11 Data Visualisation

Data visualization is the graphical representation of data or information by using elements like graphs, charts, or other visual format. It communicates the relationships of the data with images. This is important because it allows trends and patterns to be more easily seen. We need data visualization because a visual summary of information makes it easier to identify patterns and trends than looking through thousands of rows on a spreadsheet. It's the way the human brain works. Since the purpose of data analysis is to gain insights, data is much more valuable when it is visualized. There are numerous tools available to help create data visualizations. Python offers multiple great graphing libraries that come packed with lots of different features. In this section, we will learn how to create basic plots using **matplotlib** library and how to use some of its specific features.

11.1 Figures and Subplots

Generally, to create a visual output, the very first step is to import **matplotlib** library before using any of its methods. In Python, all plotting functions exist inside a sub-module of **matplotlib** called **pyplot**. **pyplot** is an interface that contains a collection of command-style functions that we can use for performing common actions on our plot. Therefore, this sub-module needs to be imported as well so its functions can be called later on in the code. Here, we will be using the `Blood` dataset as a toy example for illustrating the different built-in figure functions included in this library.

11.1.1 Scatter Plot

Scatter plots graph pairs of numerical data, one variable on each axis, to look for a relationship using different points called 'markers', e.g., dots, circles, crosses, etc. The variable on the x-axis is called the independent variable, whereas the variable on the y-axis is called the dependent variable. Scatter plots show the degree and kind of correlation between both variables. This can be one of the cases shown in Fig(61).

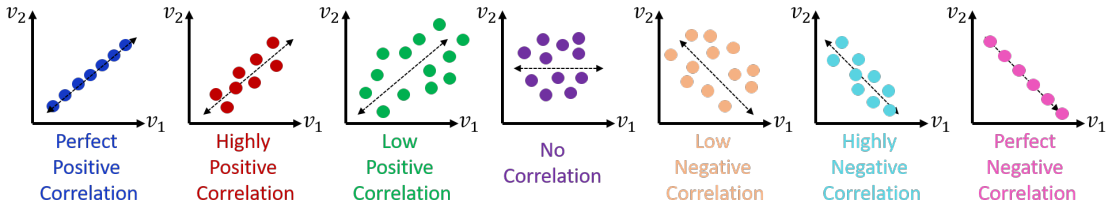

Figure 61: Different correlation patterns that can be visualised by scatter plots.

To create a scatter plot in **matplotlib** we can use the `scatter` method. As shown in Fig(62), four main steps are followed to perform such a task:

- Import relevant libraries. Two libraries here are imported: **pandas** for reading the dataset and **matplotlib** for data visualisation.
- Specify data to plot. After reading the dataset, we need to choose what data to plot. Here, we select two columns: CO2 on the x axis and O2 y axis.
- Call the `scatter` method to generate the plot.
- Show the plot by calling the `show()` method.

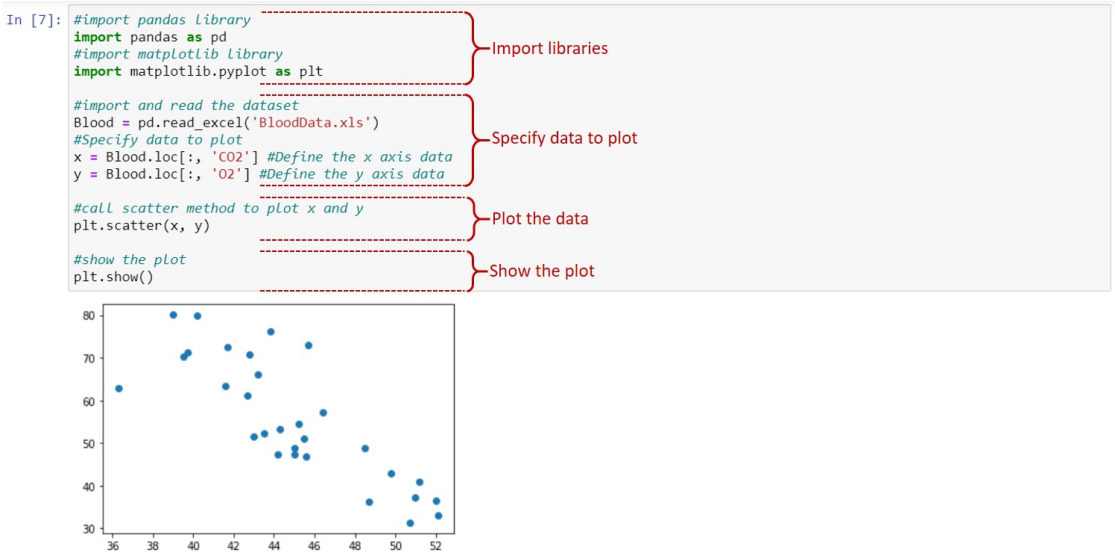

Figure 62: The main steps to generate a scatter plot in Python.

11.1.2 Line Chart

A line chart, line plot, or line graph, is a type of chart that displays the information of a data set as a series of data points connected by a straight line. In **matplotlib** we can create a line chart by calling the `plot` method. Although similar main steps to scatter plots are followed here to generate a line chart, as shown in Fig(63), two differences can be noticed:

1. Only the CO2 column is specified to plot. The numbers of this column are interpreted as the y-values to create the plot. The x axis, on the other hand, is scaled automatically by the **pyplot** interface. Otherwise, you can surely specify certain values or a range for the x axis.

2. The `plot` function is called.

```
In [13]: #import pandas library
         import pandas as pd
         #import matplotlib library
         import matplotlib.pyplot as plt

         #import and read the dataset
         Blood = pd.read_excel('BloodData.xls')
         #Specify data to plot
         x = Blood.loc[:, 'CO2'] #Define the data to plot

         #call line method to plot x
         plt.plot(x)

         #show the plot
         plt.show()
```

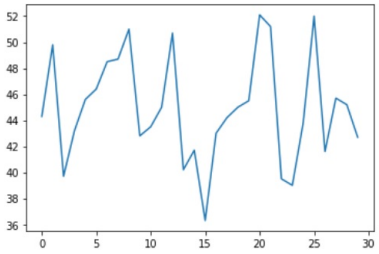

Figure 63: The main steps to generate a line chart in Python.

11.1.3 Histogram

Histograms are the most commonly used graph for showing frequency distributions, or how often each different value in a set of data occurs. This kind of graph uses vertical bars to display quantitative data. The heights of the bars indicate the frequencies of values in our data set. It is very important to mention here that Histograms should only be used for continuous data. However, for discrete (categorical) data we should use bar charts. As shown in Fig(64,a), in a histogram,

the categories are numerical (continuous, quantities) data. Bars usually are touching. However, in a bar graph in Fig(64,b), the categories are usually categorical (discrete) data. Bars usually are separated. In **matplotlib** we can create a Histogram using the `hist` method, as shown in Fig(65), following similar main steps as before.

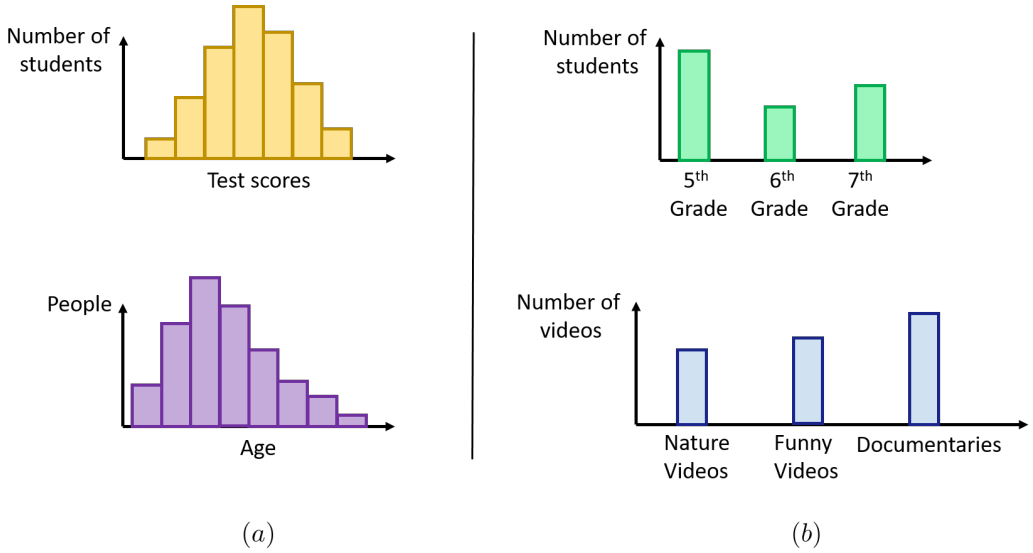

Figure 64: Difference between histograms in (a) and bar charts in (b).

```
In [14]: #import pandas library
         import pandas as pd
         #import matplotlib library
         import matplotlib.pyplot as plt

         #import and read the dataset
         Blood = pd.read_excel('BloodData.xls')
         #Specify data to plot
         x = Blood.loc[:, 'CO2']

         #call hist method
         plt.hist(x)

         #show the histogram
         plt.show()
```

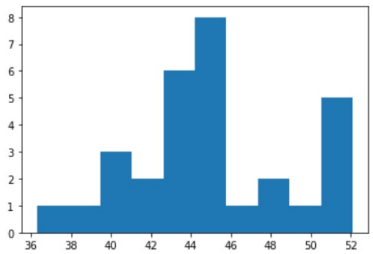

Figure 65: Histograms syntax in Python.

11.1.4 Bar Chart

As mentioned before, a bar chart or bar graph is a chart or graph that presents categorical data with rectangular bars with heights or lengths proportional to the values that they represent. The bars can be plotted vertically or horizontally. In Python, a bar chart can be created using the `bar` method. Fig(66) shows three bars that represent the CO2 values for the first three participants. You can notice that the quantities on the x-axis are categorical in this case rather than numerical as before.

```
In [42]: #import pandas library
         import pandas as pd
         #import matplotlib library
         import matplotlib.pyplot as plt

         #import and read the dataset
         Blood = pd.read_excel('BloodData.xls')
         #Set the bar labels
         participants = ['Participant 1','Participant 2', 'Participants 3']
         #select the values for each bar
         CO2 = [Blood['CO2'].iloc[0], Blood['CO2'].iloc[1], Blood['CO2'].iloc[2]]

         #plot the bar chart
         plt.bar(participants,CO2)

         #show the chart
         plt.show()
```

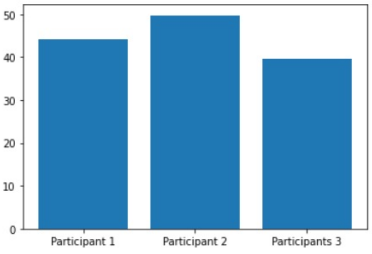

Figure 66: Bar chart syntax in Python.

11.1.5 Subplots

To draw multiple plots in one figure we use the `subplot()` function. The `subplot()` function takes three arguments that describe the layout of the figure. The layout is organized in rows and columns, which are represented by the first and second arguments. The third argument represents the index of the current plot. For example, if we want two plots to be vertically stacked with 2 rows and 1 column (meaning that the two plots will be displayed on top of each), we can write the syntax as shown in Fig(67,a). However, if we want the two plots to be horizontally stacked with 1 row and 2 columns (meaning that the two plots will be displayed side by side), we can write the syntax as shown in Fig(67,b). Generally speaking, we can plot as many subplots as the problem requires. For example, as shown in Fig(67,c), we can organise a 2×2 layout for four plots in the same figure.

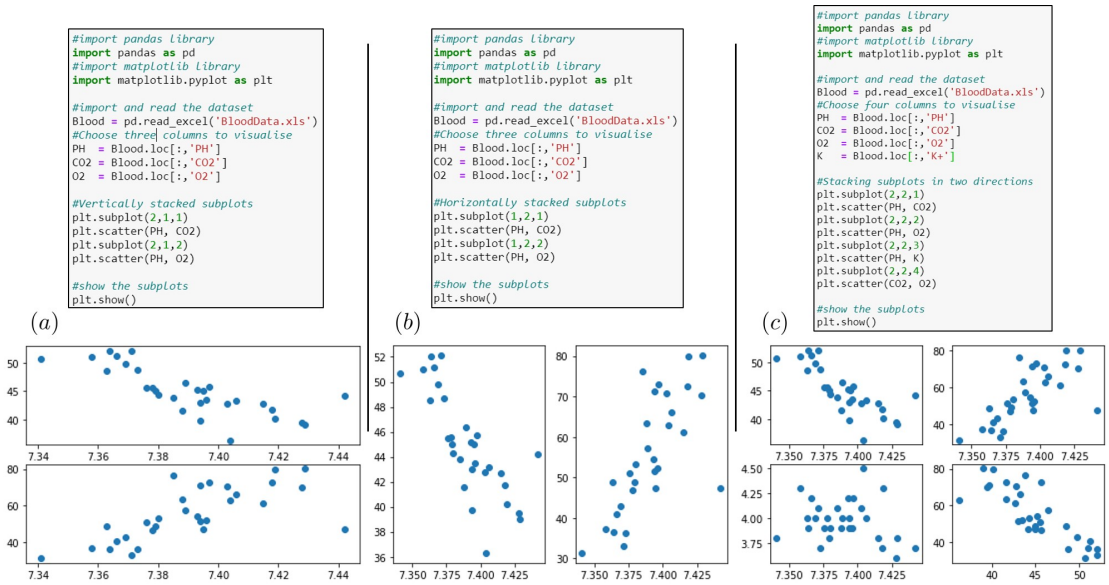

Figure 67: (a) Two subplots stacked vertically. (b) Two subplots stacked horizontally. (c) Four subplots stacked in a 2D structure.

11.2 Title, Axes Labels & Legends

It is essential to keep in mind that any plot that visualises a certain set of data needs to have the following main parts:

1. Figure.
2. Title.
3. Axes labels
4. Legend

Missing any of the above parts results in a major loss of information and misinterpreting the ideas the plot tries to show. Since we have covered how to plot several types of figures in Python earlier, we will cover here how to include the rest of the main parts. For simplicity, as the steps apply to all other types of plots, we will focus on the scatter plot for illustration. Fig(68) shows two scatter plots: the first plotting PH as a function of CO2 in blue dots and the second plotting PH as a function of O2 in orange dots. After calling the `scatter` method to plot both data sets, a title, axis labels, and legend are specified to be added to the plot. It is always important to remember to add this information as it certainly shows the value of what the plot displays.

```python
#import pandas library
import pandas as pd
#import matplotlib library
import matplotlib.pyplot as plt

#import and read the dataset
Blood = pd.read_excel('BloodData.xls')
#Specify data to plot
PH  = Blood.loc[:,'PH']
CO2 = Blood.loc[:, 'CO2']
O2  = Blood.loc[:, 'O2']

#call scatter method
plt.scatter(PH,CO2) #scatter PH as a function of CO2
plt.scatter(PH,O2) #scatter PH as a function of O2

#add title
plt.title('Blood Chemistry')

#add axis labels
plt.xlabel('CO2 and O2')
plt.ylabel('PH')

#add legend
plt.legend(['CO2','O2'])

#show the plot
plt.show()
```

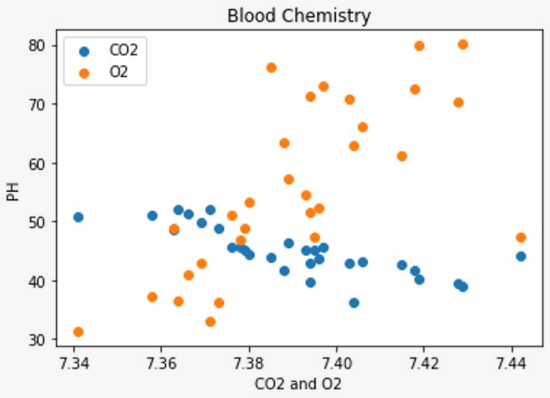

Figure 68: Title, axes labels, and legend in Python.

11.3 Problem: Ideal Gas Behaviour

Quantities such as pressure, volume, temperature, and amount of substance describe the conditions, or *state*, in which a particular material exists. These quantities are called state variables [6]. The volume V of a substance is usually determined by its pressure P, temperature T, and amount of substance, described by the mass m or the number of moles n. Ordinarily, we can't change one of these variables without causing a change in another. Measurements of the behaviour of various gases lead to three conclusions (or laws)

1. The pressure is proportional to the absolute temperature T. If we double T, keeping the volume and number of moles constant, the pressure doubles. In other words, $P \propto T$ or $P = \text{constant} \times T$ when n and V are constant. This is usually referred to as *Amontons's Law* or *Gay-Lussac's Law*.

2. The volume is proportional to the absolute temperature T. The volume increases as the temperature increases, and decreases as the temperature decreases by keeping the volume and number of moles constant. In other words, $V \propto T$ or $V = \text{constant} \times T$ when n and P are constant.The relationship between the volume and temperature of a given amount of gas at constant pressure is known as *Charles's law*.

3. Unlike the P–T and V–T relationships, pressure, and volume are not directly proportional to each other. Instead, P and V exhibit inverse proportionality: Increasing the pressure, withholding the temperatures T and number of

moles n constant, results in a decrease in the volume of the gas. In other words, $P \propto 1/V$ or $PV =$ constant when n and T are constant. This is known as *Boyle's Law*.

4. The volume V is proportional to the number of moles n. If we double n, keeping pressure and temperatures constant, the volume doubles. This relationship is usually referred to as *Avogadro's Law*.

When we combine these three relationships into a single ideal-gas equation:

$$PV = nRT \tag{8}$$

In SI units, the unit of P is Pa, and the unit of V is m^3. We might expect that the proportionality constant R in Eq(8) would have different values for different gases, but it turns out to have the same value for *all* gases. It is called the gas constant and its numerical value is equal to $R = 8.314$J/mol.K. An ideal gas is one for which Eq(8) holds precisely for *all* pressures and temperatures. This is an idealised model; it works best at very low pressures and high temperatures when the gas molecules are far apart and in rapid motion. Fig(69) shows a Python code that plots the main relationships (or laws) of an ideal gas.

In [15]:

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

#Define gas constant
R = 8.314;

#P-T relationship: Amontons's Law or Gay-Lussac's law
Temp = np.linspace(273.15,50+323.15,100); #variable
V    = 0.0132; #constant
n    = 50; #constant
P_Amonton= n*R*Temp/V; #solve for P

#V-T relationship: Charles's Law
Temp = np.linspace(273.15,50+323.15,100); #variable
P    = 15502725; #constant
n    = 50; #constant
V_Charles= n*R*Temp/P; #solve for V

#P-V relationship: Boyle's law
Volume = np.linspace(0.001,0.03,100); #variable
n      = 50; #constant
T      = 300.15; #constant
P_Boyle= n*R*T/Volume; #Solve for P

#V-n relationship: Avogadro's law
nmole = np.linspace(1,500,100); #variable
P     = 15502725; #constant
T     = 300.15; #constant
V_Avogadro= nmole*R*T/P; #Solve for V

#Stacking subplots in two directions
fig = plt.figure() #generate the figure
fig.subplots_adjust(hspace=0.9, wspace=0.6) #adjusting the spaces between the subplots
#P-T relationship: Amontons's Law or Gay-Lussac's law
plt.subplot(2,2,1)
plt.plot(Temp, P_Amonton, c='green')
plt.title('Amontons Law')
plt.xlabel('T $(K)$')
plt.ylabel('P $(Pa)$')
#V-T relationship: Charles's law
plt.subplot(2,2,2)
plt.plot(Temp, V_Charles, c='blue')
plt.title('Charles  law')
plt.xlabel('T $(K)$')
plt.ylabel('V $(m^3)$')
#P-V relationship: Boyle's law
plt.subplot(2,2,3)
plt.plot(Volume, P_Boyle, c='red')
plt.title('Boyle law')
plt.xlabel('V $(m^3)$')
plt.ylabel('P $(Pa)$')
#V-n relationship: Avogadro's law
plt.subplot(2,2,4)
plt.plot(nmole, V_Avogadro, c='orange')
plt.title('Avogadro law')
plt.xlabel('n $(mole)$')
plt.ylabel('V $(m^3)$')

#show the subplots
plt.show()
```

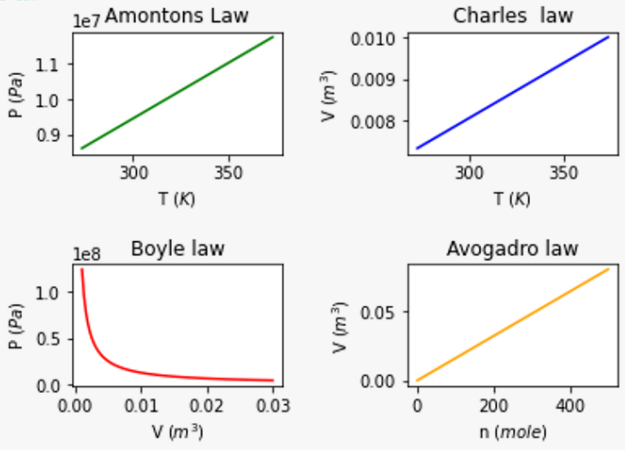

Figure 69: Python code to plot the four main laws that describe the basic relationships between the variables of state; pressure P, volume V, Temperature T, and number of moles n.