EXERCISE CLASS 1 (Part 1/3)

Introduction to Python (Jupiter notebook and Visual Studio environment) Review of basic statistical concepts

(Chapters 3 and 4, Montgomery) D.C. Montgomery, "Statistical Quality Control - an introduction", 7th ed, Wiley

Contact information

Matteo Bugatti, matteo.bugatti@polimi.it

https://politecnicomilano.webex.com/meet/matteo.bugatti

Python basic functions

These functions are embedded in Python and do not require importing libraries.

The `print()` function allows to print out a string or a combination of string and numbers.

In [ ]:
```python
name = 'Mark'
age = 23
print('Hi, my name is', name, 'and I am', age, 'years old.')
```

```
Hi, my name is Mark and I am 23 years old.
```

The printed strings can be also constructed using the appropriate syntax:

- integers `%d`
- float `%f`
- strings `%s`

In [ ]:
```python
print('Hi, my name is %s and I am %d years old.' % (name, age))
```

```
Hi, my name is Mark and I am 23 years old.
```

Introduction to `numpy`

Numpy is a powerful library for Python that is widely used for scientific and technical computing. It provides a powerful array object, as well as a variety of functions for working with arrays, such as mathematical operations, and linear algebra. The main advantages of using Numpy are its speed, convenience, and compatibility with other scientific libraries.

To use Numpy, we first need to import it.

In [ ]:
```python
# Import the libraries
import numpy as np
```

Once imported, we can create arrays using the `np.array()` function.

```
In [ ]: # Creating a one-dimensional array
        a = np.array([1, 2, 3, 4])
        print('a =', a)

        # Creating a two-dimensional array
        b = np.array([[1, 2], [3, 4]])
        print('b = ', b)
```

```
a = [1 2 3 4]
b =  [[1 2]
 [3 4]]
```

You can access individual elements of an array using brackets [ ]. Remind: elements are numbered starting from 0

```
In [ ]: print('first element of a is: ', a[0])

        print('element in position [0,1] of b is: ', b[0,1])

        print('the first column of b is: ', b[:,0])

        print('the second row of b is: ', b[1,:])
```

```
first element of a is:  1
element in position [0,1] of b is:  2
the first column of b is:  [1 3]
the second row of b is:  [3 4]
```

We can use `np.arange(start, stop, step)` function to create an evenly spaced array of values within a given range:

- `start` is the first value of the array
- `stop` is the last value of the array (the array will not include this value)
- `step` is the difference between each value in the array (default is 1)

Or, if you are not interested in a specific step size and you just want to evenly cover a range with `n` values, you can use `np.linspace(start, stop, n)` function.

```
In [ ]: c = np.arange(-5, 5, 1)
        print('c = ', c)

        d = np.linspace(-5, 5, 11)
        print('d = ', d)
```

```
c =  [-5 -4 -3 -2 -1  0  1  2  3  4]
d =  [-5. -4. -3. -2. -1.  0.  1.  2.  3.  4.  5.]
```

Introduction to `pandas`

Pandas is a powerful library for data manipulation and analysis in Python. It provides data structures such as Series (1-dimensional) and DataFrame (2-dimensional) for storing and manipulating data.

This is the library we will use to import the CSV files for all our exercise classes.

To use Pandas, we first need to import it.

In [ ]:
```python
import pandas as pd
```

One of the most basic and important functions of Pandas is the `read_csv()` function, which is used to load a CSV file into a DataFrame.

In [ ]:
```python
# Loading a CSV file into a DataFrame
df = pd.read_csv('ESE2_ex5.csv')
```

Another important function is the `head()` function, which is used to display the first n rows of a DataFrame.

In [ ]:
```python
# Display the first 3 rows of a DataFrame
df.head(3)
```

Out[ ]:

	cond1	cond2
0	19.8	14.9
1	18.5	12.7
2	17.6	11.9

You can also access and modify individual cells in the DataFrame using the `loc[]` function. For example, let's change the value in the first row and first column of the DataFrame and set it to `20.0`.

In [ ]:
```python
# Change the value of a cell
df.loc[0,'cond1'] = 20.0

# See if the change was made
df.head()
```

Out[ ]:

	cond1	cond2
0	20.0	14.9
1	18.5	12.7
2	17.6	11.9
3	16.7	11.4
4	16.7	10.1

You can also search for specific values in the DataFrame. For example, let's change the value in the first column that is equal to `17.6` and set it to `17.0`.

In [ ]:
```python
# Find the row equal to 17.6 value in the first column and change the value to 17.
df.loc[df['cond1'] == 17.6, 'cond1'] = 17.0

# See if the change was made
df.head()
```

Out[ ]:

	cond1	cond2
0	20.0	14.9
1	18.5	12.7
2	17.0	11.9
3	16.7	11.4
4	16.7	10.1

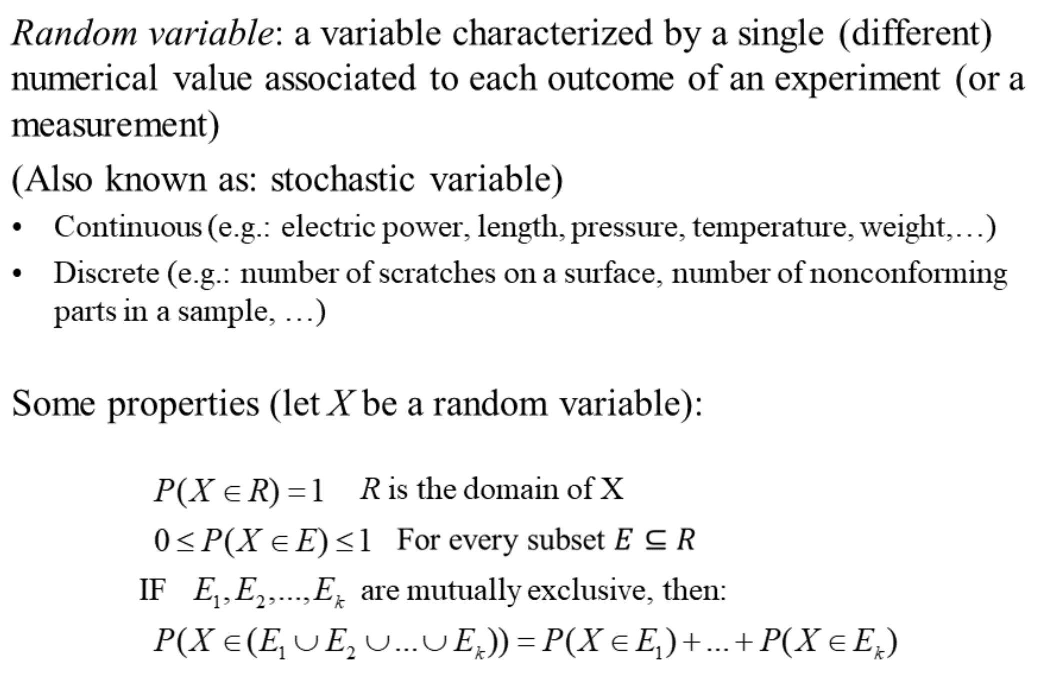
You can use the built-in functions to find the index of the maximum value in a column. For example, let's find the index of the maximum value in the first column and set it to `20.1`.

In [ ]:
```
# Find the row with the maximum value in a column and change the value
df.loc[df['cond1'].idxmax(), 'cond1'] = 20.1

# See if the change was made
df.head()
```

Out[ ]:

	cond1	cond2
0	20.1	14.9
1	18.5	12.7
2	17.0	11.9
3	16.7	11.4
4	16.7	10.1

What kind of data? Random variables



Random variable: a variable characterized by a single (different) numerical value associated to each outcome of an experiment (or a measurement)

(Also known as: stochastic variable)

- Continuous (e.g.: electric power, length, pressure, temperature, weight,…)
- Discrete (e.g.: number of scratches on a surface, number of nonconforming parts in a sample, …)

Some properties (let X be a random variable):

$P(X \in R) = 1$ $\quad R$ is the domain of X

$0 \le P(X \in E) \le 1$ $\quad$ For every subset $E \subseteq R$

IF $\quad E_1, E_2, \ldots, E_k$ are mutually exclusive, then:

$$P(X \in (E_1 \cup E_2 \cup \ldots \cup E_k)) = P(X \in E_1) + \ldots + P(X \in E_k)$$

Graphical representation: introduction to `matplotlib`

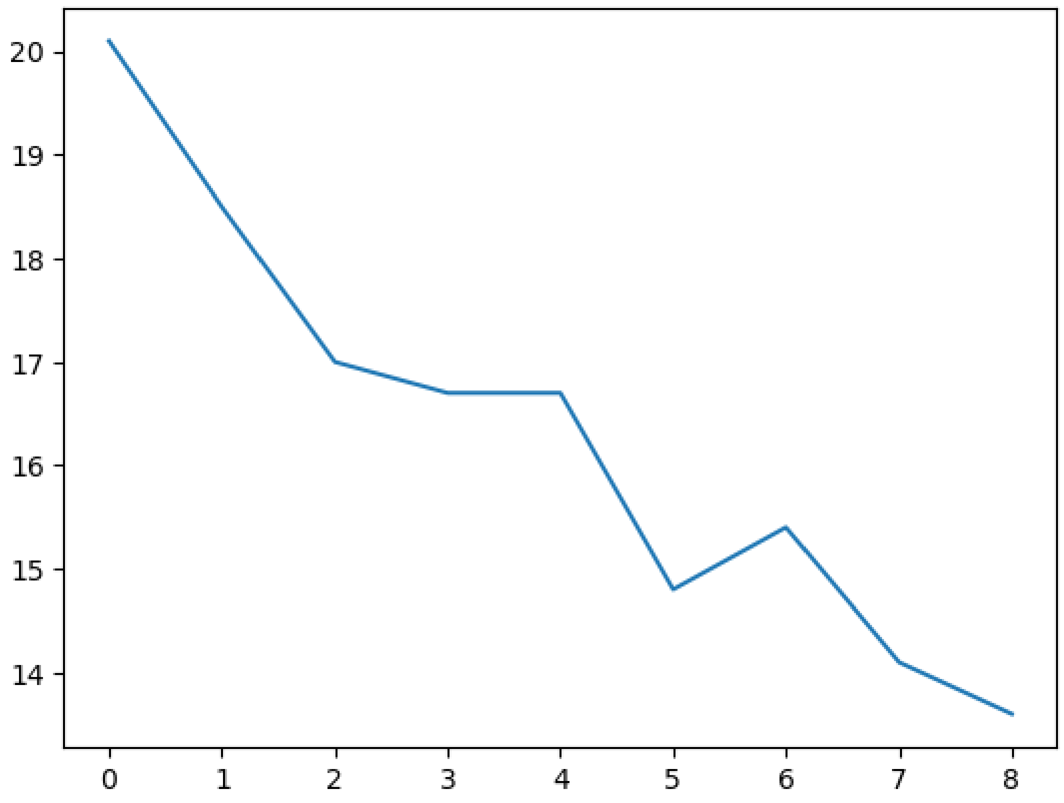
Matplotlib is a powerful library for creating static, animated, and interactive data visualizations in Python.

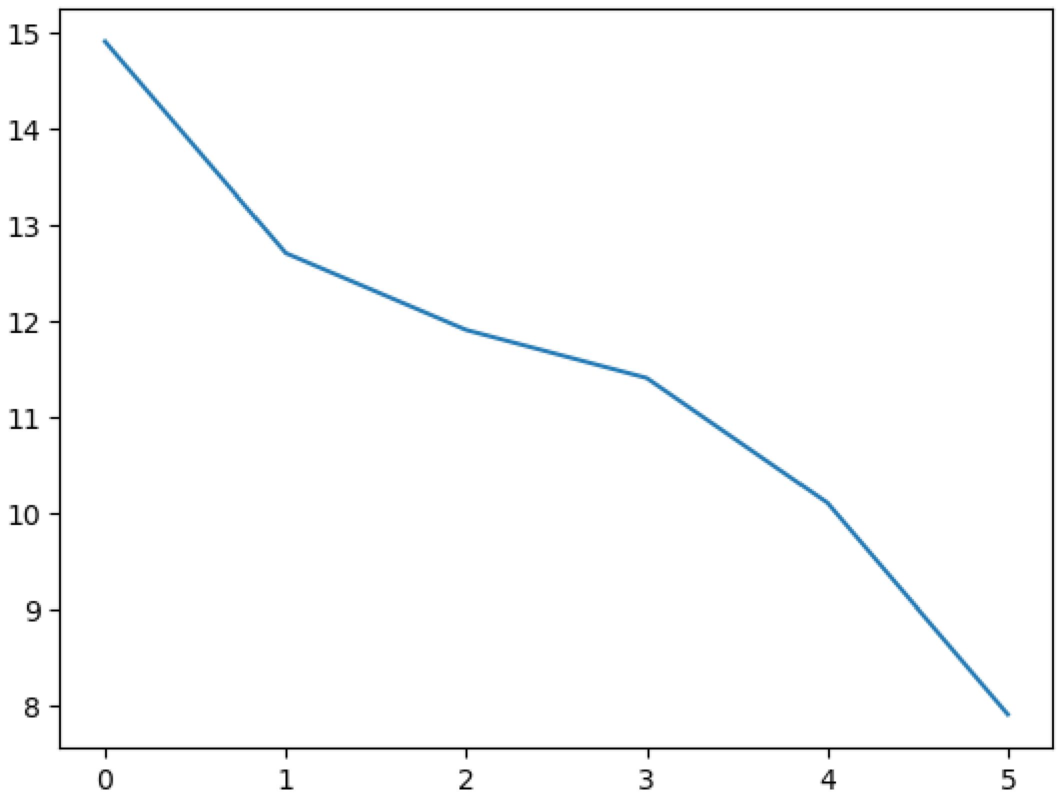
To use Matplotlib, we first need to import it.

In [ ]:
```
import matplotlib.pyplot as plt
```

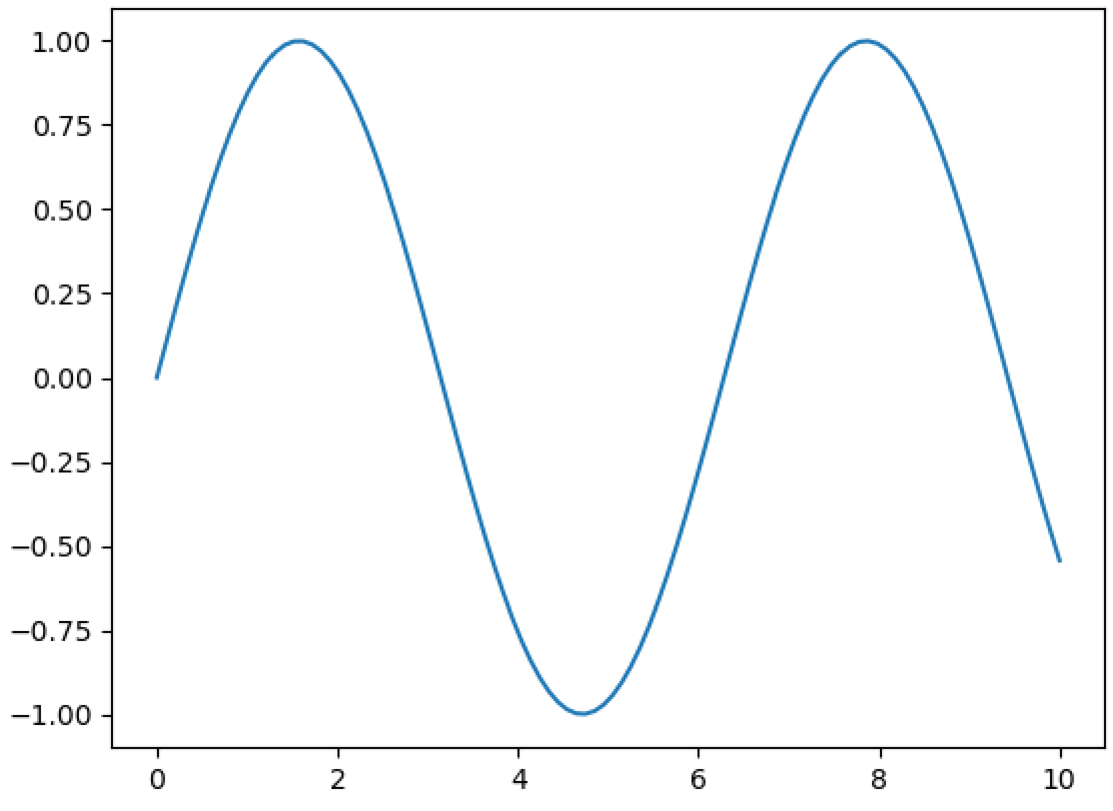
The most basic function in Matplotlib is the `plot()` function, which is used to create a line plot of data. For example:

In [ ]:
```
# Plot the two variables in the dataframe 'df' versus time order

# variable 1
plt.plot(df['cond1'])

# Show the plot
plt.show()

# variable 2
plt.plot(df['cond2'])

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





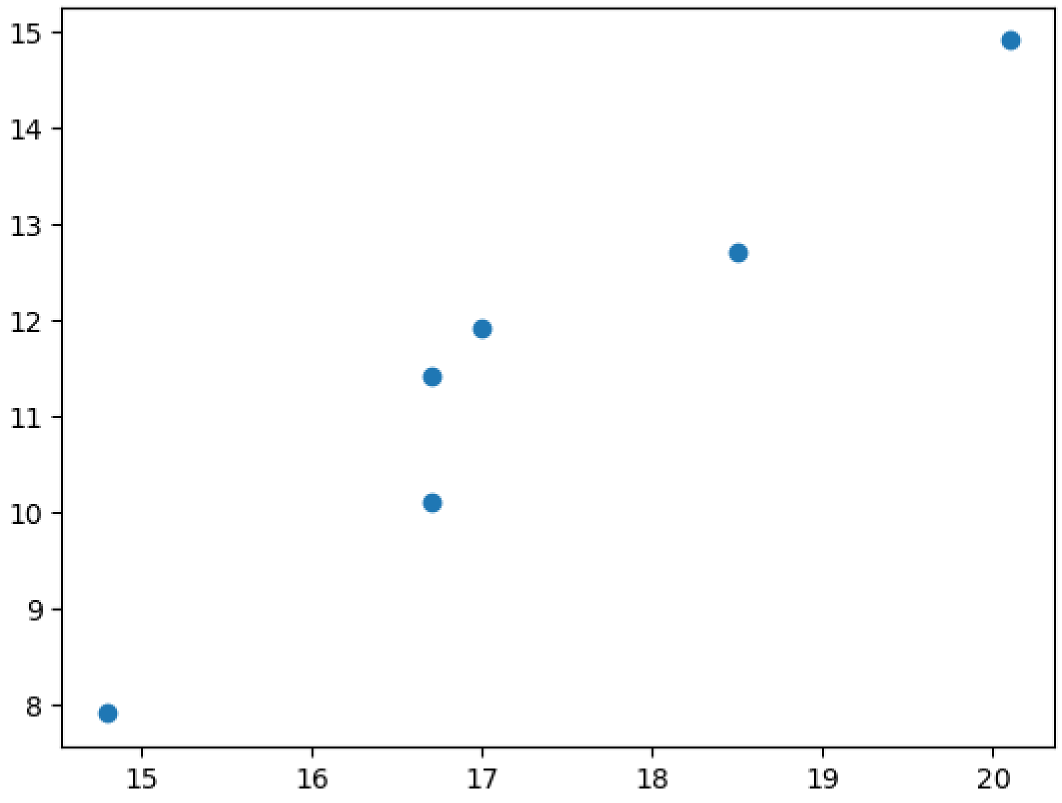
```
In [ ]: # Create some data
        x = np.linspace(0, 10, 100)
        y = np.sin(x)

        # Create a line plot
        plt.plot(x, y)

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



This will create a line plot of `y` versus `x`.

In addition to line plots, Matplotlib also provides a wide range of other plot types, such as scatter plots, bar plots, histograms, and more. For example:

In [ ]:
```
# Create a scatter plot
plt.scatter(df['cond1'], df['cond2'])

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

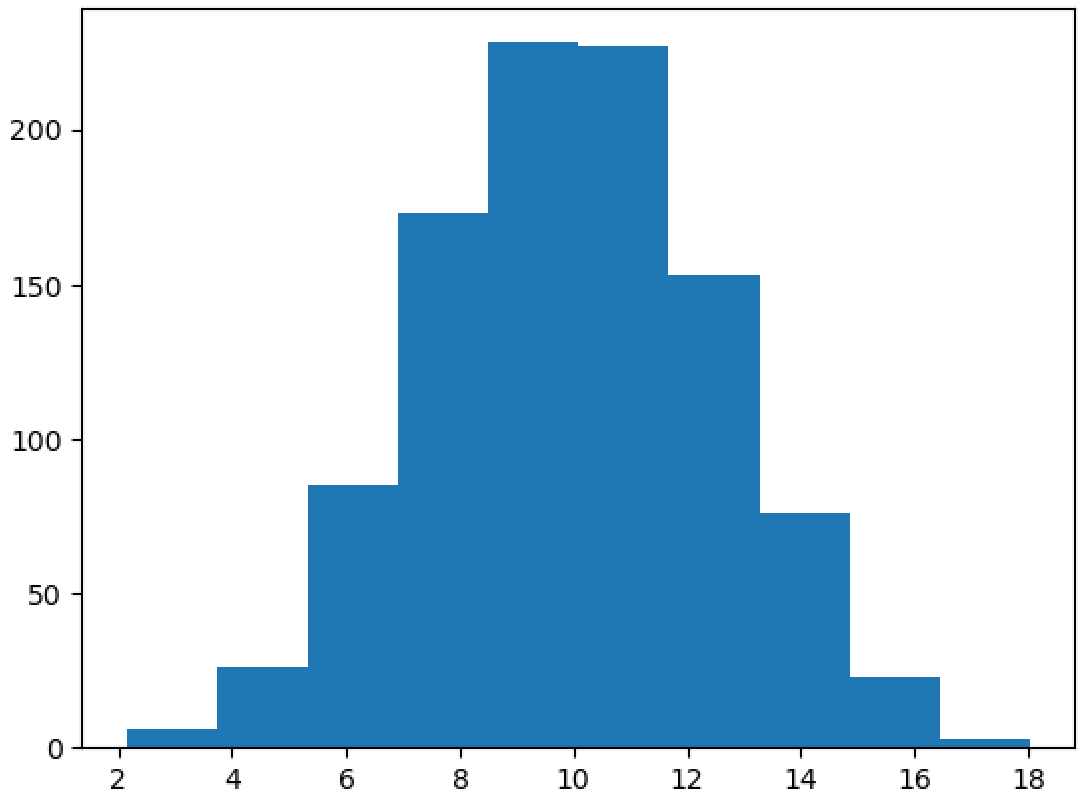


Let's generate a bigger dataset. We can draw random numbers from a given distribution (e.g., a normal distribution) Note: probability distributions will be discussed later on

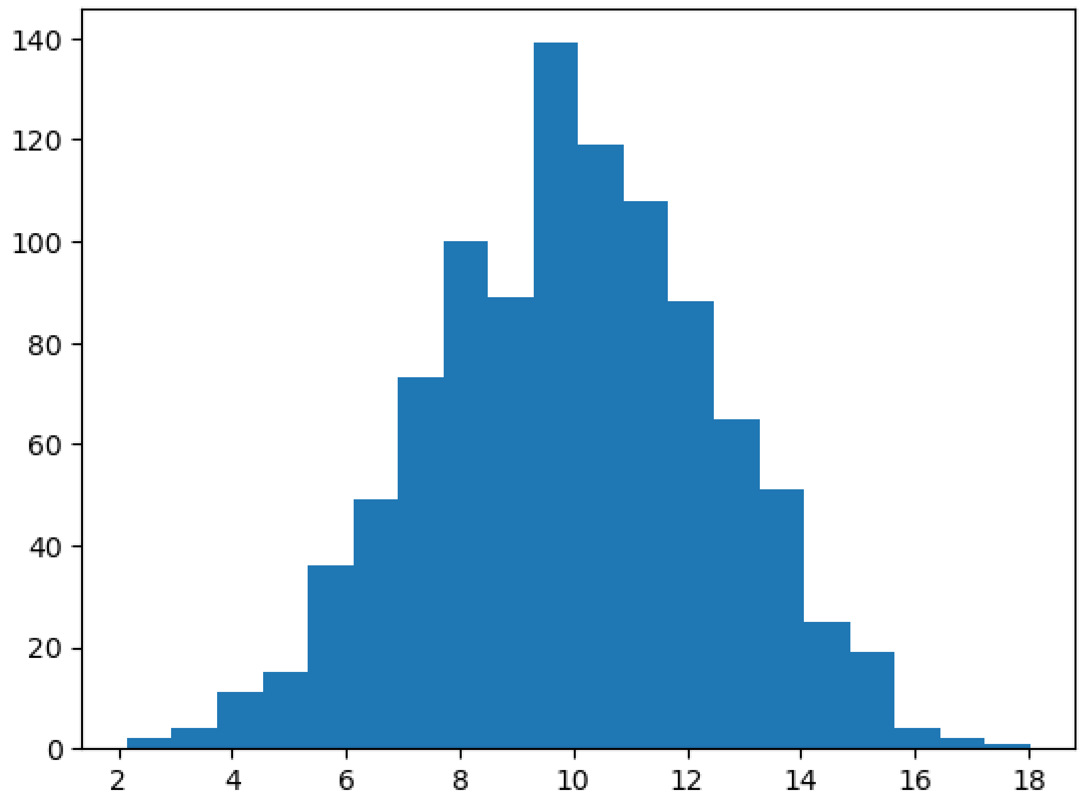
In [ ]:
```
# Generate a random array of 1000 elements from a normal distribution
mu = 10         # mean
sigma = 2.5     # standard deviation
y = np.random.normal(mu, sigma, 1000)
```

Let's plot the histogram of the random data

In [ ]:
```
# Create a histogram
plt.hist(y)

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



In [ ]:
```
# Create a histogram with a given number of bins
plt.hist(y,bins=20)

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



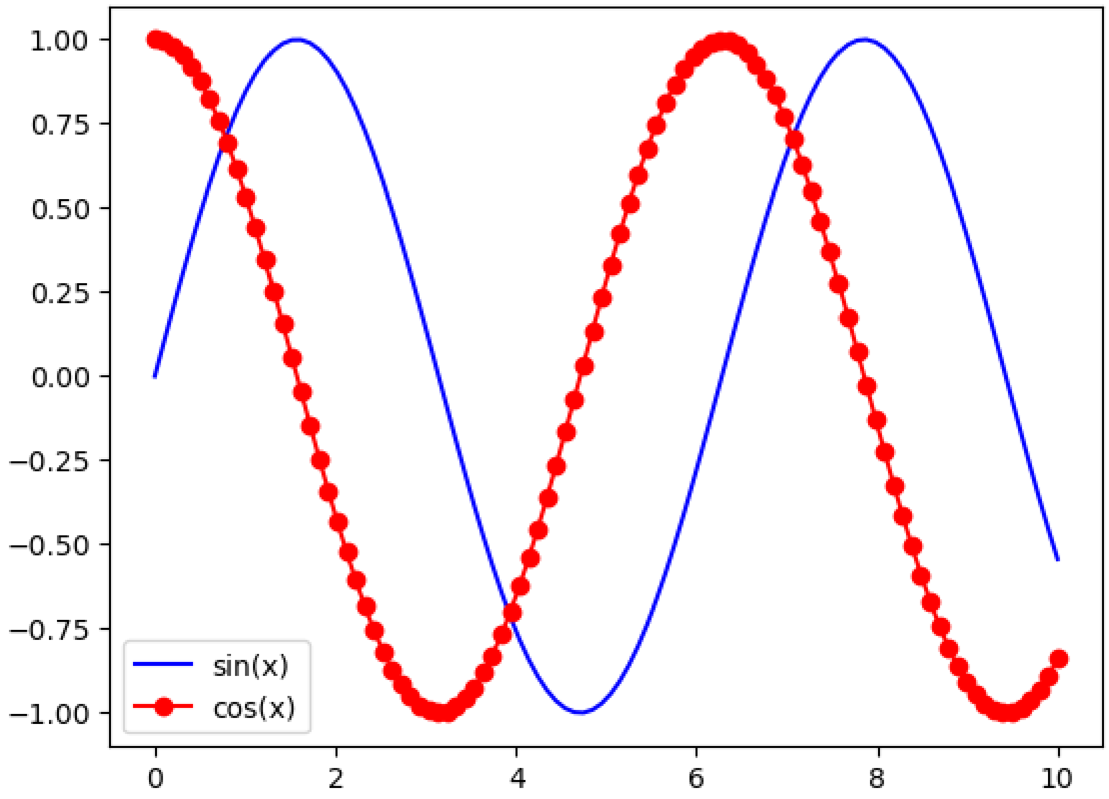
Hint: Choosing the number of bins approximately equal to the square root of the number of observations often works well in practise

There are several options within these basic functions that will allow you to improve data visualization. \ Each type of plot have both common parameters, e.g.

- `color` to set the color of the line.
- `marker` to choose the marker of the datapoint.
- `label` to assign a name to the curve that will be shown in the legend (`legend()`).

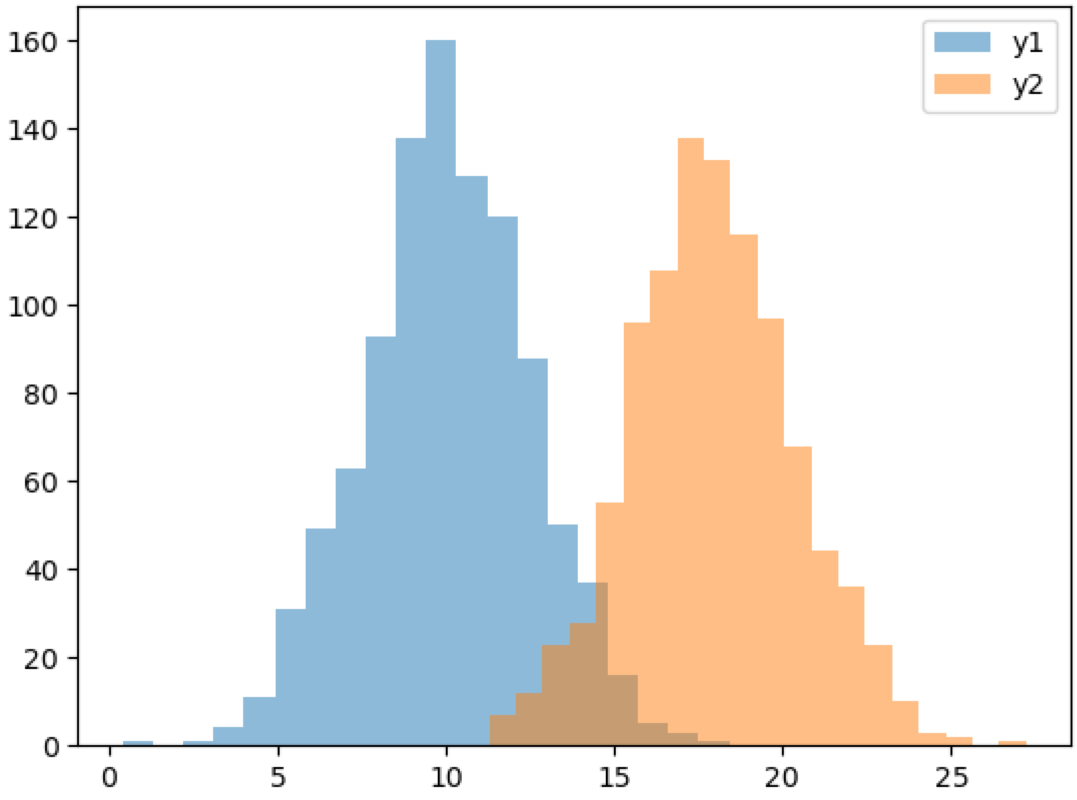
Here's a more complex example of line plot with two samples.

In [ ]:
```
# Create some data
x = np.linspace(0, 10, 100)
y1 = np.sin(x)
y2 = np.cos(x)

# Create a line plot
plt.plot(x, y1, color='blue', label='sin(x)')
plt.plot(x, y2, color='red', marker='o', label='cos(x)')

# Add a legend
plt.legend()

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



Other parameters, specific to the type of plot (e.g., the number of `bins` , or the probability `density` rather than the raw count in the histogram), can be found in the documentation.

In [ ]:
```
# Generate a random array of 1000 elements from a normal distribution
y1 = np.random.normal(mu, sigma, 1000)
y2 = np.random.normal(mu+8, sigma, 1000)

# Create a histogram
plt.hist(y1, bins=20, alpha=0.5, label='y1')
plt.hist(y2, bins=20, alpha=0.5, label='y2')
plt.legend()

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

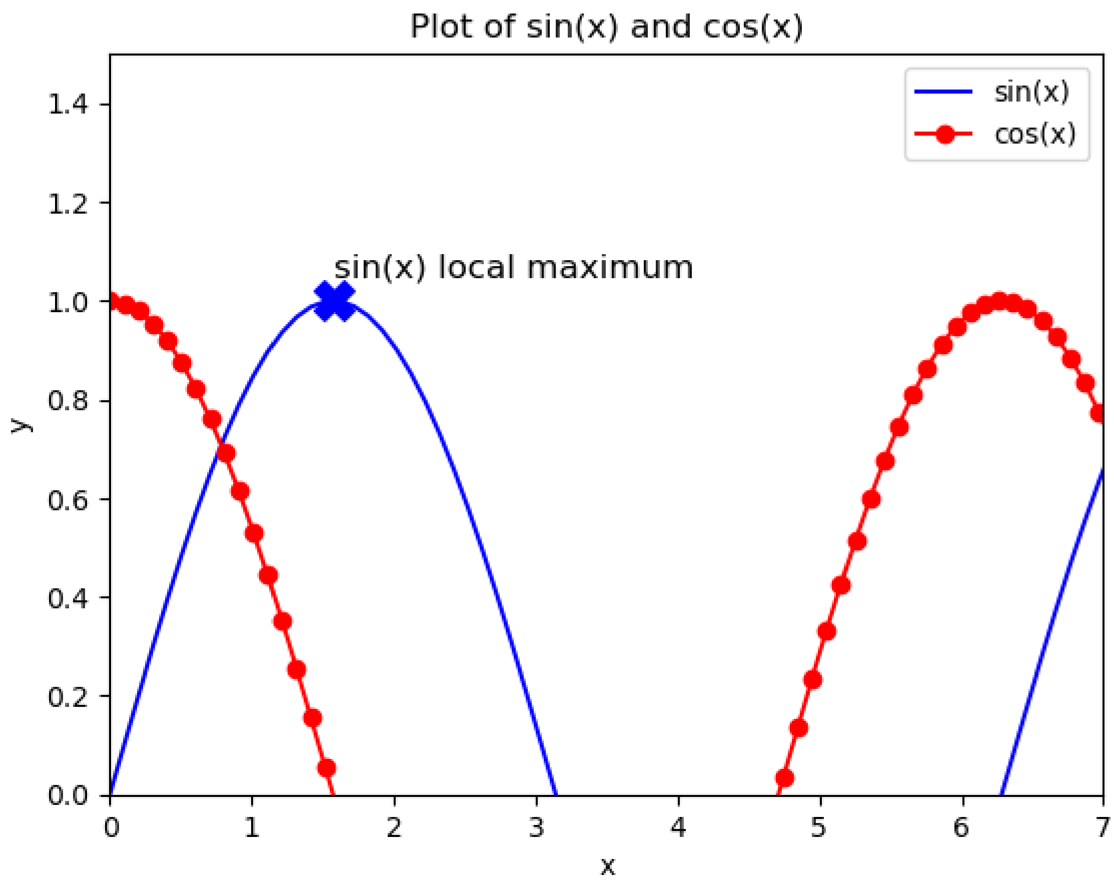


The plot itself can be further customized by:

- adding a title (`title()`)
- assigning labels to each axis (`xlabel()` and `ylabel()`)
- changing axis limits (`xlim()` and `ylim()`)
- showing the legend (`legend()`)
- adding text in a specific location (`text()`)

In [ ]:
```
# Create some data
x = np.linspace(0, 10, 100)
y1 = np.sin(x)
y2 = np.cos(x)

# Create a line plot
plt.plot(x, y1, color='blue', label='sin(x)')
plt.plot(x, y2, color='red', marker='o', label='cos(x)')

# Add a legend and title
plt.legend()
plt.title('Plot of sin(x) and cos(x)')
plt.xlabel('x')
plt.ylabel('y')
plt.xlim(0, 7)
plt.ylim(0, 1.5)

# Add text to call out a specific point
plt.text(np.pi/2, 1.05, 'sin(x) local maximum', size=12, ha='left')
plt.scatter(np.pi/2, 1, color='blue', marker='X', s=200)
```

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



Numerical summary of data (descriptive statistics)

Given a sample of n observations $x_1, x_2, \ldots, x_n$ (random variable X):

- Sample mean: $\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n}$
- Sample variance: $s^2 = \frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{n-1}$
- Sample standard dev: $s = \sqrt{\frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{n-1}}$
- Median: $P(X \le m) = P(X \ge m) = 1/2$ Note: Continuous probability function
- Quartiles: *3 points (Q1, median, Q3) that divide the dataset into 4 equal groups, each group comprising a quarter of the data*

The `describe()` function is used to generate summary statistics of the numerical columns of the DataFrame.

```
In [ ]: # Generate summary statistics for the dataframe 'df'
        df.describe()
```

Out[ ]:

	cond1	cond2
count	9.000000	6.000000
mean	16.322222	11.483333
std	2.099868	2.370162
min	13.600000	7.900000
25%	14.800000	10.425000
50%	16.700000	11.650000
75%	17.000000	12.500000
max	20.100000	14.900000

Or you can access individual descriptive statistics such as `mean()`, `stdev()`, or `var()`.

```
In [ ]: # Calculate the mean of a column
        df['cond1'].mean()
```

```
Out[ ]: 16.322222222222223
```

Numpy integrates the functions to compute the most common descriptive statistics:

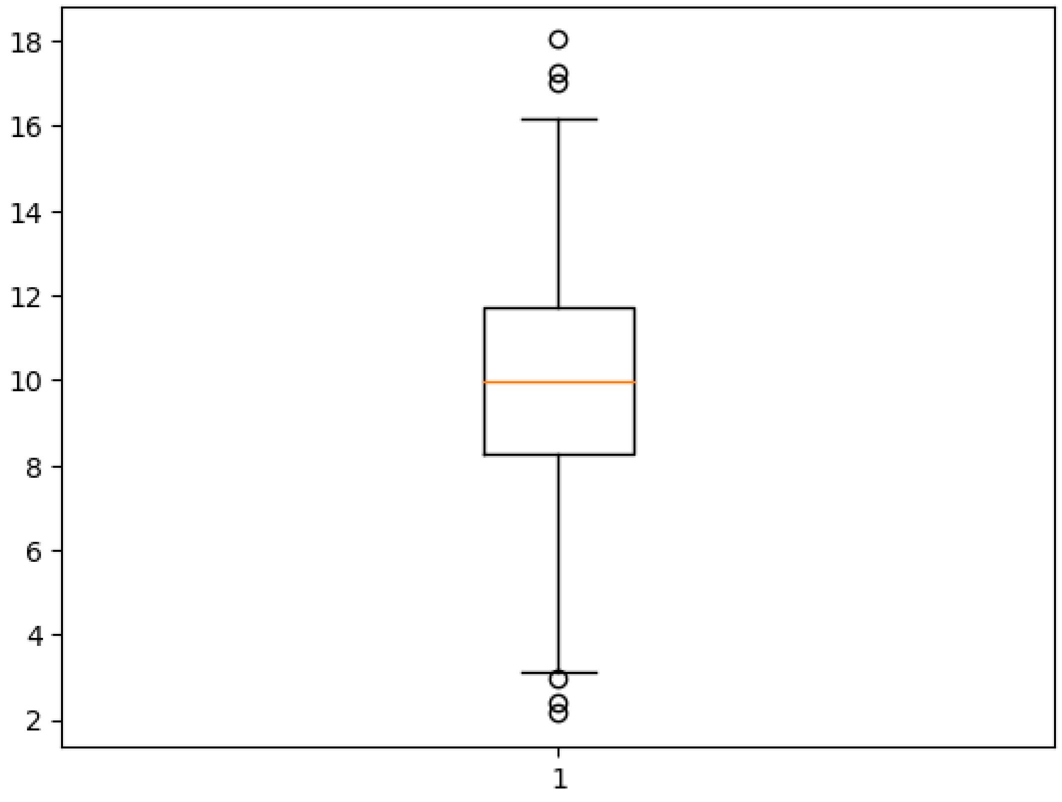
- `mean()` mean
- `std()` standard deviation
- `var()` variance
- `min()` minimum
- `max()` maximum
- `median()` median
- `percentile()` for computing any percentile or quartiles (Q1 = 25 percentile, Q3 = 75 percentile)

```
In [ ]: print("Mean:\t %.3f" % np.mean(df['cond1']))
        print("StDev:\t %.3f" % np.std(df['cond1']))
        print("Var:\t %.3f" % np.var(df['cond1']))
        print("Min:\t %.3f" % np.min(df['cond1']))
        print("Q1:\t %.3f" % np.percentile(df['cond1'], 25))
        print("Median:\t %.3f" % np.median(df['cond1']))
        print("Q3:\t %.3f" % np.percentile(df['cond1'], 75))
        print("Max:\t %.3f" % np.max(df['cond1']))
```

```
Mean:    16.322
StDev:   1.980
Var:     3.920
Min:     13.600
Q1:      14.800
Median:  16.700
Q3:      17.000
Max:     20.100
```

Another graphical tool to describe variation: Boxplot

```
In [ ]: # Boxplot of the random data drawn from the normal distribution
        plt.boxplot(y)

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



The boxplot:

- Boxplot

Upper whisker – Extends to the maximum data point within 1.5 box heights from the top of the box

Lower whisker – Extends to the minimum data point within 1.5 box heights from the bottom of the box

IF:
$x_i >$ Q3 + 1.5 (Q3 - Q1) OR
$x_i <$ Q1 - 1.5 (Q3 – Q1),

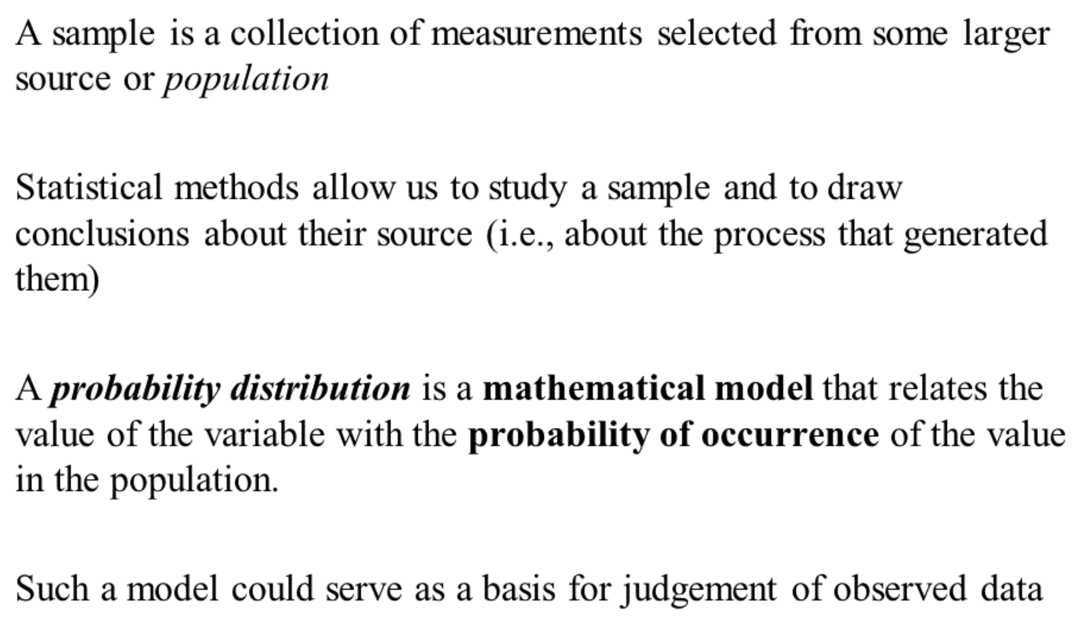
then:

x_i is signalled as an outlier in the boxplot

We can also perform mathematical operations on arrays, such as addition, subtraction, multiplication, and division. We will see more complex functions later on.

Probability distributions

Probability distributions (1/5)



A sample is a collection of measurements selected from some larger source or *population*

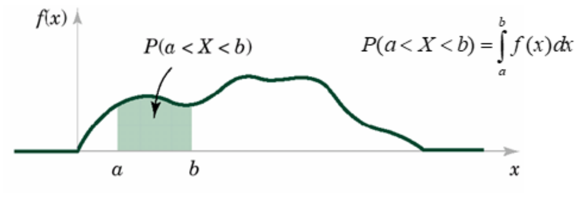
Statistical methods allow us to study a sample and to draw conclusions about their source (i.e., about the process that generated them)

A ***probability distribution*** is a **mathematical model** that relates the value of the variable with the **probability of occurrence** of the value in the population.

Such a model could serve as a basis for judgement of observed data

Probability distributions (2/5)

- *Continuous distribution*: variable expressed on a continuous scale
- **Probability density function** *f(x)* – continuous variables



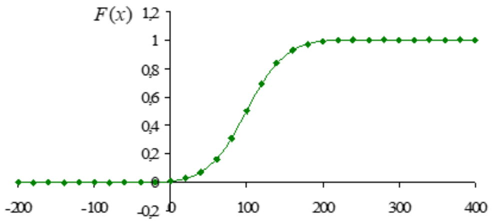
If X is acontinuous variable, then, for every $x_1 < x_2$:

$$P(x_1 \le X \le x_2) = P(x_1 < X \le x_2) =$$
$$= P(x_1 \le X < x_2) = P(x_1 < X < x_2)$$

- **Cumulative distribution function** *F(x)*:

$$F(x) = P(X \le x) = \int_{-\infty}^{x} f(u)du$$

For $-\infty < x < \infty$



Probability distributions (3/5)

If X is a random variable with probability density $f(x)$:

- Mean (expected value) $\mu = E(X) = \int_{-\infty}^{\infty} xf(x)dx$
- Variance $\sigma^2 = V(X) = \int_{-\infty}^{\infty} (x-\mu)^2 f(x)dx$
- Standard deviation $\sigma = \sqrt{V(X)}$

Try at home

Prove that: $\sigma^2 = V(X) = \int_{-\infty}^{\infty} (x-\mu)^2 f(x)dx = E(X^2) - \mu^2$

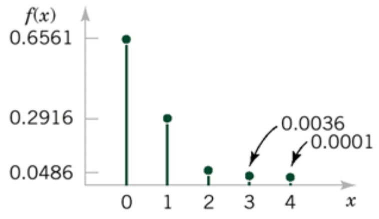
Probability distributions (4/5)

- *Discrete distribution*: variable can take on only certain values
- **Probability mass function $f(x_i)$**

Definition

For a discrete random variable X with possible values $x_1, x_2, \ldots, x_n$, the **probability mass function** is

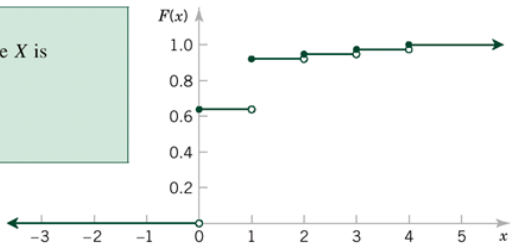
$$f(x_i) = P(X = x_i) \tag{3-6}$$



- **Cumulative distribution function:**

The **cumulative distribution function** of a discrete random variable X is

$$F(x) = P(X \le x) = \sum_{x_i \le x} f(x_i)$$



Probability distributions (5/5)

If X is a discrete random variable with probability mass $f(x_i)$:

- Mean (expected value) $\mu = E(X) = \sum_{i=1}^{n} x_i f(x_i)$
- Variance $\sigma^2 = V(X) = \sum_{i=1}^{n} (x_i - \mu)^2 f(x_i)$
- Standard deviation $\sigma = \sqrt{V(X)}$

Example of variance computation for a dataset with mean = 0.4

x	$x - 0.4$	$(x - 0.4)^2$	$f(x)$	$f(x)(x - 0.4)^2$
0	−0.4	0.16	0.6561	0.104976
1	0.6	0.36	0.2916	0.104976
2	1.6	2.56	0.0486	0.124416
3	2.6	6.76	0.0036	0.024336
4	3.6	12.96	0.0001	0.001296

$$V(X) = \sigma^2 = \sum_{i=1}^{5} f(x_i)(x_i - 0.4)^2 = 0.36$$

Probability distributions - Properties (1/2)

$$E(x)=\begin{cases}\sum_{\text{all } x_i} x_i p(x_i), & x_i \text{ is a discrete random variable}\\ \int_{-\infty}^{\infty} x f(x)dx, & x \text{ is a continuous random variable}\end{cases}$$

$$E[h(x)]=\begin{cases}\sum_{\text{all } x_i} h(x_i) p(x_i), & x_i \text{ is a discrete random variable}\\ \int_{-\infty}^{\infty} h(x) f(x)dx, & x \text{ is a continuous random variable}\end{cases}$$

$$E(c)=c$$

$$E(x)=\mu$$

$$E(cx)=cE(x)=c\mu$$

$$E[ch(x)]=cE[h(x)]$$

$$E[c_1h_1(x)+c_2h_2(x)]=c_1E[h_1(x)]+c_2E[h_2(x)]$$

$$\begin{aligned}E[(x-c)^2]&=E[x^2-2xc+c^2]\\&=E(x^2)-2cE(x)+c^2\end{aligned}$$

$$\begin{aligned}V(x)&=E[(x-\mu)^2]\\&=\sigma^2\end{aligned}$$

$$V(cx)=c^2\sigma^2$$

Probability distributions - Properties (2/2)

$$E(x_1+x_2)=\mu_1+\mu_2 \qquad \boxed{V(x_1+x_2)=\sigma_1^2+\sigma_2^2+2Cov(x_1,x_2)} \qquad Cov(x_1,x_2)=E[(x_1-\mu_1)(x_2-\mu_2)]$$

$$\boxed{V(x_1-x_2)=\sigma_1^2+\sigma_2^2-2Cov(x_1,x_2)}$$

If vars are independent:

$$E(x_1x_2)=E(x_1)E(x_2)=\mu_1\mu_2 \qquad E\left(\frac{x_1}{x_2}\right)\neq\frac{E(x_1)}{E(x_2)} \qquad V(x+x)=4\sigma^2 \qquad Cov(x,x)\equiv\sigma^2$$

Moments about the origin (not centred), about the mean (centred) of order k

$$h(x)=x^k$$

$$E(x^k)=\begin{cases}\sum_{\text{all } x_i} x_i^k p(x_i), & x_i \text{ is a discrete random variable}\\ \int_{-\infty}^{\infty} x^k f(x)dx, & x \text{ is a continuous random variable}\end{cases}$$

$$\boxed{E(x^2)=\mu^2+\sigma^2}$$

$$E[(x-\mu)^k]=\begin{cases}\sum_{\text{all } x_i} (x_i-\mu)^k p(x_i), & x_i \text{ is a discrete random variable}\\ \int_{-\infty}^{\infty} (x-\mu)^k f(x)dx, & x \text{ is a continuous random variable}\end{cases}$$

The normal (gaussian) distribution (1/3)

A random variable X with probability density function:

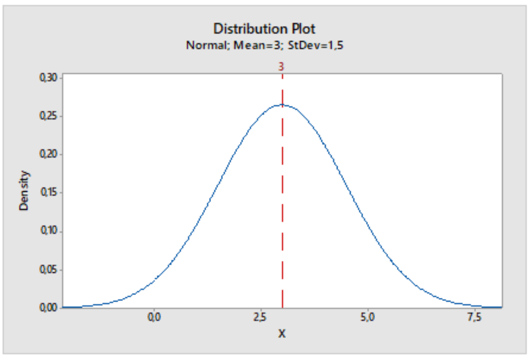
$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-(x-\mu)^2}{2\sigma^2}} \qquad \text{for } -\infty < x < \infty$$

has a normal distribution with parameters μ and σ where

$$-\infty < \mu < \infty \text{ and } \sigma > 0$$

Also,

$$E(x) = \mu \quad \text{and } V(x) = \sigma^2$$



The normal (gaussian) distribution (2/3)

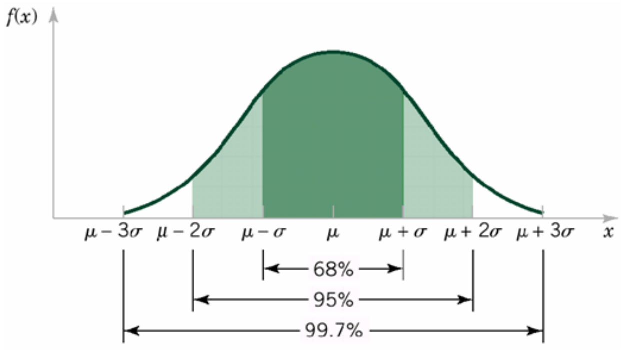
Chebyshev inequality: $P(|X-\mu| \geq k\sigma) \leq 1/k^2$ (unknown distrib.)

E.g., k=3 → $P(|X-\mu| \leq 3\sigma) \geq 88.89\%$

(we have a tool to state if an observation is unusually large or small)

If the distribution is normal:

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$



The normal (gaussian) distribution (3/3)

A normal random variable with $\mu = 0$ and $\sigma^2 = 1$ is called a *standard normal variable.*
A standard normal variable is usually denoted as Z.

Suppose X is a normal random variable with mean μ and variance σ^2.
Then

$$P(X \le x) = P\left(\frac{X-\mu}{\sigma} \le \frac{x-\mu}{\sigma}\right) = P\left(Z \le z\right) = \Phi(z)$$

Z is a standard normal random variable and $z = \left(x-\mu\right)/\sigma$ is the z-value obtained by standardizing x.

Introduction to `scipy.stats`

Scipy is a python library that is widely used for scientific and technical computing. The `stats` module in scipy is used for probability distributions and statistical functions. In this tutorial, we will learn about the basic functions in scipy.stats.

Probability Distribution Functions

scipy.stats has a wide range of probability distributions that can be used for statistical analysis. Some of the commonly used distributions are:

- Normal Distribution (`norm`)
- t-Student (`t`)
- Chi-squared (`chi2`)
- F (`f`)
- etc.

To use a probability distribution function, we first need to import the stats module from scipy.

In [ ]:
```
from scipy import stats
```

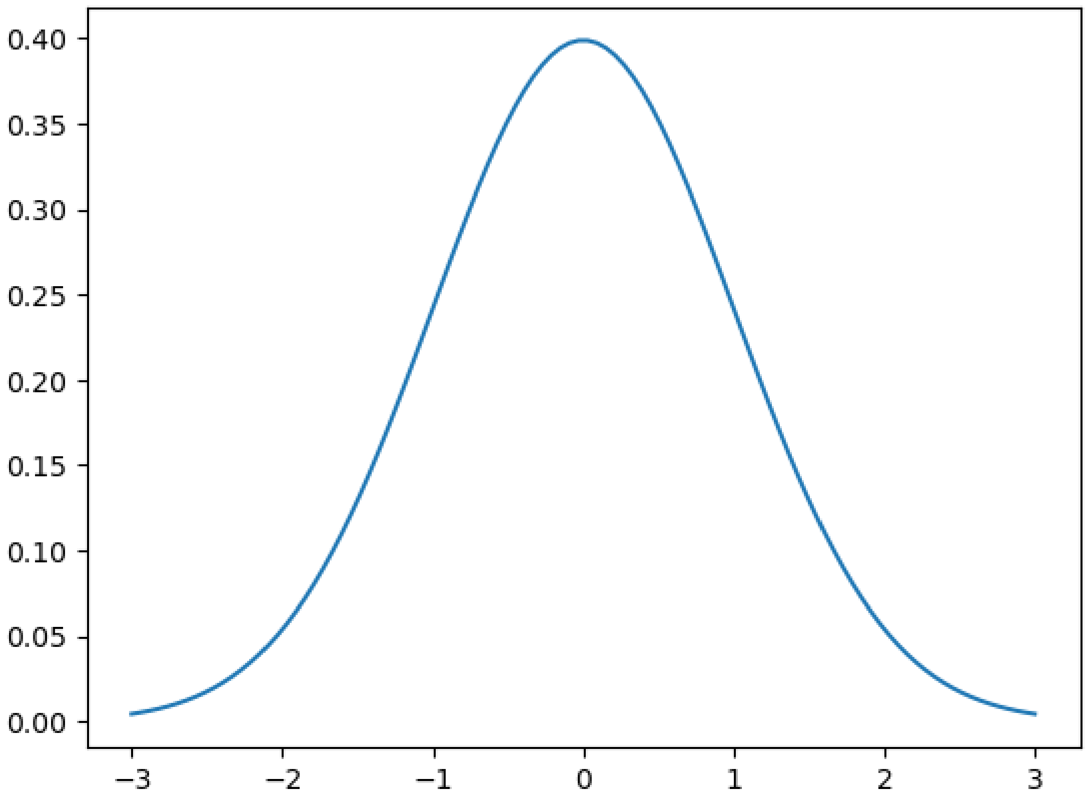
We can inspect everything we want from a distribution function using the following commands:

- Probability density function (`stats.*distribution_name*.pdf()`)
- Cumulative density function (`stats.*distribution_name*.cdf()`)

- Point probability function (or inverse cumulative probability) (`stats.*distribution_name*.ppf()`)

Let's use `stats.norm.pdf()` to plot a Normal distribution.

In [ ]:
```
# Plot the probability distribution function of a normal distribution
x = np.linspace(-3, 3, 100)
mu = 0
sigma = 1
y = stats.norm.pdf(x, mu, sigma)
plt.plot(x, y)
plt.show()
```

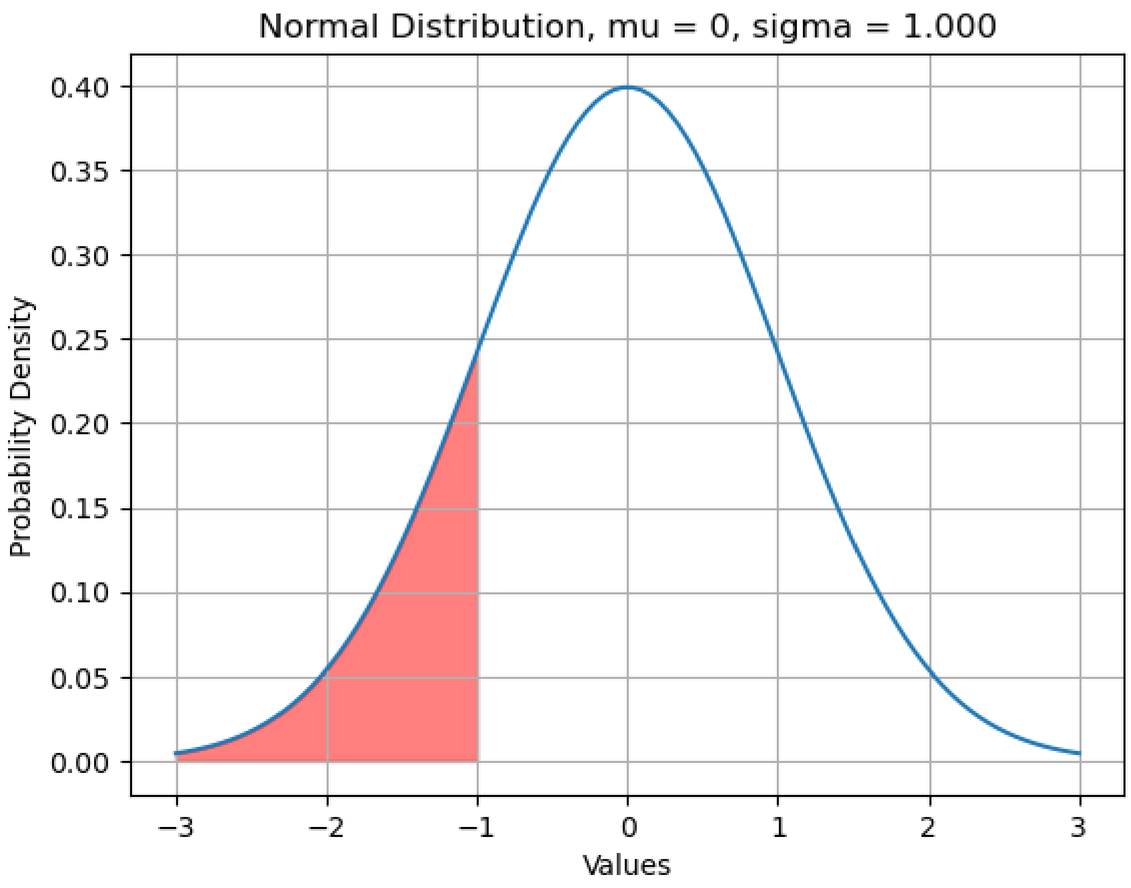


The function takes in input the points where you want to evaluate the function `x` and the parameters of the distribution (in this case, the mean `mu` and the StDev `sigma`).

We can graphically represent areas under the 'pdf' and compute the corresponding probability

In [ ]:
```
# Plot the cumulative distribution function of a normal distribution
plt.plot(x, y)

# Adding Title, Labels and Grid
plt.title("Normal Distribution, mu = %d, sigma = %.3f" % (mu, sigma))
plt.xlabel("Values")
plt.ylabel("Probability Density")
plt.grid(True)

# Filling the Probability Area that Z < z_0
z_0 = -1
x_fill = np.linspace(np.min(x), z_0, 100)
y_fill = stats.norm.pdf(x_fill, mu, sigma)
plt.fill_between(x_fill, y_fill, color='red', alpha=0.5)    #alpha is the transpar
```

```
# Showing Plot
plt.show()
```



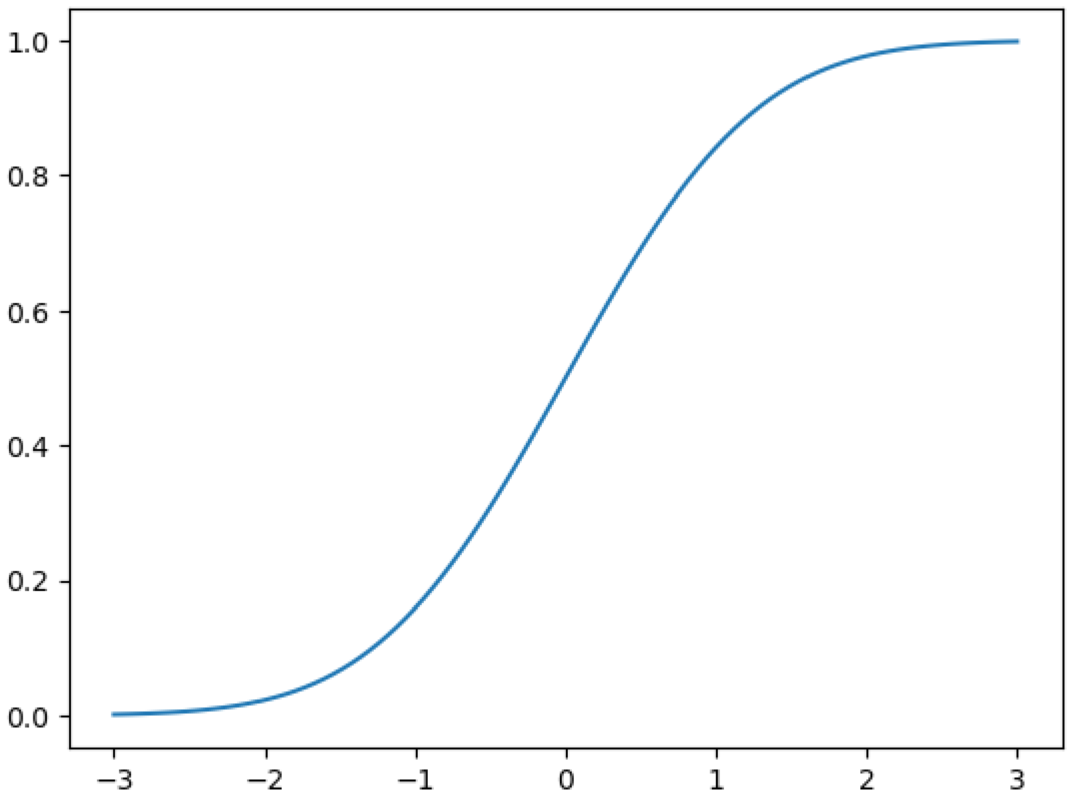
Let's compute the probability corresponding to the shaded area

In [ ]:
```
#calculate the cumulative normal of z_0
z_0 = -1
p1 = stats.norm.cdf(z_0)

print("z_0 = %.6f" % z_0)
print("Cumulative normal of z_0 = %.6f" % p1)
```

```
z_0 = -1.000000
Cumulative normal of z_0 = 0.158655
```

Let's use `stats.norm.cdf()` to plot a Normal cumulative density function.

In [ ]:
```
# Plot the cumulative density function of a normal distribution
x = np.linspace(-3, 3, 100)
mu = 0
sigma = 1
y = stats.norm.cdf(x, mu, sigma)
plt.plot(x, y)
plt.show()
```


1.0
0.8
0.6
0.4
0.2
0.0
−3
−2
−1
0
1
2
3