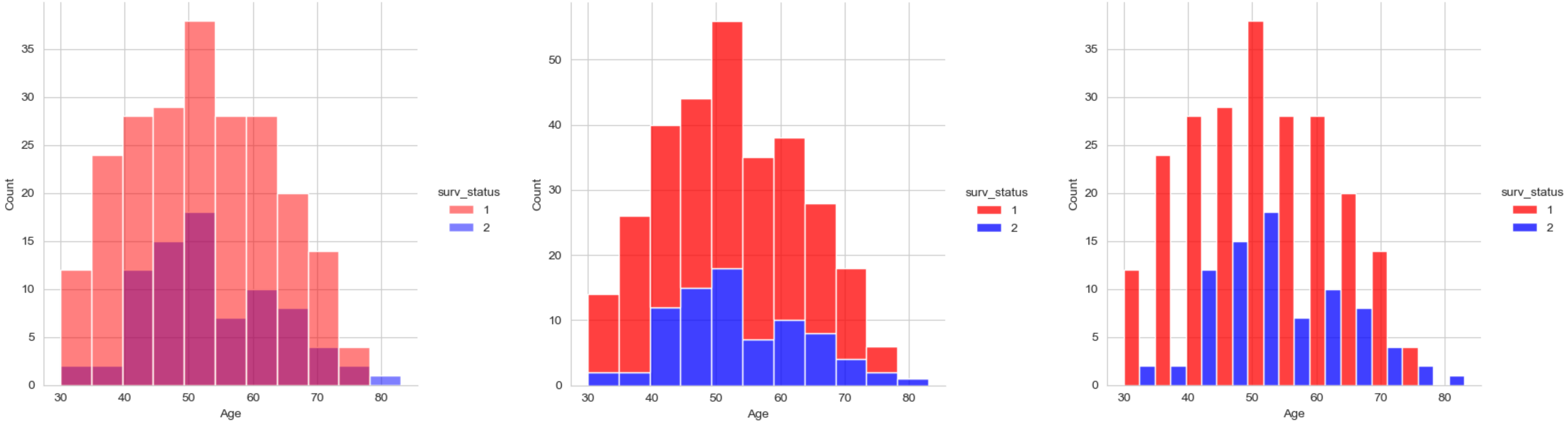
Histogram plot (B) – displot() – Seaborn

- Once you understand the distribution of a variable, the next step is often to ask whether the behavior of that distribution differs across other variables in the dataset - *use of hue, usually with categorical var*

```
plt.figure()
colors = ['red', 'blue']
sns.displot(data=df1, x='age', hue='surv_status', palette=colors,  )
plt.xlabel('Age')
```

```
multiple='stack'
```

```
multiple='dodge'
```



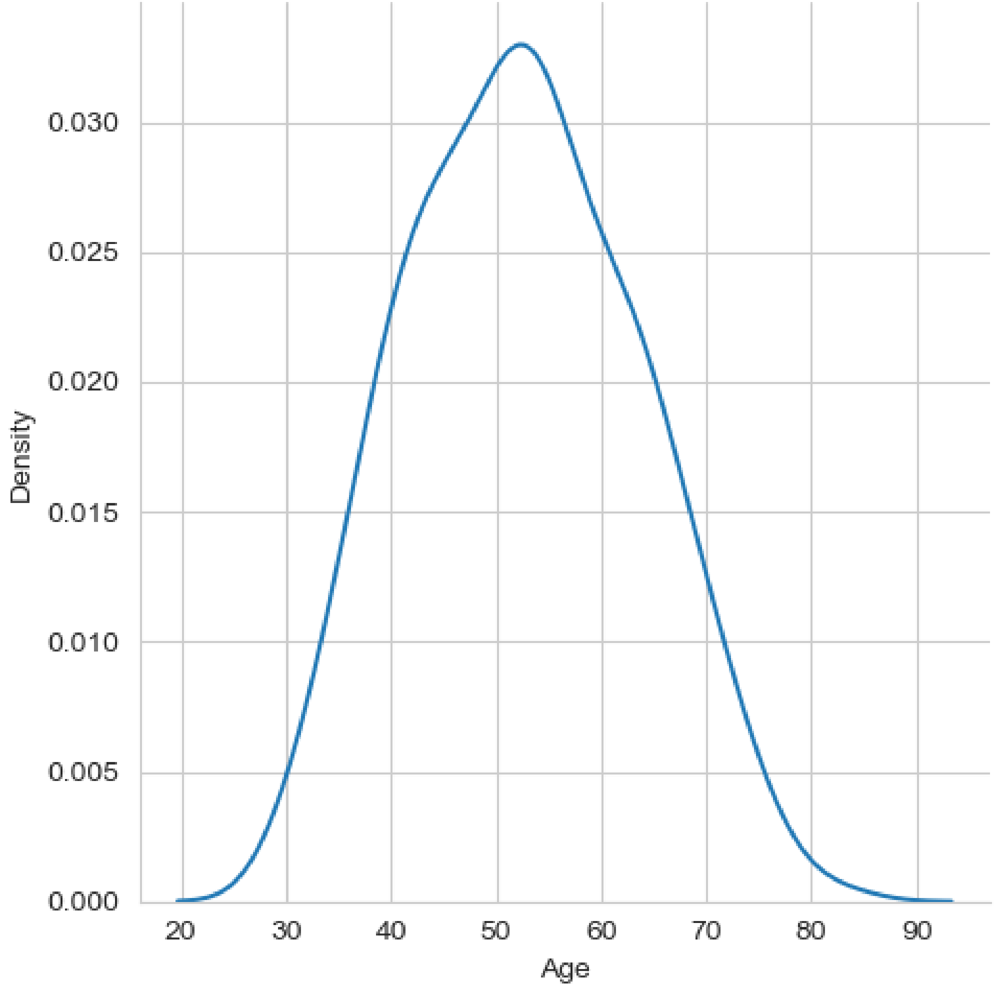
Distribution plot (U) – displot() – Seaborn

- displot() with `kind='kde'`
  - same behavior as `kdeplot()`
  - rather than using discrete bins, a Kernel density estimation (KDE) plot smooths the observations with a Gaussian kernel, producing a continuous density estimate:

```
plt.figure()
colors = ['red', 'blue']
sns.displot(data=df1, x='age', kind='kde')
plt.xlabel('Age')
```

OR

```
fig,ax = plt.subplots()
sns.kdeplot(data=df1, x='age')
ax.set_xlabel('Age')
```

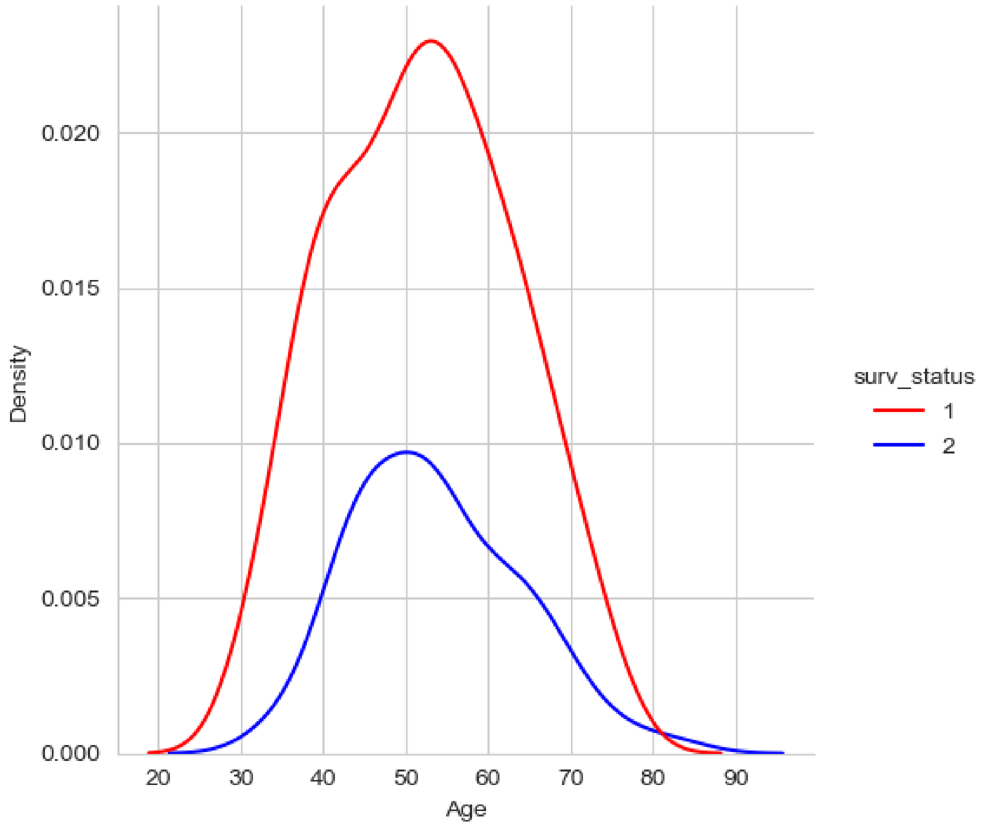


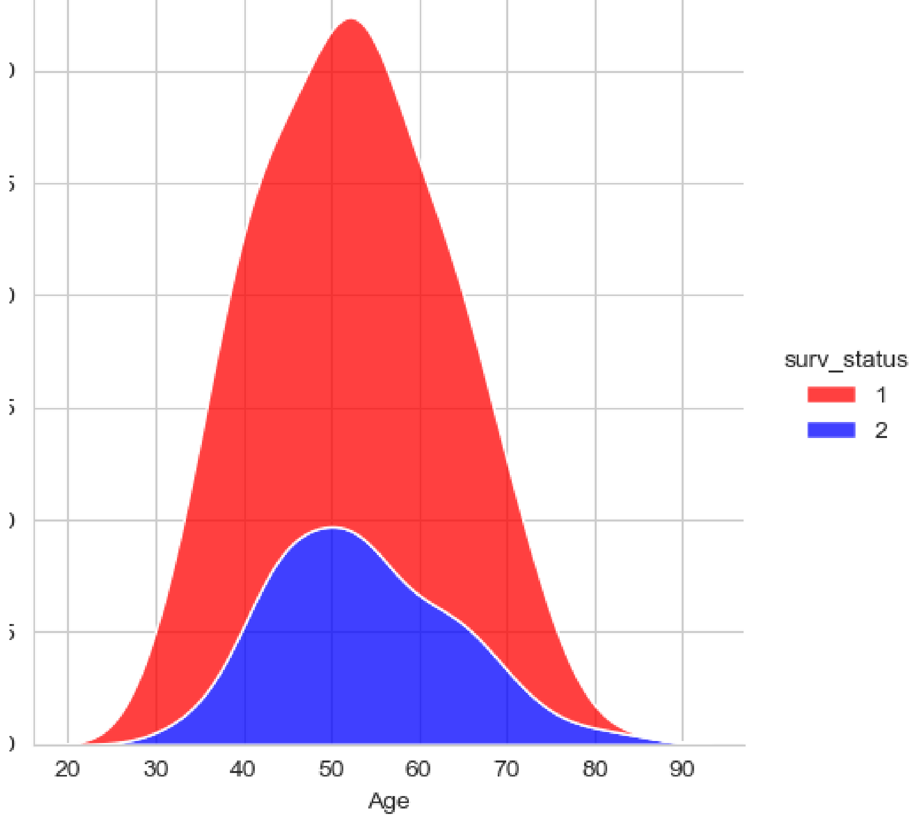
Distribution plot (B) – displot() – Seaborn

- Assigning a variable to hue will draw a separate kde plot for each of its unique values and distinguish them by color

```
colors = ['red', 'blue']
sns.displot(data=df1, x='age', kind='kde', hue='surv_status', palette=colors )
plt.xlabel('Age')
```

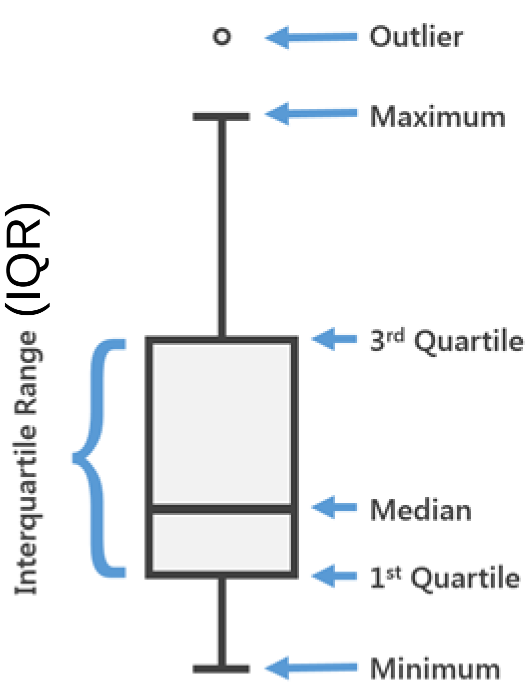
```
multiple='stack'
```





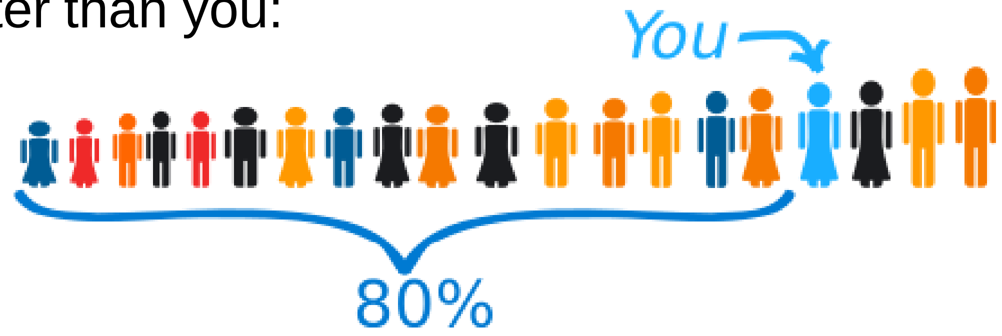
Box plots (or box-and whisker plots) (U)

- Used to extract the **statistical details** of a dataset
- Box plots also give a clear overview of **outlier points**
- Interquartile range (IQR): where the bulk of values lie
  – contains the middle **half** of the data set
- Straight lines at the maximum and minimum are called as whiskers
  – Maximum: Q3 + 1.5 * IQR
  – Minimum: Q1 – 1.5 * IQR
- Points outside of whiskers can be inferred as outliers
- The box plot gives us a representation of 25^{th} percentile (or 1^{st} quartile), 50^{th} percentile (or 2^{nd} quartile or median), 75^{th} percentile (or 3^{rd} quartile)



Percentiles

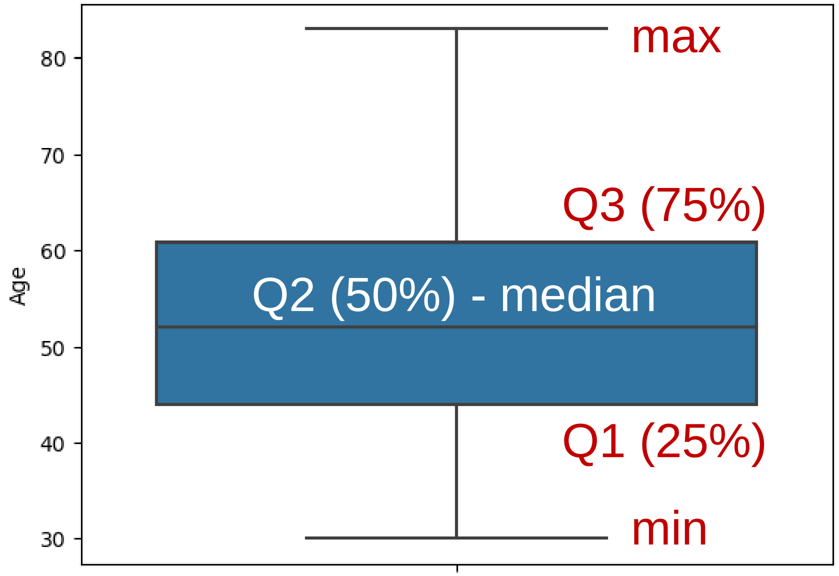
- *Percentile* is the percent of cases occurring at or below a score
- Example: You are the fourth tallest person in a group of 20
  – 80% of people are shorter than you:



  – That means you are at the **80th percentile**.
    - If your height is 1.85m then "1.85m" is the 80th percentile height in that group.
- Q1 = 25th percentile = 1st quartile
- Q2 = 50th percentile (*Median*) = 2nd quartile
- Q3 = 75th percentile = 3rd quartile
- Q4 = 100th percentile = 4th quartile

Box plots (U) – Seaborn

- boxplot() function is available in the seaborn library
  - data parameter: dataset for plotting
  - x, y, hue parameters: names of features (variables) in data
- Box plots offer univariate analysis when we are exploring one variable, however, multivariate analysis can be performed (see next slides)

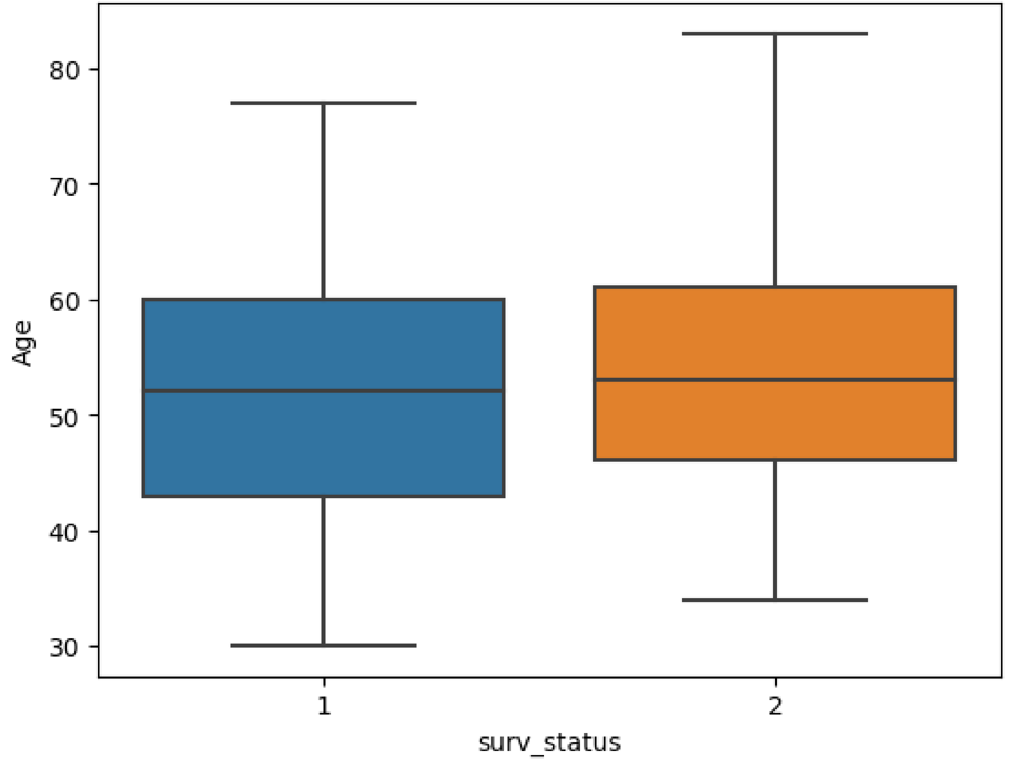


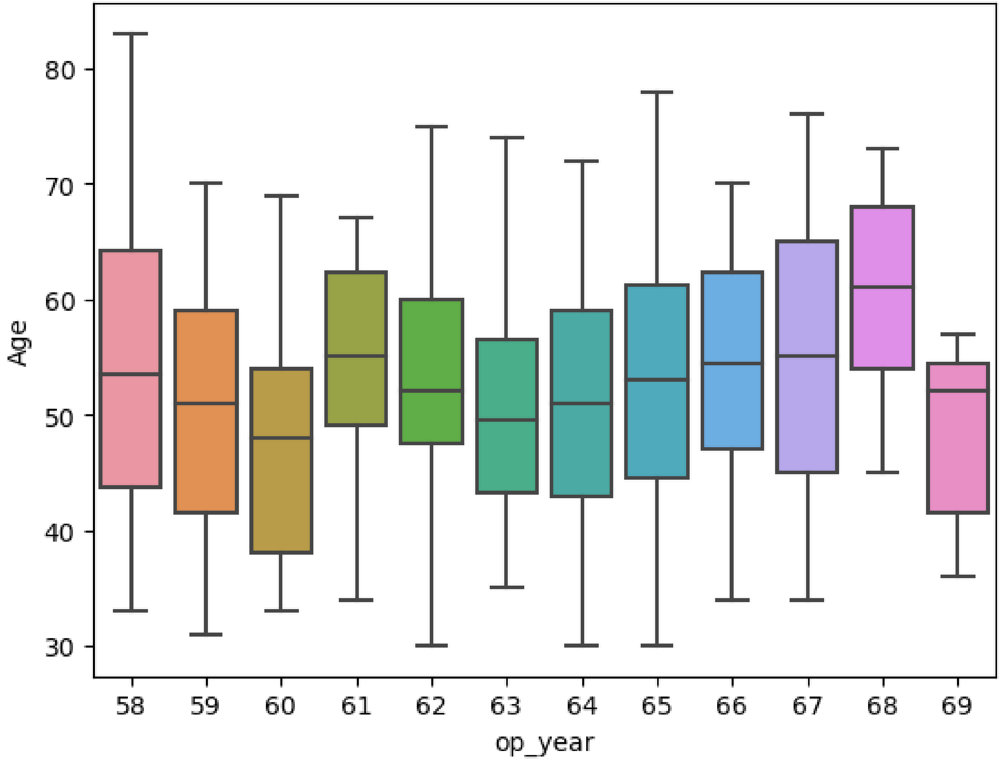

```
fig, ax = plt.subplots()
sns.boxplot(data=df1, y='age')
ax.set_ylabel('Age')
```

```
print(df1['age'].describe())
count    306.000000
mean      52.457516
std       10.803452
min       30.000000
25%       44.000000
50%       52.000000
75%       60.750000
max       83.000000
```

Box plots (B) – Seaborn

- Assign a variable to x-axis to examine the statistical details for a combination of two variables



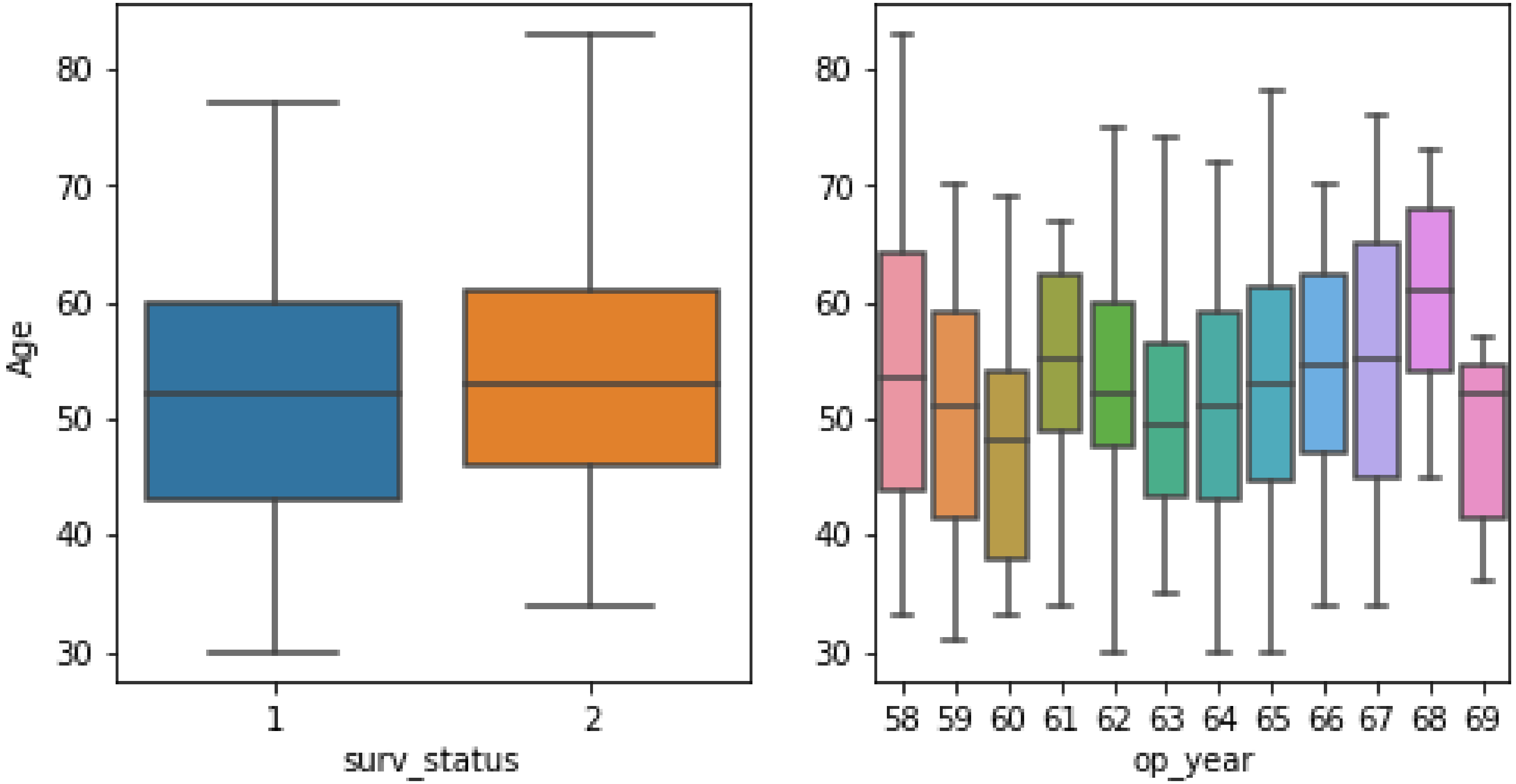



```
fig, ax = plt.subplots()
sns.boxplot(data=df1, y='age', x='surv_status')
ax.set_ylabel('Age')
```

```
fig, ax = plt.subplots()
sns.boxplot(data=df1, y='age', x='op_year')
ax.set_ylabel('Age')
```

Box plots (B) – Seaborn

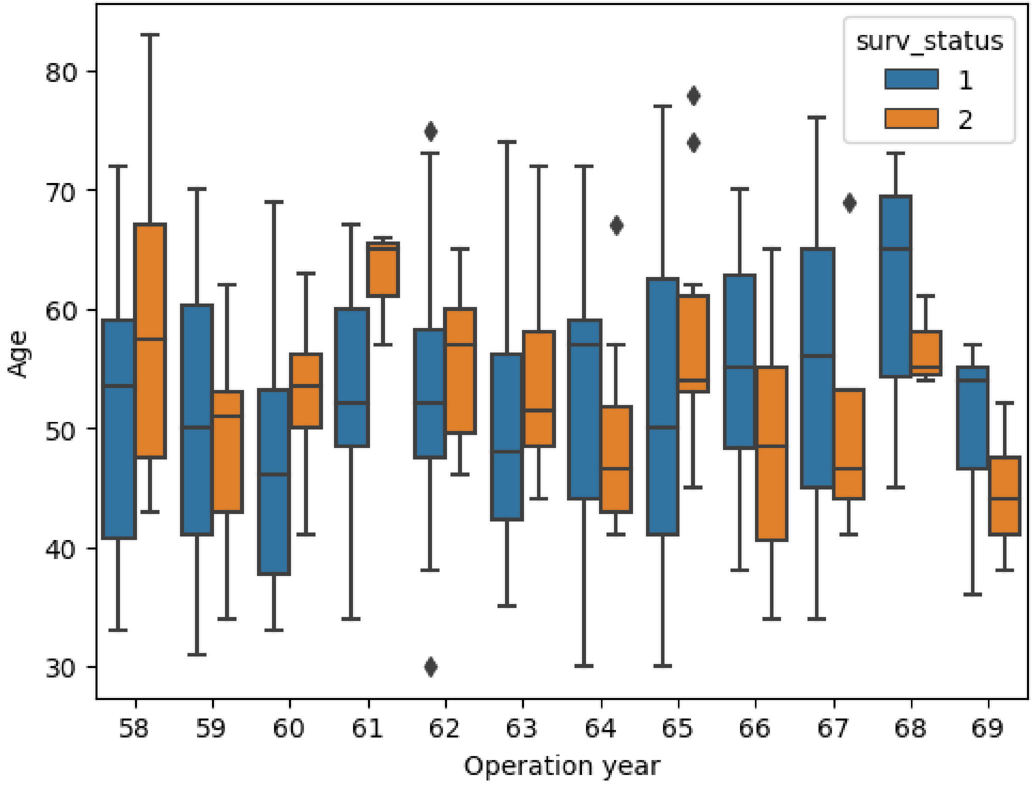
- Print both plots on the same figure




```
fig, axs = plt.subplots(1,2,figsize=(8, 4))
sns.boxplot(data=df1, y='age', x='surv_status', ax=axs[0])
sns.boxplot(data=df1, y='age', x='op_year', ax=axs[1])
axs[0].set_ylabel('Age')
axs[1].set_ylabel('')
```

Box plots (M) – Seaborn

- Assigning a variable to hue will draw a separate box plot for each of its unique values and distinguish them by color

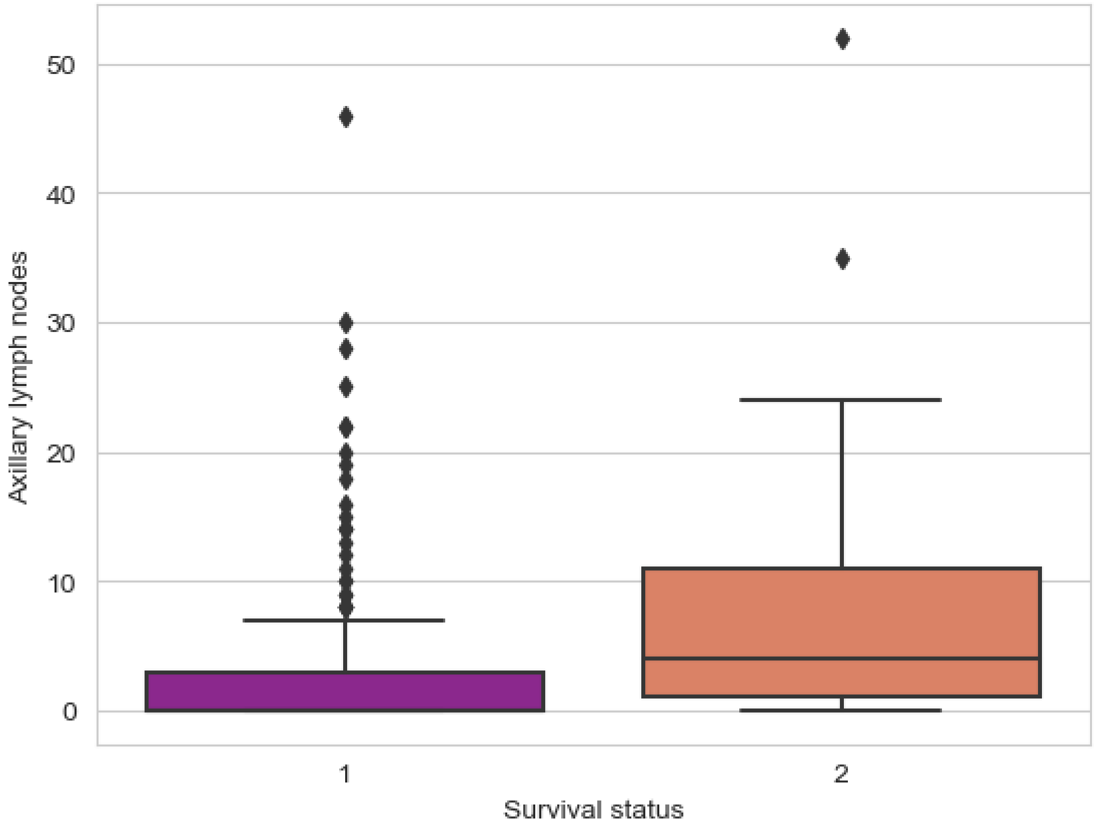



```
fig, ax = plt.subplots()
sns.boxplot(data=df1, y='age', x='op_year', hue='surv_status')
ax.set_xlabel('Operation year')
ax.set_ylabel('Age')
```

Box plots (B) for outlier detection

```
fig, ax = plt.subplots()
sns.boxplot(data=df1, y='axil_nodes', x='surv_status', palette='plasma')
ax.set_xlabel('Survival status')
ax.set_ylabel('Axillary lymph nodes')
```

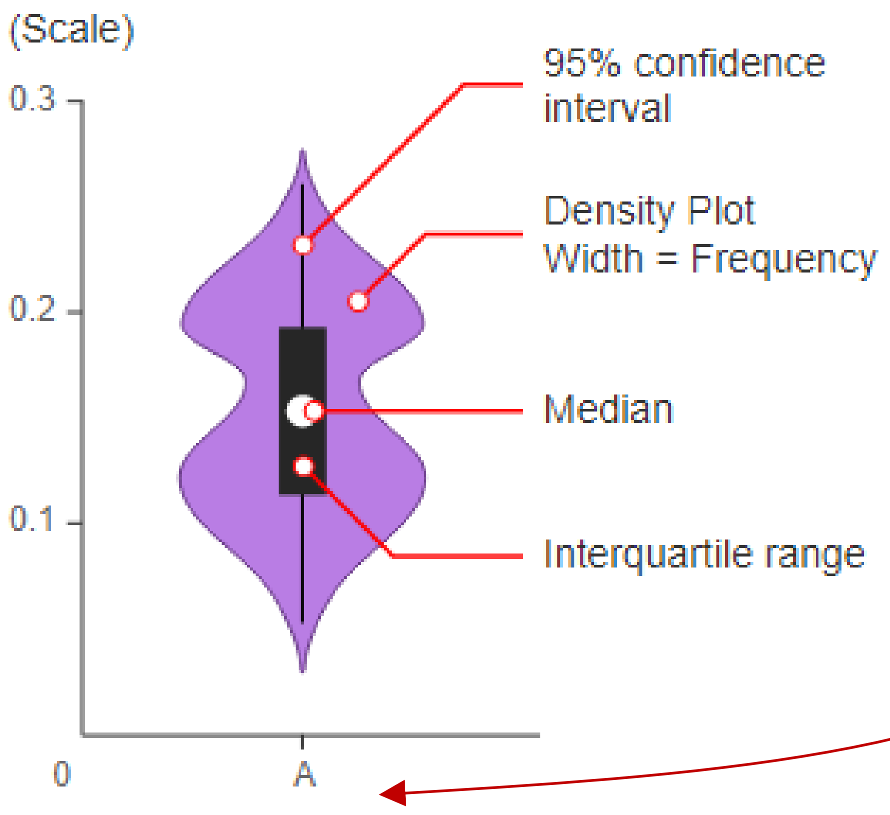
- Observations:
  - For the class 1 we can see that there are very few/no data is present between the 1st quartile and the median (2nd quartile)
  - High number of outlier points for class 1 in feature axillary nodes
    - Why is outlier observation important?
      - Many machine learning models, like linear & logistic regression, are easily impacted by the outliers in the training data.
      - Models like AdaBoost increase the weights of misclassified points on every iteration and therefore might put high weights on these outliers as they tend to be often misclassified. This can become an issue if that outlier is an error of some type, or if we want our model to generalize well and not care for extreme values.



Violin plots (U/B/M) – Seaborn

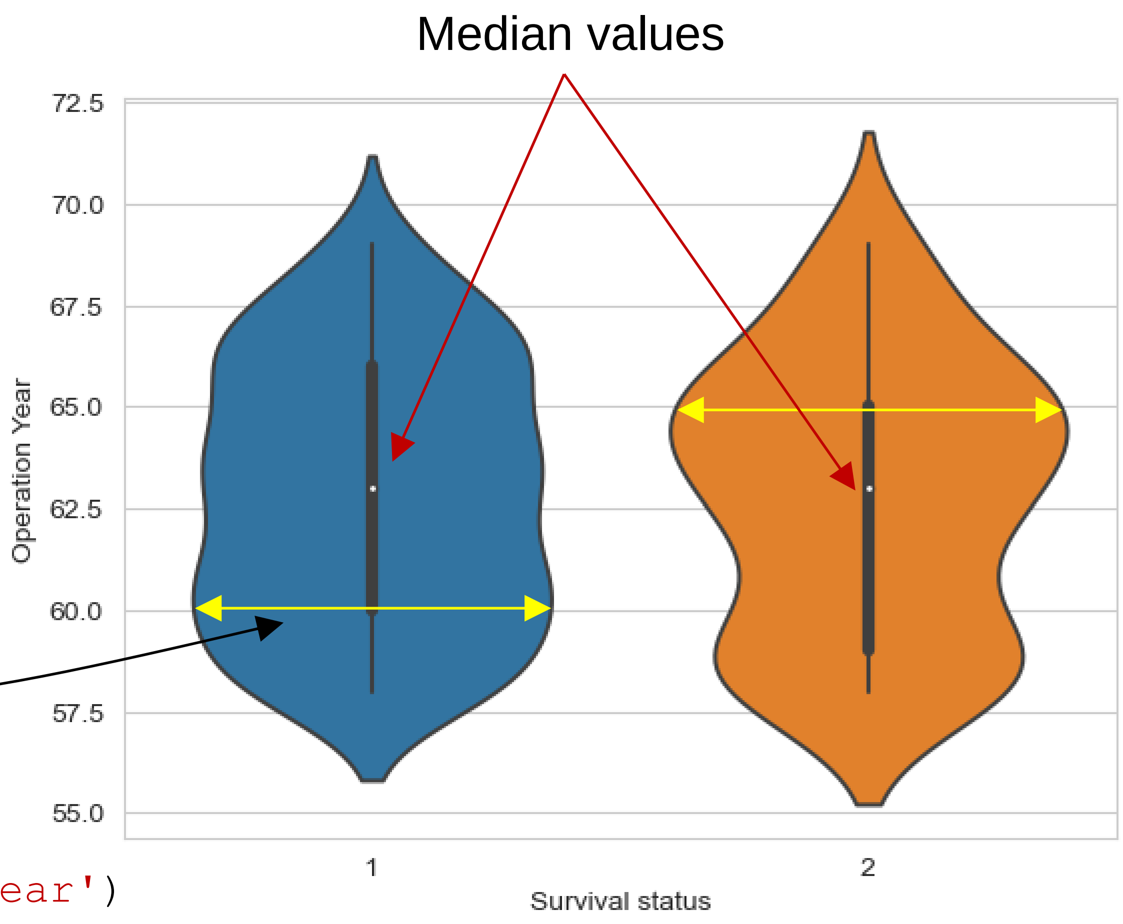
- Shows the same summary statistics (median, IQR) as box plots
- Also show shape/distribution of a single numerical feature across several levels of one (or more) categorical (target) variables

Values of a single numerical feature



Violin plots (U/B/M) – Seaborn

- Median values of both the classes are around 63
- The majority of patients from class 2 has operation year value equal to 65
- The majority of patients from class 1 has operation year value equal to 60




```
fig, ax = plt.subplots()
sns.violinplot(data=df1, x='surv_status', y='op_year')
ax.set_xlabel('Survival status')
ax.set_ylabel('Operation Year')
```

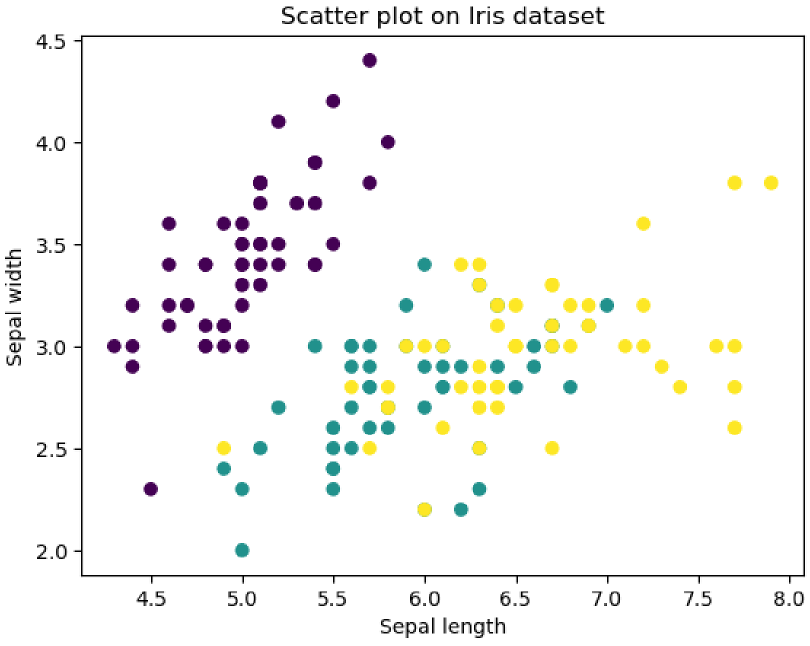
Scatter plot (B) – Matplotlib

- Gives a representation of where each point (observation) in the entire dataset is present with respect to any 2 or 3 features (dimensions).
  - Scatter plots are available in 2D as well as in 3D.
- 2D scatter plot is primarily used to find patterns/clusters and separability of the data. The code snippet for using a scatter plot from the Matplotlib library is shown below.

```
fig, ax = plt.subplots()
ax.scatter(df2['sepal_length_(cm)'],df2['sepal_width_(cm)'],c=df2['target'])
ax.set_xlabel('Sepal length')
ax.set_ylabel('Sepal width')
ax.set_title('Scatter plot on Iris dataset')
```

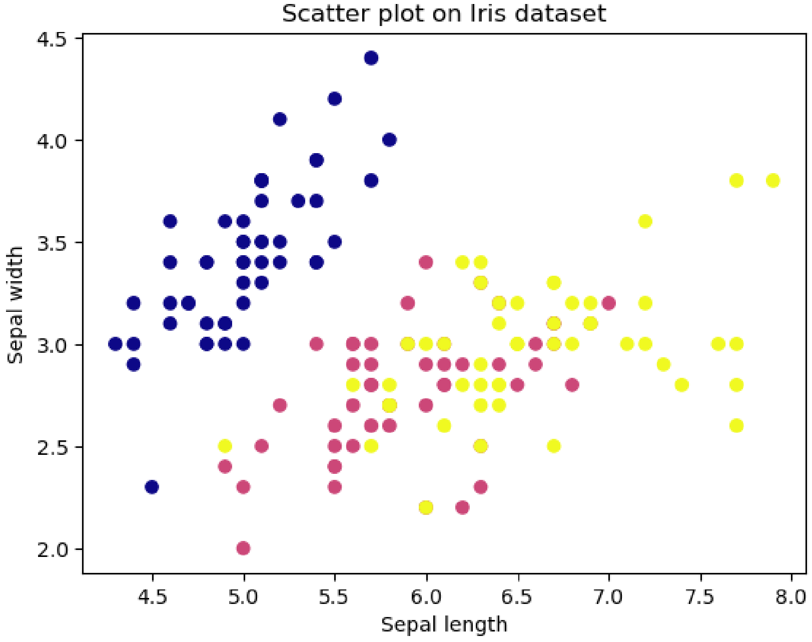
Scatter plot (B) – Matplotlib

```
fig, ax = plt.subplots()
ax.scatter(df2['sepal_length_(cm)'],df2['sepal_width_(cm)'],c=df2['target'])
```



The parameter **`c`** (or **`color`**) decides the color of each datapoints (it is a sequence of colors or sequence of numbers to be mapped to colors). Here we use the target column (the species class) in c, so that we got this plot colored in this manner.

```
fig, ax = plt.subplots()
ax.scatter(df2['sepal_length_(cm)'],df2['sepal_width_(cm)'],c=df2['target'],cmap='plasma')
```



`cmap`: colormap instance used to map data values from the interval [0,1] to RGBA colors that the respective Colormap represents. <u>It is only used if **`c`** is an array of numbers.</u>

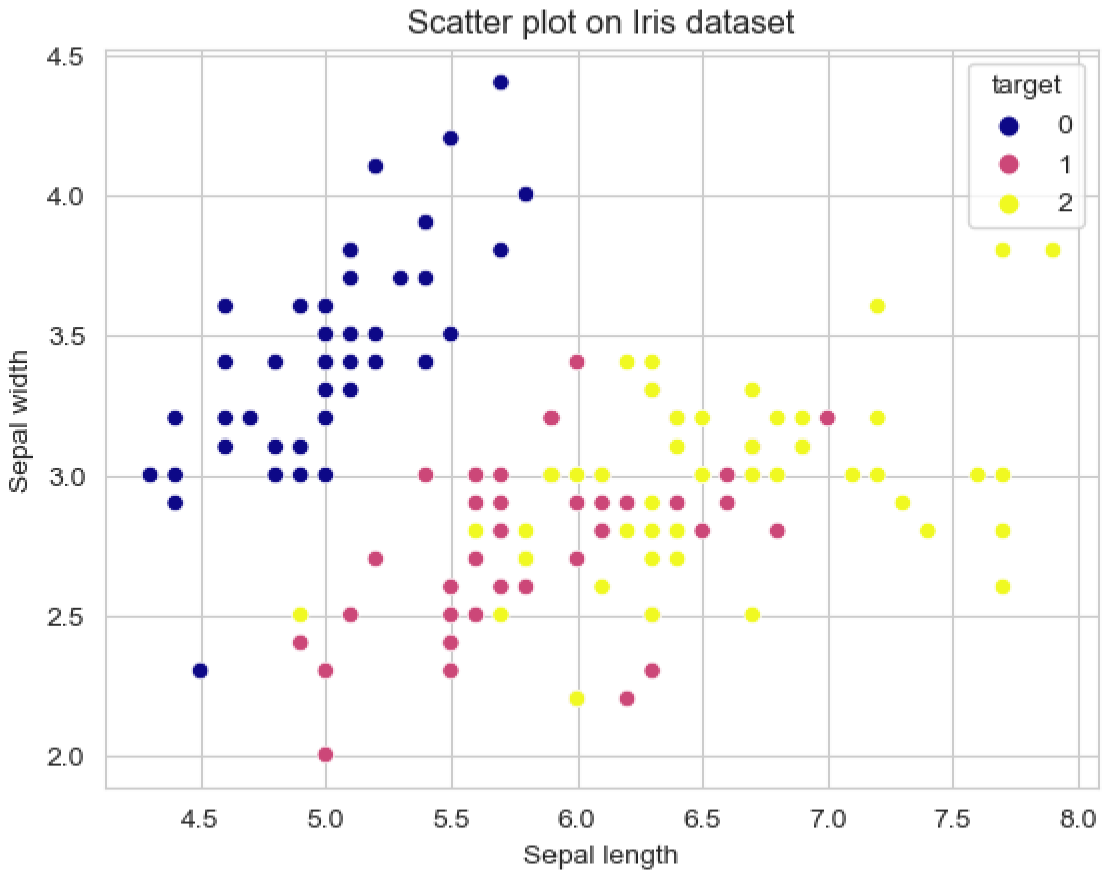
Colormap examples: `'viridis', 'plasma', 'inferno', 'magma', 'cividis'`

See more here: https://matplotlib.org/stable/tutorials/colors/colormaps.html

Scatter plot (B) – Seaborn

```
fig, ax = plt.subplots()
sns.scatterplot(data=df2, x = "sepal_width_(cm)", y = "sepal_length_(cm)", hue =
"target", palette="plasma")
```

Parameter **`palette`** decides the colormap. Same as **`cmap`** in matplotlib.



Parameter **`hue`** decides the color of each datapoints. Same as **`c`** in matplotlib

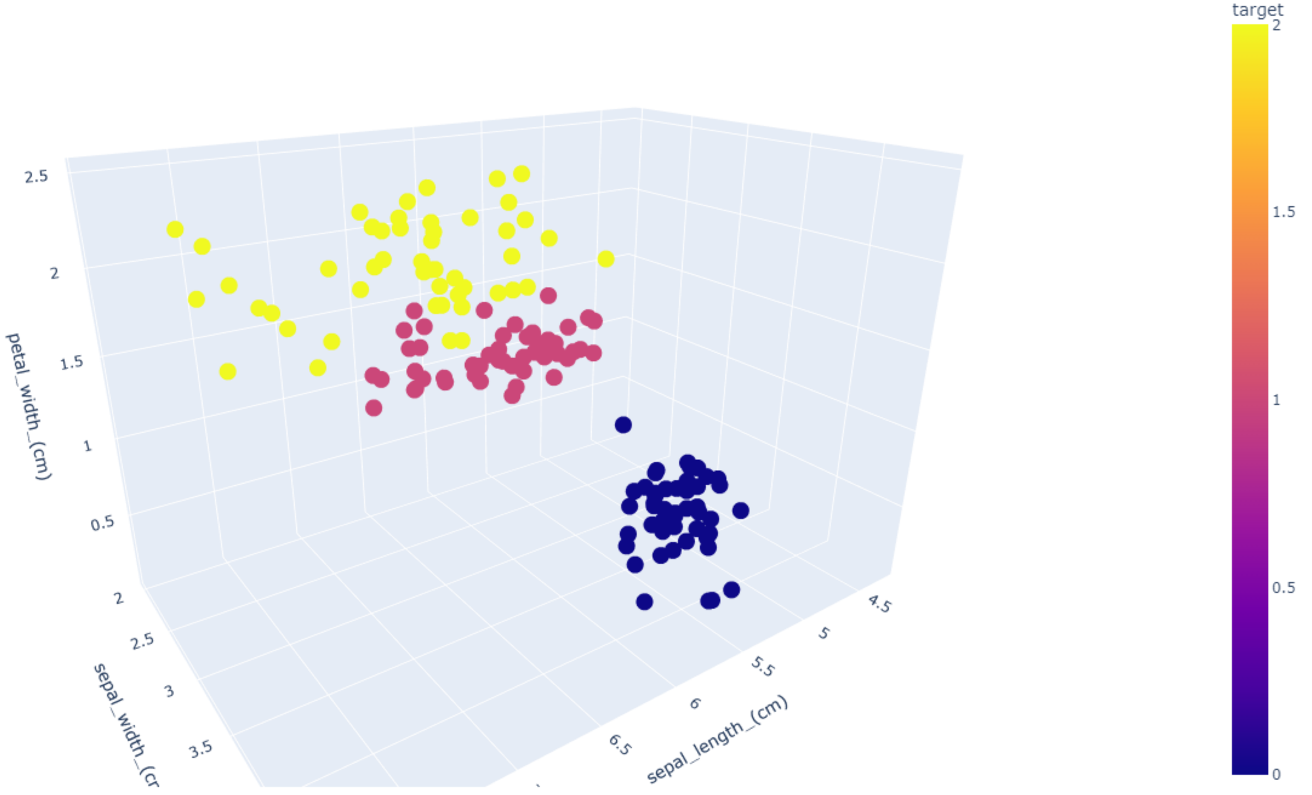
3D Scatter plot (M) – Plotly

- 3D scatter plot with Plotly Express

```
import plotly.express as px
fig = px.scatter_3d(df2, x='sepal_length_(cm)', y='sepal_width_(cm)',
z='petal_width_(cm)', color='target')
fig.show()
```

Parameter **`color`** decides the color of each datapoints.
Same as **`c`** in matplotlib and **`hue`** in seaborn

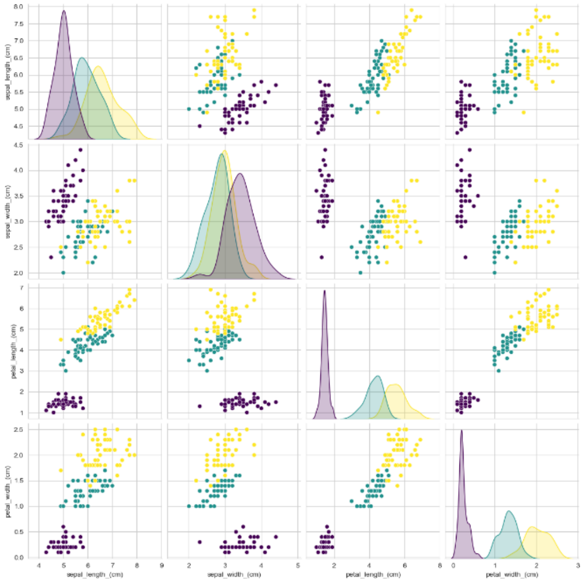
Not pre-installed in Anaconda:
`conda install -c plotly plotly`



Pair plots (M) – Seaborn

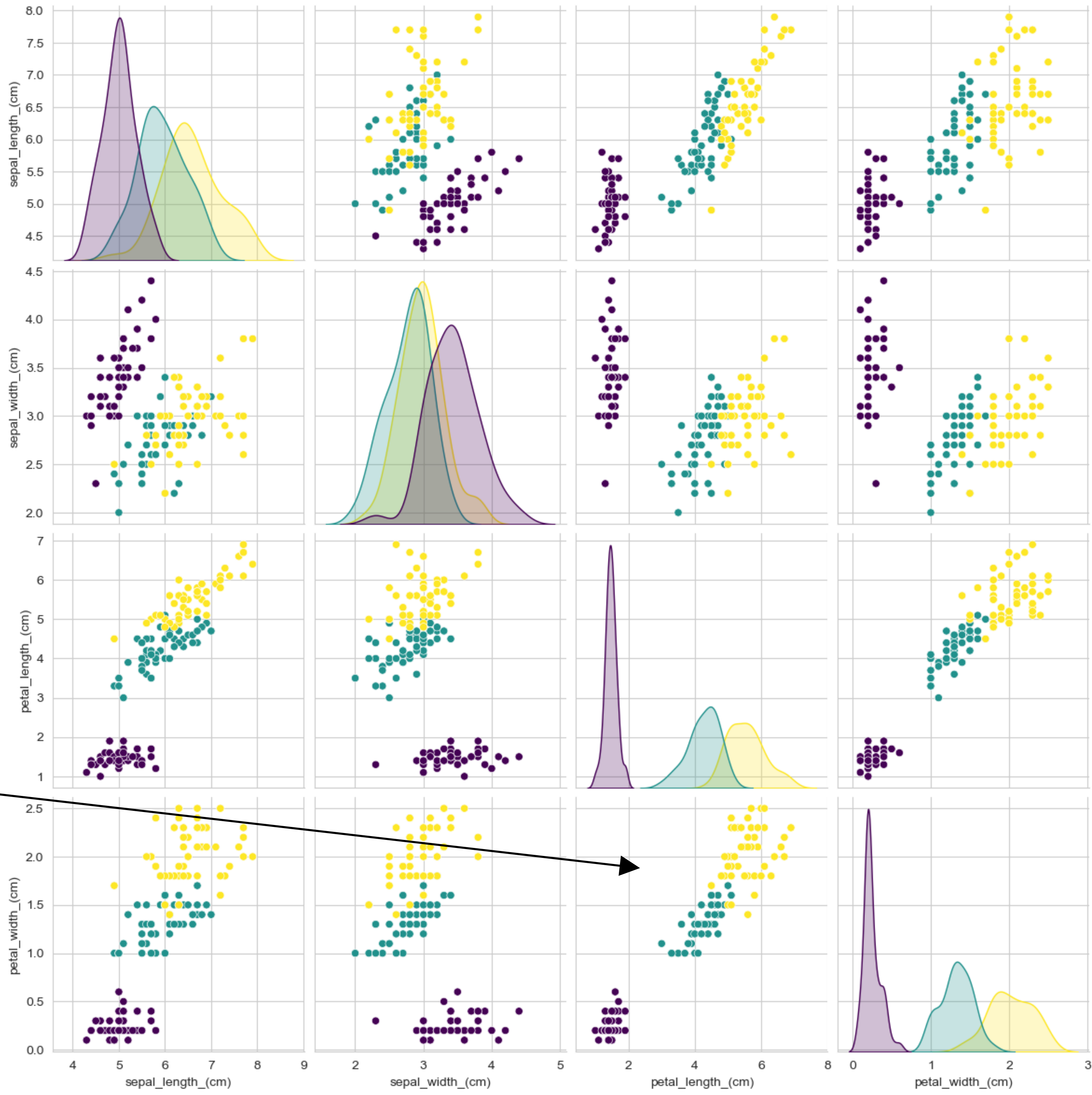
- Pair plot from seaborn: for scatter plots 4D and over
- For n features, pair plot will create a n x n figure where the diagonal plots will be univariate distribution plot of the feature corresponding to that row and rest of the plots are the combination of features from each row in y axis and feature from each column in x axis.

```
sns.pairplot(data=df2, hue='target', palette='viridis')
```



Pair plots (M) – Seabo

- We can observe which 2 features can well explain/separate the data ➔ then we can use scatter plot between those 2 features to explore further
- It seems petal length and petal width are the 2 features which can separate the data very well

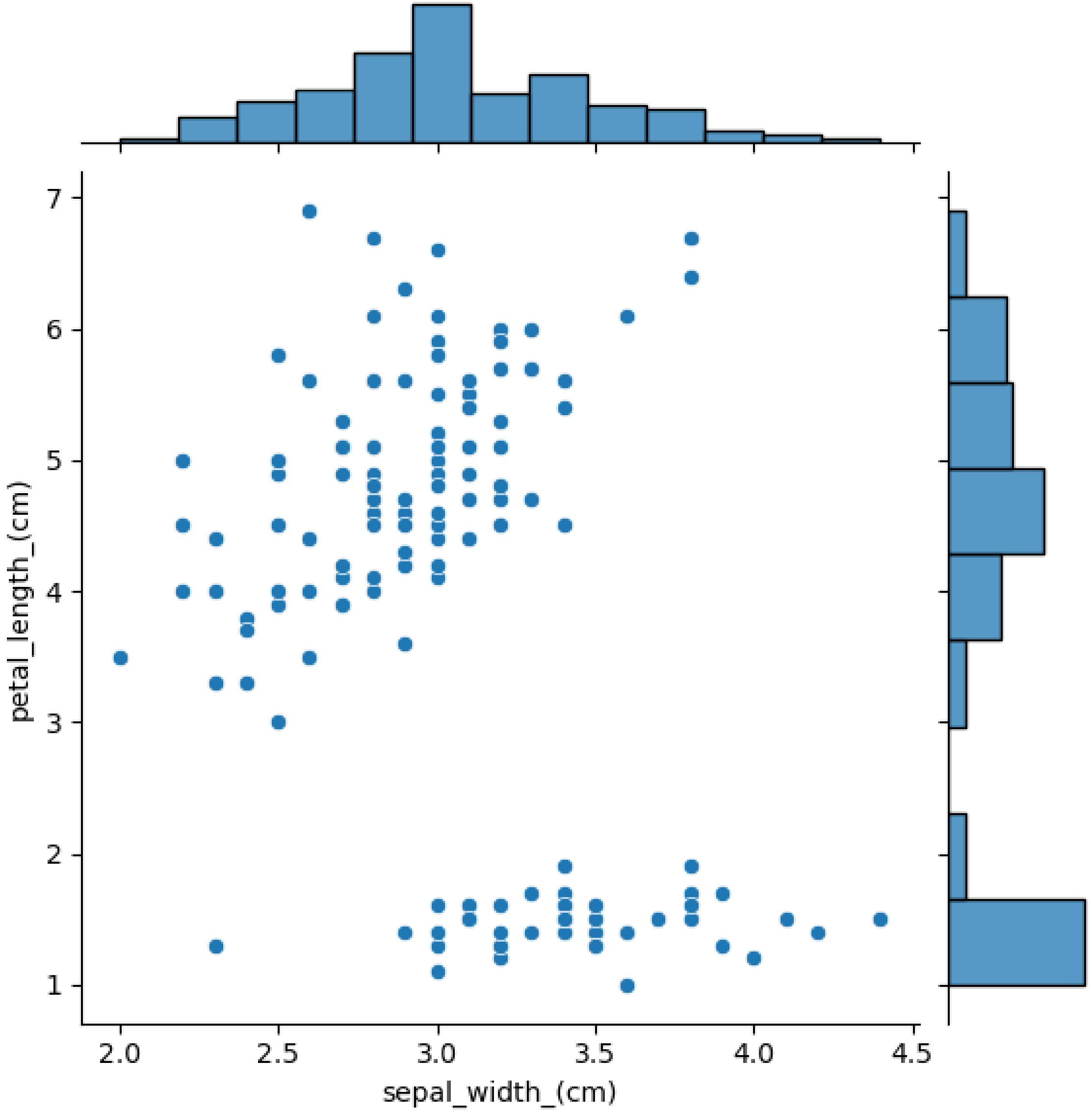


Pair plots (M) – Seaborn

- Pair plot produces `n x n` plots for `n` features
- Pair plot may become complex when we have high number of features (dimensions) say like 10 or so on.
- In such cases, a dimensionality reduction technique can be used to map data into 2d plane (by eliminating not “important” features) and visualizing it using a 2d or 3d scatter plot.

Joint plot (U/B) – Seaborn

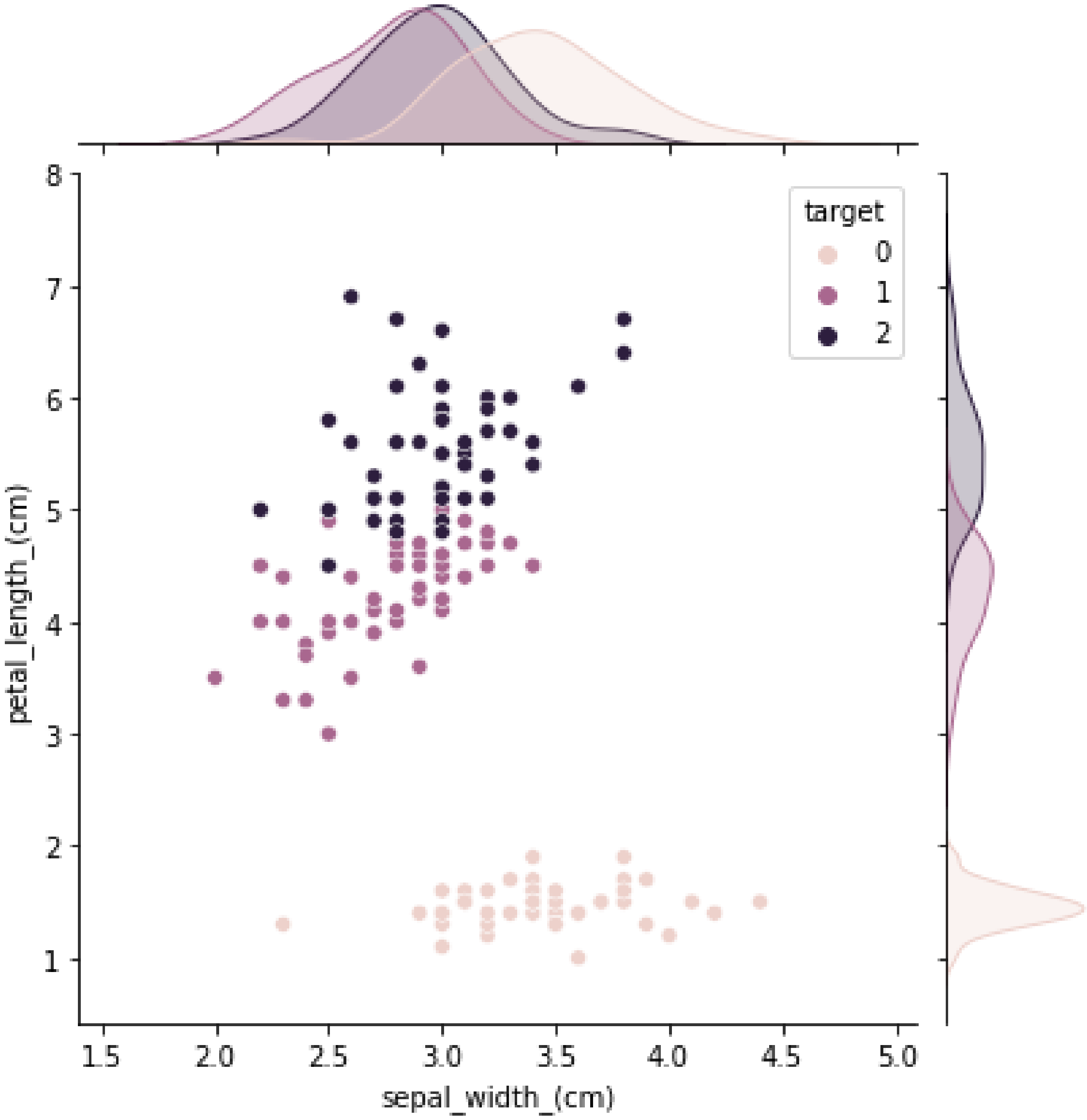
- Seaborn provides jointplot()
- Central plot involves bivariate analysis whereas on the top and right side provides univariate plots of both variables
  - By default, jointplot() represents the bivariate distribution using scatterplot() and the marginal distributions using histplot()




```
sns.jointplot(data=df2, x='sepal_width_(cm)', y='petal_length_(cm)')
```

Joint plot (U/B) – Seaborn

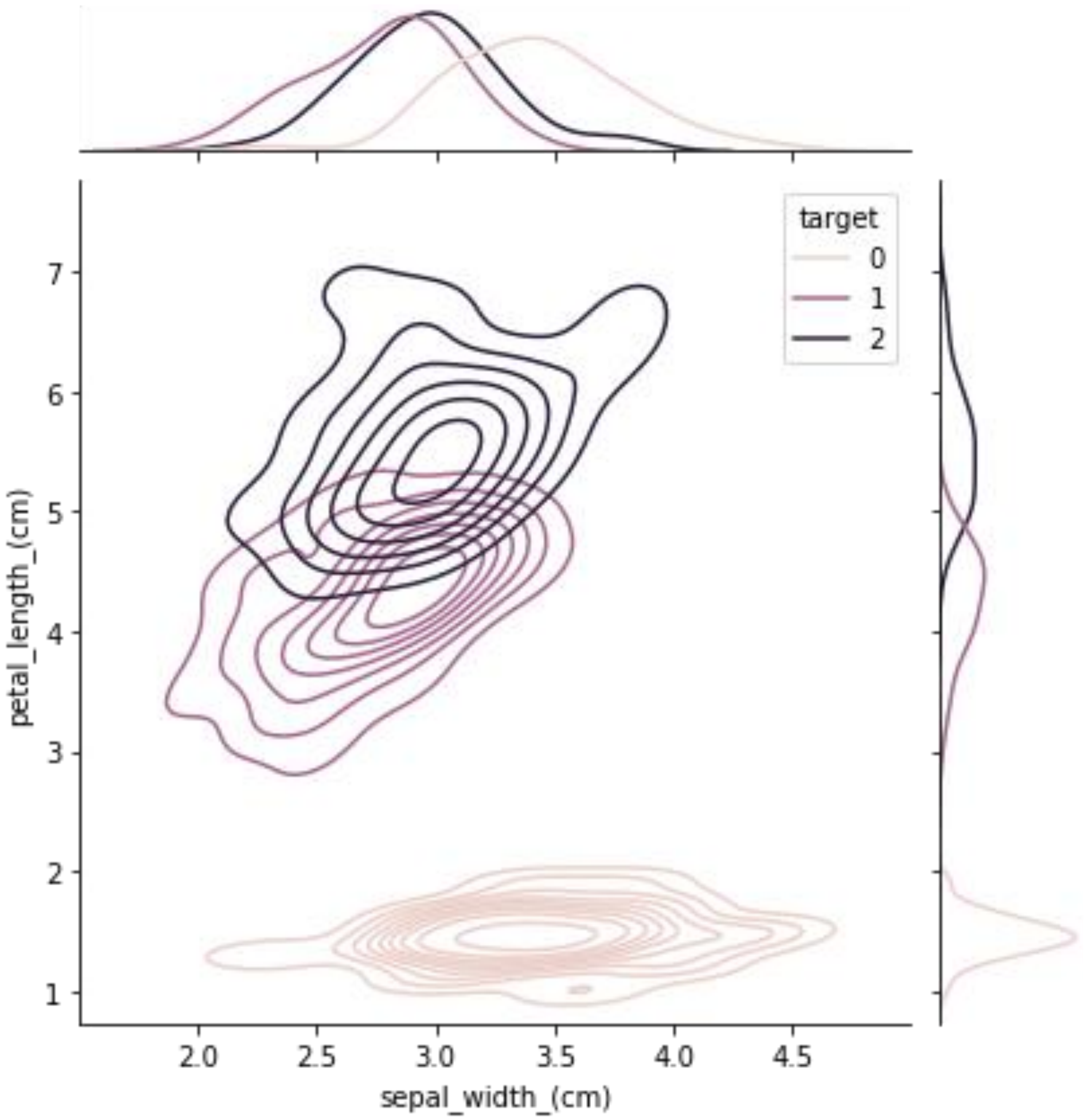
- Assigning a hue variable will add conditional colors to the scatterplot and draw separate density curves on the marginal axes




```
sns.jointplot(data=df2, x='sepal_width_(cm)', y='petal_length_(cm)', hue='target')
```

Joint plot (U/B) – Seaborn

- Several different approaches to plotting are available through the kind parameter.
- By setting kind='kde' will draw both bivariate and univariate KDEs




```
sns.jointplot(data=df2, x='sepal_width_(cm)', y='petal_length_(cm)', hue='target',
kind='kde')
```

Bar plot (B) – Seaborn

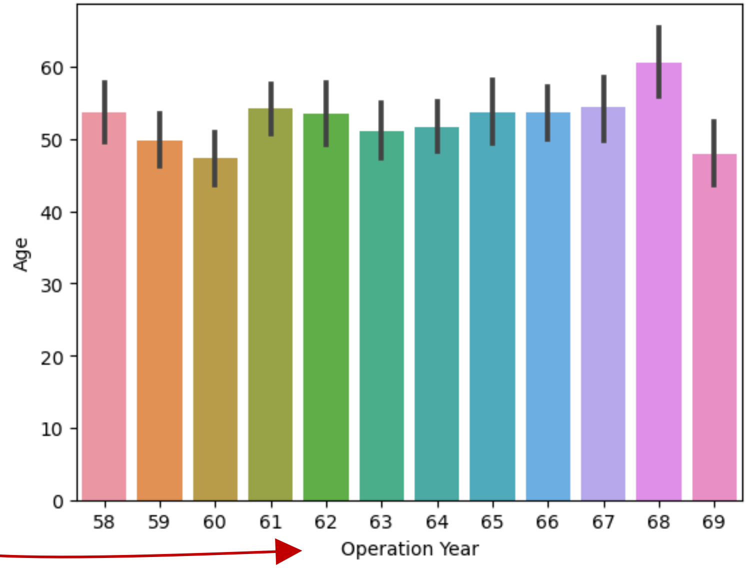
- Presents a categorical (or discrete numerical) variable with rectangular bars with heights / lengths proportional to a statistical measure (mean, sum, median) of a numerical variable
- The size of the bar represents a numeric value of that category
  - numeric value is estimated by aggregating across multiple observations of the y (numeric) variable at the same x (categorical) level – default is mean
  - indication of uncertainty (variation) around that value provided using error bars
    - Can be disabled using parameter ci=None

	age	op_year	axil_nodes	surv_status
0	30	64	1	1
1	30	62	3	1
2	30	65	0	1
3	31	59	2	1
4	31	65	4	1

category

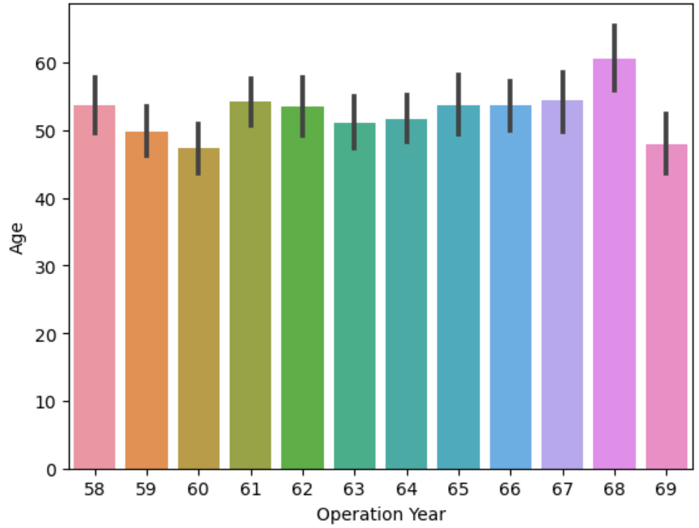
```
y='age', x='op_year'
```

Aggregate (group) by op_year and estimate the mean value of aggregated ages



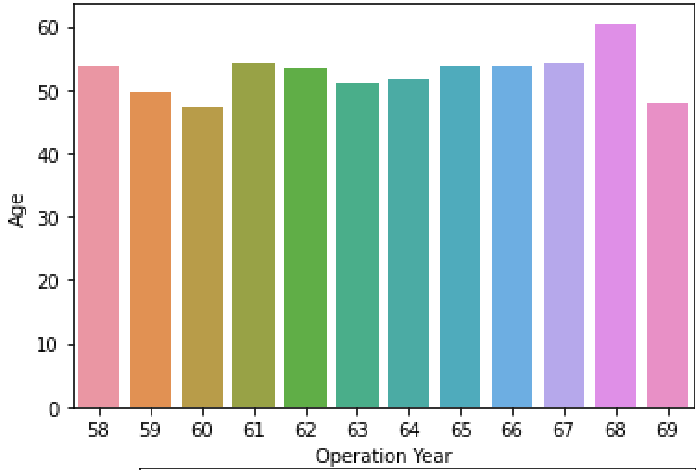
Bar plot (B) – Seaborn

```
fig, ax = plt.subplots()
sns.barplot(data=df1, y='age', x='op_year')
ax.set_xlabel('Operation Year')
ax.set_ylabel('Age')
```



Without variation

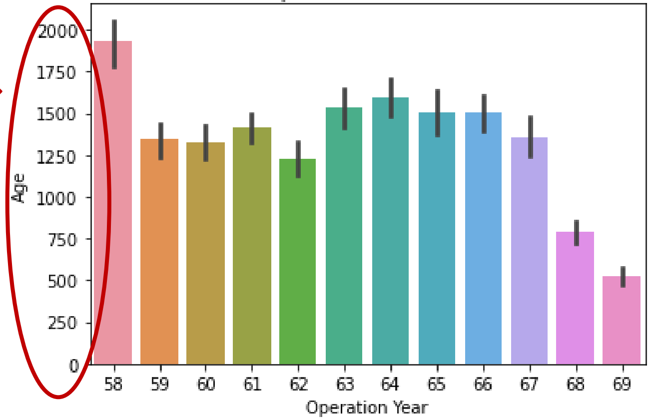
```
fig, ax = plt.subplots()
sns.barplot(data=df1, y='age', x='op_year', errorbar=None)
ax.set_xlabel('Operation Year')
ax.set_ylabel('Age')
```



Aggregation method parameter (default is mean)

```
fig, ax = plt.subplots()
sns.barplot(data=df1, y='age', x='op_year', estimator=np.sum)
ax.set_xlabel('Operation Year')
ax.set_ylabel('Age')
```

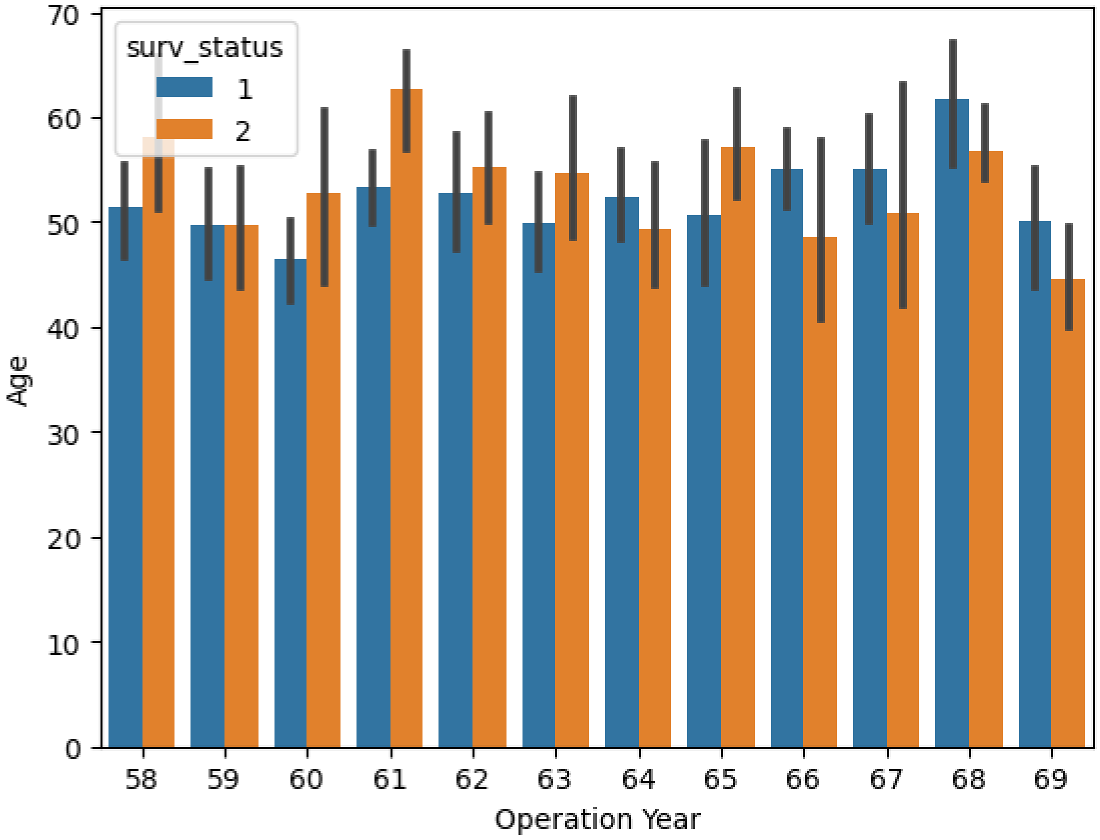
```
import numpy as np
```



Bar plot (M) – Seaborn

- Assigning a variable to hue will draw a separate bar for each of its unique values and distinguish them by a different color

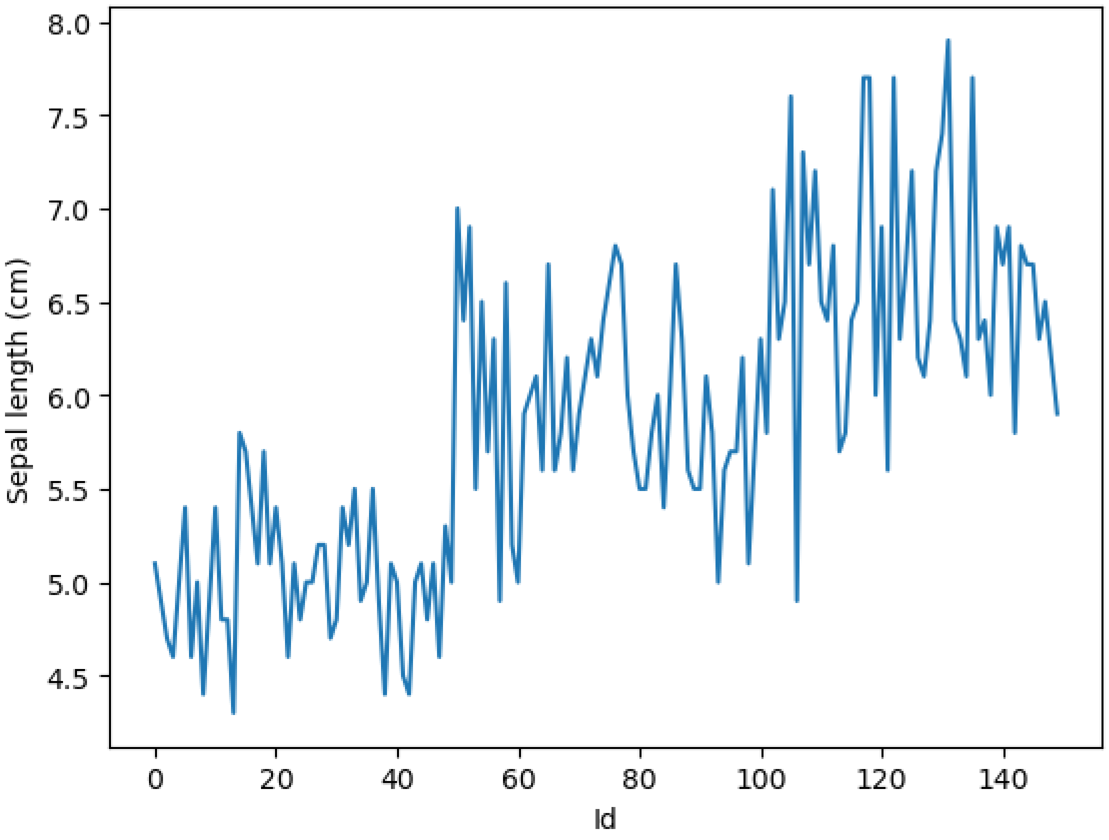
```
fig, ax = plt.subplots()
sns.barplot(data=df1, y='age', x='op_year',
hue='surv_status')
ax.set_xlabel('Operation Year')
ax.set_ylabel('Age')
```



Line plot (U) – Matplotlib & Seaborn

- A graph that displays data as points on a number line
- For variables (features) that can be ordered across another variable
  - Useful for timeseries data, where x-axis is a time-dependent variable (i.e. date)
- `plt.plot()` in Matplotlib
- `sns.lineplot()` in Seaborn

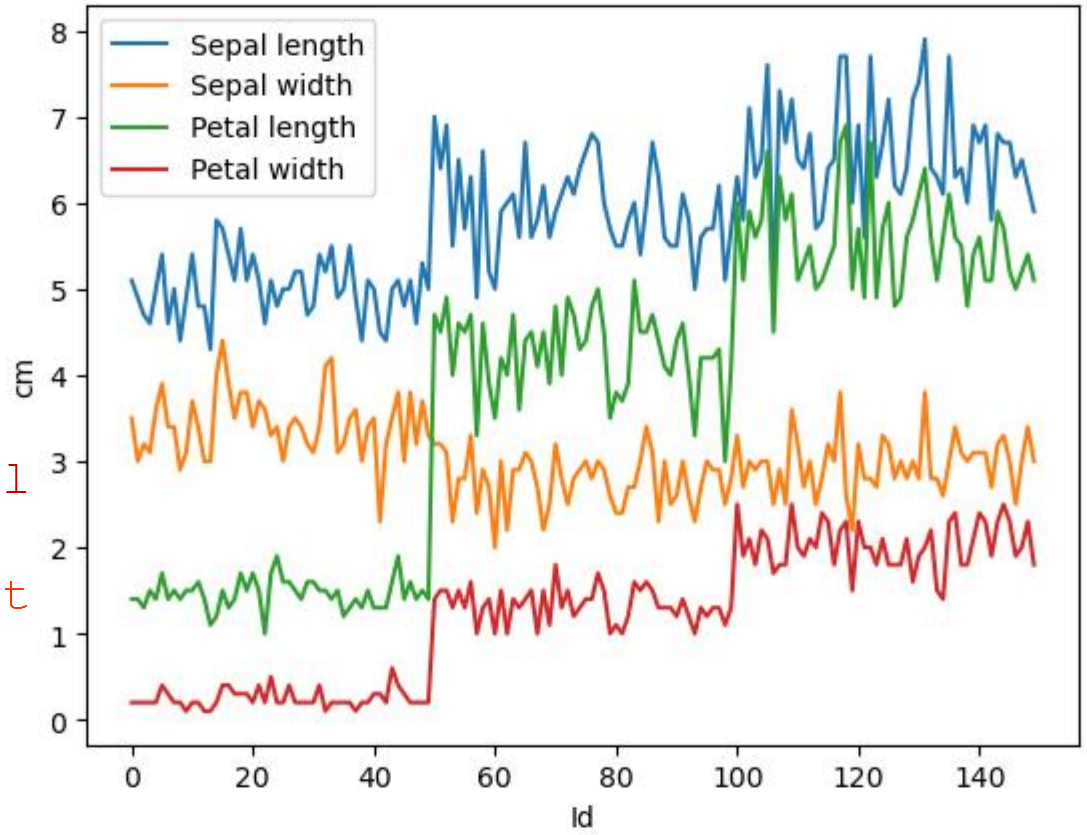
```
fig, ax = plt.subplots()
ax.plot(df2.index, df2['sepal_length_(cm)'])
ax.set_xlabel('Id')
ax.set_ylabel('Sepal length (cm)')

fig, ax = plt.subplots()
sns.lineplot(data=df2, x=df2.index,
y='sepal_length_(cm)')
ax.set_xlabel('Id')
ax.set_ylabel('Sepal length (cm)')
```



Line plot (U) – Matplotlib

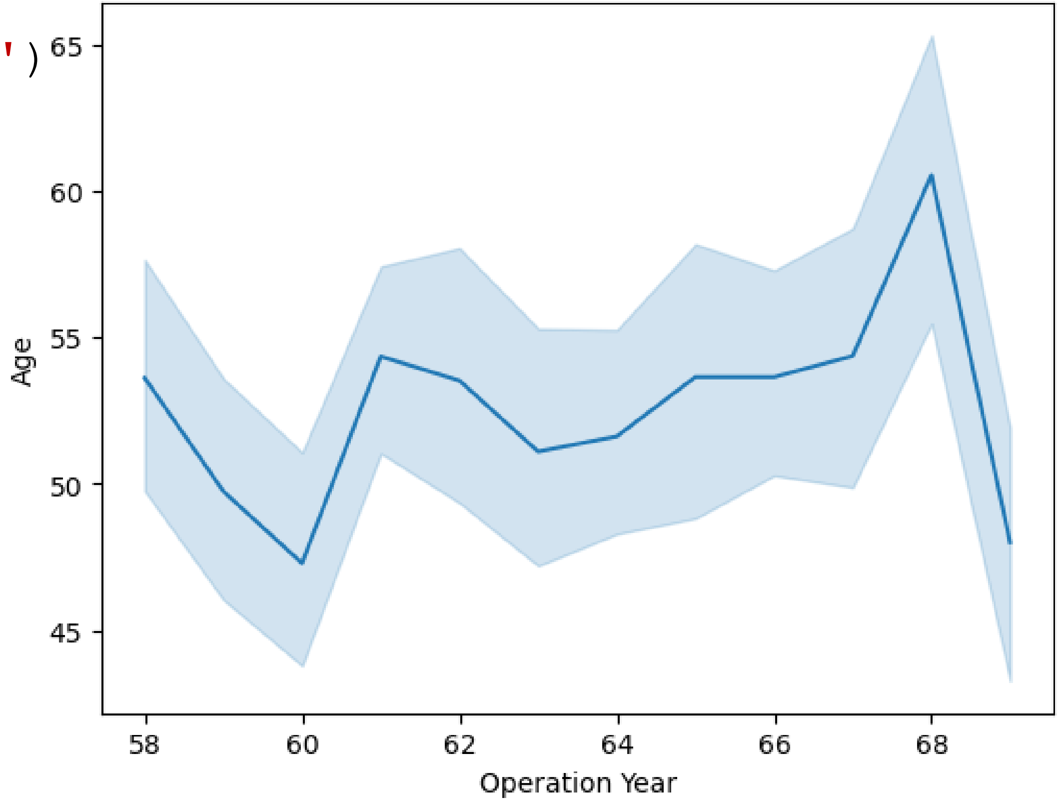
- We can plot multiple lines inside a single figure as shown below where you need to add multiple plt.plot() or sns.lineplot() commands with each line representing a different color parameter

```
fig, ax = plt.subplots()
ax.plot(df2.index, df2['sepal_length_(cm)'])
ax.plot(df2.index, df2['sepal_width_(cm)'])
ax.plot(df2.index, df2['petal_length_(cm)'])
ax.plot(df2.index, df2['petal_width_(cm)'])
ax.set_xlabel('Id')
ax.set_ylabel('cm')
ax.legend(['Sepal length', 'Sepal width', 'Petal
length', 'Petal width'])
# you could set the label parameter on each plot
# in that case legend() should not take any
input parameters => ax.legend()
```



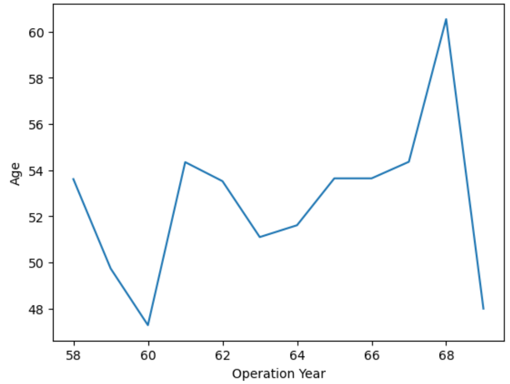
Line plot (B) – Seaborn

```
fig, ax = plt.subplots()
sns.lineplot(data=df1, x = 'op_year', y = 'age')
ax.set_xlabel('Operation Year')
ax.set_ylabel('Age')
```



- Uses `estimator` and `ci` parameters as in barplot
  - Aggregation over all ages for each operation year
    - Line goes through the mean values (since the default value for estimator is mean)
  - Confidence interval is drawn around the line (omitted if ci=None)

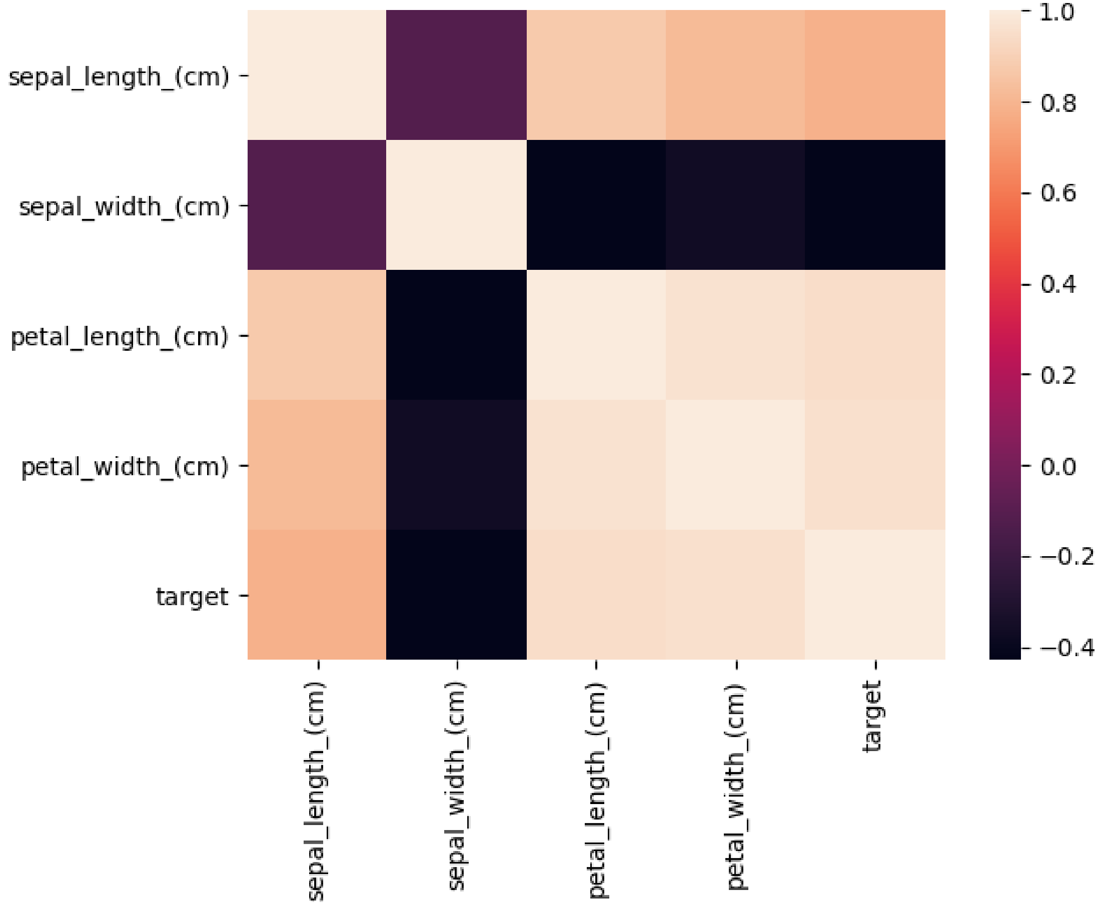
```
fig, ax = plt.subplots()
sns.lineplot(data=df1, x = 'op_year', y = 'age',
errobar=None)
ax.set_xlabel('Operation Year')
ax.set_ylabel('Age')
```



Heatmap (B) – Seaborn

- Heat Map can be used to visualize how two categorical variables relate to each other
  - Example: visualize the correlation between different features in the dataset

```
fig, ax = plt.subplots()
sns.heatmap(data=df2.corr())
```



Correlation

- Correlation methods measure the relationship between two variables
- The correlation coefficient can never be less than -1 or higher than +1
  - +1 = there is a perfect linear relationship between the variables
  - 0 = there is no linear relationship between the variables
  - -1 = there is a perfect negative linear relationship between the variables
    - Highly correlated features can be removed from dataset prior running into machine learning algorithms so as to make the learning algorithm faster
      - curse of dimensionality: less features usually mean high improvement in terms of speed
    - If speed is not an issue, perhaps don't remove these features right away. If you have correlated features but they are also correlated to the target (if target is numerical), you want to keep them
      - Some algorithms like Naive Bayes actually directly benefit from "positive" correlated features. And others like random forest may indirectly benefit from them.
    - Moral of the story, removing these features might be necessary due to speed, but remember that you might make your algorithm worse in the process

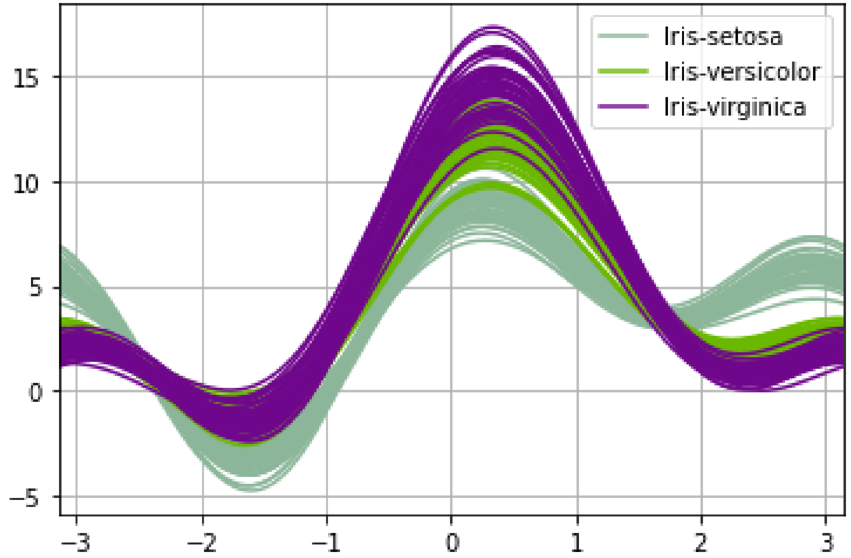
Andrew curves (M)

- representing multivariate data by curves
- useful tool for separating multivariate observation into groups that can not easily be distinguished in a tabular presentation
  - Check if observations are distinguishable on the basis of a given feature
- each multivariate observation (each line of file) $X_i = (X_{i,1}, X_{i,2}, \dots, X_{i,p})$, here p=4, is transformed (Fourier series transformation) into a curve as follows:

$$f_i(t) = \begin{cases} \frac{X_{i,1}}{\sqrt{2}} + X_{i,2}\sin(t) + X_{i,3}\cos(t) + \dots + X_{i,p-1}\sin(\frac{p-1}{2}t) + X_{i,p}\cos(\frac{p-1}{2}t) & \text{for } p \text{ odd} \\ \frac{X_{i,1}}{\sqrt{2}} + X_{i,2}\sin(t) + X_{i,3}\cos(t) + \dots + X_{i,p}\sin(\frac{p}{2}t) & \text{for } p \text{ even} \end{cases}$$

```
import pandas.plotting as pdplt
# andrews curves
pdplt.andrews_curves(df, 'class')
plt.show()
```

Data are grouped by this column



Parallel coordinates (M)

- allows to see clusters in data and to estimate other statistics visually
- each multivariate observation is represented (in parallel) by connected line segments
- each vertical line represents one feature
- points that tend to cluster will appear closer together

```
# parallel coordinates
pdplt.parallel_coordinates(df,'class')
plt.show()
```