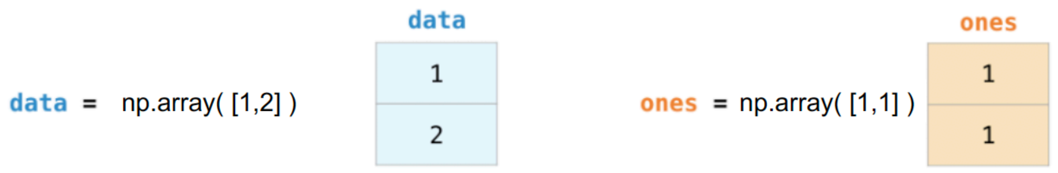
Class 3

July 28, 2025

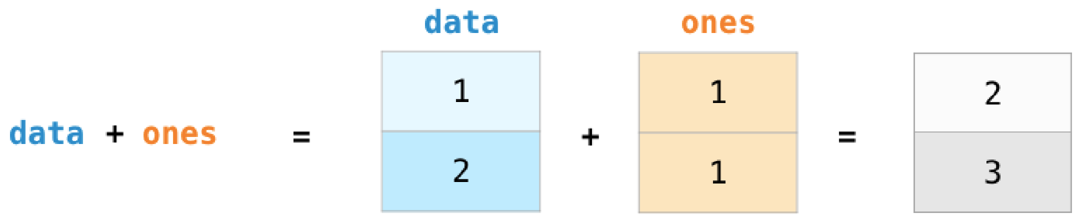
1 Review

Python Basic

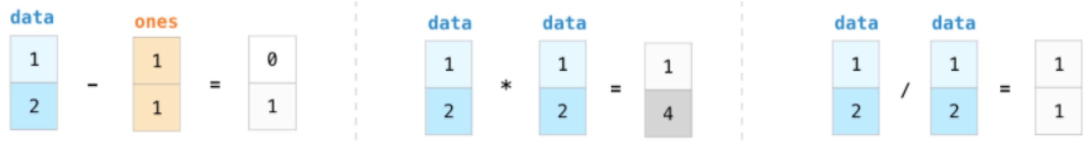
- Python as a calculator
- Variable assignment
- Standard data types
- Python lists (indexing and slicing)
- Packages
- Numpy: Numpy array is an alternative to Python list. It makes calculations over the entire arrays. Summary of array arithmetic



Adding them up element-wise (i.e. adding the values of each row) is as simple as typing `data + ones`



Of course, not just addition.



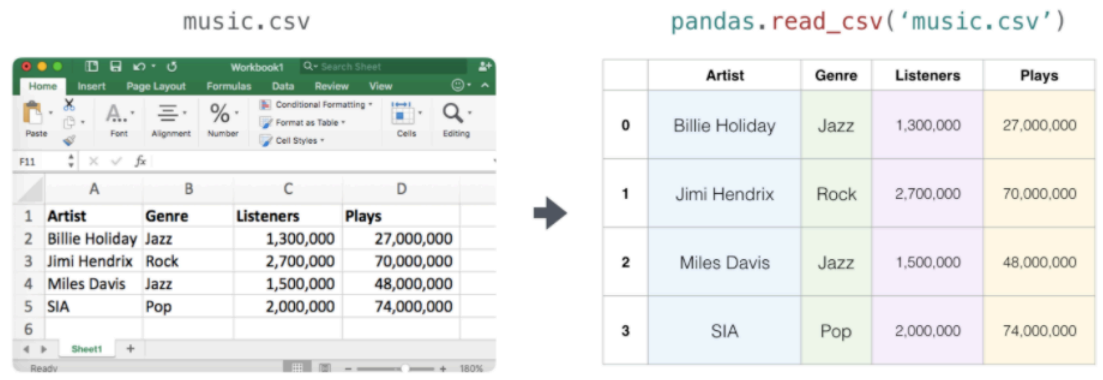
2 Data Representation

Think of all the data types you'll need to crunch and build models around (spreadsheets, images, audio, text…). So many of them are perfectly suited for representation in an n-dimensional array.

2.1 Tables and Spreadsheets

Think of the previous Iris data representation or the data representation of basketball players. It would be wonderful if we can refer to a column by its name.

A spreadsheet or a table of values is a two dimensional matrix. The most popular abstraction in python for those is the pandas dataframe, which actually uses NumPy and builds on top of it. If you would like to learn more about Pandas, you can start from here: https://colab.google/articles/pandas


music.csv

	A	B	C	D
1	Artist	Genre	Listeners	Plays
2	Billie Holiday	Jazz	1,300,000	27,000,000
3	Jimi Hendrix	Rock	2,700,000	70,000,000
4	Miles Davis	Jazz	1,500,000	48,000,000
5	SIA	Pop	2,000,000	74,000,000
6				

pandas.read_csv('music.csv')

	Artist	Genre	Listeners	Plays
0	Billie Holiday	Jazz	1,300,000	27,000,000
1	Jimi Hendrix	Rock	2,700,000	70,000,000
2	Miles Davis	Jazz	1,500,000	48,000,000
3	SIA	Pop	2,000,000	74,000,000

First, we need to upload the dataset to the cloud's virtual machine. The module **files** provides functions for interacting with the Colab file system. Then, it calls the files.upload() function. This function opens a file upload dialog in the Colab notebook, allowing you to select files from your local machine to upload. The uploaded files are stored in the Colab environment's virtual machine. **OR** you can drag/upload the csv file to the file folder on google colab.

[ ]:
```
from google.colab import files
uploaded = files.upload()
```

Pandas allows us to load a spreadsheet and manipulate it programmatically in python. The central concept in pandas is the type of object called a DataFrame – basically a table of values which has a label for each row and column. Let's load this basic CSV file containing data from music.csv.

[3]:
```
import pandas as pd
df = pd.read_csv('music.csv')
df # Now the variable df is a pandas DataFrame
```

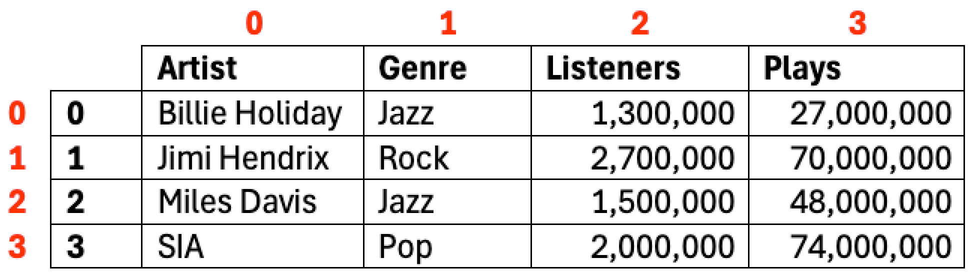
[3]:
```
           Artist Genre  Listeners     Plays
0  Billie Holiday  Jazz    1300000  27000000
1    Jimi Hendrix  Rock    2700000  70000000
2     Miles Davis  Jazz    1500000  48000000
3             SIA   Pop    2000000  74000000
```

`DataFrame.to_numpy()` gives a NumPy representation of the underlying data. Note that this can be an expensive operation when your DataFrame has columns with different data types, which comes down to a fundamental difference between pandas and NumPy: NumPy arrays have one dtype for the entire array, while pandas DataFrames have one dtype per column.

For DataFrame of all floating-point values, `DataFrame.to_numpy()` is fast and doesn't require copying data.

2.1.1 Index and Slicing by position

Select via the position of the passed integers using `iloc`.



		0	1	2	3
		Artist	**Genre**	**Listeners**	**Plays**
0	**0**	Billie Holiday	Jazz	1,300,000	27,000,000
1	**1**	Jimi Hendrix	Rock	2,700,000	70,000,000
2	**2**	Miles Davis	Jazz	1,500,000	48,000,000
3	**3**	SIA	Pop	2,000,000	74,000,000

[ ]:
```
df.iloc[3, :] # same as 2d numpy array
df.iloc[0:2, 1:3]
df.iloc[2, :3]
df.iloc[ [1,3,2] , :3]
```

2.1.2 Index and Slicing by label/name

We can also select any column or row using its row and column name, by using `loc`, but `loc` may also be used with a boolean array..

[ ]:
```
df.columns
df.loc[3,'Artist']
df.loc[1:3,'Artist']  # since now 1,2,3 are labels, *both* endpoints are␣
 ↪included:
df.loc[2, ['Artist','Plays'] ]
df.loc[:, ['Artist','Plays'] ]

df[ ['Artist','Plays'] ]
df[ 'Artist' ]
type( df['Artist'] ) # a Series, which is a one-dimensional array-like object␣
 ↪containing the data and label/name.

# `loc` can be used with boolean arrays to filter rows based on conditions.
df.loc[ df['Genre'] == 'Jazz' , :]
df.loc[ df['Genre'] == 'Jazz', ['Artist', 'Plays'] ]
```

2.1.3 Practice

We will use the Iris data, where each sample is one of three types of flowers that has had the size of its petals and sepals carefully measured. This is perhaps the best known database to be found in the pattern recognition and data mining literature. This dataset contains 150 different iris plants, with the information of its sepal length, sepal width, petal length and petal width.

```
[2]: df = pd.read_csv('iris.csv')
     df
```

```
[2]:      SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm         Species
     0              5.1           3.5            1.4           0.2     Iris-setosa
     1              4.9           3.0            1.4           0.2     Iris-setosa
     2              4.7           3.2            1.3           0.2     Iris-setosa
     3              4.6           3.1            1.5           0.2     Iris-setosa
     4              5.0           3.6            1.4           0.2     Iris-setosa
     ..             …             …              …             …             …
     145            6.7           3.0            5.2           2.3  Iris-virginica
     146            6.3           2.5            5.0           1.9  Iris-virginica
     147            6.5           3.0            5.2           2.0  Iris-virginica
     148            6.2           3.4            5.4           2.3  Iris-virginica
     149            5.9           3.0            5.1           1.8  Iris-virginica

     [150 rows x 5 columns]
```

Questions:

1. Print out the 3rd iris plant.
2. Make a new variable, sepal_width, containing the entire second column of iris dataframe.
3. What is the petal length of the 95th iris plant?

Take-home questions:

4. Find all iris plants whose species are "Iris-versicolor".
5. Find out the average petal length of all versicolors using `np.mean()`.

3 Data Visualization

Data visualization is a key part of any data science workflow. As the cliché goes, a picture is worth a thousand words.

Data exploration should really be part of your data analysis from the very beginning, as there is a lot of value and insight to be gained from just looking at your data. Summary statistics often don't tell the whole story. Furthermore, the impact of an effective visualization is difficult to match with words and will go a long way toward ensuring that your work gets the recognition it deserves.

When visualizing data, the most important factor to keep in mind is the purpose of the visualization. This is what will guide you in choosing the best plot type. It could be that you are trying to compare two quantitative variables to each other. Maybe you want to check for differences between groups. Perhaps you are interested in the way a variable is distributed. Each of these goals is best served by different plots and using the wrong one could distort your interpretation of the data or the message that you are trying to convey.

3.1 Introduction to Matplotlib

Matplotlib is the leading visualization library in Python. This tutorial is intended to help you get up-and-running with matplotlib quickly. We will go over how to create the most commonly used plots, when you would want to use each one, and highlight the parameters that you are most likely to adjust.

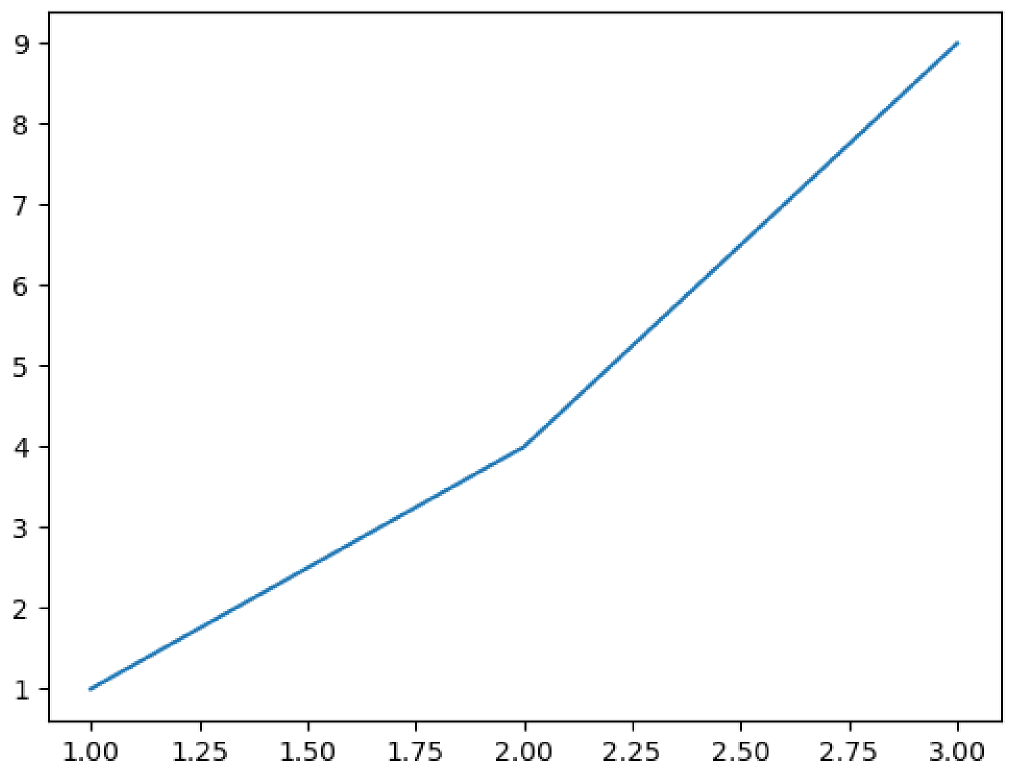
- Basic Plot: Line Chart, Scatter Plot
- Distribution Plot: Histogram, Boxplot

[4]:
```
from matplotlib import pyplot as plt
```

3.1.1 Simple Line Chart

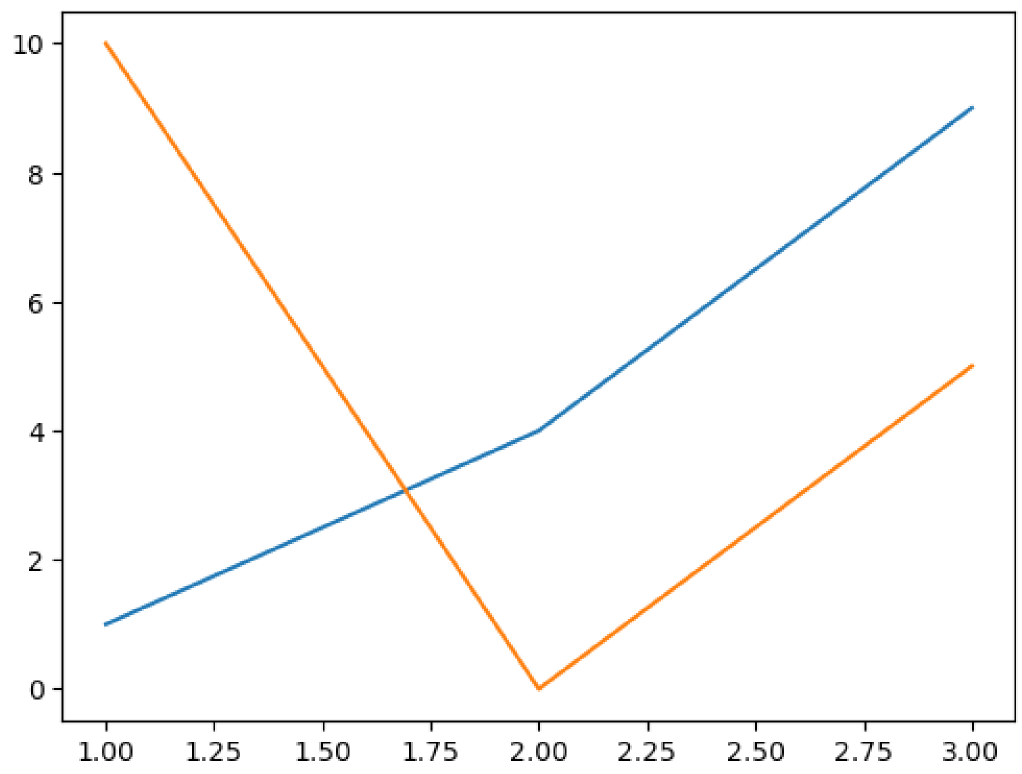
Plotting with default settings

[4]:
```
x = [1, 2, 3]
y = [1, 4, 9]

plt.plot(x,y)

plt.show()
```



Plotting multiple lines on a single chart

[5]:
```
x = [1, 2, 3]
y = [1, 4, 9]
z = [10, 0, 5]

plt.plot(x,y)
plt.plot(x,z)
plt.show()
```

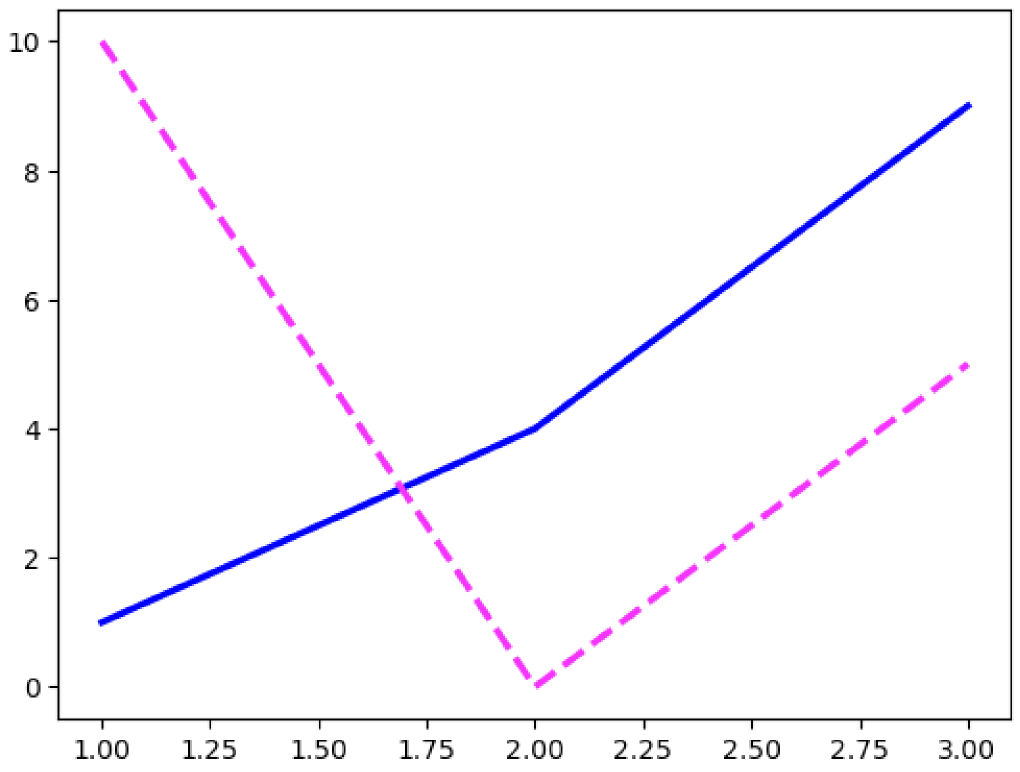


Changing colors, line widths and line styles We want to have the y in blue and the z in red and a slighty thicker line for both of them. We also want to change the line styles.

The third argument in the function call is a character that represents the type of symbols used for the plotting. The full list of available symbols can be seen in the documentation of `plt.plot`, or in Matplotlib's online **documentation**. https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.plot.html

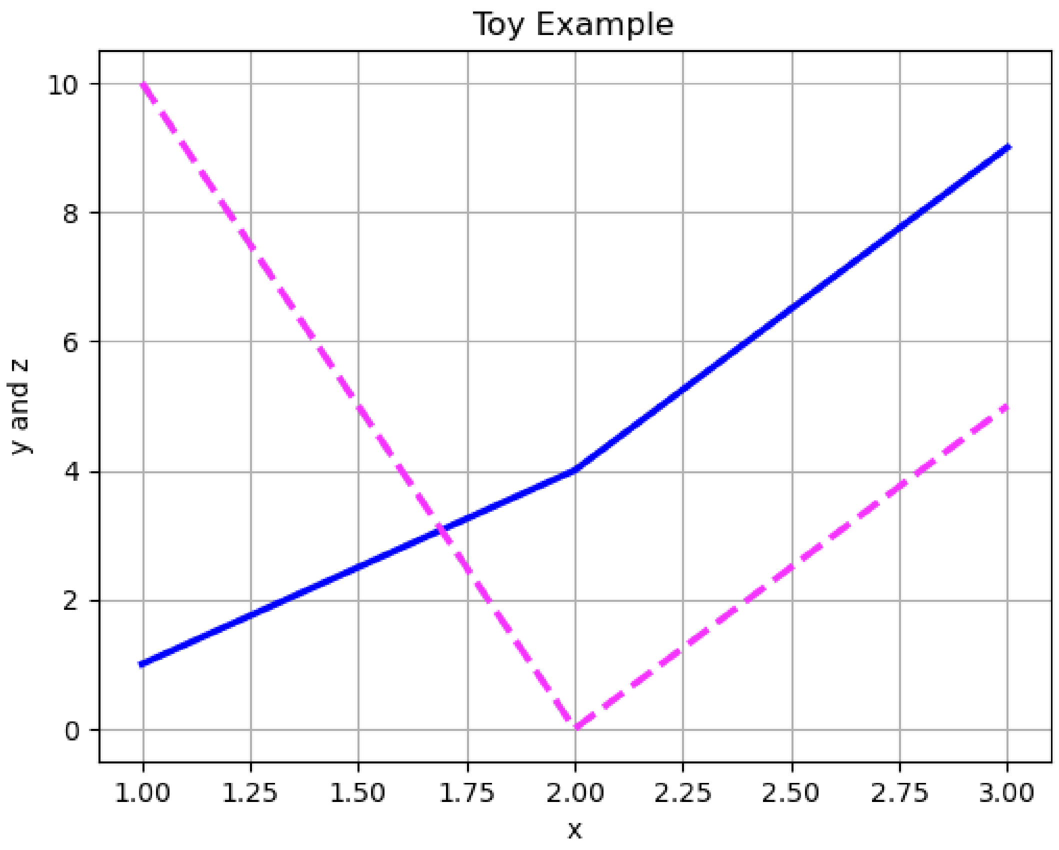
Ask Gemini:, How can I change the line color and line styles?

[7]:
```
x = [1, 2, 3]
y = [1, 4, 9]
z = [10, 0, 5]

plt.plot(x,y, "-", color = 'b', linewidth = 2.5)
plt.plot(x,z, "--", color = '#F433FF', linewidth = 2.5)
plt.show()
```



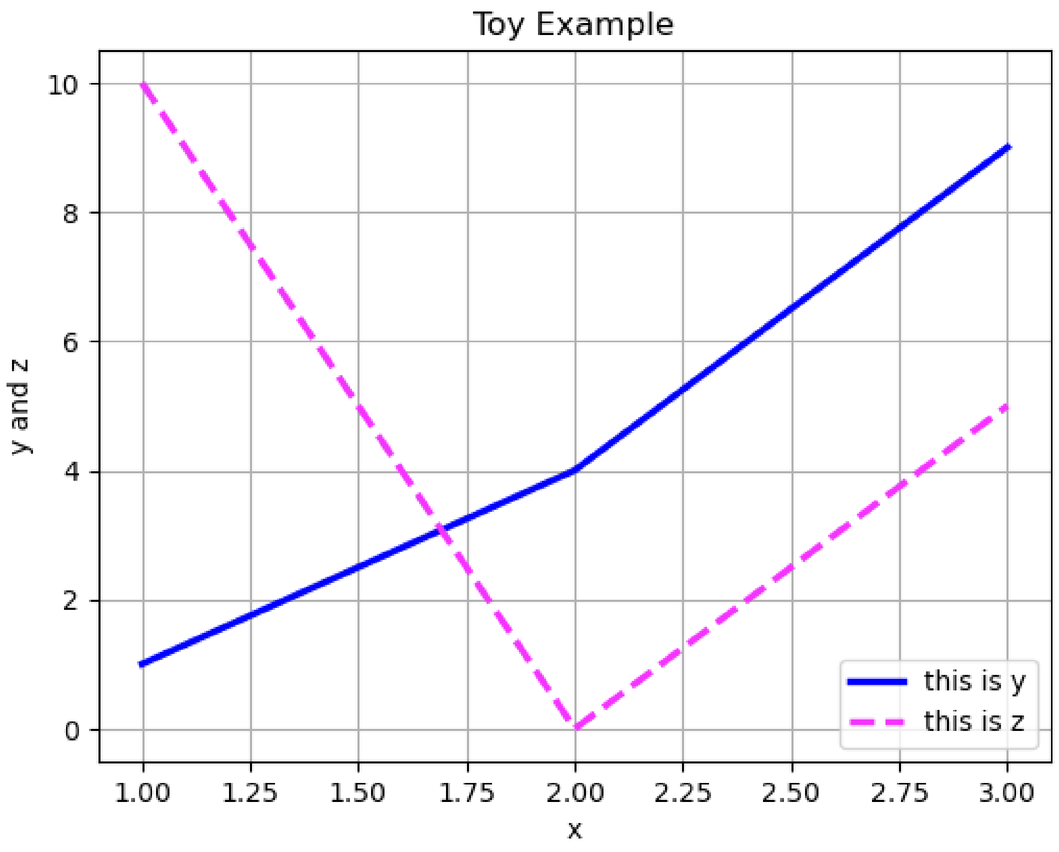
Adding titles and axis labels

[11]:
```
x = [1, 2, 3]
y = [1, 4, 9]
z = [10, 0, 5]

plt.plot(x,y, "-", color = 'b', linewidth = 2.5)
plt.plot(x,z, "--", color = '#F433FF', linewidth = 2.5)

plt.title("Toy Example")
plt.xlabel("x")
plt.ylabel("y and z")

# Optional: Add a grid
plt.grid(True)

plt.show()
```



Adding legends

[12]:
```python
x = [1, 2, 3]
y = [1, 4, 9]
z = [10, 0, 5]

plt.plot(x,y, "-", color = 'b', linewidth = 2.5, label = 'this is y')
plt.plot(x,z, "--", color = '#F433FF', linewidth = 2.5, label = 'this is z')

plt.title("Toy Example")
plt.xlabel("x")
plt.ylabel("y and z")

plt.legend()

# Optional: Add a grid
plt.grid(True)

plt.show()
```



Take-home Practice The `sales.csv` dataset contains daily sales data for three different product categories: Laptops, Printers, and Printer Ink. It covers a period of 12 days in January 2024, providing a snapshot of retail performance over a short, consistent timeframe.

Questions: 1. Visualize the sales trends of each product category. Put date on the x-axis, sales on the y-axis. You need to generate 3 lines on the same figure to represent 3 categories. Set the markers to be 'o'. Include proper x/y-axis labels, legends, and title. 2. What business insights you can get regarding retail sales trends over this specified period?

3.2 Scatter Plot

In many cases this is the least aggregated representation of your data. Displays relationship between two numerical variables.

The scatter() function makes a scatter plot with markers of varying size and/or color. The full list of available options can be seen in the documentation of `plt.scatter` **documentation**.

https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.scatter.html

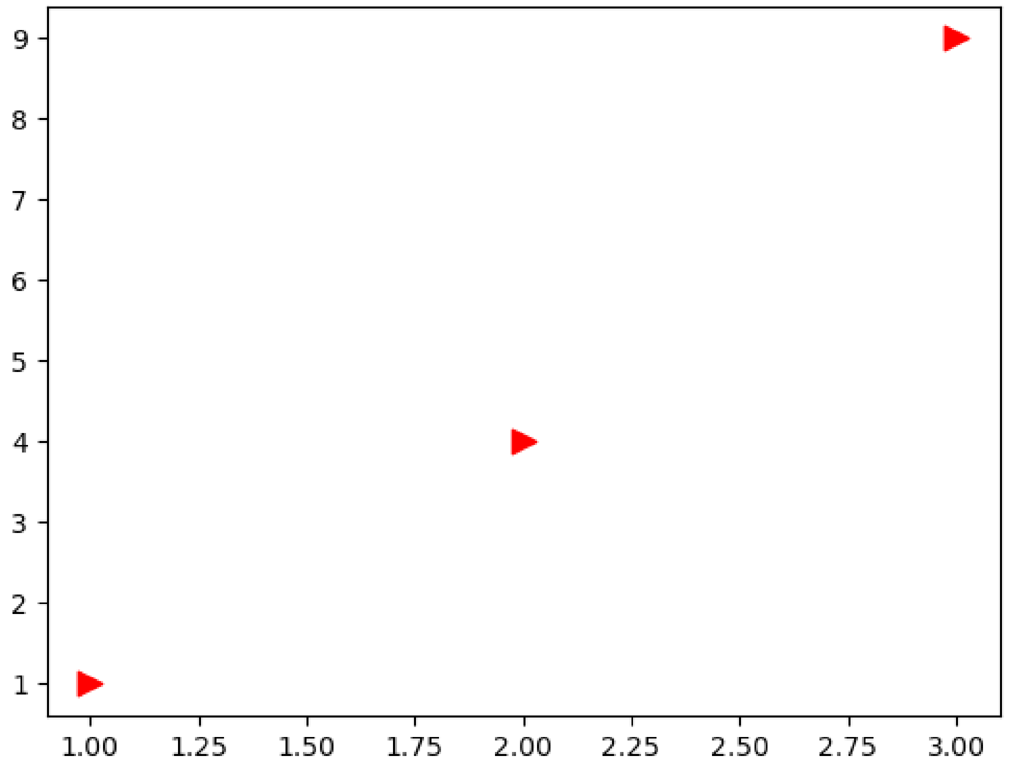
Function template:

```
matplotlib.pyplot.scatter(x, y, s=None, c=None, marker=None, cmap=None,
norm=None, vmin=None, vmax=None, alpha=None, linewidths=None, verts=None,
edgecolors=None, hold=None, data=None, **kwargs)
```

[ ]:
```python
x = [1, 2, 3]
y = [1, 4, 9]


plt.scatter(x, y)    # plot with the default setting
plt.show()
```

3.2.1 Changing colors, marker style and marker sizes

[13]:
```python
x = [1, 2, 3]
y = [1, 4, 9]

plt.scatter(x, y, s = 80, c = 'r', marker = '>')
plt.show()
```



3.2.2 Practice

We will use the Iris data, where each sample is one of three types of flowers that has had the size of its petals and sepals carefully measured.

```
[5]: df = pd.read_csv('iris.csv')
     df
```

```
[5]:      SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm         Species
     0              5.1           3.5            1.4           0.2     Iris-setosa
     1              4.9           3.0            1.4           0.2     Iris-setosa
     2              4.7           3.2            1.3           0.2     Iris-setosa
     3              4.6           3.1            1.5           0.2     Iris-setosa
     4              5.0           3.6            1.4           0.2     Iris-setosa
     ..             …             …              …             …               …
     145            6.7           3.0            5.2           2.3  Iris-virginica
     146            6.3           2.5            5.0           1.9  Iris-virginica
     147            6.5           3.0            5.2           2.0  Iris-virginica
     148            6.2           3.4            5.4           2.3  Iris-virginica
     149            5.9           3.0            5.1           1.8  Iris-virginica

     [150 rows x 5 columns]
```

Question:

1. We want to find out the relationship between sepal length and width by constructing a scatter plot. Put the sepal length as the x-axis and sepal width as the y-axis. Set the marker color to 'g'. Be sure to include proper x/y-axis labels.
2. What insights you can gain from this scatter plot?

3.2.3 Multi-dimensional visualization (Bubble chart)

We want to create scatter plots where the properties of each individual point (size, color, etc.) can be individually controlledor mapped to data. In this way, the color and size of points can be used to convey information in the visualization, in order to visualize multidimensional data.

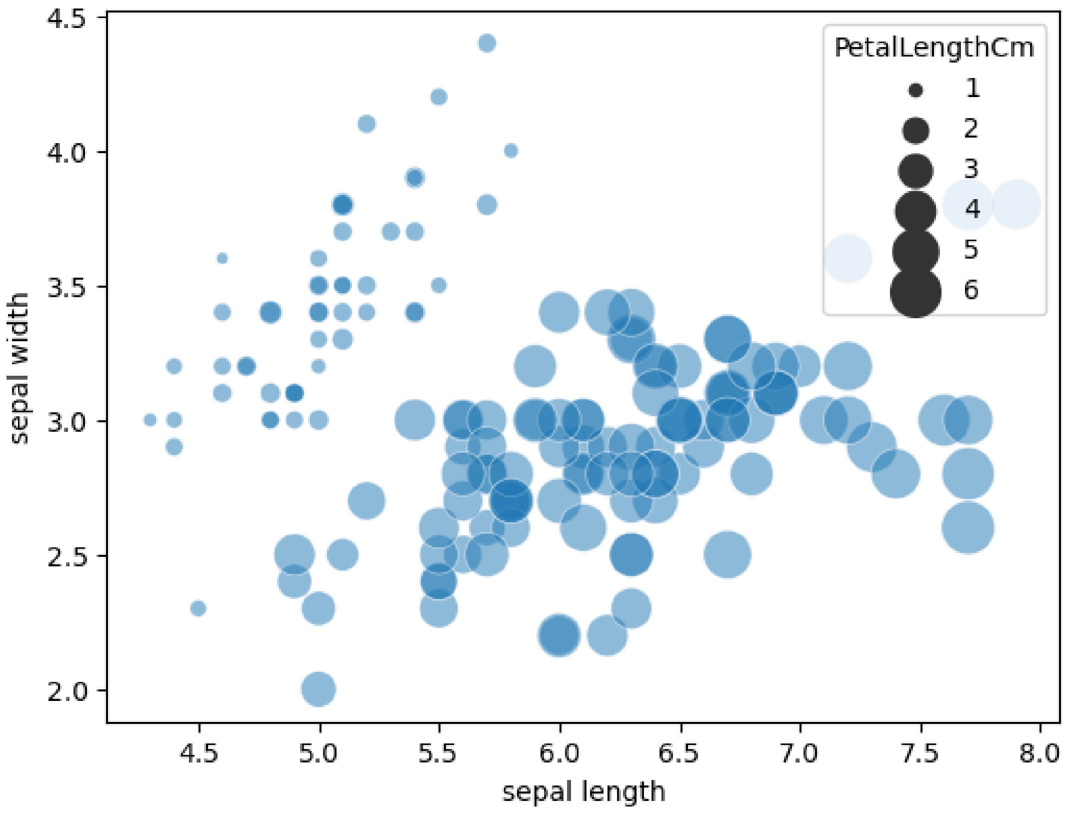
Seaborn is a Python data visualization library built on top of Matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics. Seaborn simplifies the creation of complex plots, supports themes for consistent styling, and integrates well with Pandas data structures. It offers functions for visualizing univariate and bivariate data, fitting and visualizing linear models, and creating complex plots .

Fun fact: the library Seaborn is named after Samuel Norman Seaborn, a fictional character from the TV show The West Wing. The library's standard alias is "sns", which are the character's initials.

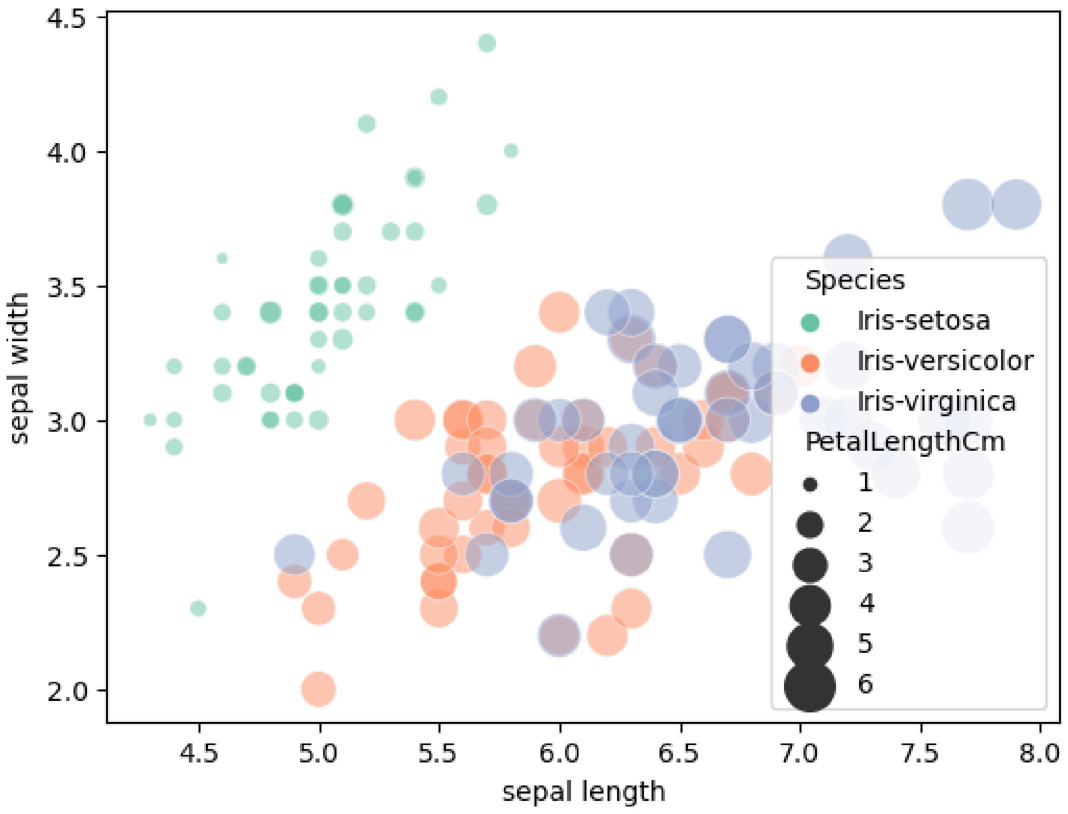
Show case for **Seaborn**: https://seaborn.pydata.org/tutorial/introduction.html#multivariate-views-on-complex-datasets

```
[21]: import seaborn as sns

      sns.scatterplot(data=df, x="SepalLengthCm", y="SepalWidthCm",␣
       ↪size="PetalLengthCm",
                      sizes=(20, 400), # Adjust the 'sizes' range
                      alpha=0.5 ) # transparency level
```

```
plt.xlabel("sepal length")
plt.ylabel("sepal width")
plt.show()
```



[8]:
```
# Create a scatter plot with sepal length as x-axis, sepal width as y-axis,
# and color-coded by species using Seaborn
sns.scatterplot(data=df, x="SepalLengthCm", y="SepalWidthCm", hue="Species",
                size="PetalLengthCm",
                sizes=(20, 400),  # Adjust the 'sizes' range
                alpha=0.5, palette='Set2') # change colormap

plt.xlabel("sepal length")
plt.ylabel("sepal width")

#plt.legend(loc='upper right')

plt.show()
```

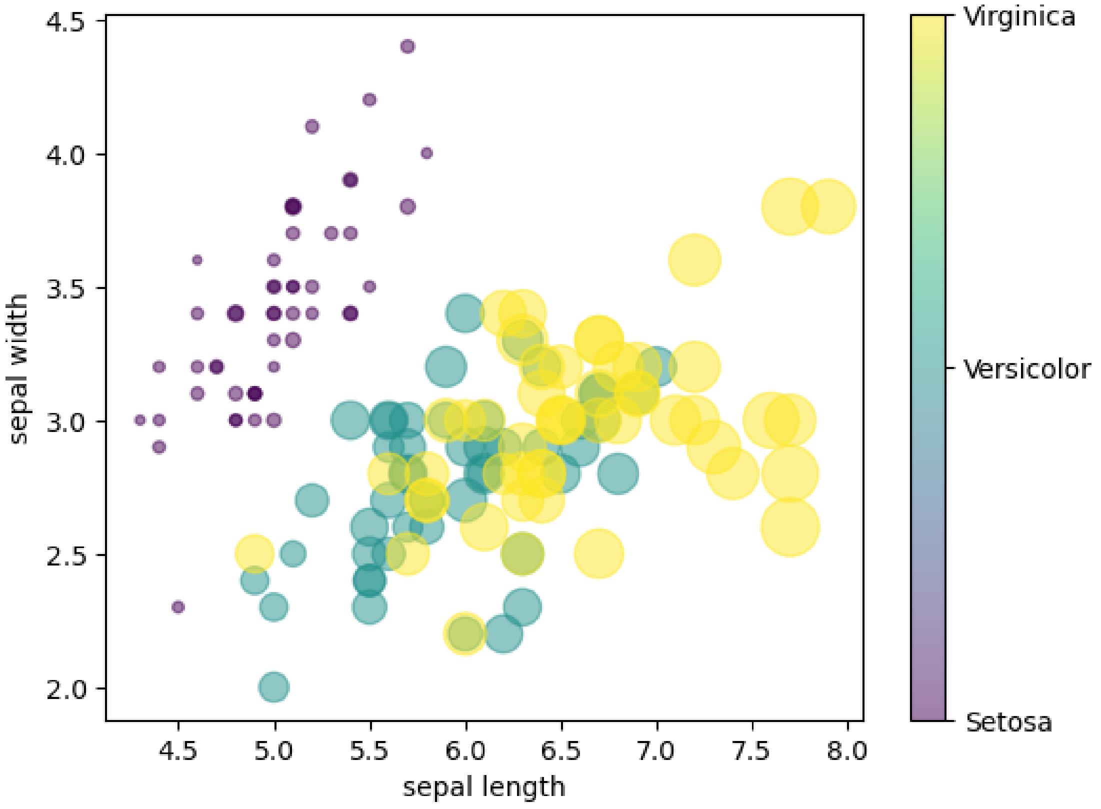


We can see that this scatter plot has given us the ability to simultaneously explore four different dimensions of the data: the (x, y) location of each point corresponds to the sepal length and width, the size of the point is related to the petal width, and the color is related to the particular species of flower. Multicolor and multifeature scatter plots like this can be useful for both exploration and presentation of data.

Below is how you can use Matplotlib to generate a similar plot. The coding is much involved, but you do have the ability to have detailed control over every aspect of a plot, allowing for highly customized visualizations.

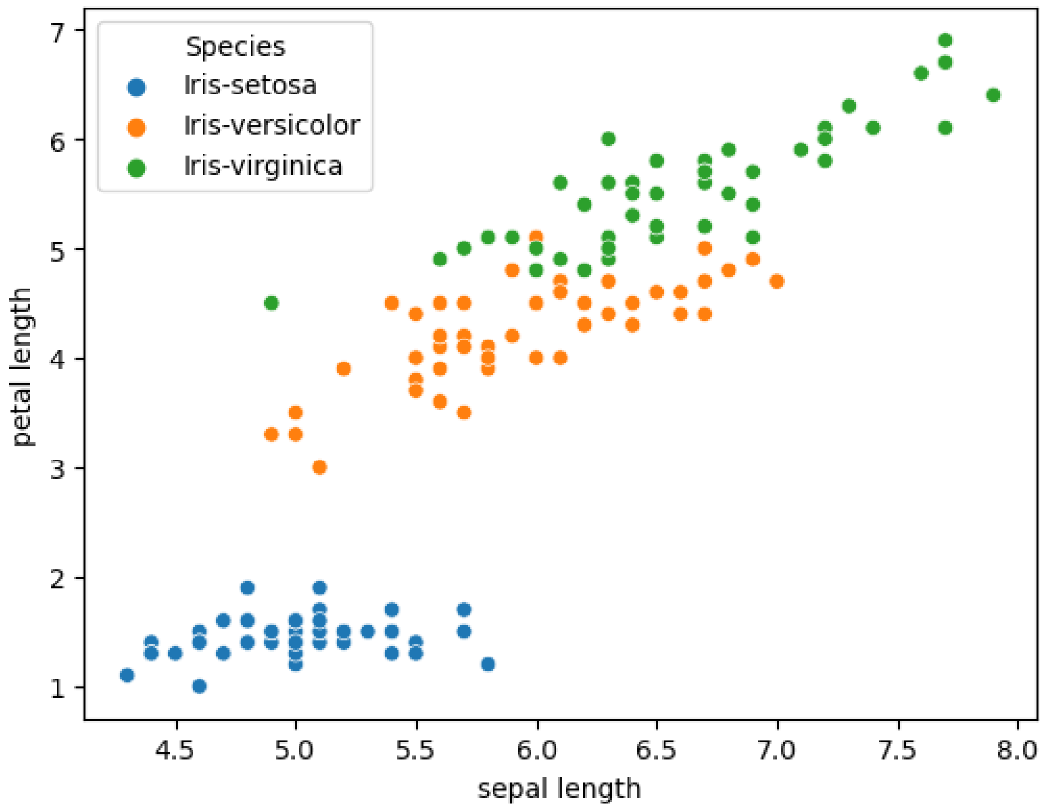
[9]:
```
# Map species to numerical values
species_mapping = {'Iris-setosa': 0, 'Iris-versicolor': 1, 'Iris-virginica': 2}
df['Species_num'] = df['Species'].map(species_mapping)

# Create a scatter plot with sepal length as x-axis and sepal width as y-axis
# Use 'Species_num' for color mapping
plt.scatter(df["SepalLengthCm"], df["SepalWidthCm"],  s = 10 *␣
  ↪df["PetalLengthCm"] ** 2,  c = df["Species_num"], alpha = 0.5)

# Add x-axis label
plt.xlabel("sepal length")
plt.ylabel("sepal width")
```

```
# Add a colorbar to show the mapping
cbar = plt.colorbar(ticks=[0, 1,  2])
cbar.set_ticklabels(['Setosa', 'Versicolor', 'Virginica'])

plt.show()
```



3.2.4 Exploring different variables for better decision making

[10]:
```
sns.scatterplot(data=df, x="SepalLengthCm", y="PetalLengthCm", hue="Species")
plt.xlabel("sepal length")
plt.ylabel("petal length")
plt.show()
```

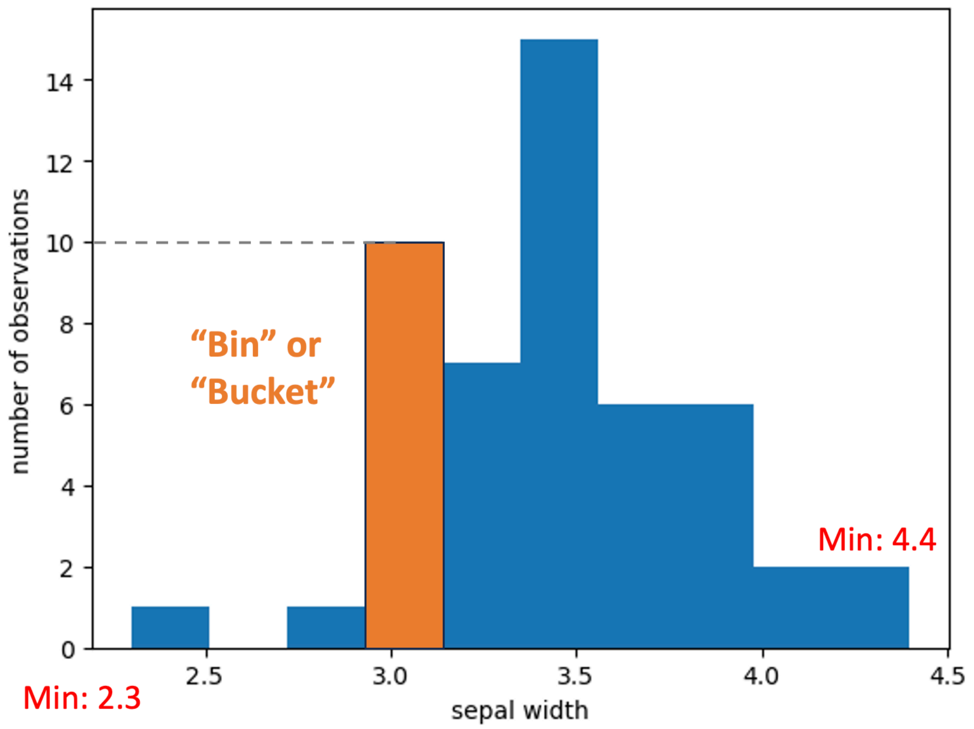


3.3 Distribution Plot: Histogram

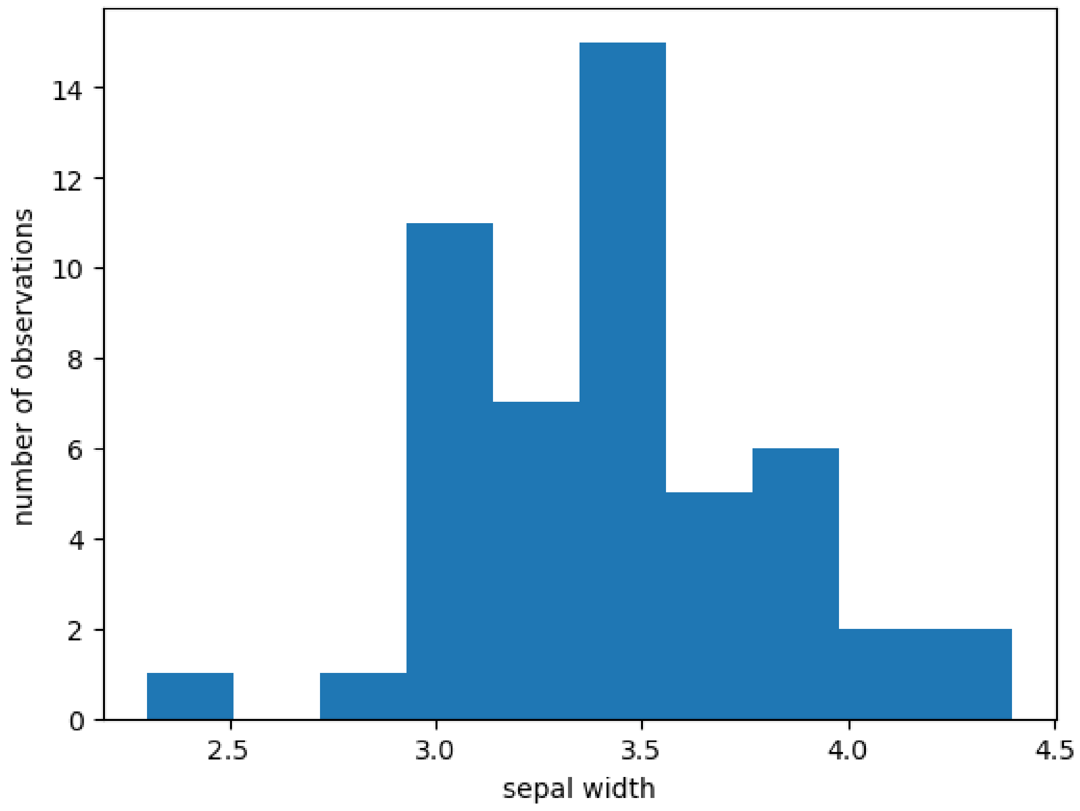
Histogram is an estimate of the probability distribution of a quantitative variable. The observed values are placed into different bins and the frequency of observations in each of those bins is calculated. A histogram is a graphical display of data using bars of different heights. Taller bars show that more data falls in that range.

It can detect possible outliers.

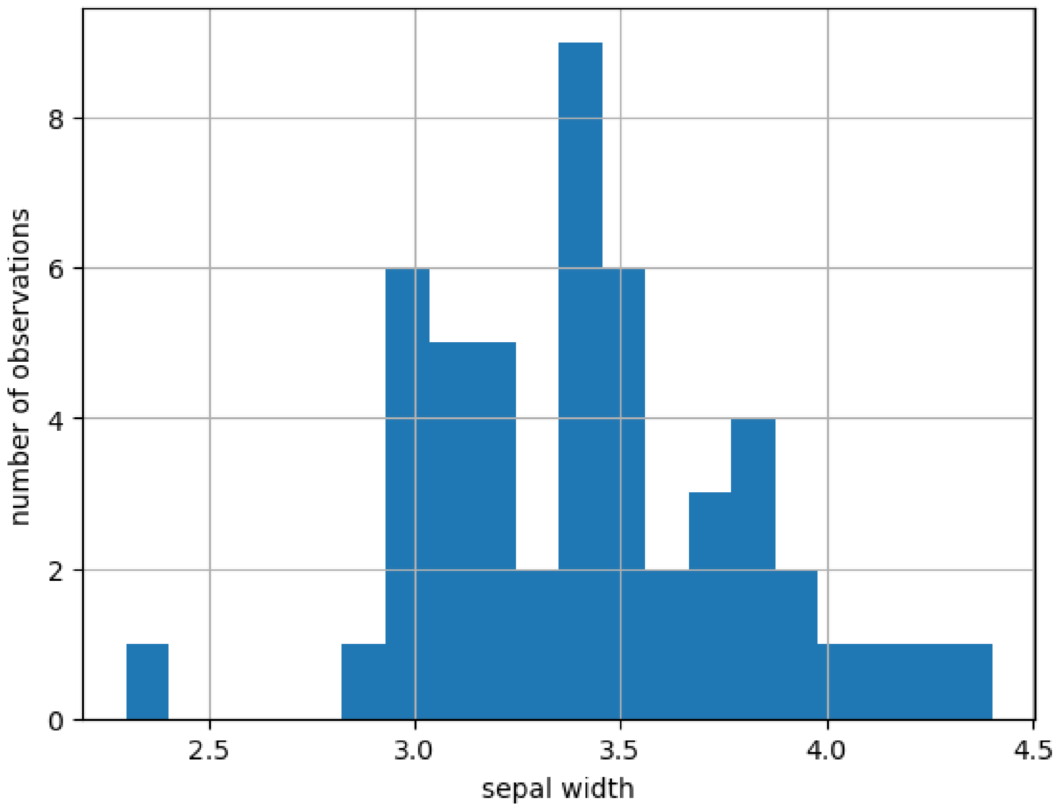
For this example, let's examine the distribution of the sepal width of all "Iris-Setosa"s.



```
[11]: index = df["Species"] == "Iris-setosa"  # generate logical values on whether␣
      ↪Setosa
      setosa = df.loc[index, :]
      plt.hist(setosa["SepalWidthCm"])  # give the frequency of observations in each␣
      ↪bin
      plt.xlabel("sepal width")
      plt.ylabel("number of observations")
      plt.show()
```



[12]:
```
num_bins = 20                              # default = 10
plt.hist(setosa["SepalWidthCm"], num_bins)  # give the frequency of␣
 ↪observations in each bin
plt.xlabel("sepal width")
plt.ylabel("number of observations")
plt.grid(True)
plt.show()
```

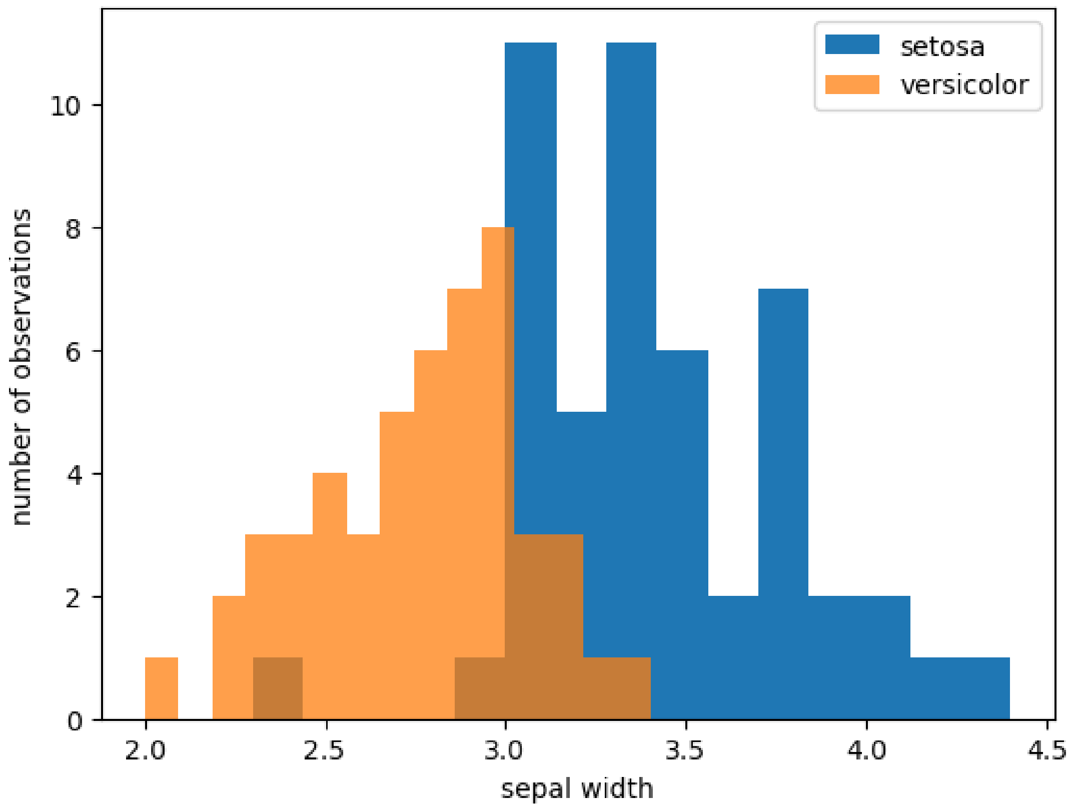


In statistics, an outlier is an observation point that is distant from other observations. An outlier may be due to variability in the measurement or it may indicate experimental error; the latter are sometimes excluded from the data set. An outlier can cause serious problems in statistical analyses.

3.3.1 Overlaid Histogram

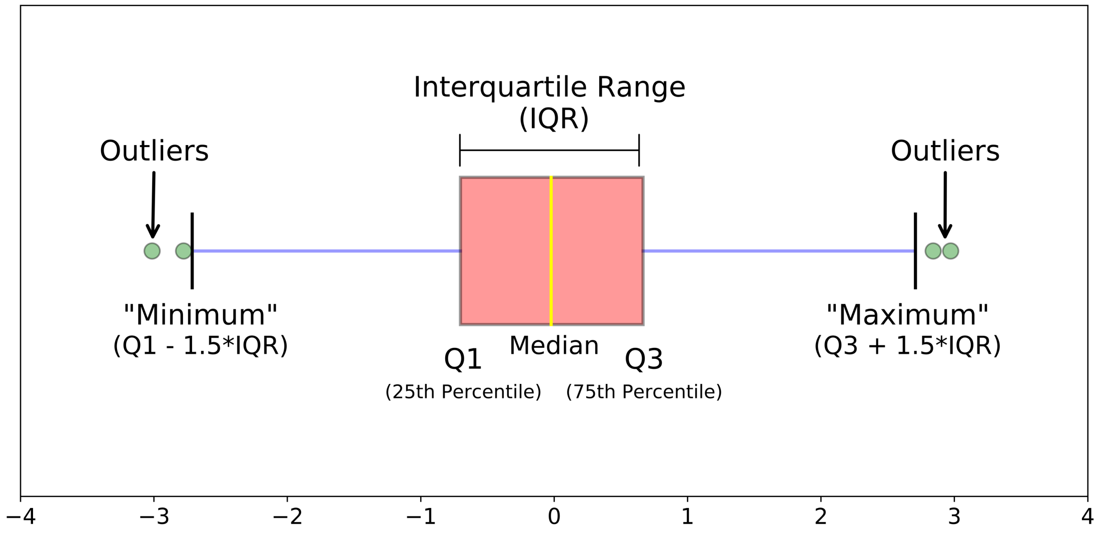
If you are looking to compare two (or more) distributions, use an overlaid histogram. Some additional care needs to be taken with these plots to ensure that they remain clear and easy to read, especially when more than two distributions are visualized. In this example, we will compare the distributions of sepal width of all “Iris-setosa”s and sepal width of all “Iris-versicolor”s.

[15]:
```
versicolor = df.loc[df["Species"] == "Iris-versicolor" , :]

num_bins = 15
plt.hist(setosa["SepalWidthCm"], num_bins, label = "setosa")
plt.hist(versicolor["SepalWidthCm"], num_bins, alpha = 0.75, label =␣
 ↪"versicolor")

plt.xlabel("sepal width")
plt.ylabel("number of observations")
plt.legend()
plt.show()
```



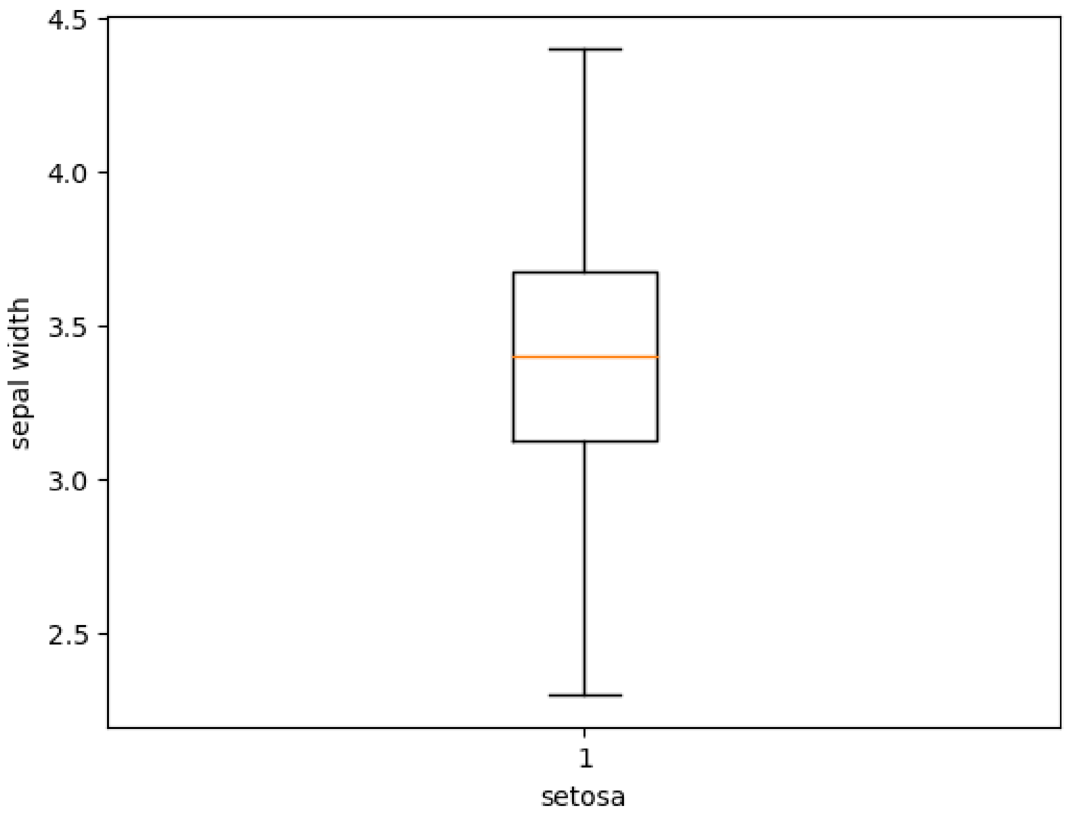
3.4 Distribution Plot: Box plot

A box plot used for graphically depicting groups of numerical data through their quartiles. Outliers can be plotted as individual points. The spacings between the different parts of the box indicate the degree of dispersion (spread) and skewness in the data, and show outliers. The basic explanations for boxplot are provided in the following figure:



Next, let's examine the distribution of the sepal width of all "Iris-setosa"s.

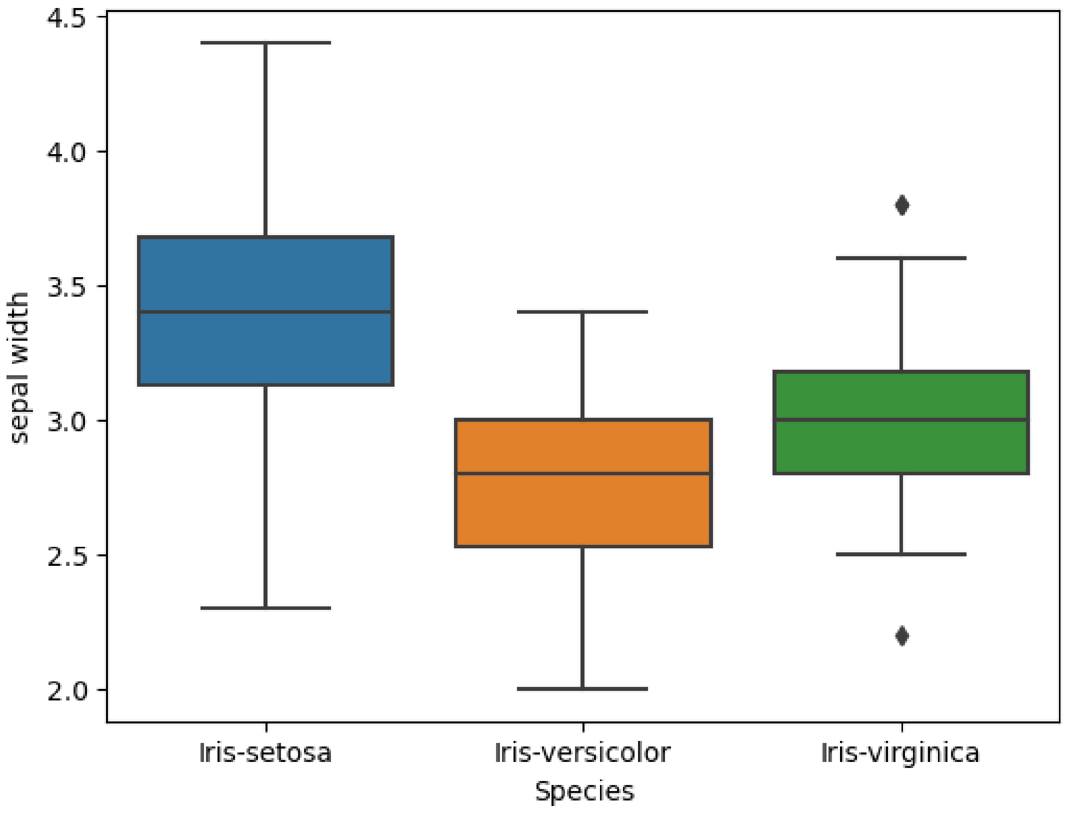
[19]:
```
plt.boxplot(setosa["SepalWidthCm"])
plt.ylabel("sepal width")
plt.xlabel("setosa")
plt.show()
```



3.4.1 Side-by-Side box plots

If you are looking to compare two (or more) distributions, you can use a side-by-side box plots. In this example, we will compare the distributions of sepal width of all “Iris-setosa”s, sepal width of all “Iris-versicolor”s, and that of all “Iris-virginica”.

[20]:
```
sns.boxplot(data=df, x='Species', y='SepalWidthCm' )
plt.xlabel('Species')
plt.ylabel('sepal width')
plt.show()
```



3.5 Visualization Practice: Bike Sharing Systems

Bike sharing systems are new generation of traditional bike rentals where whole process from membership, rental and return back has become automatic. Through these systems, user is able to easily rent a bike from a particular position and return back at another position. Currently, there are about over 500 bike-sharing programs around the world which is composed of over 500 thousands bicycles. Today, there exists great interest in these systems due to their important role in traffic, environmental and health issues.

Apart from interesting real world applications of bike sharing systems, the characteristics of data being generated by these systems make them attractive for the research. Opposed to other transport services such as bus or subway, the duration of travel, departure and arrival position is explicitly recorded in these systems. This feature turns bike sharing system into a virtual sensor network that can be used for sensing mobility in the city. Hence, it is expected that most of important events in the city could be detected via monitoring these data.

Data Description We will be using the daily version of the Capital Bikeshare System dataset from the UCI Machine Learning Repository. This data set contains information about the daily count of bike rental checkouts in Washington, D.C.'s bikeshare program between 2011 and 2012. It also includes information about the weather and seasonal/temporal features for that day (like whether it was a weekday).

- **day**: Day of the record (relative to day 1:2011-01-01)
- **season**: Season (1:winter, 2:spring, 3:summer, 4:fall)
- **weekday**: Day of the week (0=Sunday, 6=Saturday)
- **workingday**: If day is neither weekend nor holiday is 1, otherwise is 0.
- **weathersit**:
  - 1: Clear, Few clouds, Partly cloudy, Partly cloudy
  - 2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist
  - 3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds
- **temp**: Normalized temperature in Celcius
- **windspeed**: Normalized wind speed
- **casual**: Count of checkouts by casual/non-registered users
- **registered**: Count of checkouts by registered users
- **cnt**: Total checkouts

[ ]:
```python
import pandas as pd
daily = pd.read_csv('day.csv')
daily.head()
```

Questions:

1. **Understand Trends.** Generate a line chart to show the checkouts over time by using `day` column as the x-axis and `cnt` column as the y-axis. Label the x-axis as 'Day', and y-axis as 'Check Outs'. What can you conclude?

2. **Explore Relationships.** We will plot the daily count of bikes that were checked out by casual users against the temperature. Color the points to be '#539cab'. Set the transparency to be 0.7. Be sure to include appropriate labels for x-axis and y-axis. What insight can you get?

3. **Explore Relationships with Multidimensional Information.** We will plot the daily count of bikes that were checked out by casual users against the temperature. The color of each point will be set according to whether it is a working day. Set the transparency to be 0.7. Be sure to include appropriate labels for x-axis and y-axis. Change the legend to whether it is a working day. What additional insights can you get?

4. **Examine Distributions.** Let's first build a histogram of the registered bike checkouts with the number of bins as 10. Set appropriate labels. Also set the title to be "Distribution of Registered Check Outs".

5. **Compare Distributions.** We now compare the distributions of registered and casual checkouts. To make the figure easy to understand, additional to the histogram we made for the previous question, we will set the transparency of the casual one to 0.8 and the number of bins to 15. Set appropriate labels.

6. How do the temperatures change across the seasons? You need to choose the type of visualization that best serves this purpose. What are the mean and median temperatures?

7. In this question, we are exploring AI-assistance tools in python programming and visualization. By providing proper prompts to generative AI tools, generate a line chart showing average bike rentals for each day of the week. This can reveal weekly usage patterns and peak days. Interact with generative AI tools, ask questions about anything you find unclear, and try to understand the generated codes.