



**IPRC KARONGI**  
Integrated Polytechnic Regional College

P.O. Box 85 KARONGI- RWANDA  
Tel: +250 788871075  
Email: info@iprcwest.ac.rw  
[www.iprcarongi.rp.ac.rw](http://www.iprcarongi.rp.ac.rw)

---

**Department: Information and Communication Technology (ICT)**

**Option: Information Technology (IT)**

**Name of Module: Python and fundamentals of AI**

**Module code: ITLPA701**

**Number of credits: 10**

**Level: VII**

**Name of Module leader: BIZIMANA Zephania**

**Module leader signature:**

## **What is Python?**

Python is a popular programming language. It was created by Guido van Rossum and released in 1991.

### **It is used for:**

- web development (server-side),
- software development,
- mathematics,
- system scripting.

### **What can Python do?**

- Python can be used on a server to create web applications.
- Python can be used alongside software to create workflows.
- Python can connect to database systems. It can also read and modify files.
- Python can be used to handle big data and perform complex mathematics.
- Python can be used for rapid prototyping, or for production-ready software development.

### **Why Python?**

- Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
- Python has a simple syntax similar to the English language.
- Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
- Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
- Python can be treated in a procedural way, an object-oriented way or a functional way.

### **Good to know**

- The most recent major version of Python is Python 3, which we shall be using in this tutorial. However, Python 2, although not being updated with anything other than security updates, is still quite popular.
- In this tutorial Python will be written in a text editor. It is possible to write Python in an Integrated Development Environment, such as Thonny, Pycharm, Netbeans or Eclipse which are particularly useful when managing larger collections of Python files.

### **Python Syntax compared to other programming languages**

- Python was designed for readability and has some similarities to the English language with influence from mathematics.
- Python uses new lines to complete a command, as opposed to other programming languages which often use semicolons or parentheses.
- Python relies on indentation, using whitespace, to define scope; such as the scope of loops, functions and classes. Other programming languages often use curly brackets for this purpose.

## **Python versions**

For unreleased (in development) documentation, see [In Development Versions](#).

- Python 3.11.1, documentation released on 6 December 2022.
- Python 3.11.0, documentation released on 24 October 2022.
- Python 3.10.9, documentation released on 6 December 2022.
- Python 3.10.8, documentation released on 8 October 2022.
- Python 3.10.7, documentation released on 6 September 2022.
- Python 3.10.6, documentation released on 8 August 2022.
- Python 3.10.5, documentation released on 6 June 2022.
- Python 3.10.4, documentation released on 24 March 2022.
- Python 3.10.3, documentation released on 16 March 2022.
- Python 3.10.2, documentation released on 14 January 2022.
- Python 3.10.1, documentation released on 6 December 2021.
- Python 3.10.0, documentation released on 4 October 2021.
- Python 3.9.16, documentation released on 6 December 2022.
- Python 3.9.15, documentation released on 11 October 2022.
- Python 3.9.14, documentation released on 6 September 2022.
- Python 3.9.13, documentation released on 17 May 2022.
- Python 3.9.12, documentation released on 24 March 2022.
- Python 3.9.11, documentation released on 16 March 2022.
- Python 3.9.10, documentation released on 14 January 2022.
- Python 3.9.9, documentation released on 15 November 2021.
- Python 3.9.8, documentation released on 05 November 2021.

- Python 3.9.7, documentation released on 30 August 2021.
- Python 3.9.6, documentation released on 28 June 2021.
- Python 3.9.5, documentation released on 3 May 2021.
- Python 3.9.4, documentation released on 4 April 2021.
- Python 3.9.3, documentation released on 2 April 2021.
- Python 3.9.2, documentation released on 19 February 2021.
- Python 3.9.1, documentation released on 8 December 2020.
- Python 3.9.0, documentation released on 5 October 2020.
- Python 3.8.16, documentation released on 6 December 2022.
- Python 3.8.15, documentation released on 11 October 2022.
- Python 3.8.14, documentation released on 6 September 2022.
- Python 3.8.13, documentation released on 16 March 2022.
- Python 3.8.12, documentation released on 30 August 2021.
- Python 3.8.11, documentation released on 28 June 2021.
- Python 3.8.10, documentation released on 3 May 2021.
- Python 3.8.9, documentation released on 2 April 2021.
- Python 3.8.8, documentation released on 19 February 2021.
- Python 3.8.7, documentation released on 21 December 2020.
- Python 3.8.6, documentation released on 23 September 2020.
- Python 3.8.5, documentation released on 20 July 2020.
- Python 3.8.4, documentation released on 13 July 2020.
- Python 3.8.3, documentation released on 13 May 2020.
- Python 3.8.2, documentation released on 24 February 2020.
- Python 3.8.1, documentation released on 18 December 2019.
- Python 3.8.0, documentation released on 14 October 2019.
- Python 3.7.16, documentation released on 6 December 2022.
- Python 3.7.15, documentation released on 11 October 2022.
- Python 3.7.14, documentation released on 6 September 2022.

- Python 3.7.13, documentation released on 16 March 2022.
- Python 3.7.12, documentation released on 4 September 2021.
- Python 3.7.11, documentation released on 28 June 2021.
- Python 3.7.10, documentation released on 15 February 2021.
- Python 3.7.9, documentation released on 17 August 2020.
- Python 3.7.8, documentation released on 27 June 2020.
- Python 3.7.7, documentation released on 10 March 2020.
- Python 3.7.6, documentation released on 18 December 2019.
- Python 3.7.5, documentation released on 15 October 2019.
- Python 3.7.4, documentation released on 08 July 2019.
- Python 3.7.3, documentation released on 25 March 2019.
- Python 3.7.2, documentation released on 24 December 2018.
- Python 3.7.1, documentation released on 20 October 2018.
- Python 3.7.0, documentation released on 27 June 2018.
- Python 3.6.15, documentation released on 4 September 2021.
- Python 3.6.14, documentation released on 28 June 2021.
- Python 3.6.13, documentation released on 15 February 2021.
- Python 3.6.12, documentation released on 17 August 2020.
- Python 3.6.11, documentation released on 27 June 2020.
- Python 3.6.10, documentation released on 18 December 2019.
- Python 3.6.9, documentation released on 02 July 2019.
- Python 3.6.8, documentation released on 24 December 2018.
- Python 3.6.7, documentation released on 20 October 2018.
- Python 3.6.6, documentation released on 27 June 2018.
- Python 3.6.5, documentation released on 28 March 2018.
- Python 3.6.4, documentation released on 19 December 2017.
- Python 3.6.3, documentation released on 03 October 2017.
- Python 3.6.2, documentation released on 17 July 2017.

- Python 3.6.1, documentation released on 21 March 2017.
- Python 3.6.0, documentation released on 23 December 2016.
- Python 3.5.10, documentation released on 5 September 2020.
- Python 3.5.8, documentation released on 1 November 2019.
- Python 3.5.7, documentation released on 18 March 2019.
- Python 3.5.6, documentation released on 8 August 2018.
- Python 3.5.5, documentation released on 4 February 2018.
- Python 3.5.4, documentation released on 25 July 2017.
- Python 3.5.3, documentation released on 17 January 2017.
- Python 3.5.2, documentation released on 27 June 2016.
- Python 3.5.1, documentation released on 07 December 2015.
- Python 3.5.0, documentation released on 13 September 2015.
- Python 3.4.10, documentation released on 18 March 2019.
- Python 3.4.9, documentation released on 8 August 2018.
- Python 3.4.8, documentation released on 4 February 2018.
- Python 3.4.7, documentation released on 25 July 2017.
- Python 3.4.6, documentation released on 17 January 2017.
- Python 3.4.5, documentation released on 26 June 2016.
- Python 3.4.4, documentation released on 06 December 2015.
- Python 3.4.3, documentation released on 25 February 2015.
- Python 3.4.2, documentation released on 4 October 2014.
- Python 3.4.1, documentation released on 18 May 2014.
- Python 3.4.0, documentation released on 16 March 2014.
- Python 3.3.7, documentation released on 19 September 2017.
- Python 3.3.6, documentation released on 12 October 2014.
- Python 3.3.5, documentation released on 9 March 2014.
- Python 3.3.4, documentation released on 9 February 2014.
- Python 3.3.3, documentation released on 17 November 2013.

- Python 3.3.2, documentation released on 15 May 2013.
- Python 3.3.1, documentation released on 7 April 2013.
- Python 3.3.0, documentation released on 29 September 2012.
- Python 3.2.6, documentation released on 11 October 2014.
- Python 3.2.5, documentation released on 15 May 2013.
- Python 3.2.4, documentation released on 7 April 2013.
- Python 3.2.3, documentation released on 10 April 2012.
- Python 3.2.2, documentation released on 4 September 2011.
- Python 3.2.1, documentation released on 10 July 2011.
- Python 3.2, documentation released on 20 February 2011.
- Python 3.1.5, documentation released on 9 April 2012.
- Python 3.1.4, documentation released on 11 June 2011.
- Python 3.1.3, documentation released on 27 November 2010.
- Python 3.1.2, documentation released on 21 March 2010.
- Python 3.1.1, documentation released on 17 August 2009.
- Python 3.1, documentation released on 27 June 2009.
- Python 3.0.1, documentation released on 13 February 2009.
- Python 3.0, documentation released on 3 December 2008.
- Python 2.7.18, documentation released on 20 April 2020
- Python 2.7.17, documentation released on 19 October 2019
- Python 2.7.16, documentation released on 02 March 2019
- Python 2.7.15, documentation released on 30 April 2018
- Python 2.7.14, documentation released on 16 September 2017
- Python 2.7.13, documentation released on 17 December 2016
- Python 2.7.12, documentation released on 26 June 2016.
- Python 2.7.11, documentation released on 5 December 2015.
- Python 2.7.10, documentation released on 23 May 2015.
- Python 2.7.9, documentation released on 10 December 2014.

- Python 2.7.8, documentation released on 1 July 2014.
- Python 2.7.7, documentation released on 31 May 2014.
- Python 2.7.6, documentation released on 10 November 2013.
- Python 2.7.5, documentation released on 15 May 2013.
- Python 2.7.4, documentation released on 6 April 2013.
- Python 2.7.3, documentation released on 9 April 2012.
- Python 2.7.2, documentation released on 11 June 2011.
- Python 2.7.1, documentation released on 27 November 2010.
- Python 2.7, documentation released on 4 July 2010.
- Python 2.6.9, documentation released on 29 October 2013.
- Python 2.6.8, documentation released on 10 April 2012.
- Python 2.6.7, documentation released on 3 June 2011.
- Python 2.6.6, documentation released on 24 August 2010.
- Python 2.6.5, documentation released on 19 March 2010.
- Python 2.6.4, documentation released on 25 October 2009.
- Python 2.6.3, documentation released on 2 October 2009.
- Python 2.6.2, documentation released on 14 April 2009.
- Python 2.6.1, documentation released on 4 December 2008.
- Python 2.6, documentation released on 1 October 2008.
- Python 2.5.4, documentation released on 23 December 2008.
- Python 2.5.3, documentation released on 19 December 2008.
- Python 2.5.2, documentation released on 21 February 2008.
- Python 2.5.1, documentation released on 18 April 2007.
- Python 2.5, documentation released on 19 September 2006.
- Python 2.4.4, documentation released on 18 October 2006.
- Python 2.4.3, documentation released on 29 March 2006.
- Python 2.4.2, documentation released on 28 September 2005.
- Python 2.4.1, documentation released on 30 March 2005.



- Python 2.4, documentation released on 30 November 2004.
- Python 2.3.5, documentation released on 8 February 2005.
- Python 2.3.4, documentation released on 27 May 2004.
- Python 2.3.3, documentation released on 19 December 2003.
- Python 2.3.2, documentation released on 3 October 2003.
- Python 2.3.1, documentation released on 23 September 2003.
- Python 2.3, documentation released on 29 July 2003.
- Python 2.2.3, documentation released on 30 May 2003.
- Python 2.2.2, documentation released on 14 October 2002.
- Python 2.2.1, documentation released on 10 April 2002.
- Python 2.2p1, documentation released on 29 March 2002.
- Python 2.2, documentation released on 21 December 2001.
- Python 2.1.3, documentation released on 8 April 2002.
- Python 2.1.2, documentation released on 16 January 2002.
- Python 2.1.1, documentation released on 20 July 2001.
- Python 2.1, documentation released on 15 April 2001.
- Python 2.0.1, documentation released on 22 June 2001.
- Python 2.0, documentation released on 16 October 2000.
- Python 1.6, documentation released on 5 September 2000.
- Python 1.5.2p2, documentation released on 22 March 2000.
- Python 1.5.2p1, documentation released on 6 July 1999.
- Python 1.5.2, documentation released on 30 April 1999.
- Python 1.5.1p1, documentation released on 6 August 1998.
- Python 1.5.1, documentation released on 14 April 1998.
- Python 1.5, documentation released on 17 February 1998.
- Python 1.4, documentation released on 25 October 1996.

Example

```
print("Hello, World!")
```

Python Install

Many PCs and Macs will have python already installed.

To check if you have python installed on a Windows PC, search in the start bar for Python or run the following on the Command Line (cmd.exe):

```
C:\Users\Your Name>python --version
```

To check if you have python installed on a Linux or Mac, then on linux open the command line or on Mac open the Terminal and type:

```
python --version
```

If you find that you do not have Python installed on your computer, then you can download it for free from the following website: <https://www.python.org/>

## Python Quickstart

Python is an interpreted programming language, this means that as a developer you write Python (.py) files in a text editor and then put those files into the python interpreter to be executed.

The way to run a python file is like this on the command line:

```
C:\Users\Your Name>python helloworld.py
```

Where "helloworld.py" is the name of your python file.

Let's write our first Python file, called helloworld.py, which can be done in any text editor.

```
helloworld.py
```

```
print("Hello, World!")
```

Simple as that. Save your file. Open your command line, navigate to the directory where you saved your file, and run:

```
C:\Users\Your Name>python helloworld.py
```

The output should read:

Hello, World!

Congratulations, you have written and executed your first Python program.

## The Python Command Line

To test a short amount of code in python sometimes it is quickest and easiest not to write the code in a file. This is made possible because Python can be run as a command line itself.

Type the following on the Windows, Mac or Linux command line:

```
C:\Users\Your Name>python
```

Or, if the "python" command did not work, you can try "py":

```
C:\Users\Your Name>py
```

From there you can write any python, including our hello world example from earlier in the tutorial:

```
C:\Users\Your Name>python
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)]
on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello, World!")
```

Which will write "Hello, World!" in the command line:

```
C:\Users\Your Name>python
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)]
on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello, World!")
Hello, World!
```

Whenever you are done in the python command line, you can simply type the following to quit the python command line interface:

```
exit()
```

## Python Syntax

### Execute Python Syntax

As we learned in the previous page, Python syntax can be executed by writing directly in the Command Line:

```
>>> print("Hello, World!")  
Hello, World!
```

Or by creating a python file on the server, using the .py file extension, and running it in the Command Line:

```
C:\Users\Your Name>python myfile.py
```

## **Python Indentation**

Indentation refers to the spaces at the beginning of a code line.

Where in other programming languages the indentation in code is for readability only, the indentation in Python is very important.

Python uses indentation to indicate a block of code.

Example

```
if 5 > 2:  
    print("Five is greater than two!")
```

Python will give you an error if you skip the indentation:

Example

Syntax Error:

```
if 5 > 2:  
print("Five is greater than two!")
```

The number of spaces is up to you as a programmer, the most common use is four, but it has to be at least one.

Example

```
if 5 > 2:  
    print("Five is greater than two!")  
if 5 > 2:  
    print("Five is greater than two!")
```

You have to use the same number of spaces in the same block of code, otherwise Python will give you an error:

Example

Syntax Error:

```
if 5 > 2:
    print("Five is greater than two!")
    print("Five is greater than two!")
```

## Python Variables

In Python, variables are created when you assign a value to it:

### Example

Variables in Python:

```
x= 5
y = "Hello, World!"
```

Python has no command for declaring a variable.

You will learn more about variables in the [Python Variables](#) chapter.

## Comments

Python has commenting capability for the purpose of in-code documentation.

Comments start with a #, and Python will render the rest of the line as a comment:

### Example

Comments in Python:

```
#This is a comment.
print("Hello, World!")
```

## Python Comments

Comments can be used to explain Python code.

Comments can be used to make the code more readable.

Comments can be used to prevent execution when testing code.

### Creating a Comment

Comments starts with a #, and Python will ignore them:

### Example

```
#This is a comment
print("Hello, World!")
```

Comments can be placed at the end of a line, and Python will ignore the rest of the line:

Example

```
print("Hello, World!") #This is a comment
```

A comment does not have to be text that explains the code, it can also be used to prevent Python from executing code:

Example

```
#print("Hello, World!")  
print("Cheers, Mate!")
```

Multi Line Comments

Python does not really have a syntax for multi-line comments.

To add a multiline comment, you could insert a # for each line:

Example

```
#This is a comment  
#written in  
#more than just one line  
print("Hello, World!")
```

Or, not quite as intended, you can use a multiline string.

Since Python will ignore string literals that are not assigned to a variable, you can add a multiline string (triple quotes) in your code, and place your comment inside it:

Example

```
"""  
    "  
    "  
    "  
This is a comment  
written in  
more than just one line  
"""  
print("Hello, World!")
```

If the string is not assigned to a variable, Python will read the code, but then ignore it, and you have made a multiline comment.

## **Python Variables**

Variables

Variables are containers for storing data values.

## Creating Variables

Python has no command for declaring a variable.

A variable is created the moment you first assign a value to it.

### Example

```
x = 5
y = "John"
print(x)
print(y)
```

Variables do not need to be declared with any particular *type*, and can even change type after they have been set.

### Example

```
x = 4    # x is of type int
x = "Sally" # x is now of type str
print(x)
```

## Casting

If you want to specify the data type of a variable, this can be done with casting.

### Example

```
x = str(3)  # x will be '3'
y = int(3)  # y will be 3
z = float(3) # z will be 3.0
```

## Get the Type

You can get the data type of a variable with the `type()` function.

### Example

```
x = 5
y = "John"
print(type(x))
print(type(y))
```

## Single or Double Quotes?

String variables can be declared either by using single or double quotes:

Example

```
x = "John"
# is the same as
x = 'John'
```

[Try it Yourself »](#)

Case-Sensitive

Variable names are case-sensitive.

Example

This will create two variables:

```
a = 4
A = "Sally"
#A will not overwrite a
```

## Python - Variable Names

Variable Names

A variable can have a short name (like x and y) or a more descriptive name (age, carname, total\_volume). Rules for Python variables:

- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and \_ )
- Variable names are case-sensitive (age, Age and AGE are three different variables)

Example

Legal variable names:

```
myvar = "John"
my_var = "John"
_my_var = "John"
myVar = "John"
MYVAR = "John"
myvar2 = "John"
```

[Try it Yourself »](#)

Example

Illegal variable names:



```
2myvar = "John"  
my-var = "John"  
my var = "John"
```

## Multi Words Variable Names

Variable names with more than one word can be difficult to read.

There are several techniques you can use to make them more readable:

### Camel Case

Each word, except the first, starts with a capital letter:

```
myVariableName = "John"
```

### Pascal Case

Each word starts with a capital letter:

```
MyVariableName = "John"
```

### Snake Case

Each word is separated by an underscore character:

```
my_variable_name = "John"
```

## Python Variables - Assign Multiple Values

### Many Values to Multiple Variables

Python allows you to assign values to multiple variables in one line:

#### Example

```
x, y, z = "Orange", "Banana", "Cherry"  
print(x)  
print(y)  
print(z)
```

**Try it Yourself »**

**Note:** Make sure the number of variables matches the number of values, or else you will get an error.

## One Value to Multiple Variables

And you can assign the *same* value to multiple variables in one line:

Example

```
x = y = z = "Orange"
print(x)
print(y)
print(z)
```

[Try it Yourself »](#)

## Unpack a Collection

If you have a collection of values in a list, tuple etc. Python allows you to extract the values into variables. This is called *unpacking*.

Example

Unpack a list:

```
fruits = ["apple", "banana", "cherry"]
x, y, z = fruits
print(x)
print(y)
print(z)
```

## Python - Output Variables

### Output Variables

The Python print() function is often used to output variables.

Example

```
x = "Python is awesome"
print(x)
```

[Try it Yourself »](#)

In the print() function, you output multiple variables, separated by a comma:

Example

```
x = "Python"
y = "is"
z = "awesome"
print(x,y,z)
```

### Try it Yourself »

You can also use the + operator to output multiple variables:

Example

```
x = "Python "  
y = "is "  
z = "awesome"  
print(x + y + z)
```

### Try it Yourself »

Notice the space character after "Python " and "is ", without them the result would be "Pythonisawesome".

For numbers, the + character works as a mathematical operator:

Example

```
x = 5  
y = 10  
print(x + y)
```

### Try it Yourself »

In the print() function, when you try to combine a string and a number with the + operator, Python will give you an error:

Example

```
x= 5  
y= "John"  
print(x + y)
```

The best way to output multiple variables in the print() function is to separate them with commas, which even support different data types:

Example

```
x= 5  
y= "John"  
print(x, y)
```

## Python - Global Variables

Global Variables

Variables that are created outside of a function (as in all of the examples above) are known as global variables.

Global variables can be used by everyone, both inside of functions and outside.

#### Example

Create a variable outside of a function, and use it inside the function

```
x = "awesome"

def myfunc():
    print("Python is " + x)

myfunc()
```

If you create a variable with the same name inside a function, this variable will be local, and can only be used inside the function. The global variable with the same name will remain as it was, global and with the original value.

#### Example

Create a variable inside a function, with the same name as the global variable

```
x = "awesome"

def myfunc():
    x = "fantastic"
    print("Python is " + x)
myfunc()
print("Python is " + x)
```

### **The global Keyword**

Normally, when you create a variable inside a function, that variable is local, and can only be used inside that function.

To create a global variable inside a function, you can use the global keyword.

#### Example

If you use the global keyword, the variable belongs to the global scope:

```
def myfunc():
    global x
    x = "fantastic"
myfunc()
print("Python is " + x)
```

Also, use the global keyword if you want to change a global variable inside a function.

### Example

To change the value of a global variable inside a function, refer to the variable by using the global keyword:

```
x= "awesome"
```

```
def myfunc():  
    global x  
    x= "fantastic"  
myfunc()  
print("Python is " + x)
```

## Python - Variable Exercises

### Test Yourself With Exercises

Now you have learned a lot about variables, and how to use them in Python.

Are you ready for a test?

Try to insert the missing part to make the code work as expected:

Exercise:

1. Create a variable named carname and assign the value Volvo to it.

```
 = "
```

```
carname = "Volvo"
```

2. Create a variable named x and assign the value 50 to it.

```
 = 
```

Answer:

```
x = 50
```

3. Display the sum of 5 + 10, using two variables: x and y.

```
x = 5
y = 10
print(x + y)
```

4. Create a variable called z, assign x + y to it, and display the result.

- x = 5
- y = 10
- = x + y
- print()

Answer:

```
x = 5
y = 10
z = x + y
print(z)
```

5. Remove the illegal characters in the variable name:

2my-first\_name = "John"

**Answer:**

myfirst\_name = "John"

6. Insert the correct syntax to assign the same value to all three variables in one code line.

x  y  z  "Orange"

Answer:

```
x = y = z = "Orange"
```

7. Insert the correct keyword to make the variable x belong to the global scope.

def myfunc():

x

x = "fantastic"

Answer:

```
def myfunc():  
    global x  
    x = "fantastic"
```

## Python Data Types

### Built-in Data Types

In programming, data type is an important concept.

Variables can store data of different types, and different types can do different things.

Python has the following data types built-in by default, in these categories:

Text Type:	str
Numeric Types:	int, float, complex
Sequence Types:	list, tuple, range
Mapping Type:	dict
Set Types:	set, frozenset
Boolean Type:	bool
Binary Types:	bytes, bytearray, memoryview
None Type:	NoneType

### Getting the Data Type

You can get the data type of any object by using the `type()` function:

#### Example

Print the data type of the variable x:

```
x = 5  
print(type(x))
```

## Setting the Data Type

In Python, the data type is set when you assign a value to a variable:

Example	Data Type
<code>x = "Hello World"</code>	str
<code>x = 20</code>	int
<code>x = 20.5</code>	float
<code>x = 1j</code>	complex
<code>x = ["apple", "banana", "cherry"]</code>	list
<code>x = ("apple", "banana", "cherry")</code>	tuple
<code>x = range(6)</code>	range
<code>x = {"name" : "John", "age" : 36}</code>	dict
<code>x = {"apple", "banana", "cherry"}</code>	set



<code>x = frozenset({"apple", "banana", "cherry"})</code>	<code>frozenset</code>
<code>x = True</code>	<code>bool</code>
<code>x = b"Hello"</code>	<code>bytes</code>
<code>x = bytearray(5)</code>	<code>bytearray</code>
<code>x = memoryview(bytes(5))</code>	<code>memoryview</code>
<code>x = None</code>	<code>NoneType</code>

### Setting the Specific Data Type

If you want to specify the data type, you can use the following constructor functions:

<b>Example</b>	<b>Data Type</b>
<code>x = str("Hello World")</code>	<code>str</code>
<code>x = int(20)</code>	<code>int</code>

<code>x = float(20.5)</code>	float
<code>x = complex(1j)</code>	complex
<code>x = list(("apple", "banana", "cherry"))</code>	list
<code>x = tuple(("apple", "banana", "cherry"))</code>	tuple
<code>x = range(6)</code>	range
<code>x = dict(name="John", age=36)</code>	dict
<code>x = set(("apple", "banana", "cherry"))</code>	set
<code>x = frozenset(("apple", "banana", "cherry"))</code>	frozenset
<code>x = bool(5)</code>	bool
<code>x = bytes(5)</code>	bytes
<code>x = bytearray(5)</code>	bytearray

```
x = memoryview(bytes(5))
```

memoryview

Exercise:

The following code example would print the data type of x, what data type would that be?

```
x = 5  
print(type(x))
```

## Python Numbers

---

### Python Numbers

There are three numeric types in Python:

- int
- float
- complex

Variables of numeric types are created when you assign a value to them:

Example

```
x = 1 # int  
y = 2.8 # float  
z = 1j # complex
```

To verify the type of any object in Python, use the type() function:

Example

```
print(type(x))  
print(type(y))  
print(type(z))
```

[Try it Yourself »](#)

## Int

Int, or integer, is a whole number, positive or negative, without decimals, of unlimited length.

Example

Integers:

```
x = 1
y = 35656222554887711
z = -3255522
```

```
print(type(x))
print(type(y))
print(type(z))
```

[Try it Yourself »](#)

## Float

Float, or "floating point number" is a number, positive or negative, containing one or more decimals.

Example

Floats:

```
x = 1.10
y = 1.0
z = -35.59
```

```
print(type(x))
print(type(y))
print(type(z))
```

[Try it Yourself »](#)

Float can also be scientific numbers with an "e" to indicate the power of 10.

Example

Floats:

```
x = 35e3
y = 12E4
```

```
z = -87.7e100
```

```
print(type(x))  
print(type(y))  
print(type(z))
```

[Try it Yourself »](#)

## Complex

Complex numbers are written with a "j" as the imaginary part:

### Example

Complex:

```
x = 3+5j  
y = 5j  
z = -5j
```

```
print(type(x))  
print(type(y))  
print(type(z))
```

## Type Conversion

You can convert from one type to another with the `int()`, `float()`, and `complex()` methods:

### Example

Convert from one type to another:

```
x = 1    # int  
y = 2.8  # float  
z = 1j   # complex
```

```
#convert from int to float:  
a = float(x)
```

```
#convert from float to int:  
b = int(y)
```

```
#convert from int to complex:  
c = complex(x)
```

```
print(a)  
print(b)
```

```
print(c)
```

```
print(type(a))
```

```
print(type(b))
```

```
print(type(c))
```

**Note:** You cannot convert complex numbers into another number type.

## Random Number

Python does not have a `random()` function to make a random number, but Python has a built-in module called `random` that can be used to make random numbers:

### Example

Import the `random` module, and display a random number between 1 and 9:

```
import random
```

```
print(random.randrange(1, 10))
```

Exercise:

Insert the correct syntax to convert `x` into a floating point number.

```
x = 5
```

```
x =  (x)
```

---

Answer:

```
x = 5
x = float(x)
```

## Python Casting

### Specify a Variable Type

There may be times when you want to specify a type on to a variable. This can be done with casting. Python is an object-orientated language, and as such it uses classes to define data types, including its primitive types.

Casting in python is therefore done using constructor functions:

- `int()` - constructs an integer number from an integer literal, a float literal (by removing all decimals), or a string literal (providing the string represents a whole number)

- float() - constructs a float number from an integer literal, a float literal or a string literal (providing the string represents a float or an integer)
- str() - constructs a string from a wide variety of data types, including strings, integer literals and float literals

Example

Integers:

```
x = int(1) # x will be 1
y = int(2.8) # y will be 2
z = int("3") # z will be 3
```

**Try it Yourself »**

Example

Floats:

```
x = float(1) # x will be 1.0
y = float(2.8) # y will be 2.8
z = float("3") # z will be 3.0
w = float("4.2") # w will be 4.2
```

**Try it Yourself »**

Example

Strings:

```
x = str("s1") # x will be 's1'
y = str(2) # y will be '2'
z = str(3.0) # z will be '3.0'
```

## Python Strings

Strings

Strings in python are surrounded by either single quotation marks, or double quotation marks.

'hello' is the same as "hello".

You can display a string literal with the print() function:

Example

```
print("Hello")
print('Hello')
```

### Assign String to a Variable

Assigning a string to a variable is done with the variable name followed by an equal sign and the string:

#### Example

```
a = "Hello"
print(a)
```

### Multiline Strings

You can assign a multiline string to a variable by using three quotes:

#### Example

You can use three double quotes:

```
a = """Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua."""
print(a)
```

Or three single quotes:

#### Example

```
a = "Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua."
print(a)
```

**Note:** in the result, the line breaks are inserted at the same position as in the code.

### Strings are Arrays

Like many other popular programming languages, strings in Python are arrays of bytes representing unicode characters.

However, Python does not have a character data type, a single character is simply a string with a length of 1.

Square brackets can be used to access elements of the string.



## Example

Get the character at position 1 (remember that the first character has the position 0):

```
a = "Hello, World!"  
print(a[1])
```

## Looping Through a String

Since strings are arrays, we can loop through the characters in a string, with a for loop.

## Example

Loop through the letters in the word "banana":

```
for x in "banana":  
    print(x)
```

Learn more about For Loops in our [Python For Loops](#) chapter.

## String Length

To get the length of a string, use the len() function.

## Example

The len() function returns the length of a string:

```
a = "Hello, World!"  
print(len(a))
```

## Check String

To check if a certain phrase or character is present in a string, we can use the keyword in.

## Example

Check if "free" is present in the following text:

```
txt = "The best things in life are free!"  
print("free" in txt)  
Use it in an if statement:
```

## Example

Print only if "free" is present:

```
txt = "The best things in life are free!"  
if "free" in txt:  
    print("Yes, 'free' is present.")
```

Learn more about If statements in our [Python If...Else](#) chapter.

## Check if NOT

To check if a certain phrase or character is NOT present in a string, we can use the keyword not in.

Example

Check if "expensive" is NOT present in the following text:

```
txt = "The best things in life are free!"  
print("expensive" not in txt)
```

Use it in an if statement:

Example

print only if "expensive" is NOT present:

```
txt = "The best things in life are free!"  
if "expensive" not in txt:  
    print("No, 'expensive' is NOT present.")
```

## Python - Slicing Strings

Slicing

You can return a range of characters by using the slice syntax.

Specify the start index and the end index, separated by a colon, to return a part of the string.

Example

Get the characters from position 2 to position 5 (not included):

```
b = "Hello, World!"  
print(b[2:5])
```

**Note:** The first character has index 0.

Slice From the Start

By leaving out the start index, the range will start at the first character:

Example

Get the characters from the start to position 5 (not included):

```
b = "Hello, World!"  
print(b[:5])
```

Slice To the End

By leaving out the *end* index, the range will go to the end:

Example

Get the characters from position 2, and all the way to the end:

```
b = "Hello, World!"  
print(b[2:])
```

Negative Indexing

Use negative indexes to start the slice from the end of the string:

Example

Get the characters:

From: "o" in "World!" (position -5)

To, but not included: "d" in "World!" (position -2):

```
b = "Hello, World!"  
print(b[-5:-2])
```

## **Python - Modify Strings**

Python has a set of built-in methods that you can use on strings.

Upper Case

Example

The upper() method returns the string in upper case:

```
a = "Hello, World!"  
print(a.upper())
```

Lower Case

Example

The lower() method returns the string in lower case:

```
a = "Hello, World!"  
print(a.lower())
```

Remove Whitespace

Whitespace is the space before and/or after the actual text, and very often you want to remove this space.

Example

The strip() method removes any whitespace from the beginning or the end:

```
a = " Hello, World! "  
print(a.strip()) # returns "Hello, World!"
```

Replace String

Example

The replace() method replaces a string with another string:

```
a= "Hello,World!"  
print(a.replace("H", "J"))
```

Split String

The split() method returns a list where the text between the specified separator becomes the list items.

Example

The split() method splits the string into substrings if it finds instances of the separator:

```
a= "Hello,World!"  
print(a.split(",")) # returns ['Hello', ' World!']
```

String Methods

Learn more about String Methods with our [String Methods Reference](#)

## **Python - String Concatenation**

String Concatenation

To concatenate, or combine, two strings you can use the + operator.

Example

Merge variable a with variable b into variable c:

```
a= "Hello"  
b= "World"  
c=a+b  
print(c)
```

Example

To add a space between them, add a " ":

```
a = "Hello"  
b = "World"  
c = a + " " + b  
print(c)
```

## **Python - Format - Strings**

String Format

As we learned in the Python Variables chapter, we cannot combine strings and numbers like this:

Example

```
age = 36  
txt = "My name is John, I am " + age  
print(txt)
```

But we can combine strings and numbers by using the format() method!

The format() method takes the passed arguments, formats them, and places them in the string where the placeholders {} are:

Example

Use the format() method to insert numbers into strings:

```
age = 36  
txt = "My name is John, and I am {}"  
print(txt.format(age))
```

The format() method takes unlimited number of arguments, and are placed into the respective placeholders:

Example

```
quantity = 3  
itemno = 567
```

```
price = 49.95
myorder = "I want {} pieces of item {} for {} dollars."
print(myorder.format(quantity, itemno, price))
```

You can use index numbers {0} to be sure the arguments are placed in the correct placeholders:

Example

```
quantity = 3
itemno = 567
price = 49.95
myorder = "I want to pay {2} dollars for {0} pieces of item {1}."
print(myorder.format(quantity, itemno, price))
```

## Python - Escape Characters

Escape Character

To insert characters that are illegal in a string, use an escape character.

An escape character is a backslash \ followed by the character you want to insert.

An example of an illegal character is a double quote inside a string that is surrounded by double quotes:

Example

You will get an error if you use double quotes inside a string that is surrounded by double quotes:

```
txt = "We are the so-called "Vikings" from the north."
```

**Try it Yourself »**

To fix this problem, use the escape character \":

Example

The escape character allows you to use double quotes when you normally would not be allowed:

```
txt = "We are the so-called \"Vikings\" from the north."
```

**Try it Yourself »**

Escape Characters

Other escape characters used in Python:

Code	Result
\'	Single Quote
\\	Backslash
\n	New Line
\r	Carriage Return
\t	Tab
\b	Backspace
\f	Form Feed
\ooo	Octal value
\xhh	Hex value

## Python - String Methods

String Methods

Python has a set of built-in methods that you can use on strings.

**Note:** All string methods return new values. They do not change the original string.

Method	Description
<u>capitalize()</u>	Converts the first character to upper case
<u>casefold()</u>	Converts string into lower case
<u>center()</u>	Returns a centered string
<u>count()</u>	Returns the number of times a specified value occurs in a string
<u>encode()</u>	Returns an encoded version of the string
<u>endswith()</u>	Returns true if the string ends with the specified value
<u>expandtabs()</u>	Sets the tab size of the string
<u>find()</u>	Searches the string for a specified value and returns the position of where it was found
<u>format()</u>	Formats specified values in a string



<code>format_map()</code>	Formats specified values in a string
<code><u>index()</u></code>	Searches the string for a specified value and returns the position of where it was found
<code><u>isalnum()</u></code>	Returns True if all characters in the string are alphanumeric
<code><u>isalpha()</u></code>	Returns True if all characters in the string are in the alphabet
<code><u>isdecimal()</u></code>	Returns True if all characters in the string are decimals
<code><u>isdigit()</u></code>	Returns True if all characters in the string are digits
<code><u>isidentifier()</u></code>	Returns True if the string is an identifier
<code><u>islower()</u></code>	Returns True if all characters in the string are lower case
<code><u>isnumeric()</u></code>	Returns True if all characters in the string are numeric
<code><u>isprintable()</u></code>	Returns True if all characters in the string are printable
<code><u>isspace()</u></code>	Returns True if all characters in the string are whitespaces

<u>istitle()</u>	Returns True if the string follows the rules of a title
<u>isupper()</u>	Returns True if all characters in the string are upper case
<u>join()</u>	Joins the elements of an iterable to the end of the string
<u>ljust()</u>	Returns a left justified version of the string
<u>lower()</u>	Converts a string into lower case
<u>lstrip()</u>	Returns a left trim version of the string
<u>maketrans()</u>	Returns a translation table to be used in translations
<u>partition()</u>	Returns a tuple where the string is parted into three parts
<u>replace()</u>	Returns a string where a specified value is replaced with a specified value
<u>rfind()</u>	Searches the string for a specified value and returns the last position of where it was found
<u>rindex()</u>	Searches the string for a specified value and returns the last position of where it was found

<u>rstrip()</u>	Returns a right justified version of the string
<u>rpartition()</u>	Returns a tuple where the string is parted into three parts
<u>rsplit()</u>	Splits the string at the specified separator, and returns a list
<u>rstrip()</u>	Returns a right trim version of the string
<u>split()</u>	Splits the string at the specified separator, and returns a list
<u>splitlines()</u>	Splits the string at line breaks and returns a list
<u>startswith()</u>	Returns true if the string starts with the specified value
<u>strip()</u>	Returns a trimmed version of the string
<u>swapcase()</u>	Swaps cases, lower case becomes upper case and vice versa
<u>title()</u>	Converts the first character of each word to upper case
<u>translate()</u>	Returns a translated string

<u>upper()</u>	Converts a string into upper case
<u>zfill()</u>	Fills the string with a specified number of 0 values at the beginning

## Python - String Exercises

### Test Yourself With Exercises

Now you have learned a lot about Strings, and how to use them in Python.

Are you ready for a test?

Try to insert the missing part to make the code work as expected:

Exercise:

Use the len method to print the length of the string.

```
x = "Hello World"  
print()
```

Most Values are True

Almost any value is evaluated to True if it has some sort of content.

Any string is True, except empty strings.

Any number is True, except 0.

Any list, tuple, set, and dictionary are True, except empty ones.

Example

The following will return True:

```
bool("abc")  
bool(123)  
bool(["apple", "cherry", "banana"])
```

Some Values are False

In fact, there are not many values that evaluate to False, except empty values, such as (), [], {}, "", the number 0, and the value None. And of course the value False evaluates to False.

#### Example

The following will return False:

```
bool(False)
bool(None)
bool(0)
bool("")
bool()
bool([])
bool({})
```

One more value, or object in this case, evaluates to False, and that is if you have an object that is made from a class with a `__len__` function that returns 0 or False:

#### Example

```
class myclass():
    def __len__(self):
        return 0

myobj = myclass()
print(bool(myobj))
```

### **Functions can Return a Boolean**

You can create functions that returns a Boolean Value:

#### Example

Print the answer of a function:

```
def myFunction() :
    return True

print(myFunction())
```

You can execute code based on the Boolean answer of a function:

#### Example

Print "YES!" if the function returns True, otherwise print "NO!":

```
def myFunction() :  
    return True
```

```
if myFunction():  
    print("YES!")  
else:  
    print("NO!")
```

Python also has many built-in functions that return a boolean value, like the `isinstance()` function, which can be used to determine if an object is of a certain data type:

Example

Check if an object is an integer or not:

```
x = 200  
print(isinstance(x, int))
```

Exercise:

The statement below would print a Boolean value, which one?

```
print(10 > 9)
```

Answer:

```
print(10 > 9)
```

True

```
x = "Hello world"  
print(len(x))
```

## Python Booleans

Booleans represent one of two values: True or False.

## Boolean Values

In programming you often need to know if an expression is True or False.

You can evaluate any expression in Python, and get one of two answers, True or False.

When you compare two values, the expression is evaluated and Python returns the Boolean answer:

Example

```
print(10 > 9)
print(10 == 9)
print(10 < 9)
```

[Try it Yourself »](#)

When you run a condition in an if statement, Python returns True or False:

Example

Print a message based on whether the condition is True or False:

```
a = 200
b = 33

if b > a:
    print("b is greater than a")
else:
    print("b is not greater than a")
```

## Evaluate Values and Variables

The bool() function allows you to evaluate any value, and give you True or False in return,

Example

Evaluate a string and a number:

```
print(bool("Hello"))
print(bool(15))
```

Example

Evaluate two variables:

```
x = "Hello"  
y = 15
```

```
print(bool(x))  
print(bool(y))
```

Most Values are True

Almost any value is evaluated to True if it has some sort of content.

Any string is True, except empty strings.

Any number is True, except 0.

Any list, tuple, set, and dictionary are True, except empty ones.

Example

The following will return True:

```
bool("abc")  
bool(123)  
bool(["apple", "cherry", "banana"])  
Some Values are False
```

In fact, there are not many values that evaluate to False, except empty values, such as (), [], {}, "", the number 0, and the value None. And of course the value False evaluates to False.

Example

The following will return False:

```
bool(False)  
bool(None)  
bool(0)  
bool("")  
bool()  
bool([])  
bool({})
```

[Try it Yourself »](#)



One more value, or object in this case, evaluates to False, and that is if you have an object that is made from a class with a `__len__` function that returns 0 or False:

Example

```
class myclass():
    def __len__(self):
        return 0
```

```
myobj = myclass()
print(bool(myobj))
```

[Try it Yourself »](#)

## Functions can Return a Boolean

You can create functions that returns a Boolean Value:

Example

Print the answer of a function:

```
def myFunction() :
    return True
```

```
print(myFunction())
```

[Try it Yourself »](#)

You can execute code based on the Boolean answer of a function:

Example

Print "YES!" if the function returns True, otherwise print "NO!":

```
def myFunction() :
    return True
```

```
if myFunction():
    print("YES!")
else:
    print("NO!")
```

[Try it Yourself »](#)

Python also has many built-in functions that return a boolean value, like the `isinstance()` function, which can be used to determine if an object is of a certain data type:

### Example

Check if an object is an integer or not:

```
x = 200  
print(isinstance(x, int))
```

[Try it Yourself »](#)

## Test Yourself With Exercises

### Exercise:

The statement below would print a Boolean value, which one?

```
print(10 > 9)
```

answer:

```
print(10 == 9)
```

```
False
```

### Exercise:

The statement below would print a Boolean value, which one?

```
print(10 > 9)
```

Answer:

```
print(10 > 9)
```

```
True
```

The statement below would print a Boolean value, which one?

```
print(10 == 9)
```

Answer:

```
print(10 == 9)
```

```
False
```

The statement below would print a Boolean value, which one?

```
print(10 < 9)
```

Answer:

```
print(10 < 9)
```

```
False
```

The statement below would print a Boolean value, which one?

```
print(bool("abc"))
```

Answer:

```
print(bool("abc"))
```

True

Answer:

The statement below would print a Boolean value, which one?

```
print(bool(0))
```

Answer:

```
print(bool(0))
```

False

## Python Operators

### Python Operators

Operators are used to perform operations on variables and values.

In the example below, we use the + operator to add together two values:

Example

```
print(10 + 5)
```

[Run example »](#)

Python divides the operators in the following groups:

- Arithmetic operators

- Assignment operators
- Comparison operators
- Logical operators
- Identity operators
- Membership operators
- Bitwise operators

## Python Arithmetic Operators

Arithmetic operators are used with numeric values to perform common mathematical operations:

Operator	Name	Example
+	Addition	$x + y$
-	Subtraction	$x - y$
*	Multiplication	$x * y$
/	Division	$x / y$
%	Modulus	$x \% y$
**	Exponentiation	$x ** y$

//	Floor division	x // y
----	----------------	--------

## Python Assignment Operators

Assignment operators are used to assign values to variables:

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3

<code>**=</code>	<code>x **= 3</code>	<code>x = x ** 3</code>
<code>&amp;=</code>	<code>x &amp;= 3</code>	<code>x = x &amp; 3</code>
<code> =</code>	<code>x  = 3</code>	<code>x = x   3</code>
<code>^=</code>	<code>x ^= 3</code>	<code>x = x ^ 3</code>
<code>&gt;&gt;=</code>	<code>x &gt;&gt;= 3</code>	<code>x = x &gt;&gt; 3</code>
<code>&lt;&lt;=</code>	<code>x &lt;&lt;= 3</code>	<code>x = x &lt;&lt; 3</code>

## Python Comparison Operators

Comparison operators are used to compare two values:

Operator	Name	Example
<code>==</code>	Equal	<code>x == y</code>
<code>!=</code>	Not equal	<code>x != y</code>
<code>&gt;</code>	Greater than	<code>x &gt; y</code>

<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

## Python Logical Operators

Logical operators are used to combine conditional statements:

Operator	Description	Example
and	Returns True if both statements are true	x < 5 and x < 10
or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

## Python Identity Operators



Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location:

Operator	Description	Example
is	Returns True if both variables are the same object	x is y
is not	Returns True if both variables are not the same object	x is not y

### Python Membership Operators

Membership operators are used to test if a sequence is presented in an object:

Operator	Description	Example
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

### Python Bitwise Operators

Bitwise operators are used to compare (binary) numbers:

Operator	Name	Description
&	AND	Sets each bit to 1 if both bits are 1
	OR	Sets each bit to 1 if one of two bits is 1
^	XOR	Sets each bit to 1 if only one of two bits is 1
~	NOT	Inverts all the bits
<<	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off

Exercise:

Multiply 10 with 5, and print the result.

`print(10  5)`

---

Answer:

```
print(10 * 5)
```

## Python Lists

```
mylist = ["apple", "banana", "cherry"]
```

## List

Lists are used to store multiple items in a single variable.

Lists are one of 4 built-in data types in Python used to store collections of data, the other 3 are Tuple, Set, and Dictionary, all with different qualities and usage.

Lists are created using square brackets:

### Example

Create a List:

```
thislist = ["apple", "banana", "cherry"]  
print(thislist)
```

[Try it Yourself »](#)

## List Items

List items are ordered, changeable, and allow duplicate values.

List items are indexed, the first item has index [0], the second item has index [1] etc.

### Ordered

When we say that lists are ordered, it means that the items have a defined order, and that order will not change.

If you add new items to a list, the new items will be placed at the end of the list.

**Note:** There are some list methods that will change the order, but in general: the order of the items will not change.

### Changeable

The list is changeable, meaning that we can change, add, and remove items in a list after it has been created.

### Allow Duplicates

Since lists are indexed, lists can have items with the same value:

#### Example

Lists allow duplicate values:

```
thislist = ["apple", "banana", "cherry", "apple", "cherry"]  
print(thislist)
```

To determine how many items a list has, use the len() function:

#### Example

Print the number of items in the list:

```
thislist = ["apple", "banana", "cherry"]  
print(len(thislist))
```

### List Items - Data Types

List items can be of any data type:

#### Example

String, int and boolean data types:

```
list1 = ["apple", "banana", "cherry"]  
list2 = [1, 5, 7, 9, 3]  
list3 = [True, False, False]
```

A list can contain different data types:

#### Example

A list with strings, integers and boolean values:

```
list1 = ["abc", 34, True, 40, "male"]  
type()
```

From Python's perspective, lists are defined as objects with the data type 'list':

```
<class 'list'>
```

Example

What is the data type of a list?

```
mylist = ["apple", "banana", "cherry"]  
print(type(mylist))
```

The list() Constructor

It is also possible to use the list() constructor when creating a new list.

Example

Using the list() constructor to make a List:

```
thislist = list(("apple", "banana", "cherry")) # note the double round-brackets  
print(thislist)
```

Python Collections (Arrays)

There are four collection data types in the Python programming language:

- **List** is a collection which is ordered and changeable. Allows duplicate members.
- **Tuple** is a collection which is ordered and unchangeable. Allows duplicate members.
- **Set** is a collection which is unordered, unchangeable\*, and unindexed. No duplicate members.
- **Dictionary** is a collection which is ordered\*\* and changeable. No duplicate members.

\*Set *items* are unchangeable, but you can remove and/or add items whenever you like.

\*\*As of Python version 3.7, dictionaries are *ordered*. In Python 3.6 and earlier, dictionaries are *unordered*.

When choosing a collection type, it is useful to understand the properties of that type. Choosing the right type for a particular data set could mean retention of meaning, and, it could mean an increase in efficiency or security.

## Python - Access List Items

Access Items

List items are indexed and you can access them by referring to the index number:

#### Example

Print the second item of the list:

```
thislist = ["apple", "banana", "cherry"]  
print(thislist[1])
```

**Note:** The first item has index 0.

#### Negative Indexing

Negative indexing means start from the end

-1 refers to the last item, -2 refers to the second last item etc.

#### Example

Print the last item of the list:

```
thislist = ["apple", "banana", "cherry"]  
print(thislist[-1])
```

#### Range of Indexes

You can specify a range of indexes by specifying where to start and where to end the range.

When specifying a range, the return value will be a new list with the specified items.

#### Example

Return the third, fourth, and fifth item:

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]  
print(thislist[2:5])
```

#### **Try it Yourself »**

**Note:** The search will start at index 2 (included) and end at index 5 (not included).

Remember that the first item has index 0.

By leaving out the start value, the range will start at the first item:

#### Example

This example returns the items from the beginning to, but NOT including, "kiwi":

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]  
print(thislist[:4])
```

By leaving out the end value, the range will go on to the end of the list:

Example

This example returns the items from "cherry" to the end:

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]  
print(thislist[2:])
```

Range of Negative Indexes

Specify negative indexes if you want to start the search from the end of the list:

Example

This example returns the items from "orange" (-4) to, but NOT including "mango" (-1):

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]  
print(thislist[-4:-1])
```

Check if Item Exists

To determine if a specified item is present in a list use the in keyword:

Example

Check if "apple" is present in the list:

```
thislist = ["apple", "banana", "cherry"]  
if "apple" in thislist:  
    print("Yes, 'apple' is in the fruits list")
```

## **Python - Change List Items**

Change Item Value

To change the value of a specific item, refer to the index number:

Example

Change the second item:

```
thislist = ["apple", "banana", "cherry"]  
thislist[1] = "blackcurrant"  
print(thislist)
```

### Change a Range of Item Values

To change the value of items within a specific range, define a list with the new values, and refer to the range of index numbers where you want to insert the new values:

Example

Change the values "banana" and "cherry" with the values "blackcurrant" and "watermelon":

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "mango"]  
thislist[1:3] = ["blackcurrant", "watermelon"]  
print(thislist)
```

If you insert *more* items than you replace, the new items will be inserted where you specified, and the remaining items will move accordingly:

Example

Change the second value by replacing it with *two* new values:

```
thislist = ["apple", "banana", "cherry"]  
thislist[1:2] = ["blackcurrant", "watermelon"]  
print(thislist)
```

**Try it Yourself »**

**Note:** The length of the list will change when the number of items inserted does not match the number of items replaced.

If you insert *less* items than you replace, the new items will be inserted where you specified, and the remaining items will move accordingly:

Example

Change the second and third value by replacing it with *one* value:

```
thislist = ["apple", "banana", "cherry"]  
thislist[1:3] = ["watermelon"]  
print(thislist)
```



## Insert Items

To insert a new list item, without replacing any of the existing values, we can use the insert() method.

The insert() method inserts an item at the specified index:

Example

Insert "watermelon" as the third item:

```
thislist = ["apple", "banana", "cherry"]  
thislist.insert(2, "watermelon")  
print(thislist)
```

Answer:

```
['apple', 'banana', 'watermelon', 'cherry']
```

## Python - Add List Items

Append Items

To add an item to the end of the list, use the append() method:

Example

Using the append() method to append an item:

```
thislist = ["apple", "banana", "cherry"]  
thislist.append("orange")  
print(thislist)
```

Insert Items

To insert a list item at a specified index, use the insert() method.

The insert() method inserts an item at the specified index:

Example

Insert an item as the second position:

```
thislist = ["apple", "banana", "cherry"]  
thislist.insert(1, "orange")  
print(thislist)
```

[Try it Yourself »](#)

**Note:** As a result of the examples above, the lists will now contain 4 items.

### Extend List

To append elements from *another list* to the current list, use the extend() method.

#### Example

Add the elements of tropical to thislist:

```
thislist = ["apple", "banana", "cherry"]  
tropical = ["mango", "pineapple", "papaya"]  
thislist.extend(tropical)  
print(thislist)
```

The elements will be added to the *end* of the list.

### Add Any Iterable

The extend() method does not have to append *lists*, you can add any iterable object (tuples, sets, dictionaries etc.).

#### Example

Add elements of a tuple to a list:

```
thislist = ["apple", "banana", "cherry"]  
thistuple = ("kiwi", "orange")  
thislist.extend(thistuple)  
print(thislist)
```

## Python - Remove List Items

### Remove Specified Item

The remove() method removes the specified item.

#### Example

Remove "banana":

```
thislist = ["apple", "banana", "cherry"]  
thislist.remove("banana")  
print(thislist)
```

### Remove Specified Index

The pop() method removes the specified index.

#### Example

Remove the second item:

```
thislist = ["apple", "banana", "cherry"]  
thislist.pop(1)  
print(thislist)
```

If you do not specify the index, the pop() method removes the last item.

#### Example

Remove the last item:

```
thislist = ["apple", "banana", "cherry"]  
thislist.pop()  
print(thislist)
```

The del keyword also removes the specified index:

#### Example

Remove the first item:

```
thislist = ["apple", "banana", "cherry"]  
del thislist[0]  
print(thislist)
```

The del keyword can also delete the list completely.

#### Example

Delete the entire list:

```
thislist = ["apple", "banana", "cherry"]  
del thislist
```

#### Clear the List

The clear() method empties the list.

The list still remains, but it has no content.

#### Example

Clear the list content:

```
thislist = ["apple", "banana", "cherry"]  
thislist.clear()  
print(thislist)
```

**answer:**



## **Python - Loop Lists**

### Loop Through a List

You can loop through the list items by using a for loop:

#### Example

Print all items in the list, one by one:

```
thislist = ["apple", "banana", "cherry"]  
for x in thislist:  
    print(x)
```

Learn more about for loops in our [Python For Loops](#) Chapter.

### Loop Through the Index Numbers

You can also loop through the list items by referring to their index number.

Use the range() and len() functions to create a suitable iterable.

#### Example

Print all items by referring to their index number:

```
thislist = ["apple", "banana", "cherry"]  
for i in range(len(thislist)):  
    print(thislist[i])
```

The iterable created in the example above is [0, 1, 2].

### Using a While Loop

You can loop through the list items by using a while loop.

Use the len() function to determine the length of the list, then start at 0 and loop your way through the list items by referring to their indexes.

Remember to increase the index by 1 after each iteration.

Example

Print all items, using a while loop to go through all the index numbers

```
thislist = ["apple", "banana", "cherry"]
i = 0
while i < len(thislist):
    print(thislist[i])
    i = i + 1
```

Learn more about while loops in our [Python While Loops](#) Chapter.

Looping Using List Comprehension

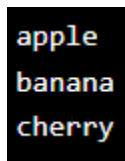
List Comprehension offers the shortest syntax for looping through lists:

Example

A short hand for loop that will print all items in a list:

```
thislist = ["apple", "banana", "cherry"]
[print(x) for x in thislist]
```

Answer:



## Python - List Comprehension

List Comprehension

List comprehension offers a shorter syntax when you want to create a new list based on the values of an existing list.

Example:

Based on a list of fruits, you want a new list, containing only the fruits with the letter "a" in the name.

Without list comprehension you will have to write a for statement with a conditional test inside:

Example

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
newlist = []
```

```
for x in fruits:
    if "a" in x:
        newlist.append(x)
```

```
print(newlist)
```

With list comprehension you can do all that with only one line of code:

Example

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
```

```
newlist = [x for x in fruits if "a" in x]
```

```
print(newlist)
```

The Syntax

```
newlist = [expression for item in iterable if condition == True]
```

The return value is a new list, leaving the old list unchanged.

Condition

The *condition* is like a filter that only accepts the items that evaluate to True.

Example

Only accept items that are not "apple":

```
newlist = [x for x in fruits if x != "apple"]
```

The condition `if x != "apple"` will return True for all elements other than "apple", making the new list contain all fruits except "apple".

The *condition* is optional and can be omitted:

Example

With no if statement:

```
newlist = [x for x in fruits]
```

Iterable

The *iterable* can be any iterable object, like a list, tuple, set etc.

Example

You can use the range() function to create an iterable:

```
newlist = [x for x in range(10)]
```

Same example, but with a condition:

Example

Accept only numbers lower than 5:

```
newlist = [x for x in range(10) if x < 5]
```

Expression

The *expression* is the current item in the iteration, but it is also the outcome, which you can manipulate before it ends up like a list item in the new list:

Example

Set the values in the new list to upper case:

```
newlist = [x.upper() for x in fruits]
```

You can set the outcome to whatever you like:

Example

Set all values in the new list to 'hello':

```
newlist = ['hello' for x in fruits]
```

The *expression* can also contain conditions, not like a filter, but as a way to manipulate the outcome:

Example

Return "orange" instead of "banana":

```
newlist = [x if x != "banana" else "orange" for x in fruits]
```

The *expression* in the example above says:

*"Return the item if it is not banana, if it is banana return orange".*

**Answer:**

```
['apple', 'orange', 'cherry', 'kiwi', 'mango']
```

## Python - Sort Lists

Sort List Alphanumerically

List objects have a sort() method that will sort the list alphanumerically, ascending, by default:

Example

Sort the list alphabetically:

```
thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]
thislist.sort()
print(thislist)
```

Example

Sort the list numerically:

```
thislist = [100, 50, 65, 82, 23]
thislist.sort()
print(thislist)
```

Sort Descending

To sort descending, use the keyword argument reverse = True:

Example

Sort the list descending:

```
thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]
thislist.sort(reverse = True)
print(thislist)
```

Example

Sort the list descending:



```
thislist = [100, 50, 65, 82, 23]
thislist.sort(reverse = True)
print(thislist)
```

### Customize Sort Function

You can also customize your own function by using the keyword argument `key = function`.

The function will return a number that will be used to sort the list (the lowest number first):

#### Example

Sort the list based on how close the number is to 50:

```
def myfunc(n):
    return abs(n - 50)

thislist = [100, 50, 65, 82, 23]
thislist.sort(key = myfunc)
print(thislist)
```

### Case Insensitive Sort

By default the `sort()` method is case sensitive, resulting in all capital letters being sorted before lower case letters:

#### Example

Case sensitive sorting can give an unexpected result:

```
thislist = ["banana", "Orange", "Kiwi", "cherry"]
thislist.sort()
print(thislist)
```

Luckily we can use built-in functions as key functions when sorting a list.

So if you want a case-insensitive sort function, use `str.lower` as a key function:

#### Example

Perform a case-insensitive sort of the list:

```
thislist = ["banana", "Orange", "Kiwi", "cherry"]
thislist.sort(key = str.lower)
print(thislist)
```

## Reverse Order

What if you want to reverse the order of a list, regardless of the alphabet?

The `reverse()` method reverses the current sorting order of the elements.

### Example

Reverse the order of the list items:

```
thislist = ["banana", "Orange", "Kiwi", "cherry"]
thislist.reverse()
print(thislist)
```

**Answer:**

```
['cherry', 'Kiwi', 'Orange', 'banana']
```

## Python - Copy Lists

### Copy a List

You cannot copy a list simply by typing `list2 = list1`, because: `list2` will only be a *reference* to `list1`, and changes made in `list1` will automatically also be made in `list2`.

There are ways to make a copy, one way is to use the built-in List method `copy()`.

### Example

Make a copy of a list with the `copy()` method:

```
thislist = ["apple", "banana", "cherry"]
mylist = thislist.copy()
print(mylist)
```

Another way to make a copy is to use the built-in method `list()`.

### Example

Make a copy of a list with the `list()` method:

```
thislist = ["apple", "banana", "cherry"]
mylist = list(thislist)
print(mylist)
```

**Answer:**

```
['apple', 'banana', 'cherry']
```

## Python - Join Lists

### Join Two Lists

There are several ways to join, or concatenate, two or more lists in Python.

One of the easiest ways are by using the + operator.

#### Example

Join two list:

```
list1 = ["a", "b", "c"]  
list2 = [1, 2, 3]
```

```
list3 = list1 + list2  
print(list3)
```

#### [Try it Yourself »](#)

Another way to join two lists is by appending all the items from list2 into list1, one by one:

#### Example

Append list2 into list1:

```
list1 = ["a", "b", "c"]  
list2 = [1, 2, 3]
```

```
for x in list2:  
    list1.append(x)
```

```
print(list1)
```

#### [Try it Yourself »](#)

Or you can use the extend() method, which purpose is to add elements from one list to another list:

#### Example

Use the extend() method to add list2 at the end of list1:

```
list1 = ["a", "b", "c"]  
list2 = [1, 2, 3]
```

```
list1.extend(list2)  
print(list1)
```

Answer:

```
['a', 'b', 'c', 1, 2, 3]
```

## Python - List Methods

### List Methods

Python has a set of built-in methods that you can use on lists.

Method	Description
<b><u>append()</u></b>	Adds an element at the end of the list
<b><u>clear()</u></b>	Removes all the elements from the list
<b><u>copy()</u></b>	Returns a copy of the list
<b><u>count()</u></b>	Returns the number of elements with the specified value
<b><u>extend()</u></b>	Add the elements of a list (or any iterable), to the end of the current list
<b><u>index()</u></b>	Returns the index of the first element with the specified value
<b><u>insert()</u></b>	Adds an element at the specified position
<b><u>pop()</u></b>	Removes the element at the specified position

<b><u>remove()</u></b>	Removes the item with the specified value
<b><u>reverse()</u></b>	Reverses the order of the list
<b><u>sort()</u></b>	Sorts the list

## Python List Exercises

### Test Yourself With Exercises

Now you have learned a lot about lists, and how to use them in Python.

Are you ready for a test?

Try to insert the missing part to make the code work as expected:

Exercise:

1. Print the second item in the fruits list.

```
fruits = ["apple", "banana", "cherry"]
print()
```

**Answer:**

```
fruits = ["apple", "banana", "cherry"]
print(fruits[1])
```

2. Change the value from "apple" to "kiwi", in the fruits list.

```
fruits = ["apple", "banana", "cherry"]
 = 
```

**Answer:**

```
fruits = ["apple", "banana", "cherry"]  
fruits[0] = "kiwi"
```

Use the append method to add "orange" to the fruits list.

```
fruits = ["apple", "banana", "cherry"]
```

**Answer:**

```
fruits = ["apple", "banana", "cherry"]  
fruits.append("orange")
```

Use the insert method to add "lemon" as the second item in the fruits list.

```
fruits = ["apple", "banana", "cherry"]
```

 "lemon")

**Answer:**

```
fruits = ["apple", "banana", "cherry"]  
fruits.insert(1, "lemon")
```

Use the remove method to remove "banana" from the fruits list.

```
fruits = ["apple", "banana", "cherry"]
```

**Answer:**

```
fruits = ["apple", "banana", "cherry"]  
fruits.remove("banana")
```

Use negative indexing to print the last item in the list.

```
fruits = ["apple", "banana", "cherry"]  
print()
```

**Answer:**

```
fruits = ["apple", "banana", "cherry"]  
print(fruits[-1])
```

Use a range of indexes to print the third, fourth, and fifth item in the list.

```
fruits = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]  
print(fruits[])
```

**Answer:**

```
fruits = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]  
print(fruits[2:5])
```

Use the correct syntax to print the number of items in the list.

```
fruits = ["apple", "banana", "cherry"]  
print()
```

**Answer:**

```
fruits = ["apple", "banana", "cherry"]  
print(len(fruits))
```

## Python Tuples

```
mytuple = ("apple", "banana", "cherry")
```

### Tuple

Tuples are used to store multiple items in a single variable.

Tuple is one of 4 built-in data types in Python used to store collections of data, the other 3 are List, Set, and Dictionary, all with different qualities and usage.

A tuple is a collection which is ordered and **unchangeable**.

Tuples are written with round brackets.

### Example

Create a Tuple:

```
thistuple = ("apple", "banana", "cherry")  
print(thistuple)
```

**Try it Yourself »**

### Tuple Items

Tuple items are ordered, unchangeable, and allow duplicate values.

Tuple items are indexed, the first item has index [0], the second item has index [1] etc.

### Ordered

When we say that tuples are ordered, it means that the items have a defined order, and that order will not change.

### Unchangeable



Tuples are unchangeable, meaning that we cannot change, add or remove items after the tuple has been created.

### Allow Duplicates

Since tuples are indexed, they can have items with the same value:

#### Example

Tuples allow duplicate values:

```
thistuple = ("apple", "banana", "cherry", "apple", "cherry")  
print(thistuple)
```

**Try it Yourself »**

### Tuple Length

To determine how many items a tuple has, use the len() function:

#### Example

Print the number of items in the tuple:

```
thistuple = ("apple", "banana", "cherry")  
print(len(thistuple))
```

#### Create Tuple With One Item

To create a tuple with only one item, you have to add a comma after the item, otherwise Python will not recognize it as a tuple.

#### Example

One item tuple, remember the comma:

```
thistuple = ("apple",)  
print(type(thistuple))
```

#NOT a tuple

```
thistuple = ("apple")  
print(type(thistuple))
```

## Try it Yourself »

### Tuple Items - Data Types

Tuple items can be of any data type:

#### Example

String, int and boolean data types:

```
tuple1 = ("apple", "banana", "cherry")
tuple2 = (1, 5, 7, 9, 3)
tuple3 = (True, False, False)
```

A tuple can contain different data types:

#### Example

A tuple with strings, integers and boolean values:

```
tuple1 = ("abc", 34, True, 40, "male")
type()
```

From Python's perspective, tuples are defined as objects with the data type 'tuple':

```
<class 'tuple'>
```

#### Example

What is the data type of a tuple?

```
mytuple = ("apple", "banana", "cherry")
print(type(mytuple))
```

### The tuple() Constructor

It is also possible to use the tuple() constructor to make a tuple.

#### Example

Using the tuple() method to make a tuple:

```
thistuple = tuple(("apple", "banana", "cherry")) # note the double round-brackets
print(thistuple)
```

## Python Collections (Arrays)

There are four collection data types in the Python programming language:

- **List** is a collection which is ordered and changeable. Allows duplicate members.
- **Tuple** is a collection which is ordered and unchangeable. Allows duplicate members.
- **Set** is a collection which is unordered, unchangeable\*, and unindexed. No duplicate members.
- **Dictionary** is a collection which is ordered\*\* and changeable. No duplicate members.

\*Set *items* are unchangeable, but you can remove and/or add items whenever you like.

\*\*As of Python version 3.7, dictionaries are *ordered*. In Python 3.6 and earlier, dictionaries are *unordered*.

When choosing a collection type, it is useful to understand the properties of that type. Choosing the right type for a particular data set could mean retention of meaning, and, it could mean an increase in efficiency or security.

## Python - Access Tuple Items

### Access Tuple Items

You can access tuple items by referring to the index number, inside square brackets:

Example

Print the second item in the tuple:

```
thistuple = ("apple", "banana", "cherry")  
print(thistuple[1])
```

**Note:** The first item has index 0.

### Negative Indexing

Negative indexing means start from the end.

-1 refers to the last item, -2 refers to the second last item etc.

Example

Print the last item of the tuple:

```
thistuple = ("apple", "banana", "cherry")  
print(thistuple[-1])
```

### Range of Indexes

You can specify a range of indexes by specifying where to start and where to end the range.

When specifying a range, the return value will be a new tuple with the specified items.

#### Example

Return the third, fourth, and fifth item:

```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")  
print(thistuple[2:5])
```

**Note:** The search will start at index 2 (included) and end at index 5 (not included).

Remember that the first item has index 0.

By leaving out the start value, the range will start at the first item:

#### Example

This example returns the items from the beginning to, but NOT included, "kiwi":

```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")  
print(thistuple[:4])
```

By leaving out the end value, the range will go on to the end of the list:

#### Example

This example returns the items from "cherry" and to the end:

```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")  
print(thistuple[2:])
```

### Range of Negative Indexes

Specify negative indexes if you want to start the search from the end of the tuple:

#### Example

This example returns the items from index -4 (included) to index -1 (excluded)

```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")  
print(thistuple[-4:-1])
```

Check if Item Exists

To determine if a specified item is present in a tuple use the in keyword:

Example

Check if "apple" is present in the tuple:

```
thistuple = ("apple", "banana", "cherry")  
if "apple" in thistuple:  
    print("Yes, 'apple' is in the fruits tuple")
```

## Python - Update Tuples

Tuples are unchangeable, meaning that you cannot change, add, or remove items once the tuple is created.

But there are some workarounds.

Change Tuple Values

Once a tuple is created, you cannot change its values. Tuples are **unchangeable**, or **immutable** as it also is called.

But there is a workaround. You can convert the tuple into a list, change the list, and convert the list back into a tuple.

Example

Convert the tuple into a list to be able to change it:

```
x = ("apple", "banana", "cherry")  
y = list(x)  
y[1] = "kiwi"  
x = tuple(y)
```

```
print(x)
```

Add Items

Since tuples are immutable, they do not have a build-in `append()` method, but there are other ways to add items to a tuple.

1. **Convert into a list:** Just like the workaround for *changing* a tuple, you can convert it into a list, add your item(s), and convert it back into a tuple.

Example

Convert the tuple into a list, add "orange", and convert it back into a tuple:

```
thistuple = ("apple", "banana", "cherry")
y = list(thistuple)
y.append("orange")
thistuple = tuple(y)
```

2. **Add tuple to a tuple.** You are allowed to add tuples to tuples, so if you want to add one item, (or many), create a new tuple with the item(s), and add it to the existing tuple:

Example

Create a new tuple with the value "orange", and add that tuple:

```
thistuple = ("apple", "banana", "cherry")
y = ("orange",)
thistuple += y

print(thistuple)
```

**Try it Yourself »**

**Note:** When creating a tuple with only one item, remember to include a comma after the item, otherwise it will not be identified as a tuple.

ADVERTISEMENT

Remove Items

**Note:** You cannot remove items in a tuple.

Tuples are **unchangeable**, so you cannot remove items from it, but you can use the same workaround as we used for changing and adding tuple items:

### Example

Convert the tuple into a list, remove "apple", and convert it back into a tuple:

```
thistuple = ("apple", "banana", "cherry")
y = list(thistuple)
y.remove("apple")
thistuple = tuple(y)
```

Or you can delete the tuple completely:

### Example

The del keyword can delete the tuple completely:

```
thistuple = ("apple", "banana", "cherry")
del thistuple
print(thistuple) #this will raise an error because the tuple no longer exists
```

### Answer:

```
Traceback (most recent call last):
  File "./prog.py", line 3, in <module>
NameError: name 'thistuple' is not defined
```

## Python - Unpack Tuples

### Unpacking a Tuple

When we create a tuple, we normally assign values to it. This is called "packing" a tuple:

### Example

Packing a tuple:

```
fruits = ("apple", "banana", "cherry")
```

But, in Python, we are also allowed to extract the values back into variables. This is called "unpacking":

### Example

Unpacking a tuple:

```
fruits = ("apple", "banana", "cherry")
```

```
(green, yellow, red) = fruits
```

```
print(green)
```

```
print(yellow)
print(red)
```

**Note:** The number of variables must match the number of values in the tuple, if not, you must use an asterisk to collect the remaining values as a list.

Using Asterisk\*

If the number of variables is less than the number of values, you can add an \* to the variable name and the values will be assigned to the variable as a list:

Example

Assign the rest of the values as a list called "red":

```
fruits = ("apple", "banana", "cherry", "strawberry", "raspberry")
```

```
(green, yellow, *red) = fruits
```

```
print(green)
print(yellow)
print(red)
```

**Try it Yourself »**

If the asterisk is added to another variable name than the last, Python will assign values to the variable until the number of values left matches the number of variables left.

Example

Add a list of values the "tropic" variable:

```
fruits = ("apple", "mango", "papaya", "pineapple", "cherry")
```

```
(green, *tropic, red) = fruits
```

```
print(green)
print(tropic)
print(red)
```

## **Python - Loop Tuples**

Loop Through a Tuple

You can loop through the tuple items by using a for loop.



### Example

Iterate through the items and print the values:

```
thistuple = ("apple", "banana", "cherry")
for x in thistuple:
    print(x)
```

[Try it Yourself »](#)

Learn more about for loops in our [Python For Loops](#) Chapter.

### Loop Through the Index Numbers

You can also loop through the tuple items by referring to their index number.

Use the range() and len() functions to create a suitable iterable.

### Example

Print all items by referring to their index number:

```
thistuple = ("apple", "banana", "cherry")
for i in range(len(thistuple)):
    print(thistuple[i])
```

[Try it Yourself »](#)

## ADVERTISEMENT

### Using a While Loop

You can loop through the tuple items by using a while loop.

Use the len() function to determine the length of the tuple, then start at 0 and loop your way through the tuple items by referring to their indexes.

Remember to increase the index by 1 after each iteration.

### Example

Print all items, using a while loop to go through all the index numbers:

```
thistuple = ("apple", "banana", "cherry")
i = 0
while i < len(thistuple):
    print(thistuple[i])
    i = i + 1
```

**Answer:**

```
apple
banana
cherry
```

## Python - Join Tuples

Join Two Tuples

To join two or more tuples you can use the + operator:

Example

Join two tuples:

```
tuple1 = ("a", "b", "c")
tuple2 = (1, 2, 3)
```

```
tuple3 = tuple1 + tuple2
print(tuple3)
```

Multiply Tuples

If you want to multiply the content of a tuple a given number of times, you can use the \* operator:

Example

Multiply the fruits tuple by 2:

```
fruits = ("apple", "banana", "cherry")
mytuple = fruits * 2
```

```
print(mytuple)
```

**Answer:**

```
('apple', 'banana', 'cherry', 'apple', 'banana', 'cherry')
```

## Python - Tuple Methods

### Tuple Methods

Python has two built-in methods that you can use on tuples.

Method	Description
<u>count()</u>	Returns the number of times a specified value occurs in a tuple
<u>index()</u>	Searches the tuple for a specified value and returns the position of where it was found

## Python Sets

```
myset = {"apple", "banana", "cherry"}
```

### Set

Sets are used to store multiple items in a single variable.

Set is one of 4 built-in data types in Python used to store collections of data, the other 3 are List, Tuple, and Dictionary, all with different qualities and usage.

A set is a collection which is *unordered*, *unchangeable*\*, and *unindexed*.

\* **Note:** Set *items* are unchangeable, but you can remove items and add new items.

Sets are written with curly brackets.

### Example

Create a Set:

```
thisset = {"apple", "banana", "cherry"}  
print(thisset)
```

**Note:** Sets are unordered, so you cannot be sure in which order the items will appear.

## Set Items

Set items are unordered, unchangeable, and do not allow duplicate values.

### Unordered

Unordered means that the items in a set do not have a defined order.

Set items can appear in a different order every time you use them, and cannot be referred to by index or key.

### Unchangeable

Set items are unchangeable, meaning that we cannot change the items after the set has been created.

Once a set is created, you cannot change its items, but you can remove items and add new items.

### Duplicates Not Allowed

Sets cannot have two items with the same value.

### Example

Duplicate values will be ignored:

```
thisset = {"apple", "banana", "cherry", "apple"}
```

```
print(thisset)
```

### Get the Length of a Set

To determine how many items a set has, use the len() function.

### Example

Get the number of items in a set:

```
thisset = {"apple", "banana", "cherry"}
```

```
print(len(thisset))
```

## Set Items - Data Types

Set items can be of any data type:

## Example

String, int and boolean data types:

```
set1 = {"apple", "banana", "cherry"}
set2 = {1, 5, 7, 9, 3}
set3 = {True, False, False}
```

A set can contain different data types:

## Example

A set with strings, integers and boolean values:

```
set1 = {"abc", 34, True, 40, "male"}

type()
```

From Python's perspective, sets are defined as objects with the data type 'set':

```
<class 'set'>
```

## Example

What is the data type of a set?

```
myset = {"apple", "banana", "cherry"}
print(type(myset))
```

## The set() Constructor

It is also possible to use the set() constructor to make a set.

## Example

Using the set() constructor to make a set:

```
thisset = set(("apple", "banana", "cherry")) # note the double round-brackets
print(thisset)
```

## Python Collections (Arrays)

There are four collection data types in the Python programming language:

- **List** is a collection which is ordered and changeable. Allows duplicate members.
- **Tuple** is a collection which is ordered and unchangeable. Allows duplicate members.

- **Set** is a collection which is unordered, unchangeable\*, and unindexed. No duplicate members.
- **Dictionary** is a collection which is ordered\*\* and changeable. No duplicate members.

\*Set *items* are unchangeable, but you can remove items and add new items.

\*\*As of Python version 3.7, dictionaries are *ordered*. In Python 3.6 and earlier, dictionaries are *unordered*.

When choosing a collection type, it is useful to understand the properties of that type. Choosing the right type for a particular data set could mean retention of meaning, and, it could mean an increase in efficiency or security.

## Creating Date Objects

To create a date, we can use the `datetime()` class (constructor) of the `datetime` module.

The `datetime()` class requires three parameters to create a date: year, month, day.

### Example

Create a date object:

```
import datetime
```

```
x = datetime.datetime(2020, 5, 17)
```

```
print(x)
```

### Try it Yourself »

The `datetime()` class also takes parameters for time and timezone (hour, minute, second, microsecond, tzzone), but they are optional, and has a default value of 0, (None for timezone).

## The `strftime()` Method

The `datetime` object has a method for formatting date objects into readable strings.

The method is called `strftime()`, and takes one parameter, `format`, to specify the format of the returned string:

### Example

Display the name of the month:

```
import datetime
```

```
x = datetime.datetime(2018, 6, 1)
```

```
print(x.strftime("%B"))
```

A reference of all the legal format codes:

Directive	Description	Example
%a	Weekday, short version	Wed
%A	Weekday, full version	Wednesday
%w	Weekday as a number 0-6, 0 is Sunday	3
%d	Day of month 01-31	31
%b	Month name, short version	Dec
%B	Month name, full version	December
%m	Month as a number 01-12	12
%y	Year, short version, without century	18
%Y	Year, full version	2018
%H	Hour 00-23	17

%I	Hour 00-12	05
%p	AM/PM	PM
%M	Minute 00-59	41
%S	Second 00-59	08
%f	Microsecond 000000-999999	548513
%z	UTC offset	+0100
%Z	Timezone	CST
%j	Day number of year 001-366	365
%U	Week number of year, Sunday as the first day of week, 00-53	52
%W	Week number of year, Monday as the first day of week, 00-53	52
%c	Local version of date and time	Mon Dec 31 17:41:00 2018
%C	Century	20
%x	Local version of date	12/31/18



%X	Local version of time	17:41:00
%%	A % character	%
%G	ISO 8601 year	2018
%u	ISO 8601 weekday (1-7)	1
%V	ISO 8601 weeknumber (01-53)	01

## Python Math

Python has a set of built-in math functions, including an extensive math module, that allows you to perform mathematical tasks on numbers.

### Built-in Math Functions

The min() and max() functions can be used to find the lowest or highest value in an iterable:

#### Example

```
x= min(5, 10, 25)
y= max(5, 10, 25)
```

```
print(x)
print(y)
```

The abs() function returns the absolute (positive) value of the specified number:

#### Example

```
x= abs(-7.25)
```

```
print(x)
```

The pow(x, y) function returns the value of x to the power of y ( $x^y$ ).

#### Example

Return the value of 4 to the power of 3 (same as  $4 * 4 * 4$ ):

```
x=pow(4, 3)
```

```
print(x)
```

The Math Module

Python has also a built-in module called math, which extends the list of mathematical functions.

To use it, you must import the math module:

```
import math
```

When you have imported the math module, you can start using methods and constants of the module.

The math.sqrt() method for example, returns the square root of a number:

Example

```
import math
```

```
x=math.sqrt(64)
```

```
print(x)
```

The math.ceil() method rounds a number upwards to its nearest integer, and the math.floor() method rounds a number downwards to its nearest integer, and returns the result:

Example

```
import math
```

```
x=math.ceil(1.4)
```

```
y=math.floor(1.4)
```

```
print(x) # returns 2
```

```
print(y) # returns 1
```

The math.pi constant, returns the value of PI (3.14...):

Example

```
import math
```

```
x=math.pi
```

```
print(x)
```

## **Python JSON**

JSON is a syntax for storing and exchanging data.

JSON is text, written with JavaScript object notation.

## **JSON in Python**

Python has a built-in package called json, which can be used to work with JSON data.

### **Example**

Import the json module:

```
import json
```

Parse JSON - Convert from JSON to Python

If you have a JSON string, you can parse it by using the json.loads() method.

The result will be a Python dictionary.

Example

Convert from JSON to Python:

```
import json
```

```
#someJSON:
```

```
x= '{"name":"John","age":30,"city":"New York"}'
```

```
# parse x:
```

```
y = json.loads(x)
```

```
# the result is a Python dictionary:  
print(y["age"])
```

[Try it Yourself »](#)

## Convert from Python to JSON

If you have a Python object, you can convert it into a JSON string by using the `json.dumps()` method.

### Example

Convert from Python to JSON:

```
import json  
  
# a Python object (dict):  
x = {  
    "name": "John",  
    "age": 30,  
    "city": "New York"  
}  
  
# convert into JSON:  
y = json.dumps(x)  
  
# the result is a JSON string:  
print(y)
```

[Try it Yourself »](#)

You can convert Python objects of the following types, into JSON strings:

- dict
- list
- tuple
- string
- int
- float
- True
- False
- None

### Example

Convert Python objects into JSON strings, and print the values:

```
import json

print(json.dumps({"name": "John", "age": 30}))
print(json.dumps(["apple", "bananas"]))
print(json.dumps(("apple", "bananas")))
print(json.dumps("hello"))
print(json.dumps(42))
print(json.dumps(31.76))
print(json.dumps(True))
print(json.dumps(False))
print(json.dumps(None))
```

[Try it Yourself »](#)

When you convert from Python to JSON, Python objects are converted into the JSON (JavaScript) equivalent:

Python	JSON
dict	Object
list	Array
tuple	Array
str	String
int	Number

float	Number
True	true
False	false
None	null

### Example

Convert a Python object containing all the legal data types:

```
import json
```

```
x = {  
    "name": "John",  
    "age": 30,  
    "married": True,  
    "divorced": False,  
    "children": ("Ann","Billy"),  
    "pets": None,  
    "cars": [  
        {"model": "BMW 230", "mpg": 27.5},  
        {"model": "Ford Edge", "mpg": 24.1}  
    ]  
}
```

```
print(json.dumps(x))
```

### Format the Result

The example above prints a JSON string, but it is not very easy to read, with no indentations and line breaks.

The `json.dumps()` method has parameters to make it easier to read the result:

#### Example

Use the `indent` parameter to define the numbers of indents:

```
json.dumps(x, indent=4)
```

You can also define the separators, default value is `(" ", ": ")`, which means using a comma and a space to separate each object, and a colon and a space to separate keys from values:

#### Example

Use the `separators` parameter to change the default separator:

```
json.dumps(x, indent=4, separators=(" ", "= "))
```

#### Order the Result

The `json.dumps()` method has parameters to order the keys in the result:

#### Example

Use the `sort_keys` parameter to specify if the result should be sorted or not:

```
json.dumps(x, indent=4, sort_keys=True)
```

### **Python RegEx**

A RegEx, or Regular Expression, is a sequence of characters that forms a search pattern.

RegEx can be used to check if a string contains the specified search pattern.

#### RegEx Module

Python has a built-in package called `re`, which can be used to work with Regular Expressions.

Import the `re` module:

```
import re
```

### **RegEx in Python**

When you have imported the re module, you can start using regular expressions:

### Example

Search the string to see if it starts with "The" and ends with "Spain":

```
import re
```

```
txt = "The rain in Spain"
```

```
x = re.search("^The.*Spain$", txt)
```

[Try it Yourself »](#)

### RegEx Functions

The re module offers a set of functions that allows us to search a string for a match:

Function	Description
<u><a href="#">findall</a></u>	Returns a list containing all matches
<u><a href="#">search</a></u>	Returns a <u><a href="#">Match object</a></u> if there is a match anywhere in the string
<u><a href="#">split</a></u>	Returns a list where the string has been split at each match
<u><a href="#">sub</a></u>	Replaces one or many matches with a string

### Metacharacters

Metacharacters are characters with a special meaning:

Character	Description	Example
-----------	-------------	---------



[ ]	A set of characters	"[a-m]"
\	Signals a special sequence (can also be used to escape special characters)	"\d"
.	Any character (except newline character)	"he..o"
^	Starts with	"^hello"
\$	Ends with	"planet\$"
*	Zero or more occurrences	"he.*o"
+	One or more occurrences	"he.+o"
?	Zero or one occurrences	"he.?o"
{ }	Exactly the specified number of occurrences	"he.{2}o"
	Either or	"falls stays"
()	Capture and group	

## Special Sequences

A special sequence is a `\` followed by one of the characters in the list below, and has a special meaning:

Character	Description	Example
<code>\A</code>	Returns a match if the specified characters are at the beginning of the string	<code>"\AThe"</code>
<code>\b</code>	Returns a match where the specified characters are at the beginning or at the end of a word (the "r" in the beginning is making sure that the string is being treated as a "raw string")	<code>r"\bain"</code> <code>r"ain\b"</code>
<code>\B</code>	Returns a match where the specified characters are present, but NOT at the beginning (or at the end) of a word (the "r" in the beginning is making sure that the string is being treated as a "raw string")	<code>r"\Bain"</code> <code>r"ain\B"</code>
<code>\d</code>	Returns a match where the string contains digits (numbers from 0-9)	<code>"\d"</code>
<code>\D</code>	Returns a match where the string DOES NOT contain digits	<code>"\D"</code>
<code>\s</code>	Returns a match where the string contains a white space character	<code>"\s"</code>

<code>\S</code>	Returns a match where the string DOES NOT contain a white space character	<code>"\S"</code>
<code>\w</code>	Returns a match where the string contains any word characters (characters from a to Z, digits from 0-9, and the underscore _ character)	<code>"\w"</code>
<code>\W</code>	Returns a match where the string DOES NOT contain any word characters	<code>"\W"</code>
<code>\Z</code>	Returns a match if the specified characters are at the end of the string	<code>"Spain\Z"</code>

## Sets

A set is a set of characters inside a pair of square brackets `[]` with a special meaning:

Set	Description
<code>[arn]</code>	Returns a match where one of the specified characters (a, r, or n) is present
<code>[a-n]</code>	Returns a match for any lower case character, alphabetically between a and n
<code>[^arn]</code>	Returns a match for any character EXCEPT a, r, and n

[0123]	Returns a match where any of the specified digits (0, 1, 2, or 3) are present
[0-9]	Returns a match for any digit between 0 and 9
[0-5][0-9]	Returns a match for any two-digit numbers from 00 and 59
[a-zA-Z]	Returns a match for any character alphabetically between a and z, lower case OR upper case
[+]	In sets, +, *, .,  , (), \$, {} has no special meaning, so [+] means: return a match for any + character in the string

### The findall() Function

The findall() function returns a list containing all matches.

#### Example

Print a list of all matches:

```
import re

txt = "The rain in Spain"
x = re.findall("ai", txt)
print(x)
```

The list contains the matches in the order they are found.

If no matches are found, an empty list is returned:

#### Example

Return an empty list if no match was found:

```
import re
```

```
txt = "The rain in Spain"  
x = re.findall("Portugal", txt)  
print(x)
```

### The search() Function

The search() function searches the string for a match, and returns a Match object if there is a match.

If there is more than one match, only the first occurrence of the match will be returned:

#### Example

Search for the first white-space character in the string:

```
import re  
  
txt = "The rain in Spain"  
x = re.search("\s", txt)  
  
print("The first white-space character is located in position:", x.start())
```

If no matches are found, the value None is returned:

#### Example

Make a search that returns no match:

```
import re  
  
txt = "The rain in Spain"  
x = re.search("Portugal", txt)  
print(x)
```

### The split() Function

The split() function returns a list where the string has been split at each match:

#### Example

Split at each white-space character:

```
import re
```

```
txt = "The rain in Spain"  
x = re.split("\s", txt)  
print(x)
```

You can control the number of occurrences by specifying the maxsplit parameter:

Example

Split the string only at the first occurrence:

```
import re
```

```
txt = "The rain in Spain"  
x = re.split("\s", txt, 1)  
print(x)
```

The sub() Function

The sub() function replaces the matches with the text of your choice:

Example

Replace every white-space character with the number 9:

```
import re
```

```
txt = "The rain in Spain"  
x = re.sub("\s", "9", txt)  
print(x)
```

You can control the number of replacements by specifying the count parameter:

Example

Replace the first 2 occurrences:

```
import re
```

```
txt = "The rain in Spain"  
x = re.sub("\s", "9", txt, 2)  
print(x)
```

Match Object

A Match Object is an object containing information about the search and the result.

**Note:** If there is no match, the value None will be returned, instead of the Match Object.

Example

Do a search that will return a Match Object:

```
import re

txt = "The rain in Spain"
x = re.search("ai", txt)
print(x) #this will print an object
```

The Match object has properties and methods used to retrieve information about the search, and the result:

- .span() returns a tuple containing the start-, and end positions of the match.
- .string returns the string passed into the function
- .group() returns the part of the string where there was a match

Example

Print the position (start- and end-position) of the first match occurrence.

The regular expression looks for any words that starts with an upper case "S":

```
import re

txt = "The rain in Spain"
x = re.search(r"\bS\w+", txt)
print(x.span())
```

Example

Print the string passed into the function:

```
import re

txt = "The rain in Spain"
x = re.search(r"\bS\w+", txt)
print(x.string)
```

Example

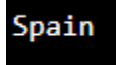
Print the part of the string where there was a match.

The regular expression looks for any words that starts with an upper case "S":

```
import re
```

```
txt = "The rain in Spain"  
x = re.search(r"\bS\w+", txt)  
print(x.group())
```

answer:



## Python PIP

What is PIP?

The pip module is an important tool in the Python ecosystem that allows you to install and manage packages (libraries) for your Python projects.

The pip module provides the following benefits:

**Package Management:** pip makes it easy to install and manage packages (libraries) for your Python projects. You can search for packages, view their information, and install the ones you need with a single command.

**Dependency Resolution:** When you install a package using pip, it automatically resolves and installs any dependencies that the package requires, saving you the time and effort of manually installing them.

**Version Management:** pip allows you to manage multiple versions of the same package in your system, which is useful when different projects require different versions of the same package.

**Package Distribution:** The pip module makes it easy for developers to distribute their packages and for users to install them. You can use pip to install packages



from the Python Package Index (PyPI), the official repository of Python packages, as well as from other sources.

To use pip, you can run the following command in your terminal or command prompt:

Copy code

```
pip install package_name
```

Replace `package_name` with the name of the package you want to install. For example, to install the numpy package, you would run:

Copy code

```
pip install numpy.
```

## List Packages

Use the list command to list all the packages installed on your system:

Example

List installed packages:

```
C:\Users\Your Name\AppData\Local\Programs\Python\Python36-32\Scripts>pip list
```

Result:

Package	Version
-----	
camelcase	0.2
mysql-connector	2.1.6
pip	18.1
pymongo	3.6.1
setuptools	39.0.1

## Python Try Except

The try block lets you test a block of code for errors.

The except block lets you handle the error.

The else block lets you execute code when there is no error.

The finally block lets you execute code, regardless of the result of the try- and except blocks.

### Exception Handling

When an error occurs, or exception as we call it, Python will normally stop and generate an error message.

These exceptions can be handled using the try statement:

#### Example

The try block will generate an exception, because x is not defined:

```
try:
    print(x)
except:
    print("An exception occurred")
```

Since the try block raises an error, the except block will be executed.

Without the try block, the program will crash and raise an error:

#### Example

This statement will raise an error, because x is not defined:

```
print(x)
```

### Many Exceptions

You can define as many exception blocks as you want, e.g. if you want to execute a special block of code for a special kind of error:

#### Example

Print one message if the try block raises a NameError and another for other errors:

```
try:
    print(x)
except NameError:
    print("Variable x is not defined")
except:
    print("Something else went wrong")
```

Else

You can use the else keyword to define a block of code to be executed if no errors were raised:

Example

In this example, the try block does not generate any error:

```
try:
    print("Hello")
except:
    print("Something went wrong")
else:
    print("Nothing went wrong")
```

Finally

The finally block, if specified, will be executed regardless if the try block raises an error or not.

Example

```
try:
    print(x)
except:
    print("Something went wrong")
finally:
    print("The 'try except' is finished")
```

This can be useful to close objects and clean up resources:

Example

Try to open and write to a file that is not writable:

```
try:
    f = open("demofile.txt")
    try:
```

```
f.write("Lorum Ipsum")
except:
    print("Something went wrong when writing to the file")
finally:
    f.close()
except:
    print("Something went wrong when opening the file")
```

The program can continue, without leaving the file object open.

Raise an exception

As a Python developer you can choose to throw an exception if a condition occurs.

To throw (or raise) an exception, use the raise keyword.

Example

Raise an error and stop the program if x is lower than 0:

```
x = -1

if x < 0:
    raise Exception("Sorry, no numbers below zero")
```

The raise keyword is used to raise an exception.

You can define what kind of error to raise, and the text to print to the user.

Example

Raise a TypeError if x is not an integer:

```
x = "hello"

if not type(x) is int:
    raise TypeError("Only integers are allowed")
```

## **Python User Input**

User Input

Python allows for user input.

That means we are able to ask the user for input.

The method is a bit different in Python 3.6 than Python 2.7.

Python 3.6 uses the `input()` method.

Python 2.7 uses the `raw_input()` method.

The following example asks for the username, and when you entered the username, it gets printed on the screen:

Python 3.6

```
username = input("Enter username:")  
print("Username is: " + username)
```

[Run Example »](#)

Python 2.7

```
username = raw_input("Enter username:")  
print("Username is: " + username)
```

## Python String Formatting

To make sure a string will display as expected, we can format the result with the `format()` method.

### String `format()`

The `format()` method allows you to format selected parts of a string.

Sometimes there are parts of a text that you do not control, maybe they come from a database, or user input?

To control such values, add placeholders (curly brackets `{ }`) in the text, and run the values through the `format()` method:

Example

Add a placeholder where you want to display the price:

```
price = 49  
txt = "The price is { } dollars"  
print(txt.format(price))
```

You can add parameters inside the curly brackets to specify how to convert the value:

#### Example

Format the price to be displayed as a number with two decimals:

```
txt = "The price is {:.2f} dollars"
```

Check out all formatting types in our [String format\(\) Reference](#).

### Multiple Values

If you want to use more values, just add more values to the format() method:

```
print(txt.format(price, itemno, count))
```

And add more placeholders:

#### Example

```
quantity = 3
```

```
itemno = 567
```

```
price = 49
```

```
myorder = "I want {} pieces of item number {} for {:.2f} dollars."
```

```
print(myorder.format(quantity, itemno, price))
```

### Index Numbers

You can use index numbers (a number inside the curly brackets {0}) to be sure the values are placed in the correct placeholders:

#### Example

```
quantity = 3
```

```
itemno = 567
```

```
price = 49
```

```
myorder = "I want {0} pieces of item number {1} for {2:.2f} dollars."
```

```
print(myorder.format(quantity, itemno, price))
```

Also, if you want to refer to the same value more than once, use the index number:

#### Example

```
age = 36
```

```
name = "John"
```

```
txt = "His name is {1}. {1} is {0} years old."
```

```
print(txt.format(age, name))
```

## Named Indexes

You can also use named indexes by entering a name inside the curly brackets {carname}, but then you must use names when you pass the parameter values `txt.format(carname = "Ford")`:

### Example

```
myorder = "I have a {carname}, it is a {model}."  
print(myorder.format(carname = "Ford", model = "Mustang"))
```

## Python File Open

File handling is an important part of any web application.

Python has several functions for creating, reading, updating, and deleting files.

### File Handling

The key function for working with files in Python is the `open()` function.

The `open()` function takes two parameters; *filename*, and *mode*.

There are four different methods (modes) for opening a file:

"r" - Read - Default value. Opens a file for reading, error if the file does not exist

"a" - Append - Opens a file for appending, creates the file if it does not exist

"w" - Write - Opens a file for writing, creates the file if it does not exist

"x" - Create - Creates the specified file, returns an error if the file exists

In addition you can specify if the file should be handled as binary or text mode

"t" - Text - Default value. Text mode

"b" - Binary - Binary mode (e.g. images)

### Syntax

To open a file for reading it is enough to specify the name of the file:

```
f = open("demofile.txt")
```

The code above is the same as:

```
f = open("demofile.txt", "rt")
```

Because "r" for read, and "t" for text are the default values, you do not need to specify them.

### Open a File on the Server

Assume we have the following file, located in the same folder as Python:

demofile.txt

Hello! Welcome to demofile.txt  
This file is for testing purposes.  
Good Luck!

To open the file, use the built-in open() function.

The open() function returns a file object, which has a read() method for reading the content of the file:

#### Example

```
f = open("demofile.txt", "r")  
print(f.read())
```

If the file is located in a different location, you will have to specify the file path, like this:

#### Example

Open a file on a different location:

```
f = open("D:\\myfiles\\welcome.txt", "r")  
print(f.read())
```

### Read Only Parts of the File

By default the read() method returns the whole text, but you can also specify how many characters you want to return:

#### Example

Return the 5 first characters of the file:



```
f = open("demofile.txt", "r")  
print(f.read(5))
```

## Read Lines

You can return one line by using the `readline()` method:

### Example

Read one line of the file:

```
f = open("demofile.txt", "r")  
print(f.readline())
```

By calling `readline()` two times, you can read the two first lines:

### Example

Read two lines of the file:

```
f = open("demofile.txt", "r")  
print(f.readline())  
print(f.readline())
```

By looping through the lines of the file, you can read the whole file, line by line:

### Example

Loop through the file line by line:

```
f = open("demofile.txt", "r")  
for x in f:  
    print(x)
```

## Close Files

It is a good practice to always close the file when you are done with it.

### Example

Close the file when you are finish with it:

```
f = open("demofile.txt", "r")  
print(f.readline())  
f.close()
```

## Python File Write

### Write to an Existing File

To write to an existing file, you must add a parameter to the open() function:

"a" - Append - will append to the end of the file

"w" - Write - will overwrite any existing content

#### Example

Open the file "demofile2.txt" and append content to the file:

```
f = open("demofile2.txt", "a")
f.write("Now the file has more content!")
f.close()
```

#open and read the file after the appending:

```
f = open("demofile2.txt", "r")
print(f.read())
```

[Run Example »](#)

#### Example

Open the file "demofile3.txt" and overwrite the content:

```
f = open("demofile3.txt", "w")
f.write("Woops! I have deleted the content!")
f.close()
```

#open and read the file after the appending:

```
f = open("demofile3.txt", "r")
print(f.read())
```

[Run Example »](#)

**Note:** the "w" method will overwrite the entire file.

### Create a New File

To create a new file in Python, use the open() method, with one of the following parameters:

"x" - Create - will create a file, returns an error if the file exist

"a" - Append - will create a file if the specified file does not exist

"w" - Write - will create a file if the specified file does not exist

Example

Create a file called "myfile.txt":

```
f = open("myfile.txt", "x")
```

Result: a new empty file is created!

Example

Create a new file if it does not exist:

```
f = open("myfile.txt", "w")
```

## **Python Delete File**

Delete a File

To delete a file, you must import the OS module, and run its `os.remove()` function:

Example

Remove the file "demofile.txt":

```
import os
os.remove("demofile.txt")
```

## **Check if File exist:**

To avoid getting an error, you might want to check if the file exists before you try to delete it:

Example

Check if file exists, *then* delete it:

```
import os
if os.path.exists("demofile.txt"):
    os.remove("demofile.txt")
else:
    print("The file does not exist")
```

## **Delete Folder**

To delete an entire folder, use the `os.rmdir()` method:

Example

Remove the folder "myfolder":

```
import os  
os.rmdir("myfolder")
```

## Machine Learning

Machine Learning is making the computer learn from studying data and statistics.

Machine Learning is a step into the direction of artificial intelligence (AI).

Machine Learning is a program that analyses data and learns to predict the outcome.

Where To Start?

In this tutorial we will go back to mathematics and study statistics, and how to calculate important numbers based on data sets.

We will also learn how to use various Python modules to get the answers we need.

And we will learn how to make functions that are able to predict the outcome based on what we have learned.

## Data Set

In the mind of a computer, a data set is any collection of data. It can be anything from an array to a complete database.

Example of an array:

[99,86,87,88,111,86,103,87,94,78,77,85,86]

Example of a database:

Carname	Color	Age	Speed	AutoPass
BMW	red	5	99	Y
Volvo	black	7	86	Y

VW	gray	8	87	N
VW	white	7	88	Y
Ford	white	2	111	Y
VW	white	17	86	Y
Tesla	red	2	103	Y
BMW	black	9	87	Y
Volvo	gray	4	94	N
Ford	white	11	78	N
Toyota	gray	12	77	N
VW	white	9	85	N
Toyota	blue	6	86	Y

By looking at the array, we can guess that the average value is probably around 80 or 90, and we are also able to determine the highest value and the lowest value, but what else can we do?

And by looking at the database we can see that the most popular color is white, and the oldest car is 17 years, but what if we could predict if a car had an AutoPass, just by looking at the other values?

That is what Machine Learning is for! Analyzing data and predicting the outcome!

In Machine Learning it is common to work with very large data sets. In this tutorial we will try to make it as easy as possible to understand the different concepts of machine learning, and we will work with small easy-to-understand data sets.

## Data Types

To analyze data, it is important to know what type of data we are dealing with.

We can split the data types into three main categories:

- **Numerical**
- **Categorical**
- **Ordinal**

**Numerical** data are numbers, and can be split into two numerical categories:

- **Discrete Data**
  - numbers that are limited to integers. Example: The number of cars passing by.
- **Continuous Data**
  - numbers that are of infinite value. Example: The price of an item, or the size of an item

**Categorical** data are values that cannot be measured up against each other. Example: a color value, or any yes/no values.

**Ordinal** data are like categorical data, but can be measured up against each other. Example: school grades where A is better than B and so on.

By knowing the data type of your data source, you will be able to know what technique to use when analyzing them.

You will learn more about statistics and analyzing data in the next chapters.

## Machine Learning - Mean Median Mode

Mean, Median, and Mode

What can we learn from looking at a group of numbers?

In Machine Learning (and in mathematics) there are often three values that interests us:

- **Mean** - The average value
- **Median** - The mid point value
- **Mode** - The most common value

Example: We have registered the speed of 13 cars:

```
speed = [99,86,87,88,111,86,103,87,94,78,77,85,86]
```

What is the average, the middle, or the most common speed value?

## Mean

The mean value is the average value.

To calculate the mean, find the sum of all values, and divide the sum by the number of values:

$$(99+86+87+88+111+86+103+87+94+78+77+85+86) / 13 = 89.77$$

The NumPy module has a method for this. Learn about the NumPy module in our [NumPy Tutorial](#).

Example

Use the NumPy mean() method to find the average speed:

```
import numpy
```

```
speed = [99,86,87,88,111,86,103,87,94,78,77,85,86]
```

```
x = numpy.mean(speed)
```

```
print(x)
```

## Median

The median value is the value in the middle, after you have sorted all the values:

77, 78, 85, 86, 86, 87, 87, 88, 94, 99, 103, 111

It is important that the numbers are sorted before you can find the median.

The NumPy module has a method for this:

Example

Use the NumPy median() method to find the middle value:

```
import numpy
```

```
speed = [99,86,87,88,111,86,103,87,94,78,77,85,86]
```

```
x = numpy.median(speed)
```

```
print(x)
```

If there are two numbers in the middle, divide the sum of those numbers by two.

77, 78, 85, 86, 86, 87, 87, 94, 98, 99, 103

$$(86 + 87) / 2 = 86.5$$

Example

Using the NumPy module:

```
import numpy
```

```
speed = [99,86,87,88,86,103,87,94,78,77,85,86]
```

```
x = numpy.median(speed)
```

```
print(x)
```

## Mode

The Mode value is the value that appears the most number of times:

99, 86, 87, 88, 111, 86, 103, 87, 94, 78, 77, 85, 86 = 86

The SciPy module has a method for this. Learn about the SciPy module in our [SciPy Tutorial](#).

Example

Use the SciPy mode() method to find the number that appears the most:

```
from scipy import stats
```

```
speed = [99,86,87,88,111,86,103,87,94,78,77,85,86]
```

```
x = stats.mode(speed)
```

```
print(x)
```

## Chapter Summary



The Mean, Median, and Mode are techniques that are often used in Machine Learning, so it is important to understand the concept behind them.

### **LU3: Machine Learning - Standard Deviation**

What is Standard Deviation?

Standard deviation is a number that describes how spread out the values are.

A low standard deviation means that most of the numbers are close to the mean (average) value.

A high standard deviation means that the values are spread out over a wider range.

Example: This time we have registered the speed of 7 cars:

`speed = [86,87,88,86,87,85,86]`

The standard deviation is:

0.9

Meaning that most of the values are within the range of 0.9 from the mean value, which is 86.4.

Let us do the same with a selection of numbers with a wider range:

`speed = [32,111,138,28,59,77,97]`

The standard deviation is:

37.85

Meaning that most of the values are within the range of 37.85 from the mean value, which is 77.4.

As you can see, a higher standard deviation indicates that the values are spread out over a wider range.

The NumPy module has a method to calculate the standard deviation:

Example

Use the NumPy `std()` method to find the standard deviation:

```
import numpy

speed = [86,87,88,86,87,85,86]

x = numpy.std(speed)

print(x)
```

Example

```
import numpy

speed = [32,111,138,28,59,77,97]

x = numpy.std(speed)

print(x)
```

[Try it Yourself »](#)

## ADVERTISEMENT

### Variance

Variance is another number that indicates how spread out the values are.

In fact, if you take the square root of the variance, you get the standard deviation!

Or the other way around, if you multiply the standard deviation by itself, you get the variance!

To calculate the variance you have to do as follows:

1. Find the mean:

$$(32+111+138+28+59+77+97) / 7 = 77.4$$

2. For each value: find the difference from the mean:

$$32 - 77.4 = -45.4$$

$$111 - 77.4 = 33.6$$

$$138 - 77.4 = 60.6$$

$$28 - 77.4 = -49.4$$

$$59 - 77.4 = -18.4$$

$$77 - 77.4 = -0.4$$

$$97 - 77.4 = 19.6$$

3. For each difference: find the square value:

$$(-45.4)^2 = 2061.16$$

$$(33.6)^2 = 1128.96$$

$$(60.6)^2 = 3672.36$$

$$(-49.4)^2 = 2440.36$$

$$(-18.4)^2 = 338.56$$

$$(-0.4)^2 = 0.16$$

$$(19.6)^2 = 384.16$$

4. The variance is the average number of these squared differences:

$$(2061.16 + 1128.96 + 3672.36 + 2440.36 + 338.56 + 0.16 + 384.16) / 7 = 1432.2$$

Luckily, NumPy has a method to calculate the variance:

Example

Use the NumPy var() method to find the variance:

```
import numpy
```

```
speed = [32,111,138,28,59,77,97]
```

```
x = numpy.var(speed)
```

```
print(x)
```

Standard Deviation

As we have learned, the formula to find the standard deviation is the square root of the variance:

$$\sqrt{1432.25} = 37.85$$

Or, as in the example from before, use the NumPy to calculate the standard deviation:

Example

Use the NumPy std() method to find the standard deviation:

```
import numpy

speed = [32,111,138,28,59,77,97]

x = numpy.std(speed)

print(x)
```

## Symbols

Standard Deviation is often represented by the symbol Sigma:  $\sigma$

Variance is often represented by the symbol Sigma Squared:  $\sigma^2$

## Chapter Summary

The Standard Deviation and Variance are terms that are often used in Machine Learning, so it is important to understand how to get them, and the concept behind them.

## LU 3.1: Machine Learning - Percentiles

### What are Percentiles?

Percentiles are used in statistics to give you a number that describes the value that a given percent of the values are lower than.

Example: Let's say we have an array of the ages of all the people that live in a street.

```
ages = [5,31,43,48,50,41,7,11,15,39,80,82,32,2,8,6,25,36,27,61,31]
```

What is the 75. percentile? The answer is 43, meaning that 75% of the people are 43 or younger.

The NumPy module has a method for finding the specified percentile:

### Example

Use the NumPy percentile() method to find the percentiles:

```
import numpy

ages = [5,31,43,48,50,41,7,11,15,39,80,82,32,2,8,6,25,36,27,61,31]
```

```
x = numpy.percentile(ages, 75)
```

```
print(x)
```

Example

What is the age that 90% of the people are younger than?

```
import numpy
```

```
ages = [5,31,43,48,50,41,7,11,15,39,80,82,32,2,8,6,25,36,27,61,31]
```

```
x = numpy.percentile(ages, 90)
```

```
print(x)
```

## **Machine Learning - Data Distribution**

Data Distribution

Earlier in this tutorial we have worked with very small amounts of data in our examples, just to understand the different concepts.

In the real world, the data sets are much bigger, but it can be difficult to gather real world data, at least at an early stage of a project.

### **How Can we Get Big Data Sets?**

To create big data sets for testing, we use the Python module NumPy, which comes with a number of methods to create random data sets, of any size.

Example

Create an array containing 250 random floats between 0 and 5:

```
import numpy
```

```
x = numpy.random.uniform(0.0, 5.0, 250)
```

```
print(x)
```

Histogram

To visualize the data set we can draw a histogram with the data we collected.

We will use the Python module Matplotlib to draw a histogram.

Learn about the Matplotlib module in our [Matplotlib Tutorial](#).

### Example

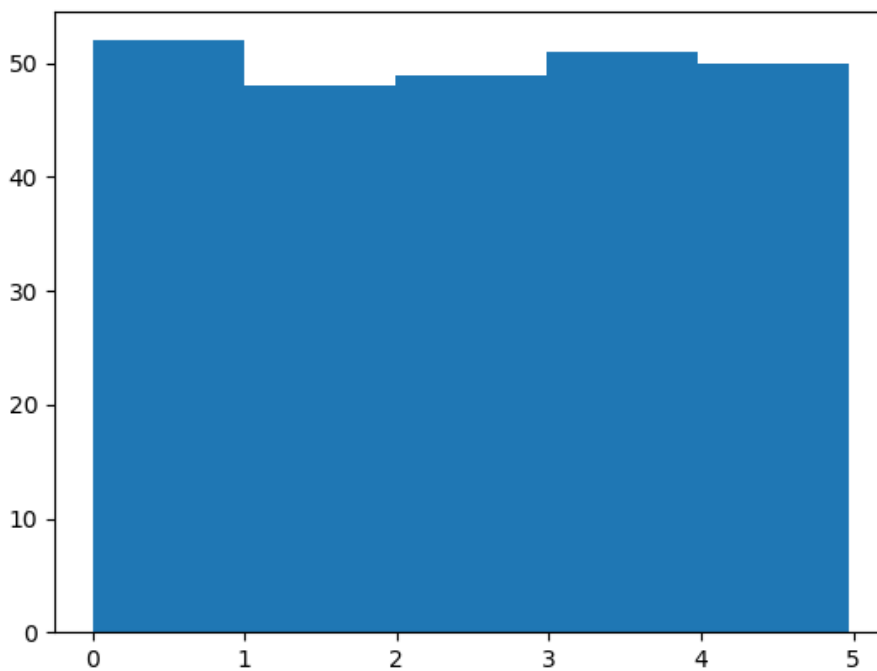
Draw a histogram:

```
import numpy
import matplotlib.pyplot as plt

x = numpy.random.uniform(0.0, 5.0, 250)
```

```
plt.hist(x, 5)
plt.show()
```

Result:



[Run example »](#)

### Histogram Explained

We use the array from the example above to draw a histogram with 5 bars.

The first bar represents how many values in the array are between 0 and 1.

The second bar represents how many values are between 1 and 2.

Etc.

Which gives us this result:

- 52 values are between 0 and 1
- 48 values are between 1 and 2
- 49 values are between 2 and 3
- 51 values are between 3 and 4
- 50 values are between 4 and 5

**Note:** The array values are random numbers and will not show the exact same result on your computer.

## Big Data Distributions

An array containing 250 values is not considered very big, but now you know how to create a random set of values, and by changing the parameters, you can create the data set as big as you want.

### Example

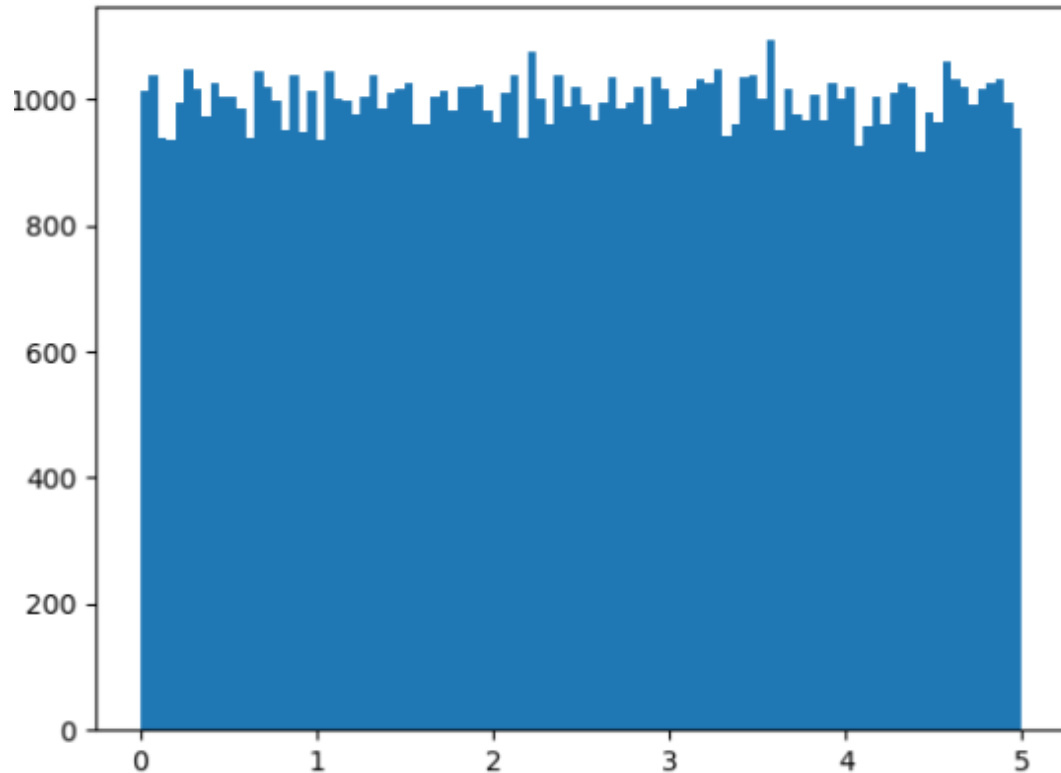
Create an array with 100000 random numbers, and display them using a histogram with 100 bars:

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

```
x = numpy.random.uniform(0.0, 5.0, 100000)
```

```
plt.hist(x, 100)
plt.show()
```

answer:



### Normal Data Distribution

In the previous chapter we learned how to create a completely random array, of a given size, and between two given values.

In this chapter we will learn how to create an array where the values are concentrated around a given value.

In probability theory this kind of data distribution is known as the *normal data distribution*, or the *Gaussian data distribution*, after the mathematician Carl Friedrich Gauss who came up with the formula of this data distribution.

### Example

A typical normal data distribution:

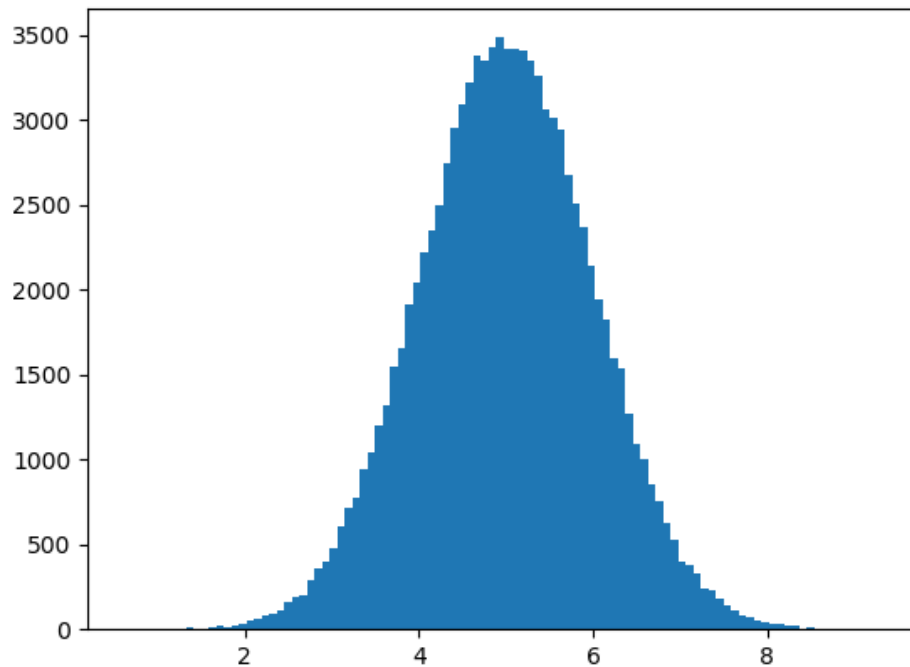
```
import numpy
import matplotlib.pyplot as plt

x = numpy.random.normal(5.0, 1.0, 100000)

plt.hist(x, 100)
plt.show()
```



Result:



[Run example »](#)

**Note:** A normal distribution graph is also known as the *bell curve* because of its characteristic shape of a bell.

### Histogram Explained

We use the array from the `numpy.random.normal()` method, with 100000 values, to draw a histogram with 100 bars.

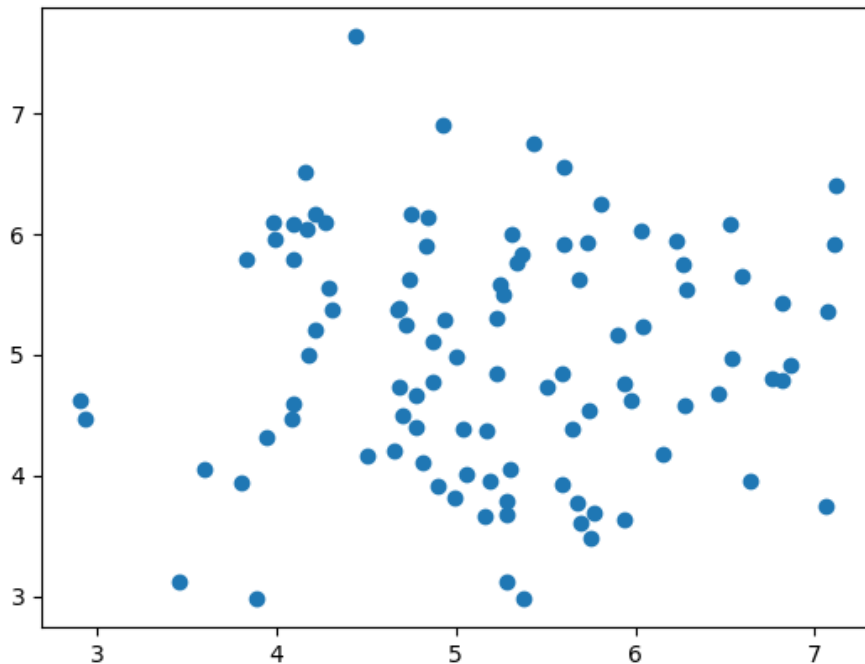
We specify that the mean value is 5.0, and the standard deviation is 1.0.

Meaning that the values should be concentrated around 5.0, and rarely further away than 1.0 from the mean.

And as you can see from the histogram, most values are between 4.0 and 6.0, with a top at approximately 5.0.

### Scatter Plot

A scatter plot is a diagram where each value in the data set is represented by a dot.



The Matplotlib module has a method for drawing scatter plots, it needs two arrays of the same length, one for the values of the x-axis, and one for the values of the y-axis:

```
x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
```

```
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]
```

The x array represents the age of each car.

The y array represents the speed of each car.

Example

Use the scatter() method to draw a scatter plot diagram:

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

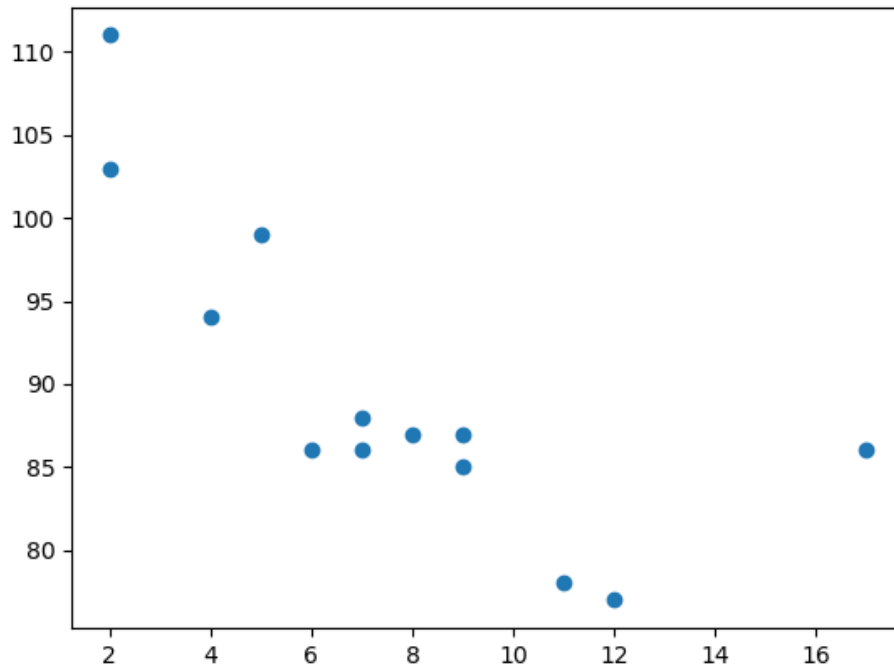
```
x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
```

```
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]
```

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

```
plt.show()
```

Result:



[Run example »](#)

## Scatter Plot Explained

The x-axis represents ages, and the y-axis represents speeds.

What we can read from the diagram is that the two fastest cars were both 2 years old, and the slowest car was 12 years old.

**Note:** It seems that the newer the car, the faster it drives, but that could be a coincidence, after all we only registered 13 cars.

## Random Data Distributions

In Machine Learning the data sets can contain thousands-, or even millions, of values.

You might not have real world data when you are testing an algorithm, you might have to use randomly generated values.

As we have learned in the previous chapter, the NumPy module can help us with that!

Let us create two arrays that are both filled with 1000 random numbers from a normal data distribution.

The first array will have the mean set to 5.0 with a standard deviation of 1.0.

The second array will have the mean set to 10.0 with a standard deviation of 2.0:

Example

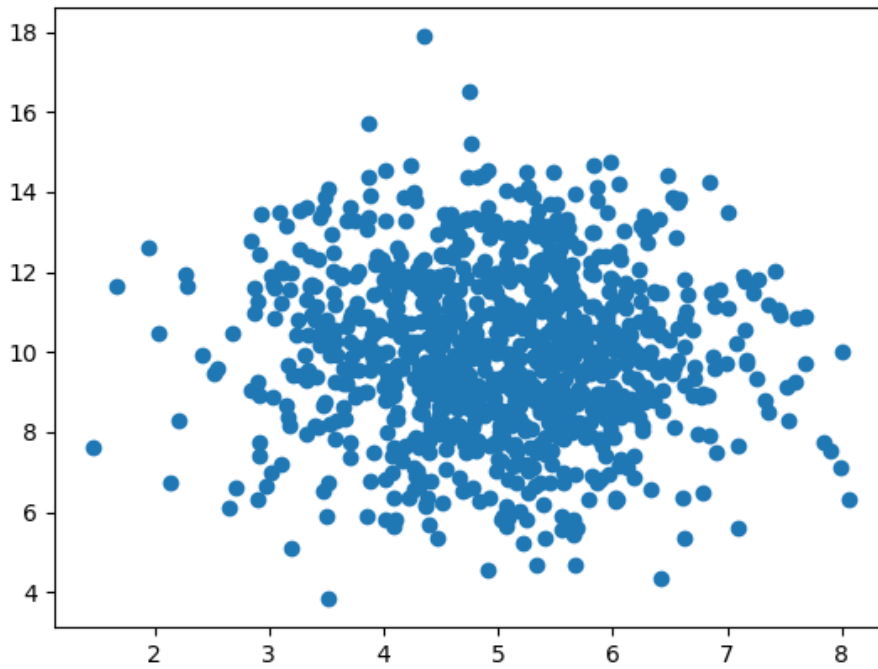
A scatter plot with 1000 dots:

```
import numpy
import matplotlib.pyplot as plt

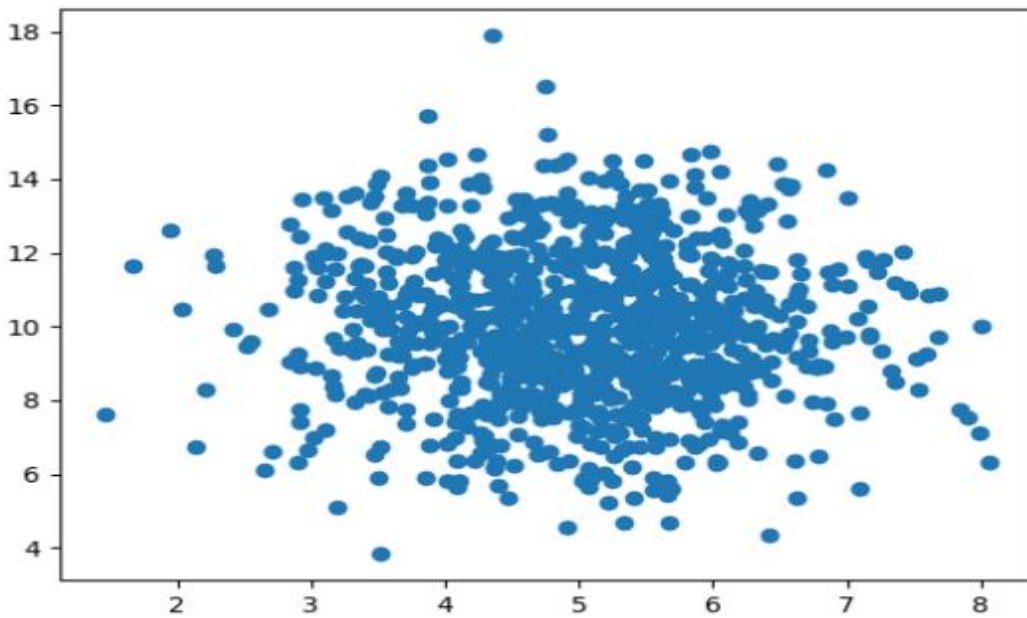
x = numpy.random.normal(5.0, 1.0, 1000)
y = numpy.random.normal(10.0, 2.0, 1000)

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

Result:



Answer:



Scatter Plot Explained

We can see that the dots are concentrated around the value 5 on the x-axis, and 10 on the y-axis.

We can also see that the spread is wider on the y-axis than on the x-axis.

## Regression

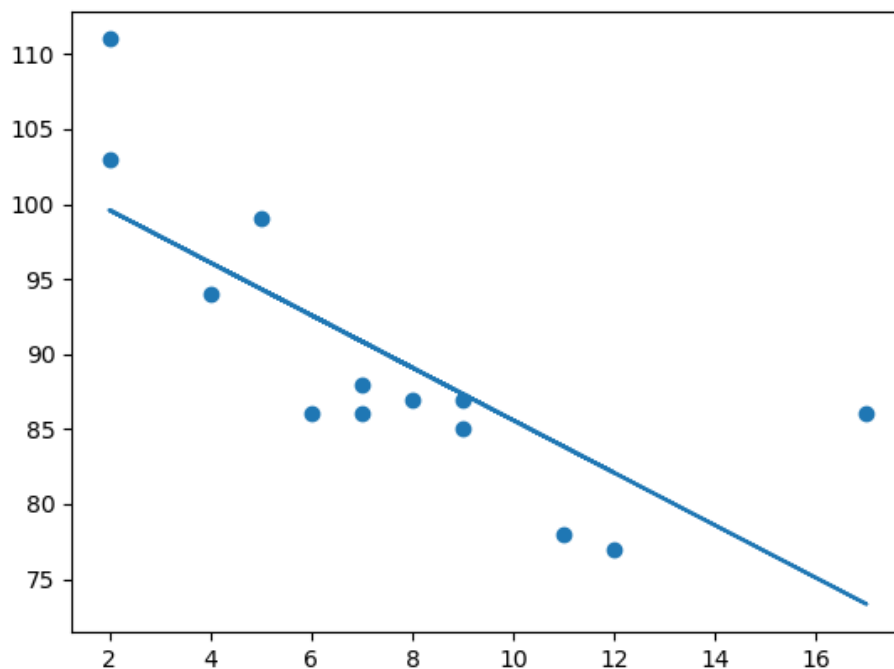
The term regression is used when you try to find the relationship between variables.

In Machine Learning, and in statistical modeling, that relationship is used to predict the outcome of future events.

## Linear Regression

Linear regression uses the relationship between the data-points to draw a straight line through all them.

This line can be used to predict future values.



In Machine Learning, predicting the future is very important.

## How Does it Work?

Python has methods for finding a relationship between data-points and to draw a line of linear regression. We will show you how to use these methods instead of going through the mathematic formula.

In the example below, the x-axis represents age, and the y-axis represents speed. We have registered the age and speed of 13 cars as they were passing a tollbooth. Let us see if the data we collected could be used in a linear regression:

Example

Start by drawing a scatter plot:

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

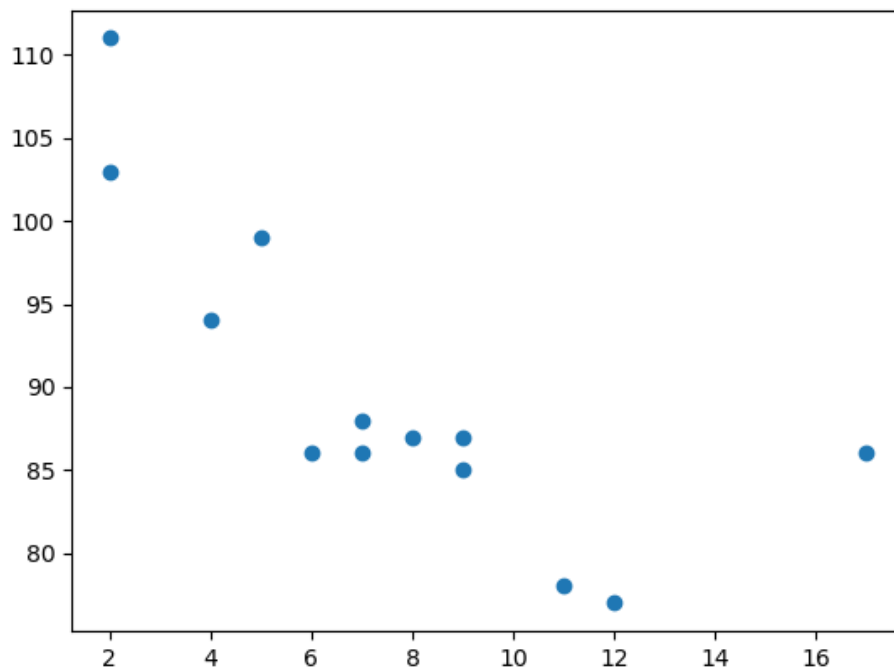
```
x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
```

```
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]
```

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

```
plt.show()
```

Result:



[Run example »](#)

Example

Import scipy and draw the line of Linear Regression:

```

import matplotlib.pyplot as plt
from scipy import stats

x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]

slope, intercept, r, p, std_err = stats.linregress(x, y)

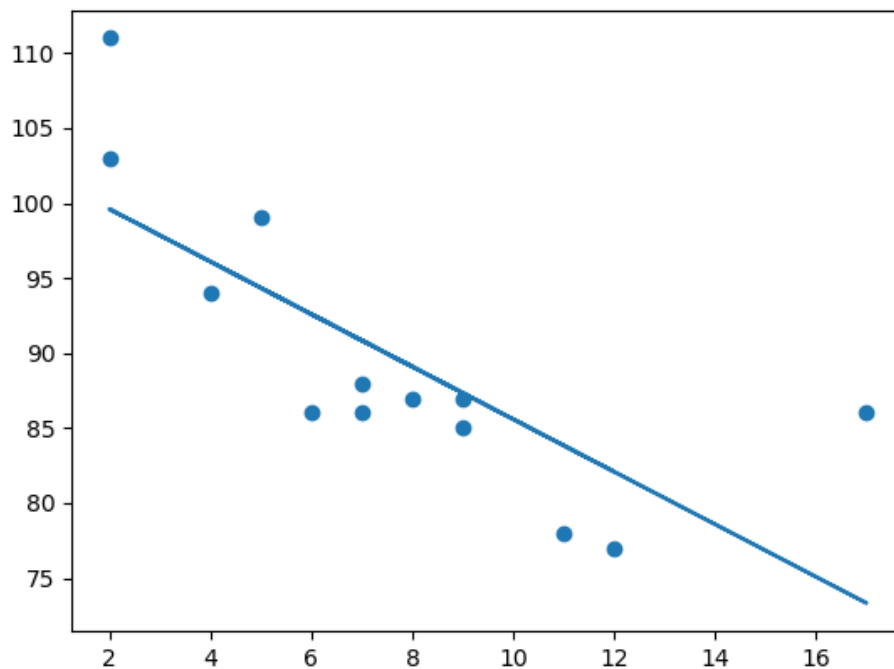
def myfunc(x):
    return slope * x + intercept

mymodel = list(map(myfunc, x))

plt.scatter(x, y)
plt.plot(x, mymodel)
plt.show()

```

Result:



[Run example »](#)

Example Explained



Import the modules you need.

You can learn about the Matplotlib module in our [Matplotlib Tutorial](#).

You can learn about the SciPy module in our [SciPy Tutorial](#).

```
import matplotlib.pyplot as plt
from scipy import stats
```

Create the arrays that represent the values of the x and y axis:

```
x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]
```

Execute a method that returns some important key values of Linear Regression:

```
slope, intercept, r, p, std_err = stats.linregress(x, y)
```

Create a function that uses the slope and intercept values to return a new value. This new value represents where on the y-axis the corresponding x value will be placed:

```
def myfunc(x):
    return slope * x + intercept
```

Run each value of the x array through the function. This will result in a new array with new values for the y-axis:

```
mymodel = list(map(myfunc, x))
```

Draw the original scatter plot:

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

Draw the line of linear regression:

```
plt.plot(x, mymodel)
```

Display the diagram:

```
plt.show()
```

## R for Relationship

It is important to know how the relationship between the values of the x-axis and the values of the y-axis is, if there are no relationship the linear regression can not be used to predict anything.

This relationship - the coefficient of correlation - is called r.

The r value ranges from -1 to 1, where 0 means no relationship, and 1 (and -1) means 100% related.

Python and the Scipy module will compute this value for you, all you have to do is feed it with the x and y values.

### Example

How well does my data fit in a linear regression?

```
from scipy import stats
```

```
x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
```

```
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]
```

```
slope, intercept, r, p, std_err = stats.linregress(x, y)
```

```
print(r)
```

### Try it Yourself »

**Note:** The result -0.76 shows that there is a relationship, not perfect, but it indicates that we could use linear regression in future predictions.

### Predict Future Values

Now we can use the information we have gathered to predict future values.

Example: Let us try to predict the speed of a 10 years old car.

To do so, we need the same myfunc() function from the example above:

```
def myfunc(x):  
    return slope * x + intercept
```

### Example

Predict the speed of a 10 years old car:

```
from scipy import stats
```

```
x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
```

```
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]
```

```
slope, intercept, r, p, std_err = stats.linregress(x, y)
```

```
def myfunc(x):
```

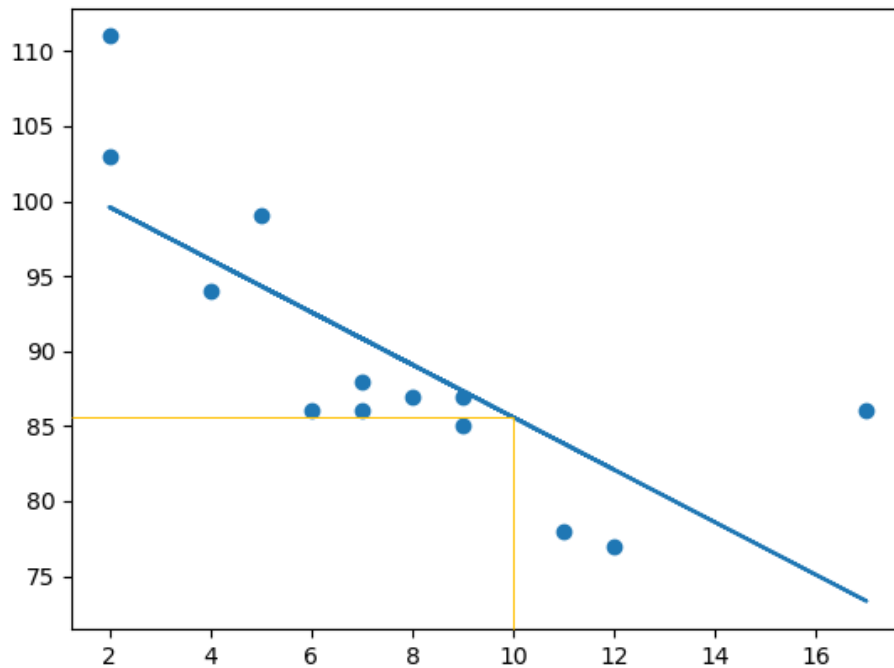
```
    return slope * x + intercept
```

```
speed = myfunc(10)
```

```
print(speed)
```

[Run example »](#)

The example predicted a speed at 85.6, which we also could read from the diagram:



## Bad Fit?

Let us create an example where linear regression would not be the best method to predict future values.

### Example

These values for the x- and y-axis should result in a very bad fit for linear regression:

```
import matplotlib.pyplot as plt
from scipy import stats
```

```
x = [89,43,36,36,95,10,66,34,38,20,26,29,48,64,6,5,36,66,72,40]
y = [21,46,3,35,67,95,53,72,58,10,26,34,90,33,38,20,56,2,47,15]
```

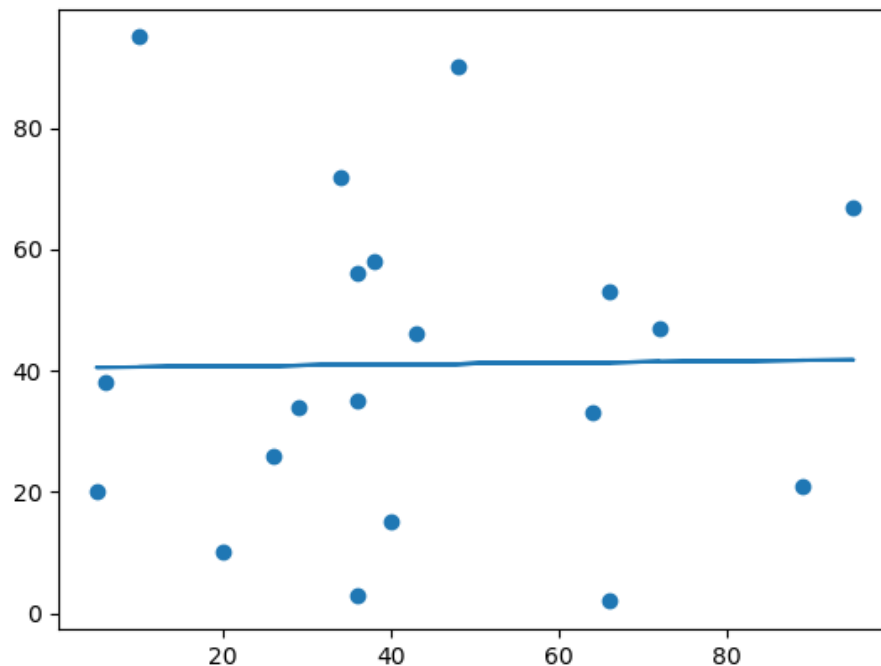
```
slope, intercept, r, p, std_err = stats.linregress(x, y)
```

```
def myfunc(x):
    return slope * x + intercept
```

```
mymodel = list(map(myfunc, x))
```

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

Result:



[Run example »](#)

And the r for relationship?

Example

You should get a very low r value.

```
import numpy
from scipy import stats

x = [89,43,36,36,95,10,66,34,38,20,26,29,48,64,6,5,36,66,72,40]
y = [21,46,3,35,67,95,53,72,58,10,26,34,90,33,38,20,56,2,47,15]
```

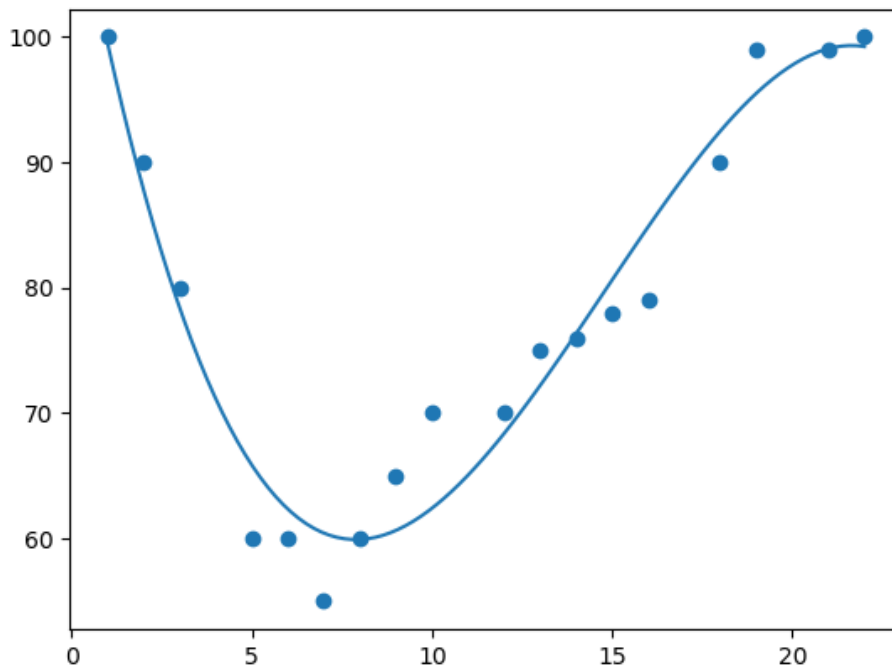
```
slope, intercept, r, p, std_err = stats.linregress(x, y)
```

```
print(r)
```

## Polynomial Regression

If your data points clearly will not fit a linear regression (a straight line through all data points), it might be ideal for polynomial regression.

Polynomial regression, like linear regression, uses the relationship between the variables x and y to find the best way to draw a line through the data points.



### How Does it Work?

Python has methods for finding a relationship between data-points and to draw a line of polynomial regression. We will show you how to use these methods instead of going through the mathematic formula.

In the example below, we have registered 18 cars as they were passing a certain tollbooth.

We have registered the car's speed, and the time of day (hour) the passing occurred.

The x-axis represents the hours of the day and the y-axis represents the speed:

Example

Start by drawing a scatter plot:

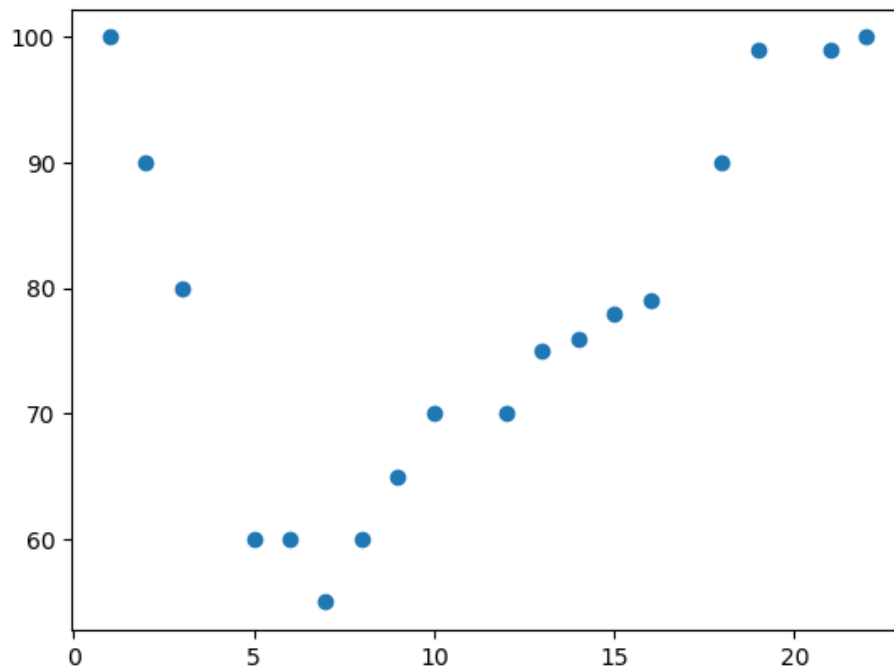
```
import matplotlib.pyplot as plt
```

```
x = [1,2,3,5,6,7,8,9,10,12,13,14,15,16,18,19,21,22]
```

```
y = [100,90,80,60,60,55,60,65,70,70,75,76,78,79,90,99,99,100]
```

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

Result:



[Run example »](#)

Example

Import numpy and matplotlib then draw the line of Polynomial Regression:

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

```
x = [1,2,3,5,6,7,8,9,10,12,13,14,15,16,18,19,21,22]
y = [100,90,80,60,60,55,60,65,70,70,75,76,78,79,90,99,99,100]
```

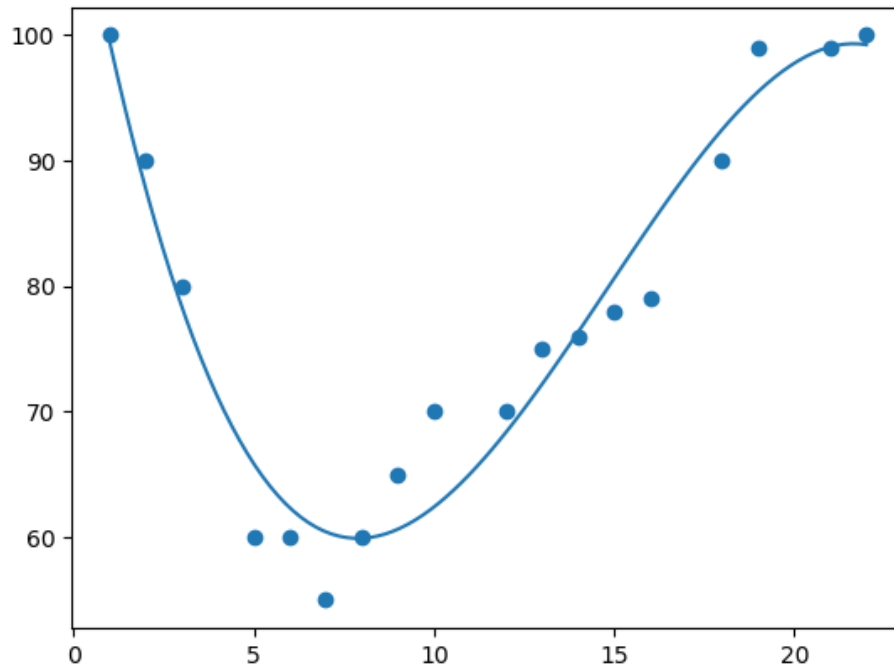
```
mymodel = numpy.poly1d(numpy.polyfit(x, y, 3))
```

```
myline = numpy.linspace(1, 22, 100)
```

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

```
plt.plot(myline, mymodel(myline))  
plt.show()
```

Result:



[Run example »](#)

Example Explained

Import the modules you need.

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

Create the arrays that represent the values of the x and y axis:

```
x = [1,2,3,5,6,7,8,9,10,12,13,14,15,16,18,19,21,22]  
y = [100,90,80,60,60,55,60,65,70,70,75,76,78,79,90,99,99,100]
```

NumPy has a method that lets us make a polynomial model:

```
mymodel = numpy.poly1d(numpy.polyfit(x, y, 3))
```



Then specify how the line will display, we start at position 1, and end at position 22:

```
myline = numpy.linspace(1, 22, 100)
```

Draw the original scatter plot:

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

Draw the line of polynomial regression:

```
plt.plot(myline, mymodel(myline))
```

Display the diagram:

```
plt.show()
```

## **R-Squared**

It is important to know how well the relationship between the values of the x- and y-axis is, if there are no relationship the polynomial regression can not be used to predict anything.

The relationship is measured with a value called the r-squared.

The r-squared value ranges from 0 to 1, where 0 means no relationship, and 1 means 100% related.

Python and the Sklearn module will compute this value for you, all you have to do is feed it with the x and y arrays:

Example

How well does my data fit in a polynomial regression?

```
import numpy
```

```
from sklearn.metrics import r2_score
```

```
x = [1,2,3,5,6,7,8,9,10,12,13,14,15,16,18,19,21,22]
```

```
y = [100,90,80,60,60,55,60,65,70,70,75,76,78,79,90,99,99,100]
```

```
mymodel = numpy.poly1d(numpy.polyfit(x, y, 3))
```

```
print(r2_score(y, mymodel(x)))
```

**Note:** The result 0.94 shows that there is a very good relationship, and we can use polynomial regression in future predictions.

### Predict Future Values

Now we can use the information we have gathered to predict future values.

Example: Let us try to predict the speed of a car that passes the tollbooth at around the time 17:00:

To do so, we need the same mymodel array from the example above:

```
mymodel = numpy.poly1d(numpy.polyfit(x, y, 3))
```

### Example

Predict the speed of a car passing at 17:00:

```
import numpy
from sklearn.metrics import r2_score
```

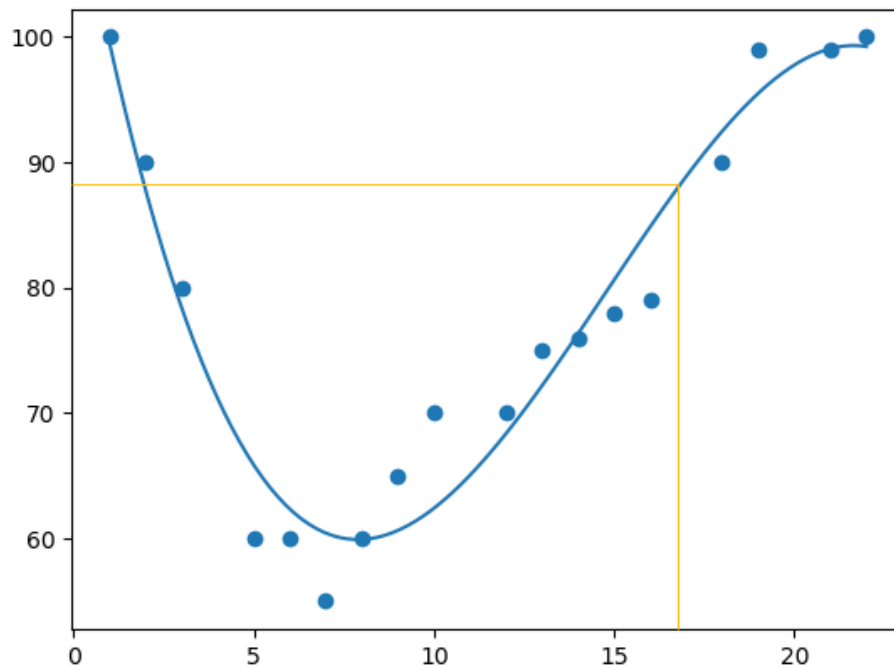
```
x = [1,2,3,5,6,7,8,9,10,12,13,14,15,16,18,19,21,22]
y = [100,90,80,60,60,55,60,65,70,70,75,76,78,79,90,99,99,100]
```

```
mymodel = numpy.poly1d(numpy.polyfit(x, y, 3))
```

```
speed = mymodel(17)
print(speed)
```

[Run example »](#)

The example predicted a speed to be 88.87, which we also could read from the diagram:



## Bad Fit?

Let us create an example where polynomial regression would not be the best method to predict future values.

### Example

These values for the x- and y-axis should result in a very bad fit for polynomial regression:

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

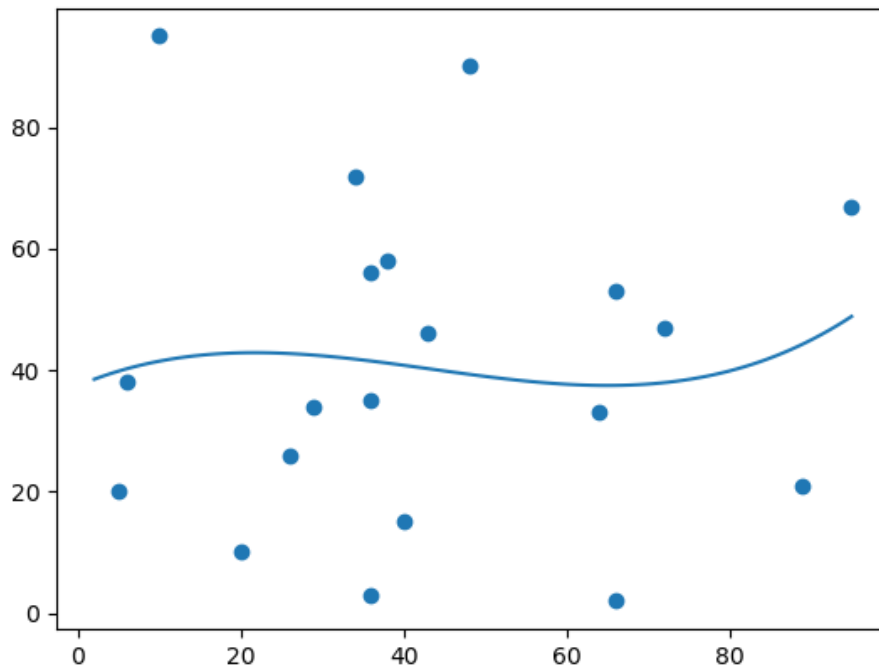
```
x = [89,43,36,36,95,10,66,34,38,20,26,29,48,64,6,5,36,66,72,40]
y = [21,46,3,35,67,95,53,72,58,10,26,34,90,33,38,20,56,2,47,15]
```

```
mymodel = numpy.poly1d(numpy.polyfit(x, y, 3))
```

```
myline = numpy.linspace(2, 95, 100)
```

```
plt.scatter(x, y)
plt.plot(myline, mymodel(myline))
plt.show()
```

Result:



[Run example »](#)

**And the r-squared value?**

Example

You should get a very low r-squared value.

```
import numpy
from sklearn.metrics import r2_score

x = [89,43,36,36,95,10,66,34,38,20,26,29,48,64,6,5,36,66,72,40]
y = [21,46,3,35,67,95,53,72,58,10,26,34,90,33,38,20,56,2,47,15]
```

```
mymodel = numpy.poly1d(numpy.polyfit(x, y, 3))
```

```
print(r2_score(y, mymodel(x)))
```

Multiple Regression

Multiple regression is like linear regression, but with more than one independent value, meaning that we try to predict a value based on **two or more** variables.

Take a look at the data set below, it contains some information about cars.

Car	Model	Volume	Weight	CO2
Toyota	Aygo	1000	790	99
Mitsubishi	Space Star	1200	1160	95
Skoda	Citigo	1000	929	95
Fiat	500	900	865	90
Mini	Cooper	1500	1140	105
VW	Up!	1000	929	105
Skoda	Fabia	1400	1109	90
Mercedes	A-Class	1500	1365	92
Ford	Fiesta	1500	1112	98
Audi	A1	1600	1150	99
Hyundai	I20	1100	980	99
Suzuki	Swift	1300	990	101
Ford	Fiesta	1000	1112	99
Honda	Civic	1600	1252	94
Hundai	I30	1600	1326	97
Opel	Astra	1600	1330	97

BMW	1	1600	1365	99
Mazda	3	2200	1280	104
Skoda	Rapid	1600	1119	104
Ford	Focus	2000	1328	105
Ford	Mondeo	1600	1584	94
Opel	Insignia	2000	1428	99
Mercedes	C-Class	2100	1365	99
Skoda	Octavia	1600	1415	99
Volvo	S60	2000	1415	99
Mercedes	CLA	1500	1465	102
Audi	A4	2000	1490	104
Audi	A6	2000	1725	114
Volvo	V70	1600	1523	109
BMW	5	2000	1705	114
Mercedes	E-Class	2100	1605	115
Volvo	XC70	2000	1746	117
Ford	B-Max	1600	1235	104
BMW	2	1600	1390	108

Opel	Zafira	1600	1405	109
Mercedes	SLK	2500	1395	120

We can predict the CO2 emission of a car based on the size of the engine, but with multiple regression we can throw in more variables, like the weight of the car, to make the prediction more accurate.

### How Does it Work?

In Python we have modules that will do the work for us. Start by importing the Pandas module.

```
import pandas
```

Learn about the Pandas module in our [Pandas Tutorial](#).

The Pandas module allows us to read csv files and return a DataFrame object.

The file is meant for testing purposes only, you can download it here: [data.csv](#)

```
df = pandas.read_csv("data.csv")
```

Then make a list of the independent values and call this variable X.

Put the dependent values in a variable called y.

```
X = df[['Weight', 'Volume']]  
y = df['CO2']
```

**Tip:** It is common to name the list of independent values with a upper case X, and the list of dependent values with a lower case y.

We will use some methods from the sklearn module, so we will have to import that module as well:

```
from sklearn import linear_model
```

From the sklearn module we will use the LinearRegression() method to create a linear regression object.

This object has a method called fit() that takes the independent and dependent values as parameters and fills the regression object with data that describes the relationship:

```
regr = linear_model.LinearRegression()  
regr.fit(X, y)
```

Now we have a regression object that are ready to predict CO2 values based on a car's weight and volume:

```
#predict the CO2 emission of a car where the weight is 2300kg, and the volume is  
1300cm3:  
predictedCO2 = regr.predict([[2300, 1300]])
```

### Example

See the whole example in action:

```
import pandas  
from sklearn import linear_model
```

```
df = pandas.read_csv("data.csv")
```

```
X = df[['Weight', 'Volume']]  
y = df['CO2']
```

```
regr = linear_model.LinearRegression()  
regr.fit(X, y)
```

```
#predict the CO2 emission of a car where the weight is 2300kg, and the volume is  
1300cm3:  
predictedCO2 = regr.predict([[2300, 1300]])
```

```
print(predictedCO2)
```

Result:

```
[107.2087328]
```

[Run example »](#)

We have predicted that a car with 1.3 liter engine, and a weight of 2300 kg, will release approximately 107 grams of CO2 for every kilometer it drives.

### Coefficient

The coefficient is a factor that describes the relationship with an unknown variable.

Example: if x is a variable, then 2x is x two times. x is the unknown variable, and the number 2 is the coefficient.



In this case, we can ask for the coefficient value of weight against CO2, and for volume against CO2. The answer(s) we get tells us what would happen if we increase, or decrease, one of the independent values.

### Example

Print the coefficient values of the regression object:

```
import pandas
from sklearn import linear_model

df = pandas.read_csv("data.csv")

X = df[['Weight', 'Volume']]
y = df['CO2']

regr = linear_model.LinearRegression()
regr.fit(X, y)

print(regr.coef_)
```

Result:

```
[0.00755095 0.00780526]
```

[Run example »](#)

### Result Explained

The result array represents the coefficient values of weight and volume.

Weight: 0.00755095  
Volume: 0.00780526

These values tell us that if the weight increase by 1kg, the CO2 emission increases by 0.00755095g.

And if the engine size (Volume) increases by 1 cm<sup>3</sup>, the CO2 emission increases by 0.00780526 g.

I think that is a fair guess, but let test it!

We have already predicted that if a car with a 1300cm<sup>3</sup> engine weighs 2300kg, the CO2 emission will be approximately 107g.

What if we increase the weight with 1000kg?

## Example

Copy the example from before, but change the weight from 2300 to 3300:

```
import pandas
from sklearn import linear_model

df = pandas.read_csv("data.csv")

X = df[['Weight', 'Volume']]
y = df['CO2']

regr = linear_model.LinearRegression()
regr.fit(X, y)

predictedCO2 = regr.predict([[3300, 1300]])

print(predictedCO2)
```

Result:

```
[114.75968007]
```

We have predicted that a car with 1.3 liter engine, and a weight of 3300 kg, will release approximately 115 grams of CO2 for every kilometer it drives.

Which shows that the coefficient of 0.00755095 is correct:

$$107.2087328 + (1000 * 0.00755095) = 114.75968$$

## Scale Features

When your data has different values, and even different measurement units, it can be difficult to compare them. What is kilograms compared to meters? Or altitude compared to time?

The answer to this problem is scaling. We can scale data into new values that are easier to compare.

Take a look at the table below, it is the same data set that we used in the multiple regression chapter, but this time the **volume** column contains values in *liters* instead of  $cm^3$  (1.0 instead of 1000).

Car	Model	Volume	Weight	CO2
-----	-------	--------	--------	-----

Toyota	Aygo	1.0	790	99
Mitsubishi	Space Star	1.2	1160	95
Skoda	Citigo	1.0	929	95
Fiat	500	0.9	865	90
Mini	Cooper	1.5	1140	105
VW	Up!	1.0	929	105
Skoda	Fabia	1.4	1109	90
Mercedes	A-Class	1.5	1365	92
Ford	Fiesta	1.5	1112	98
Audi	A1	1.6	1150	99
Hyundai	I20	1.1	980	99
Suzuki	Swift	1.3	990	101
Ford	Fiesta	1.0	1112	99
Honda	Civic	1.6	1252	94
Hundai	I30	1.6	1326	97
Opel	Astra	1.6	1330	97
BMW	1	1.6	1365	99
Mazda	3	2.2	1280	104

Skoda	Rapid	1.6	1119	104
Ford	Focus	2.0	1328	105
Ford	Mondeo	1.6	1584	94
Opel	Insignia	2.0	1428	99
Mercedes	C-Class	2.1	1365	99
Skoda	Octavia	1.6	1415	99
Volvo	S60	2.0	1415	99
Mercedes	CLA	1.5	1465	102
Audi	A4	2.0	1490	104
Audi	A6	2.0	1725	114
Volvo	V70	1.6	1523	109
BMW	5	2.0	1705	114
Mercedes	E-Class	2.1	1605	115
Volvo	XC70	2.0	1746	117
Ford	B-Max	1.6	1235	104
BMW	2	1.6	1390	108
Opel	Zafira	1.6	1405	109
Mercedes	SLK	2.5	1395	120

It can be difficult to compare the volume 1.0 with the weight 790, but if we scale them both into comparable values, we can easily see how much one value is compared to the other.

There are different methods for scaling data, in this tutorial we will use a method called standardization.

The standardization method uses this formula:

$$z = (x - u) / s$$

Where z is the new value, x is the original value, u is the mean and s is the standard deviation.

If you take the **weight** column from the data set above, the first value is 790, and the scaled value will be:

$$(790 - \underline{1292.23}) / \underline{238.74} = -2.1$$

If you take the **volume** column from the data set above, the first value is 1.0, and the scaled value will be:

$$(1.0 - \underline{1.61}) / \underline{0.38} = -1.59$$

Now you can compare -2.1 with -1.59 instead of comparing 790 with 1.0.

You do not have to do this manually, the Python sklearn module has a method called StandardScaler() which returns a Scaler object with methods for transforming data sets.

Example

Scale all values in the Weight and Volume columns:

```
import pandas
from sklearn import linear_model
from sklearn.preprocessing import StandardScaler
scale = StandardScaler()
```

```
df = pandas.read_csv("data.csv")
```

```
X = df[['Weight', 'Volume']]
```

```
scaledX = scale.fit_transform(X)
```

```
print(scaledX)
```

Result:

Note that the first two values are -2.1 and -1.59, which corresponds to our calculations:

```
[[-2.10389253 -1.59336644]
 [-0.55407235 -1.07190106]
 [-1.52166278 -1.59336644]
 [-1.78973979 -1.85409913]
 [-0.63784641 -0.28970299]
 [-1.52166278 -1.59336644]
 [-0.76769621 -0.55043568]
 [ 0.3046118  -0.28970299]
 [-0.7551301  -0.28970299]
 [-0.59595938 -0.0289703 ]
 [-1.30803892 -1.33263375]
 [-1.26615189 -0.81116837]
 [-0.7551301  -1.59336644]
 [-0.16871166 -0.0289703 ]
 [ 0.14125238 -0.0289703 ]
 [ 0.15800719 -0.0289703 ]
 [ 0.3046118  -0.0289703 ]
 [-0.05142797  1.53542584]
 [-0.72580918 -0.0289703 ]
 [ 0.14962979  1.01396046]
 [ 1.2219378  -0.0289703 ]
 [ 0.5685001  1.01396046]
 [ 0.3046118  1.27469315]
 [ 0.51404696 -0.0289703 ]
 [ 0.51404696  1.01396046]
 [ 0.72348212 -0.28970299]
 [ 0.8281997  1.01396046]
 [ 1.81254495  1.01396046]
 [ 0.96642691 -0.0289703 ]
 [ 1.72877089  1.01396046]
 [ 1.30990057  1.27469315]
 [ 1.90050772  1.01396046]
 [-0.23991961 -0.0289703 ]
 [ 0.40932938 -0.0289703 ]
 [ 0.47215993 -0.0289703 ]
 [ 0.4302729  2.31762392]]
```

Predict CO2 Values

The task in the Multiple Regression chapter was to predict the CO2 emission from a car when you only knew its weight and volume.

When the data set is scaled, you will have to use the scale when you predict values:

Example

Predict the CO2 emission from a 1.3 liter car that weighs 2300 kilograms:

```
import pandas
from sklearn import linear_model
from sklearn.preprocessing import StandardScaler
scale = StandardScaler()
```

```
df = pandas.read_csv("data.csv")
```

```
X = df[['Weight', 'Volume']]
y = df['CO2']
```

```
scaledX = scale.fit_transform(X)
```

```
regr = linear_model.LinearRegression()
regr.fit(scaledX, y)
```

```
scaled = scale.transform([[2300, 1.3]])
```

```
predictedCO2 = regr.predict([scaled[0]])
print(predictedCO2)
```

Result:

```
[107.2087328]
```

## **Machine Learning - Train/Test**

### **Evaluate Your Model**

In Machine Learning we create models to predict the outcome of certain events, like in the previous chapter where we predicted the CO2 emission of a car when we knew the weight and engine size.

To measure if the model is good enough, we can use a method called Train/Test.

## What is Train/Test

Train/Test is a method to measure the accuracy of your model.

It is called Train/Test because you split the the data set into two sets: a training set and a testing set.

80% for training, and 20% for testing.

You *train* the model using the training set.

You *test* the model using the testing set.

*Train* the model means *create* the model.

*Test* the model means test the accuracy of the model.

## Start With a Data Set

Start with a data set you want to test.

Our data set illustrates 100 customers in a shop, and their shopping habits.

Example

```
import numpy
import matplotlib.pyplot as plt
numpy.random.seed(2)
```

```
x = numpy.random.normal(3, 1, 100)
y = numpy.random.normal(150, 40, 100) / x
```

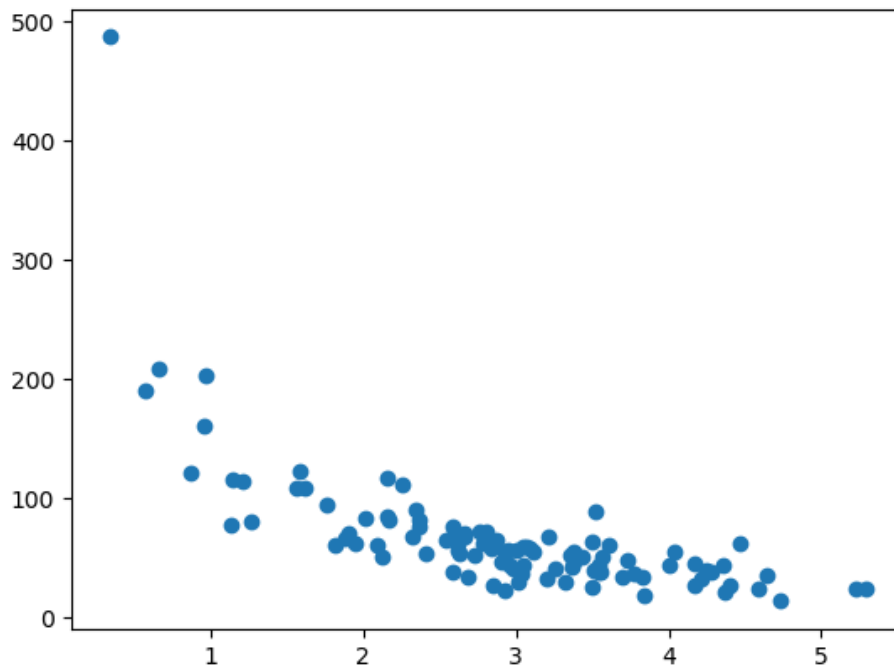
```
plt.scatter(x, y)
plt.show()
```

Result:

The x axis represents the number of minutes before making a purchase.

The y axis represents the amount of money spent on the purchase.





[Run example »](#)

### Split Into Train/Test

The *training* set should be a random selection of 80% of the original data.

The *testing* set should be the remaining 20%.

```
train_x = x[:80]  
train_y = y[:80]
```

```
test_x = x[80:]  
test_y = y[80:]
```

### Display the Training Set

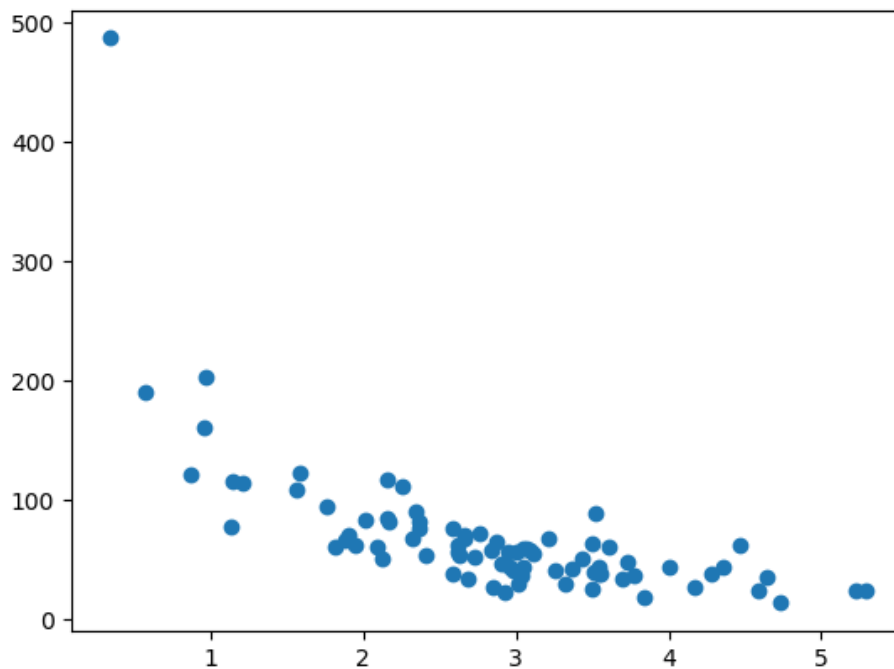
Display the same scatter plot with the training set:

Example

```
plt.scatter(train_x, train_y)  
plt.show()
```

Result:

It looks like the original data set, so it seems to be a fair selection:



[Run example »](#)

### Display the Testing Set

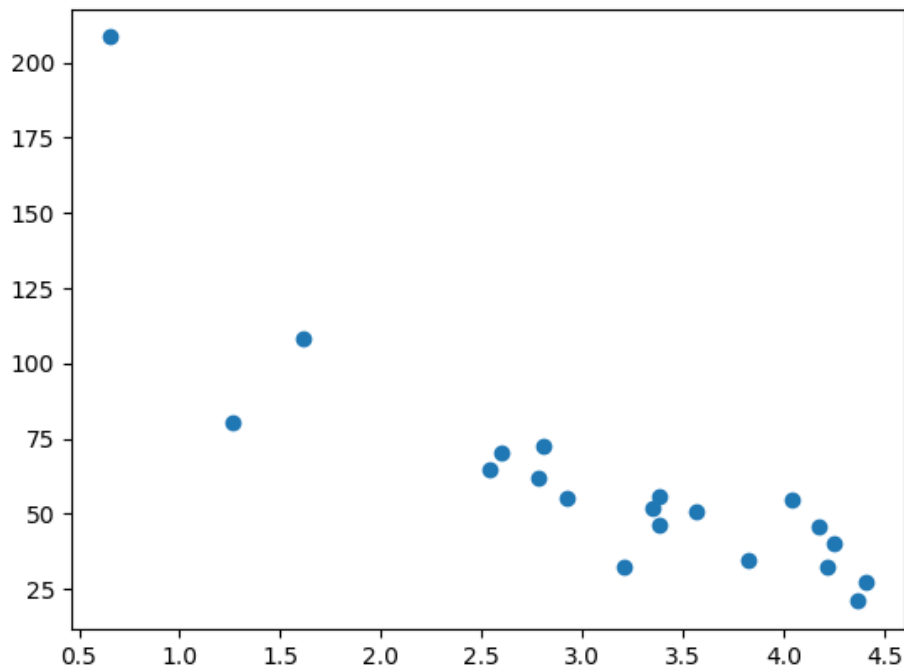
To make sure the testing set is not completely different, we will take a look at the testing set as well.

Example

```
plt.scatter(test_x, test_y)
plt.show()
```

Result:

The testing set also looks like the original data set:



[Run example »](#)

## Fit the Data Set

What does the data set look like? In my opinion I think the best fit would be a polynomial regression, so let us draw a line of polynomial regression.

To draw a line through the data points, we use the `plot()` method of the `matplotlib` module:

### Example

Draw a polynomial regression line through the data points:

```
import numpy
import matplotlib.pyplot as plt
numpy.random.seed(2)

x = numpy.random.normal(3, 1, 100)
y = numpy.random.normal(150, 40, 100) / x

train_x = x[:80]
train_y = y[:80]
```

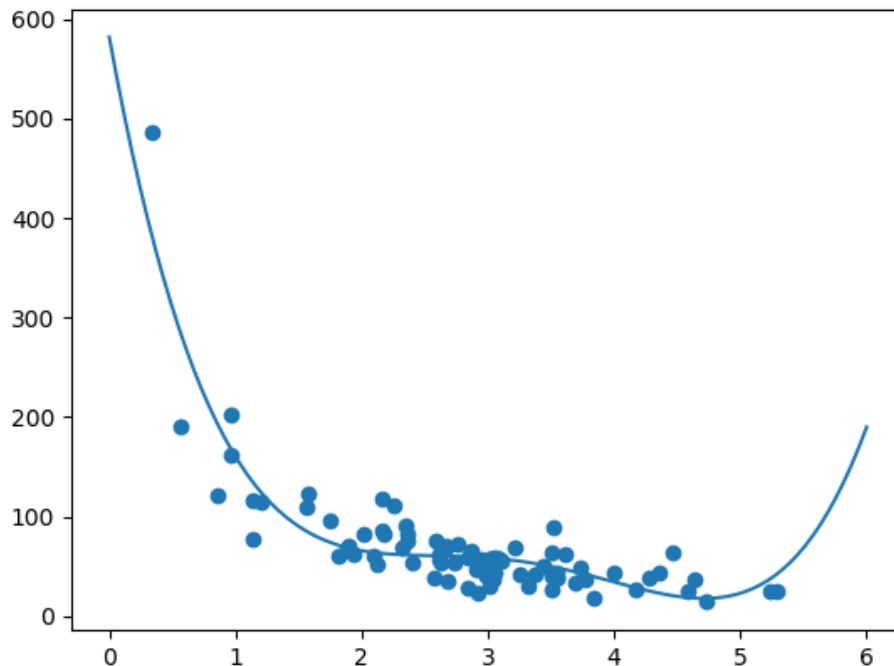
```
test_x = x[80:]
test_y = y[80:]

mymodel = numpy.poly1d(numpy.polyfit(train_x, train_y, 4))

myline = numpy.linspace(0, 6, 100)

plt.scatter(train_x, train_y)
plt.plot(myline, mymodel(myline))
plt.show()
```

Result:



[Run example »](#)

The result can back my suggestion of the data set fitting a polynomial regression, even though it would give us some weird results if we try to predict values outside of the data set. Example: the line indicates that a customer spending 6 minutes in the shop would make a purchase worth 200. That is probably a sign of overfitting.

But what about the R-squared score? The R-squared score is a good indicator of how well my data set is fitting the model.

## R<sup>2</sup>

Remember R<sup>2</sup>, also known as R-squared?

It measures the relationship between the x axis and the y axis, and the value ranges from 0 to 1, where 0 means no relationship, and 1 means totally related.

The sklearn module has a method called `r2_score()` that will help us find this relationship.

In this case we would like to measure the relationship between the minutes a customer stays in the shop and how much money they spend.

### Example

How well does my training data fit in a polynomial regression?

```
import numpy
from sklearn.metrics import r2_score
numpy.random.seed(2)

x = numpy.random.normal(3, 1, 100)
y = numpy.random.normal(150, 40, 100) / x

train_x = x[:80]
train_y = y[:80]

test_x = x[80:]
test_y = y[80:]

mymodel = numpy.poly1d(numpy.polyfit(train_x, train_y, 4))

r2 = r2_score(train_y, mymodel(train_x))

print(r2)
```

**Try it Yourself »**

**Note:** The result 0.799 shows that there is a OK relationship.

### Bring in the Testing Set

Now we have made a model that is OK, at least when it comes to training data.

Now we want to test the model with the testing data as well, to see if gives us the same result.

### Example

Let us find the R2 score when using testing data:

```
import numpy
from sklearn.metrics import r2_score
numpy.random.seed(2)

x = numpy.random.normal(3, 1, 100)
y = numpy.random.normal(150, 40, 100) / x

train_x = x[:80]
train_y = y[:80]

test_x = x[80:]
test_y = y[80:]

mymodel = numpy.poly1d(numpy.polyfit(train_x, train_y, 4))

r2 = r2_score(test_y, mymodel(test_x))

print(r2)
```

[Try it Yourself »](#)

**Note:** The result 0.809 shows that the model fits the testing set as well, and we are confident that we can use the model to predict future values.

### Predict Values

Now that we have established that our model is OK, we can start predicting new values.

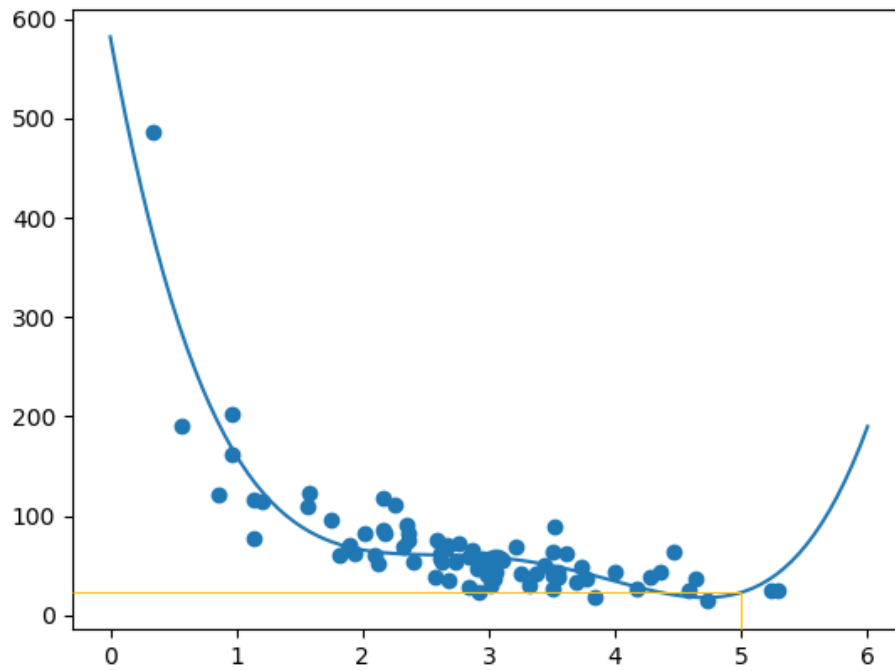
### Example

How much money will a buying customer spend, if she or he stays in the shop for 5 minutes?

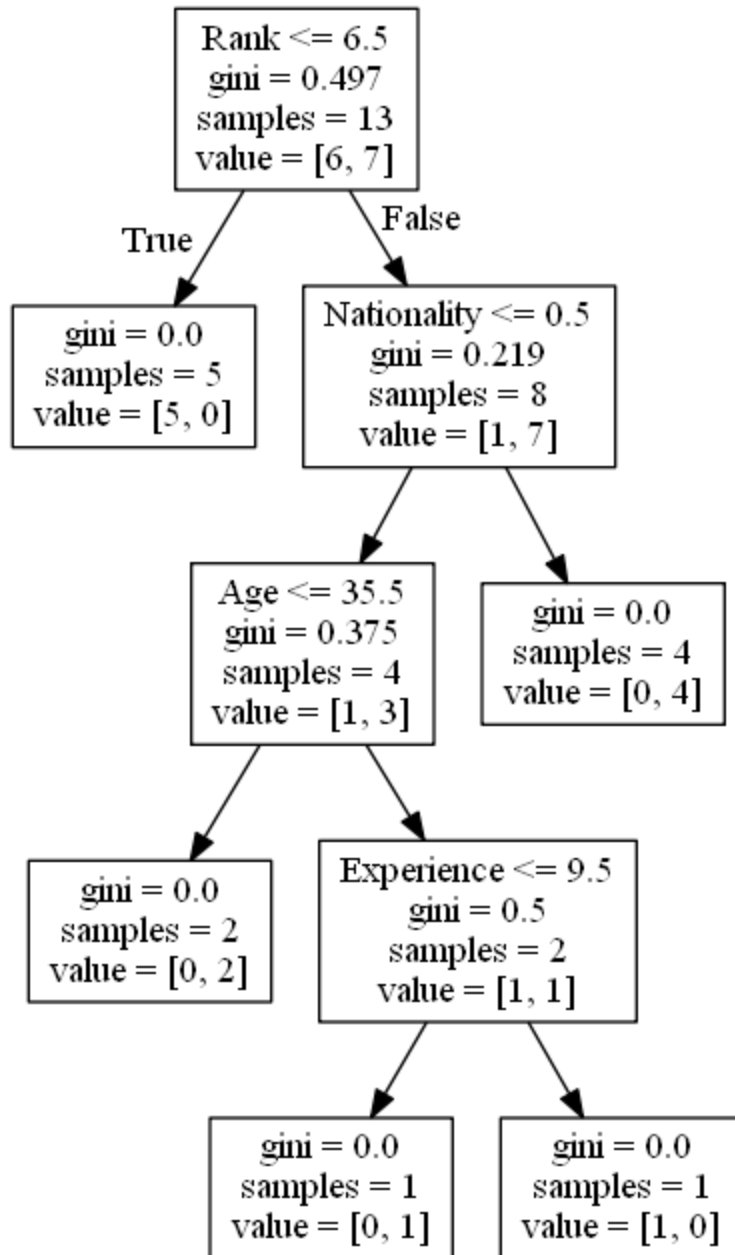
```
print(mymodel(5))
```

[Run example »](#)

The example predicted the customer to spend 22.88 dollars, as seems to correspond to the diagram:



## Machine Learning - Decision Tree



## Decision Tree

In this chapter we will show you how to make a "Decision Tree". A Decision Tree is a Flow Chart, and can help you make decisions based on previous experience.

In the example, a person will try to decide if he/she should go to a comedy show or not.



Luckily our example person has registered every time there was a comedy show in town, and registered some information about the comedian, and also registered if he/she went or not.

Age	Experience	Rank	Nationality	Go
36	10	9	UK	NO
42	12	4	USA	NO
23	4	6	N	NO
52	4	4	USA	NO
43	21	8	USA	YES
44	14	5	UK	NO
66	3	7	N	YES
35	14	9	UK	YES
52	13	7	N	YES
35	5	9	N	YES
24	3	5	USA	NO
18	3	7	UK	YES

45	9	9	UK	YES
----	---	---	----	-----

Now, based on this data set, Python can create a decision tree that can be used to decide if any new shows are worth attending to.

## How Does it Work?

First, read the dataset with pandas:

Example

Read and print the data set:

```
import pandas
```

```
df = pandas.read_csv("data.csv")
```

```
print(df)
```

[Run example »](#)

To make a decision tree, all data has to be numerical.

We have to convert the non numerical columns 'Nationality' and 'Go' into numerical values.

Pandas has a map() method that takes a dictionary with information on how to convert the values.

```
{'UK': 0, 'USA': 1, 'N': 2}
```

Means convert the values 'UK' to 0, 'USA' to 1, and 'N' to 2.

Example

Change string values into numerical values:

```
d = {'UK': 0, 'USA': 1, 'N': 2}
```

```
df['Nationality'] = df['Nationality'].map(d)
```

```
d = {'YES': 1, 'NO': 0}
```

```
df['Go'] = df['Go'].map(d)
```

```
print(df)
```

[Run example »](#)

Then we have to separate the *feature* columns from the *target* column.

The feature columns are the columns that we try to predict *from*, and the target column is the column with the values we try to predict.

Example

X is the feature columns, y is the target column:

```
features = ['Age', 'Experience', 'Rank', 'Nationality']
```

```
X = df[features]
```

```
y = df['Go']
```

```
print(X)
```

```
print(y)
```

[Run example »](#)

Now we can create the actual decision tree, fit it with our details. Start by importing the modules we need:

Example

Create and display a Decision Tree:

```
import pandas
```

```
from sklearn import tree
```

```
from sklearn.tree import DecisionTreeClassifier
```

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

```
df = pandas.read_csv("data.csv")
```

```
d = {'UK': 0, 'USA': 1, 'N': 2}
```

```
df['Nationality'] = df['Nationality'].map(d)
```

```
d = {'YES': 1, 'NO': 0}
```

```
df['Go'] = df['Go'].map(d)
```

```
features = ['Age', 'Experience', 'Rank', 'Nationality']
```

```
X = df[features]
```

```
y = df['Go']
```

```
dtree = DecisionTreeClassifier()
dtree = dtree.fit(X, y)

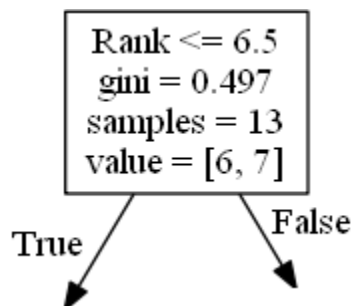
tree.plot_tree(dtree, feature_names=features)
```

[Run example »](#)

## Result Explained

The decision tree uses your earlier decisions to calculate the odds for you to wanting to go see a comedian or not.

Let us read the different aspects of the decision tree:



Rank

Rank <= 6.5 means that every comedian with a rank of 6.5 or lower will follow the True arrow (to the left), and the rest will follow the False arrow (to the right).

gini = 0.497 refers to the quality of the split, and is always a number between 0.0 and 0.5, where 0.0 would mean all of the samples got the same result, and 0.5 would mean that the split is done exactly in the middle.

samples = 13 means that there are 13 comedians left at this point in the decision, which is all of them since this is the first step.

value = [6, 7] means that of these 13 comedians, 6 will get a "NO", and 7 will get a "GO".

## Gini

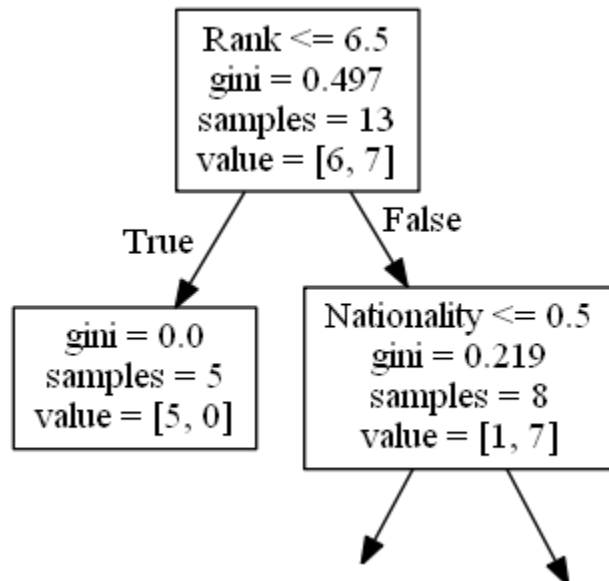
There are many ways to split the samples, we use the GINI method in this tutorial.

The Gini method uses this formula:

$$\text{Gini} = 1 - (x/n)^2 - (y/n)^2$$

Where x is the number of positive answers("GO"), n is the number of samples, and y is the number of negative answers ("NO"), which gives us this calculation:

$$1 - (7 / 13)^2 + (6 / 13)^2 = 0.497$$



The next step contains two boxes, one box for the comedians with a 'Rank' of 6.5 or lower, and one box with the rest.

True - 5 Comedians End Here:

gini = 0.0 means all of the samples got the same result.

samples = 5 means that there are 5 comedians left in this branch (5 comedian with a Rank of 6.5 or lower).

value = [5, 0] means that 5 will get a "NO" and 0 will get a "GO".

False - 8 Comedians Continue:

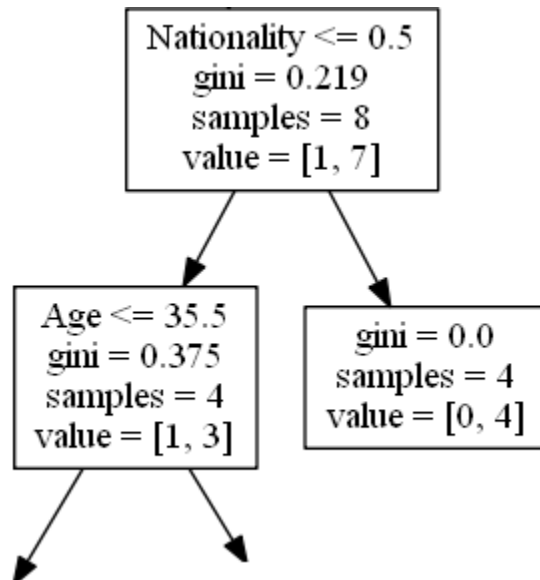
Nationality

Nationality <= 0.5 means that the comedians with a nationality value of less than 0.5 will follow the arrow to the left (which means everyone from the UK, ), and the rest will follow the arrow to the right.

gini = 0.219 means that about 22% of the samples would go in one direction.

samples = 8 means that there are 8 comedians left in this branch (8 comedian with a Rank higher than 6.5).

value = [1, 7] means that of these 8 comedians, 1 will get a "NO" and 7 will get a "GO".



True - 4 Comedians Continue:

Age

Age <= 35.5 means that comedians at the age of 35.5 or younger will follow the arrow to the left, and the rest will follow the arrow to the right.

gini = 0.375 means that about 37,5% of the samples would go in one direction.

samples = 4 means that there are 4 comedians left in this branch (4 comedians from the UK).

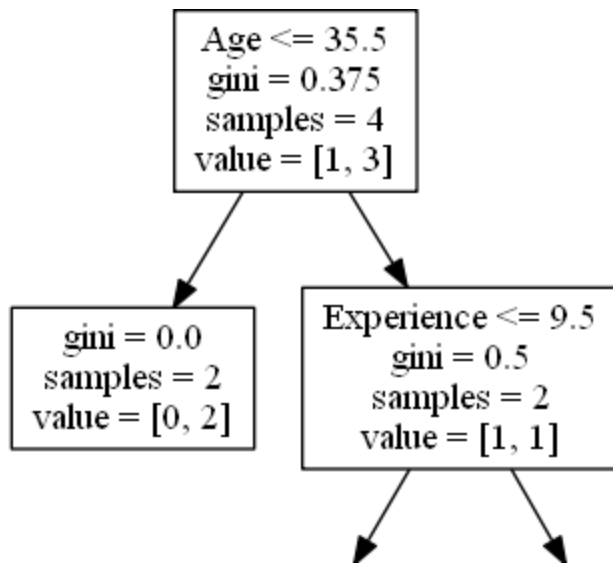
value = [1, 3] means that of these 4 comedians, 1 will get a "NO" and 3 will get a "GO".

False - 4 Comedians End Here:

gini = 0.0 means all of the samples got the same result.

samples = 4 means that there are 4 comedians left in this branch (4 comedians not from the UK).

value = [0, 4] means that of these 4 comedians, 0 will get a "NO" and 4 will get a "GO".



True - 2 Comedians End Here:

$\text{gini} = 0.0$  means all of the samples got the same result.

$\text{samples} = 2$  means that there are 2 comedians left in this branch (2 comedians at the age 35.5 or younger).

$\text{value} = [0, 2]$  means that of these 2 comedians, 0 will get a "NO" and 2 will get a "GO".

False - 2 Comedians Continue:

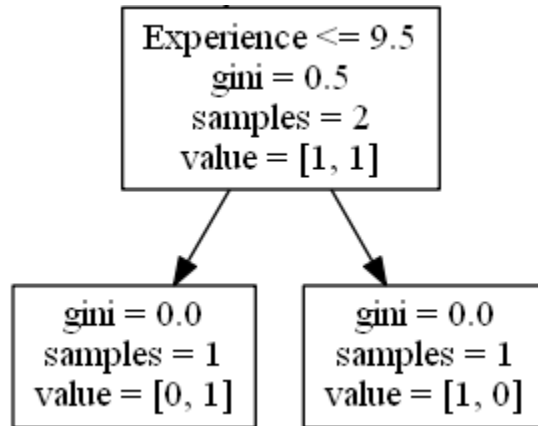
Experience

$\text{Experience} \leq 9.5$  means that comedians with 9.5 years of experience, or less, will follow the arrow to the left, and the rest will follow the arrow to the right.

$\text{gini} = 0.5$  means that 50% of the samples would go in one direction.

$\text{samples} = 2$  means that there are 2 comedians left in this branch (2 comedians older than 35.5).

$\text{value} = [1, 1]$  means that of these 2 comedians, 1 will get a "NO" and 1 will get a "GO".



True - 1 Comedian Ends Here:

gini = 0.0 means all of the samples got the same result.

samples = 1 means that there is 1 comedian left in this branch (1 comedian with 9.5 years of experience or less).

value = [0, 1] means that 0 will get a "NO" and 1 will get a "GO".

False - 1 Comedian Ends Here:

gini = 0.0 means all of the samples got the same result.

samples = 1 means that there is 1 comedians left in this branch (1 comedian with more than 9.5 years of experience).

value = [1, 0] means that 1 will get a "NO" and 0 will get a "GO".

## Predict Values

We can use the Decision Tree to predict new values.

Example: Should I go see a show starring a 40 years old American comedian, with 10 years of experience, and a comedy ranking of 7?

Example

Use predict() method to predict new values:

```
print(dtree.predict([[40, 10, 7, 1]]))
```

[Run example »](#)

Example



What would the answer be if the comedy rank was 6?

```
print(dtrees.predict([[40, 10, 6, 1]]))
```

[Run example »](#)

## Different Results

You will see that the Decision Tree gives you different results if you run it enough times, even if you feed it with the same data.

That is because the Decision Tree does not give us a 100% certain answer. It is based on the probability of an outcome, and the answer will vary.

What is a confusion matrix?

It is a table that is used in classification problems to assess where errors in the model were made.

The rows represent the actual classes the outcomes should have been. While the columns represent the predictions we have made. Using this table it is easy to see which predictions are wrong.

## Creating a Confusion Matrix

Confusion matrixes can be created by predictions made from a logistic regression.

For now we will generate actual and predicted values by utilizing NumPy:

```
import numpy
```

Next we will need to generate the numbers for "actual" and "predicted" values.

```
actual = numpy.random.binomial(1, 0.9, size = 1000)
predicted = numpy.random.binomial(1, 0.9, size = 1000)
```

In order to create the confusion matrix we need to import metrics from the sklearn module.

```
from sklearn import metrics
```

Once metrics is imported we can use the confusion matrix function on our actual and predicted values.

```
confusion_matrix = metrics.confusion_matrix(actual, predicted)
```

To create a more interpretable visual display we need to convert the table into a confusion matrix display.

```
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix =  
confusion_matrix, display_labels = [False, True])
```

Vizualizing the display requires that we import pyplot from matplotlib.

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

Finally to display the plot we can use the functions plot() and show() from pyplot.

```
cm_display.plot()  
plt.show()
```

See the whole example in action:

Example

```
import matplotlib.pyplot as plt  
import numpy  
from sklearn import metrics
```

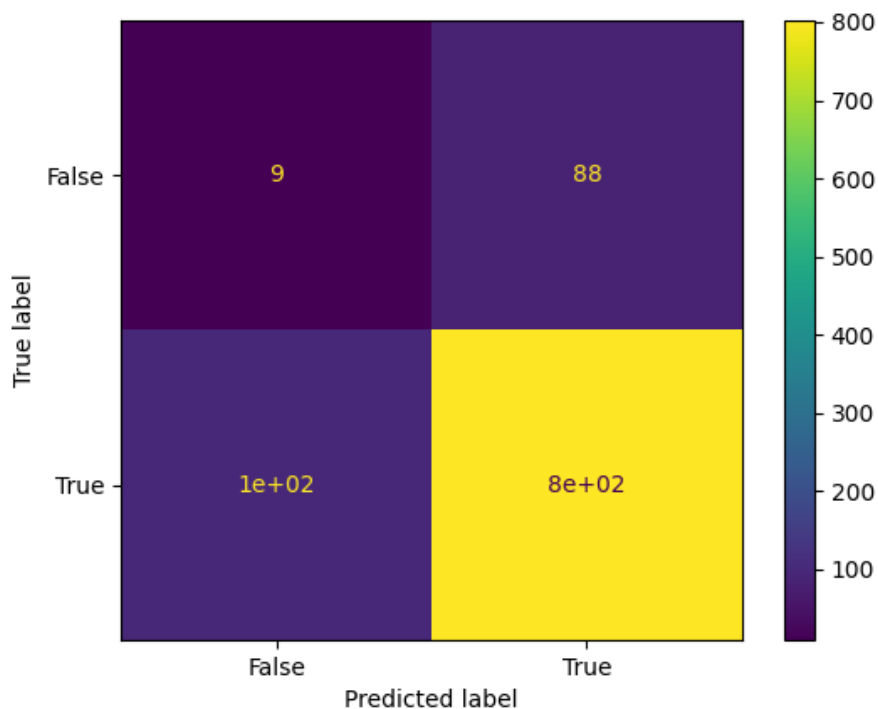
```
actual = numpy.random.binomial(1,.9,size = 1000)  
predicted = numpy.random.binomial(1,.9,size = 1000)
```

```
confusion_matrix = metrics.confusion_matrix(actual, predicted)
```

```
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix =  
confusion_matrix, display_labels = [False, True])
```

```
cm_display.plot()  
plt.show()
```

Result



[Run example »](#)

## Results Explained

The Confusion Matrix created has four different quadrants:

- True Negative (Top-Left Quadrant)
- False Positive (Top-Right Quadrant)
- False Negative (Bottom-Left Quadrant)
- True Positive (Bottom-Right Quadrant)

True means that the values were accurately predicted, False means that there was an error or wrong prediction.

Now that we have made a Confusion Matrix, we can calculate different measures to quantify the quality of the model. First, let's look at Accuracy.

## Created Metrics

The matrix provides us with many useful metrics that help us to evaluate our classification model.

The different measures include: Accuracy, Precision, Sensitivity (Recall), Specificity, and the F-score, explained below.

## Accuracy

Accuracy measures how often the model is correct.

How to Calculate

$(\text{True Positive} + \text{True Negative}) / \text{Total Predictions}$

Example

Accuracy = `metrics.accuracy_score(actual, predicted)`

[Run example »](#)

## Precision

Of the positives predicted, what percentage is truly positive?

How to Calculate

$\text{True Positive} / (\text{True Positive} + \text{False Positive})$

Precision does not evaluate the correctly predicted negative cases:

Example

Precision = `metrics.precision_score(actual, predicted)`

[Run example »](#)

## Sensitivity (Recall)

Of all the positive cases, what percentage are predicted positive?

Sensitivity (sometimes called Recall) measures how good the model is at predicting positives.

This means it looks at true positives and false negatives (which are positives that have been incorrectly predicted as negative).

How to Calculate

$\text{True Positive} / (\text{True Positive} + \text{False Negative})$

Sensitivity is good at understanding how well the model predicts something is positive:

Example

```
Sensitivity_recall = metrics.recall_score(actual, predicted)
```

[Run example »](#)

## Specificity

How well the model is at predicting negative results?

Specificity is similar to sensitivity, but looks at it from the perspective of negative results.

## How to Calculate

$\text{True Negative} / (\text{True Negative} + \text{False Positive})$

Since it is just the opposite of Recall, we use the `recall_score` function, taking the opposite position label:

Example

```
Specificity = metrics.recall_score(actual, predicted, pos_label=0)
```

[Run example »](#)

## F-score

F-score is the "harmonic mean" of precision and sensitivity.

It considers both false positive and false negative cases and is good for imbalanced datasets.

## How to Calculate

$2 * ((\text{Precision} * \text{Sensitivity}) / (\text{Precision} + \text{Sensitivity}))$

This score does not take into consideration the True Negative values:

Example

```
F1_score = metrics.f1_score(actual, predicted)
```

[Run example »](#)

All calculations in one:

## Example

```
#metrics
print({"Accuracy":Accuracy,"Precision":Precision,"Sensitivity_recall":Sensitivity_
recall,"Specificity":Specificity,"F1_score":F1_score})
```

## Hierarchical Clustering

Hierarchical clustering is an unsupervised learning method for clustering data points. The algorithm builds clusters by measuring the dissimilarities between data. Unsupervised learning means that a model does not have to be trained, and we do not need a "target" variable. This method can be used on any data to visualize and interpret the relationship between individual data points.

Here we will use hierarchical clustering to group data points and visualize the clusters using both a dendrogram and scatter plot.

### How does it work?

We will use Agglomerative Clustering, a type of hierarchical clustering that follows a bottom up approach. We begin by treating each data point as its own cluster. Then, we join clusters together that have the shortest distance between them to create larger clusters. This step is repeated until one large cluster is formed containing all of the data points.

Hierarchical clustering requires us to decide on both a distance and linkage method. We will use euclidean distance and the Ward linkage method, which attempts to minimize the variance between clusters.

## Example

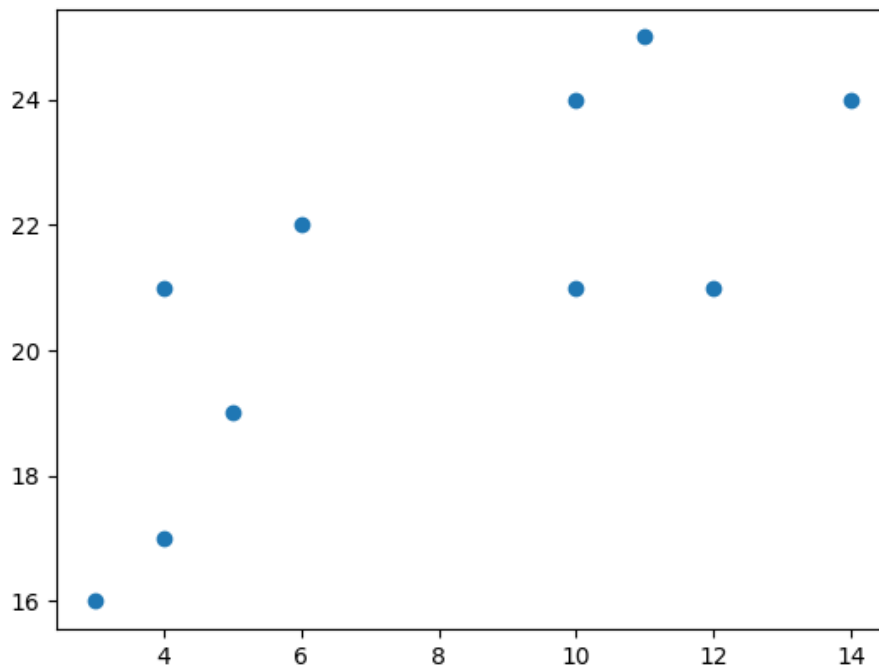
Start by visualizing some data points:

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

x = [4, 5, 10, 4, 3, 11, 14, 6, 10, 12]
y = [21, 19, 24, 17, 16, 25, 24, 22, 21, 21]

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

## Result



Now we compute the ward linkage using euclidean distance, and visualize it using a dendrogram:

Example

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage
```

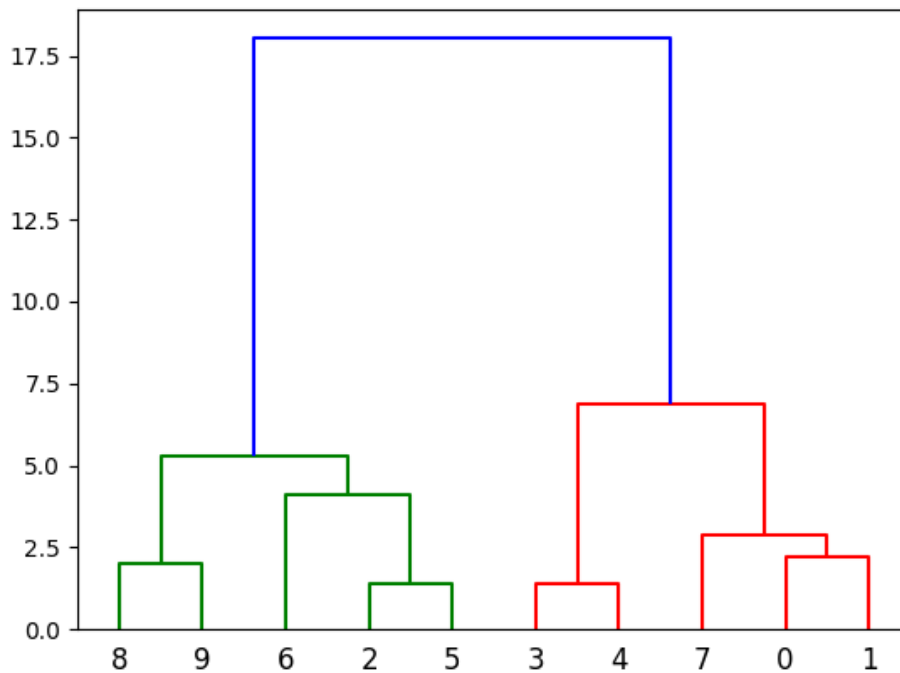
```
x = [4, 5, 10, 4, 3, 11, 14, 6, 10, 12]
y = [21, 19, 24, 17, 16, 25, 24, 22, 21, 21]
```

```
data = list(zip(x, y))
```

```
linkage_data = linkage(data, method='ward', metric='euclidean')
dendrogram(linkage_data)
```

```
plt.show()
```

Result



[Run example »](#)

Here, we do the same thing with Python's scikit-learn library. Then, visualize on a 2-dimensional plot:

Example

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import AgglomerativeClustering
```

```
x = [4, 5, 10, 4, 3, 11, 14, 6, 10, 12]
y = [21, 19, 24, 17, 16, 25, 24, 22, 21, 21]
```

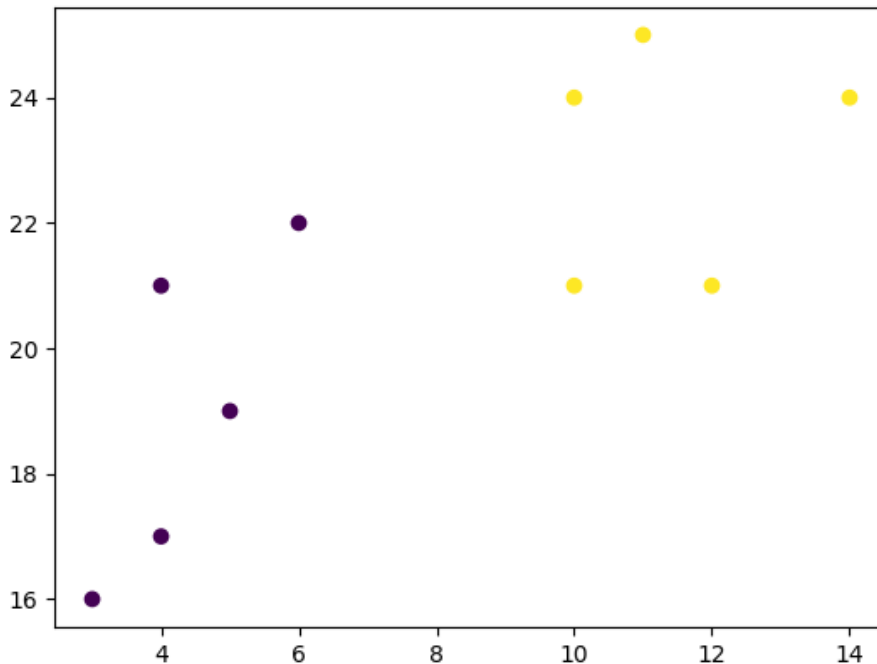
```
data = list(zip(x, y))
```

```
hierarchical_cluster = AgglomerativeClustering(n_clusters=2,
affinity='euclidean', linkage='ward')
labels = hierarchical_cluster.fit_predict(data)
```

```
plt.scatter(x, y, c=labels)
plt.show()
```

Result





[Run example »](#)

Example Explained

Import the modules you need.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage
from sklearn.cluster import AgglomerativeClustering
```

You can learn about the Matplotlib module in our ["Matplotlib Tutorial"](#).

You can learn about the SciPy module in our [SciPy Tutorial](#).

NumPy is a library for working with arrays and matrices in Python, you can learn about the NumPy module in our [NumPy Tutorial](#).

scikit-learn is a popular library for machine learning.

Create arrays that resemble two variables in a dataset. Note that while we only use two variables here, this method will work with any number of variables:

```
x = [4, 5, 10, 4, 3, 11, 14, 6, 10, 12]
y = [21, 19, 24, 17, 16, 25, 24, 22, 21, 21]
```

Turn the data into a set of points:

```
data = list(zip(x, y))
print(data)
```

Result:

```
[(4, 21), (5, 19), (10, 24), (4, 17), (3, 16), (11, 25), (14, 24), (6, 22), (10, 21),
(12, 21)]
```

Compute the linkage between all of the different points. Here we use a simple euclidean distance measure and Ward's linkage, which seeks to minimize the variance between clusters.

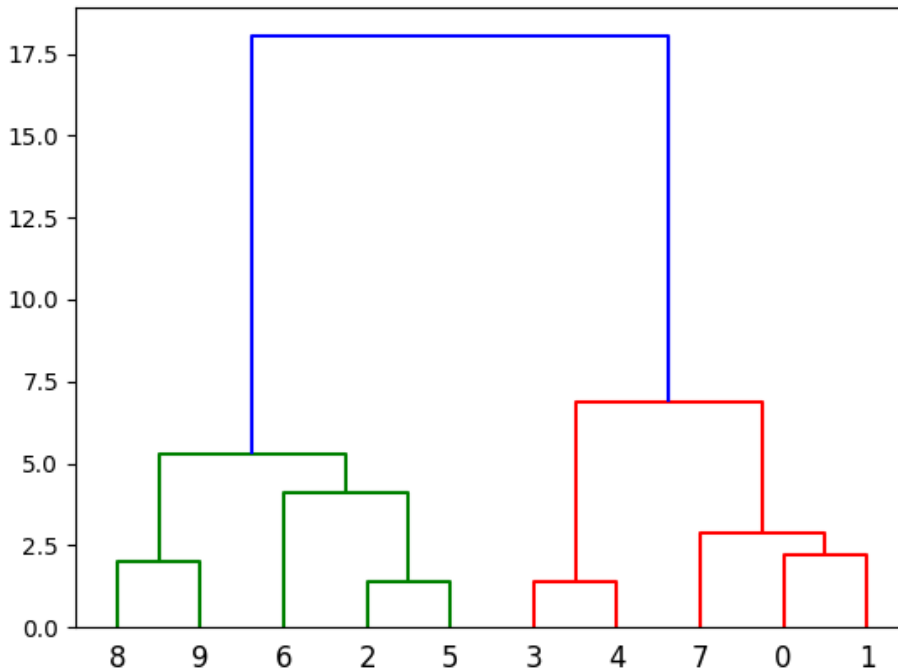
```
linkage_data = linkage(data, method='ward', metric='euclidean')
```

Finally, plot the results in a dendrogram. This plot will show us the hierarchy of clusters from the bottom (individual points) to the top (a single cluster consisting of all data points).

plt.show() lets us visualize the dendrogram instead of just the raw linkage data.

```
dendrogram(linkage_data)
plt.show()
```

Result:



The scikit-learn library allows us to use hierarchical clustering in a different manner. First, we initialize the `AgglomerativeClustering` class with 2 clusters, using the same euclidean distance and Ward linkage.

```
hierarchical_cluster = AgglomerativeClustering(n_clusters=2, affinity='euclidean', linkage='ward')
```

The `.fit_predict` method can be called on our data to compute the clusters using the defined parameters across our chosen number of clusters.

```
labels = hierarchical_cluster.fit_predict(data) print(labels)
```

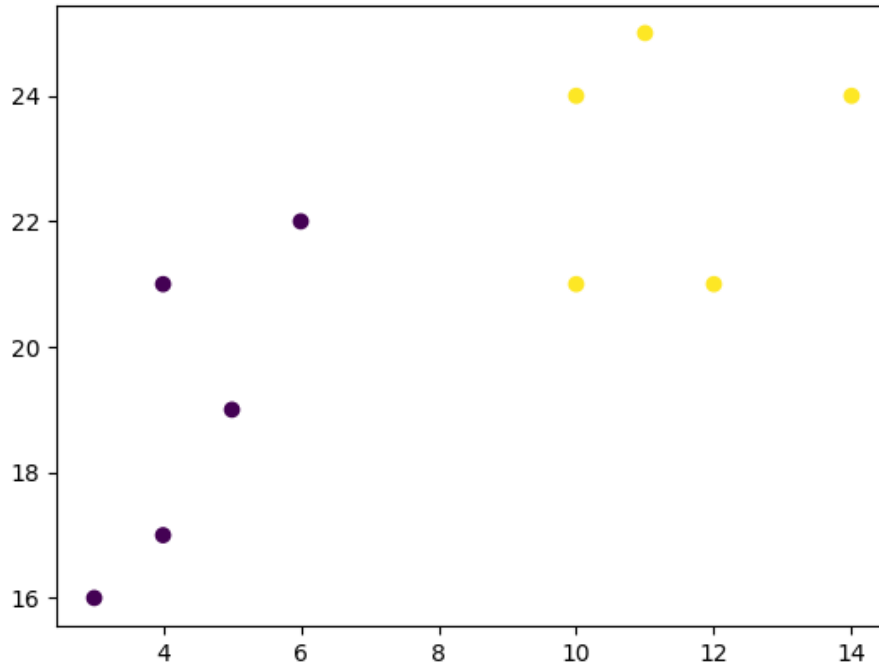
Result:

```
[0 0 1 0 0 1 1 0 1 1]
```

Finally, if we plot the same data and color the points using the labels assigned to each index by the hierarchical clustering method, we can see the cluster each point was assigned to:

```
plt.scatter(x, y, c=labels)
plt.show()
```

Result:



## Logistic Regression

Logistic regression aims to solve classification problems. It does this by predicting categorical outcomes, unlike linear regression that predicts a continuous outcome.

In the simplest case there are two outcomes, which is called binomial, an example of which is predicting if a tumor is malignant or benign. Other cases have more than two outcomes to classify, in this case it is called multinomial. A common example for multinomial logistic regression would be predicting the class of an iris flower between 3 different species.

Here we will be using basic logistic regression to predict a binomial variable. This means it has only two possible outcomes.

## How does it work?

In Python we have modules that will do the work for us. Start by importing the NumPy module.

```
import numpy
```

Store the independent variables in X.

Store the dependent variable in y.

Below is a sample dataset:

```
#X represents the size of a tumor in centimeters.  
X =  
numpy.array([3.78, 2.44, 2.09, 0.14, 1.72, 1.65, 4.92, 4.37, 4.96, 4.52, 3.69, 5.88]).r  
eshape(-1,1)
```

#Note: X has to be reshaped into a column from a row for the LogisticRegression() function to work.

```
#y represents whether or not the tumor is cancerous (0 for "No", 1 for "Yes").  
y = numpy.array([0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1])
```

We will use a method from the sklearn module, so we will have to import that module as well:

```
from sklearn import linear_model
```

From the sklearn module we will use the LogisticRegression() method to create a logistic regression object.

This object has a method called fit() that takes the independent and dependent values as parameters and fills the regression object with data that describes the relationship:

```
logr = linear_model.LogisticRegression()  
logr.fit(X,y)
```

Now we have a logistic regression object that is ready to whether a tumor is cancerous based on the tumor size:

```
#predict if tumor is cancerous where the size is 3.46mm:  
predicted = logr.predict(numpy.array([3.46]).reshape(-1,1))
```

Example

See the whole example in action:

```
import numpy  
from sklearn import linear_model
```

#Reshaped for Logistic function.

```
X =  
numpy.array([3.78, 2.44, 2.09, 0.14, 1.72, 1.65, 4.92, 4.37, 4.96, 4.52, 3.69, 5.88]).r
```

```

reshape(-1,1)
y = numpy.array([0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1])

logr = linear_model.LogisticRegression()
logr.fit(X,y)

#predict if tumor is cancerous where the size is 3.46mm:
predicted = logr.predict(numpy.array([3.46]).reshape(-1,1))
print(predicted)

```

Result

[0]

## Coefficient

In logistic regression the coefficient is the expected change in log-odds of having the outcome per unit change in X.

This does not have the most intuitive understanding so let's use it to create something that makes more sense, odds.

Example

See the whole example in action:

```

import numpy
from sklearn import linear_model

#Reshaped for Logistic function.
X =
numpy.array([3.78, 2.44, 2.09, 0.14, 1.72, 1.65, 4.92, 4.37, 4.96, 4.52, 3.69, 5.88]).r
eshape(-1,1)
y = numpy.array([0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1])

logr = linear_model.LogisticRegression()
logr.fit(X,y)

log_odds = logr.coef_
odds = numpy.exp(log_odds)

print(odds)

```

Result

[4.03541657]

[Run example »](#)

This tells us that as the size of a tumor increases by 1mm the odds of it being a tumor increases by 4x.

## Probability

The coefficient and intercept values can be used to find the probability that each tumor is cancerous.

Create a function that uses the model's coefficient and intercept values to return a new value. This new value represents probability that the given observation is a tumor:

```
def logit2prob(logr,x):  
    log_odds = logr.coef_ * x + logr.intercept_  
    odds = numpy.exp(log_odds)  
    probability = odds / (1 + odds)  
    return(probability)
```

## Function Explained

To find the log-odds for each observation, we must first create a formula that looks similar to the one from linear regression, extracting the coefficient and the intercept.

```
log_odds = logr.coef_ * x + logr.intercept_
```

To then convert the log-odds to odds we must exponentiate the log-odds.

```
odds = numpy.exp(log_odds)
```

Now that we have the odds, we can convert it to probability by dividing it by 1 plus the odds.

```
probability = odds / (1 + odds)
```

Let us now use the function with what we have learned to find out the probability that each tumor is cancerous.

## Example

See the whole example in action:

```

import numpy
from sklearn import linear_model

X =
numpy.array([3.78, 2.44, 2.09, 0.14, 1.72, 1.65, 4.92, 4.37, 4.96, 4.52, 3.69, 5.88]).r
eshape(-1,1)
y = numpy.array([0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1])

logr = linear_model.LogisticRegression()
logr.fit(X,y)

def logit2prob(logr, X):
    log_odds = logr.coef_ * X + logr.intercept_
    odds = numpy.exp(log_odds)
    probability = odds / (1 + odds)
    return(probability)

print(logit2prob(logr, X))

```

Result

```

[[0.60749955]
 [0.19268876]
 [0.12775886]
 [0.00955221]
 [0.08038616]
 [0.07345637]
 [0.88362743]
 [0.77901378]
 [0.88924409]
 [0.81293497]
 [0.57719129]
 [0.96664243]]

```

[Run example »](#)

Results Explained

3.78 0.61 The probability that a tumor with the size 3.78cm is cancerous is 61%.

2.44 0.19 The probability that a tumor with the size 2.44cm is cancerous is 19%.

2.09 0.13 The probability that a tumor with the size 2.09cm is cancerous is 13%.

**Grid Search**



The majority of machine learning models contain parameters that can be adjusted to vary how the model learns. For example, the logistic regression model, from sklearn, has a parameter C that controls regularization, which affects the complexity of the model.

How do we pick the best value for C? The best value is dependent on the data used to train the model.

### **How does it work?**

One method is to try out different values and then pick the value that gives the best score. This technique is known as a **grid search**. If we had to select the values for two or more parameters, we would evaluate all combinations of the sets of values thus forming a grid of values.

Before we get into the example it is good to know what the parameter we are changing does. Higher values of C tell the model, the training data resembles real world information, place a greater weight on the training data. While lower values of C do the opposite.

### **Using Default Parameters**

First let's see what kind of results we can generate without a grid search using only the base parameters.

To get started we must first load in the dataset we will be working with.

```
from sklearn import datasets
iris = datasets.load_iris()
```

Next in order to create the model we must have a set of independent variables X and a dependant variable y.

```
X = iris['data']
y = iris['target']
```

Now we will load the logistic model for classifying the iris flowers.

```
from sklearn.linear_model import LogisticRegression
```

Creating the model, setting max\_iter to a higher value to ensure that the model finds a result.

Keep in mind the default value for C in a logistic regression model is 1, we will compare this later.

In the example below, we look at the iris data set and try to train a model with varying values for C in logistic regression.

```
logit = LogisticRegression(max_iter = 10000)
```

After we create the model, we must fit the model to the data.

```
print(logit.fit(X,y))
```

To evaluate the model we run the score method.

```
print(logit.score(X,y))
```

Example

```
from sklearn import datasets
from sklearn.linear_model import LogisticRegression
```

```
iris = datasets.load_iris()
```

```
X = iris['data']
y = iris['target']
```

```
logit = LogisticRegression(max_iter = 10000)
```

```
print(logit.fit(X,y))
```

```
print(logit.score(X,y))
```

[Run example »](#)

With the default setting of  $C = 1$ , we achieved a score of 0.973.

Let's see if we can do any better by implementing a grid search with different values of 0.973.

## Implementing Grid Search

We will follow the same steps of before except this time we will set a range of values for C.

Knowing which values to set for the searched parameters will take a combination of domain knowledge and practice.

Since the default value for C is 1, we will set a range of values surrounding it.

```
C = [0.25, 0.5, 0.75, 1, 1.25, 1.5, 1.75, 2]
```

Next we will create a for loop to change out the values of C and evaluate the model with each change.

First we will create an empty list to store the score within.

```
scores = []
```

To change the values of C we must loop over the range of values and update the parameter each time.

```
for choice in C:  
    logit.set_params(C=choice)  
    logit.fit(X, y)  
    scores.append(logit.score(X, y))
```

With the scores stored in a list, we can evaluate what the best choice of C is.

```
print(scores)
```

Example

```
from sklearn import datasets  
from sklearn.linear_model import LogisticRegression
```

```
iris = datasets.load_iris()
```

```
X = iris['data']  
y = iris['target']
```

```
logit = LogisticRegression(max_iter = 10000)
```

```
C = [0.25, 0.5, 0.75, 1, 1.25, 1.5, 1.75, 2]
```

```
scores = []
```

```
for choice in C:  
    logit.set_params(C=choice)  
    logit.fit(X, y)  
    scores.append(logit.score(X, y))
```

```
print(scores)
```

[Run example »](#)

## Results Explained

We can see that the lower values of C performed worse than the base parameter of 1. However, as we increased the value of C to 1.75 the model experienced increased accuracy.

It seems that increasing C beyond this amount does not help increase model accuracy.

## Note on Best Practices

We scored our logistic regression model by using the same data that was used to train it. If the model corresponds too closely to that data, it may not be great at predicting unseen data. This statistical error is known as **over fitting**.

To avoid being misled by the scores on the training data, we can put aside a portion of our data and use it specifically for the purpose of testing the model. Refer to the lecture on train/test splitting to avoid being misled and overfitting.

## Categorical Data

When your data has categories represented by strings, it will be difficult to use them to train machine learning models which often only accepts numeric data.

Instead of ignoring the categorical data and excluding the information from our model, you can transform the data so it can be used in your models.

Take a look at the table below, it is the same data set that we used in the multiple regression chapter.

### Example

```
import pandas as pd
```

```
cars = pd.read_csv('data.csv')  
print(cars.to_string())
```

### Result

	Car	Model	Volume	Weight	CO2
0	Toyoty	Aygo	1000	790	99
1	Mitsubishi	Space Star	1200	1160	95
2	Skoda	Citigo	1000	929	95
3	Fiat	500	900	865	90
4	Mini	Cooper	1500	1140	105
5	VW	Up!	1000	929	105
6	Skoda	Fabia	1400	1109	90

7	Mercedes	A-Class	1500	1365	92
8	Ford	Fiesta	1500	1112	98
9	Audi	A1	1600	1150	99
10	Hyundai	I20	1100	980	99
11	Suzuki	Swift	1300	990	101
12	Ford	Fiesta	1000	1112	99
13	Honda	Civic	1600	1252	94
14	Hundai	I30	1600	1326	97
15	Opel	Astra	1600	1330	97
16	BMW	1	1600	1365	99
17	Mazda	3	2200	1280	104
18	Skoda	Rapid	1600	1119	104
19	Ford	Focus	2000	1328	105
20	Ford	Mondeo	1600	1584	94
21	Opel	Insignia	2000	1428	99
22	Mercedes	C-Class	2100	1365	99
23	Skoda	Octavia	1600	1415	99
24	Volvo	S60	2000	1415	99
25	Mercedes	CLA	1500	1465	102
26	Audi	A4	2000	1490	104
27	Audi	A6	2000	1725	114
28	Volvo	V70	1600	1523	109
29	BMW	5	2000	1705	114
30	Mercedes	E-Class	2100	1605	115
31	Volvo	XC70	2000	1746	117
32	Ford	B-Max	1600	1235	104
33	BMW	216	1600	1390	108
34	Opel	Zafira	1600	1405	109
35	Mercedes	SLK	2500	1395	120

[Run example »](#)

In the multiple regression chapter, we tried to predict the CO2 emitted based on the volume of the engine and the weight of the car but we excluded information about the car brand and model.

The information about the car brand or the car model might help us make a better prediction of the CO2 emitted.

## One Hot Encoding

We cannot make use of the Car or Model column in our data since they are not numeric. A linear relationship between a categorical variable, Car or Model, and a numeric variable, CO2, cannot be determined.

To fix this issue, we must have a numeric representation of the categorical variable. One way to do this is to have a column representing each group in the category.

For each column, the values will be 1 or 0 where 1 represents the inclusion of the group and 0 represents the exclusion. This transformation is called one hot encoding.

You do not have to do this manually, the Python Pandas module has a function that called `get_dummies()` which does one hot encoding.

Learn about the Pandas module in our [Pandas Tutorial](#).

### Example

### One Hot Encode the Car column:

```
import pandas as pd
```

```
cars = pd.read_csv('data.csv')
ohe_cars = pd.get_dummies(cars[['Car']])
```

```
print(ohe_cars.to_string())
```

## Result

	Car_Audi	Car_BMW	Car_Fiat	Car_Ford	Car_Honda	Car_Hundai				
Car_Hyundai	Car_Mazda	Car_Mercedes	Car_Mini	Car_Mitsubishi	Car_Opel					
Car_Skoda	Car_Suzuki	Car_Toyoty	Car_VW	Car_Volvo						
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0				
1	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0				
2	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0				
3	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0				
4	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0				
5	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0				
6	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0				

7	0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0				
8	0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0				
9	1	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0				
10	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0				
11	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0				
12	0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0				
13	0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0				
14	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0				
15	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0				
16	0	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0				
17	0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0				
18	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0				
19	0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0				
20	0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0				
21	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0				
22	0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0				
23	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0				
24	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1				
25	0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0				
26	1	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0				
27	1	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0				
28	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1				
29	0	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0				

30	0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0				
31	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1				
32	0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0				
33	0	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0				
34	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0				
35	0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0				

[Run example »](#)

Results

A column was created for every car brand in the Car column.

## Predict CO2

We can use this additional information alongside the volume and weight to predict CO2

To combine the information, we can use the `concat()` function from pandas.

First we will need to import a couple modules.

We will start with importing the Pandas.

```
import pandas
```

The pandas module allows us to read csv files and manipulate DataFrame objects:

```
cars = pandas.read_csv("data.csv")
```

It also allows us to create the dummy variables:

```
ohe_cars = pandas.get_dummies(cars[['Car']])
```

Then we must select the independent variables (X) and add the dummy variables columnwise.

Also store the dependent variable in y.



```
X = pandas.concat([cars[['Volume', 'Weight']], ohe_cars], axis=1)
y = cars['CO2']
```

We also need to import a method from sklearn to create a linear model

Learn about [linear regression](#).

```
from sklearn import linear_model
```

Now we can fit the data to a linear regression:

```
regr = linear_model.LinearRegression()
regr.fit(X,y)
```

Finally we can predict the CO2 emissions based on the car's weight, volume, and manufacturer.

```
##predict the CO2 emission of a Volvo where the weight is 2300kg, and the
volume is 1300cm3:
predictedCO2 = regr.predict([[2300, 1300,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0]])
```

Example

```
import pandas
from sklearn import linear_model

cars = pandas.read_csv("data.csv")
ohe_cars = pandas.get_dummies(cars[['Car']])
```

```
X = pandas.concat([cars[['Volume', 'Weight']], ohe_cars], axis=1)
y = cars['CO2']
```

```
regr = linear_model.LinearRegression()
regr.fit(X,y)
```

```
##predict the CO2 emission of a Volvo where the weight is 2300kg, and the
volume is 1300cm3:
predictedCO2 = regr.predict([[2300, 1300,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0]])
```

```
print(predictedCO2)
```

Result

```
[122.45153299]
```

[Run example »](#)

We now have a coefficient for the volume, the weight, and each car brand in the data set

## Dummifying

It is not necessary to create one column for each group in your category. The information can be retained using 1 column less than the number of groups you have.

For example, you have a column representing colors and in that column, you have two colors, red and blue.

Example

```
import pandas as pd
```

```
colors = pd.DataFrame({'color': ['blue', 'red']})
```

```
print(colors)
```

Result

```
  color
0  blue
1   red
```

[Run example »](#)

You can create 1 column called red where 1 represents red and 0 represents not red, which means it is blue.

To do this, we can use the same function that we used for one hot encoding, `get_dummies`, and then drop one of the columns. There is an argument, `drop_first`, which allows us to exclude the first column from the resulting table.

Example

```
import pandas as pd
```

```
colors = pd.DataFrame({'color': ['blue', 'red']})
dummies = pd.get_dummies(colors, drop_first=True)
```

```
print(dummies)
```

Result

```
  color_red
0         0
```

1 1

[Run example »](#)

What if you have more than 2 groups? How can the multiple groups be represented by 1 less column?

Let's say we have three colors this time, red, blue and green. When we get\_dummies while dropping the first column, we get the following table.

Example

```
import pandas as pd

colors = pd.DataFrame({'color': ['blue', 'red', 'green']})
dummies = pd.get_dummies(colors, drop_first=True)
dummies['color'] = colors['color']

print(dummies)
```

Result

	color_green	color_red	color
0	0	0	blue
1	0	1	red
2	1	0	green

K-means

K-means is an unsupervised learning method for clustering data points. The algorithm iteratively divides data points into K clusters by minimizing the variance in each cluster.

Here, we will show you how to estimate the best value for K using the elbow method, then use K-means clustering to group the data points into clusters.

### How does it work?

First, each data point is randomly assigned to one of the K clusters. Then, we compute the centroid (functionally the center) of each cluster, and reassign each data point to the cluster with the closest centroid. We repeat this process until the cluster assignments for each data point are no longer changing.

K-means clustering requires us to select K, the number of clusters we want to group the data into. The elbow method lets us graph the inertia (a distance-based metric) and visualize the point at which it starts decreasing linearly. This point is referred to as the "elbow" and is a good estimate for the best value for K based on our data.

### Example

Start by visualizing some data points:

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

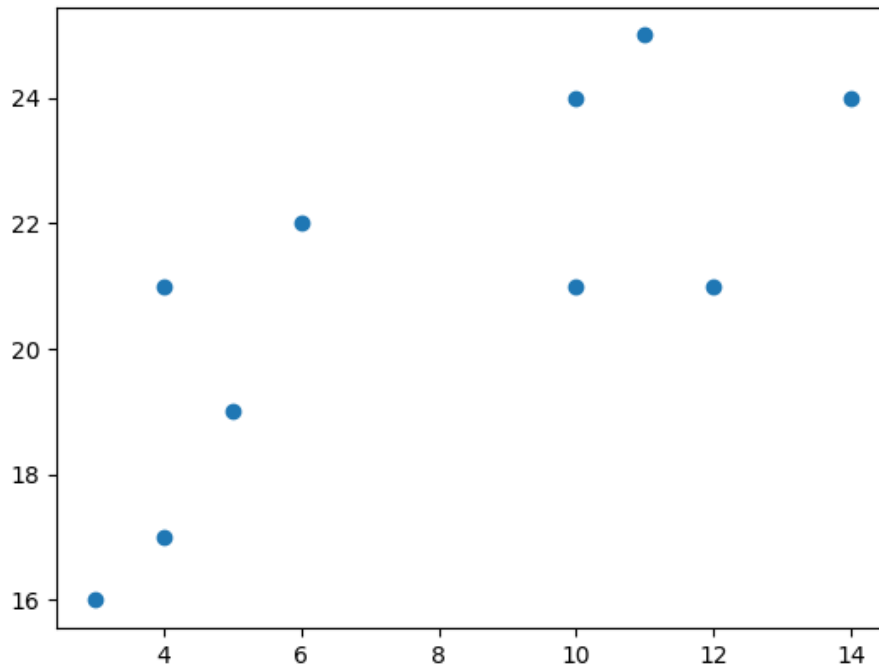
```
x = [4, 5, 10, 4, 3, 11, 14, 6, 10, 12]
```

```
y = [21, 19, 24, 17, 16, 25, 24, 22, 21, 21]
```

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

```
plt.show()
```

Result



Now we utilize the elbow method to visualize the inertia for different values of K:

### Example

```
from sklearn.cluster import KMeans
```

```
data = list(zip(x, y))
```

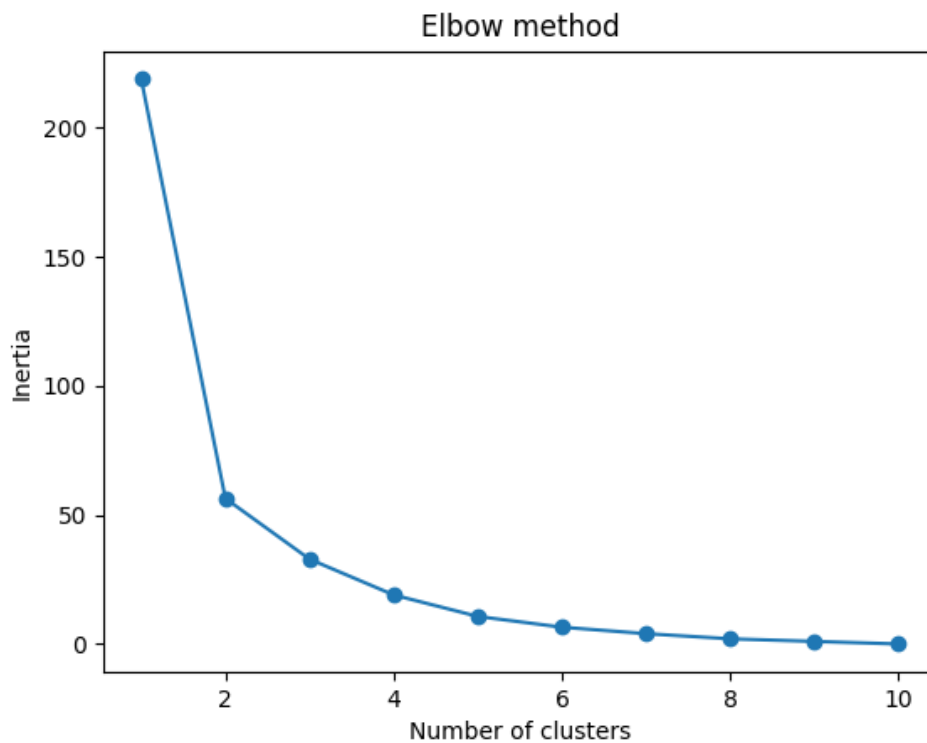
```
inertias = []
```

```
for i in range(1,11):
```

```
kmeans = KMeans(n_clusters=i)
kmeans.fit(data)
inertias.append(kmeans.inertia_)
```

```
plt.plot(range(1,11), inertias, marker='o')
plt.title('Elbow method')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.show()
```

Result



[Run example »](#)

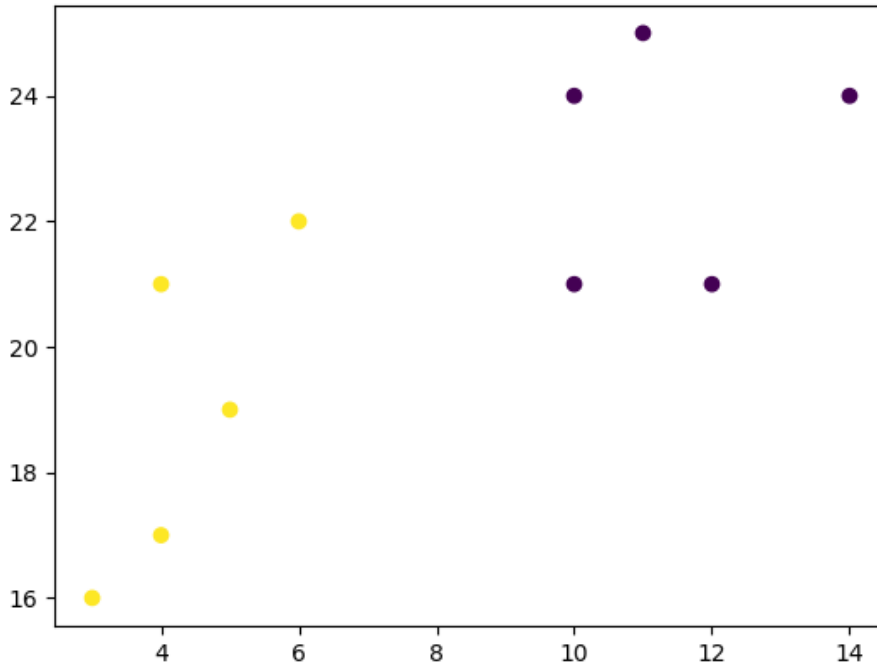
The elbow method shows that 2 is a good value for K, so we retrain and visualize the result:

Example

```
kmeans = KMeans(n_clusters=2)
kmeans.fit(data)
```

```
plt.scatter(x, y, c=kmeans.labels_)
plt.show()
```

## Result



[Run example »](#)

### Example Explained

Import the modules you need.

```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
```

You can learn about the Matplotlib module in our ["Matplotlib Tutorial"](#).

scikit-learn is a popular library for machine learning.

Create arrays that resemble two variables in a dataset. Note that while we only use two variables here, this method will work with any number of variables:

```
x = [4, 5, 10, 4, 3, 11, 14, 6, 10, 12]
y = [21, 19, 24, 17, 16, 25, 24, 22, 21, 21]
```

Turn the data into a set of points:

```
data = list(zip(x, y))
print(data)
```

Result:

```
[(4, 21), (5, 19), (10, 24), (4, 17), (3, 16), (11, 25), (14, 24), (6, 22), (10, 21),
(12, 21)]
```

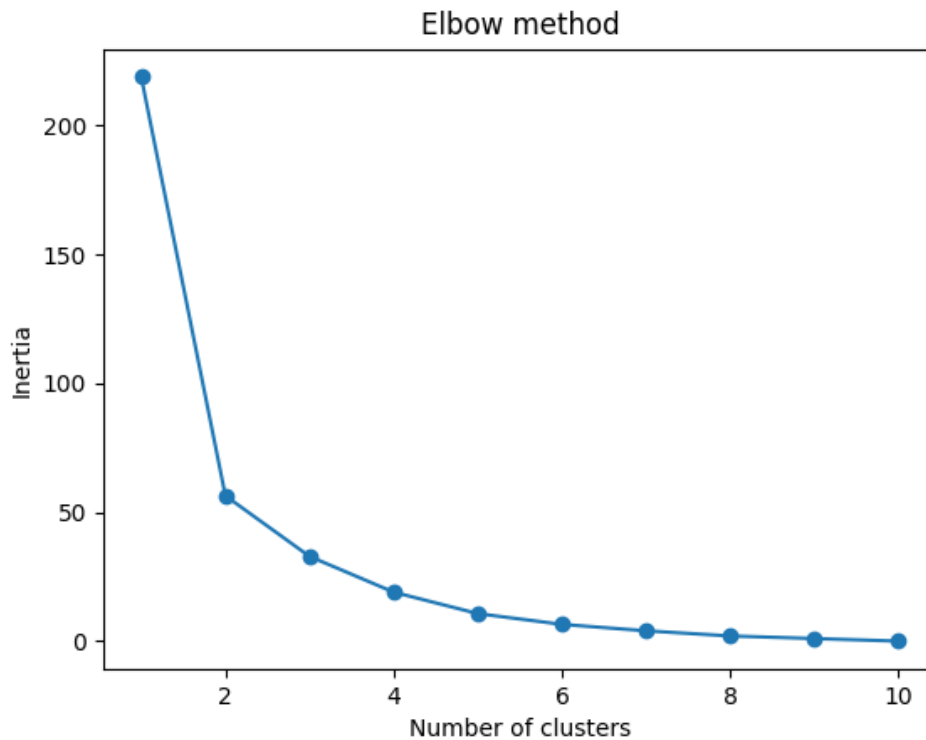
In order to find the best value for K, we need to run K-means across our data for a range of possible values. We only have 10 data points, so the maximum number of clusters is 10. So for each value K in range(1,11), we train a K-means model and plot the inertia at that number of clusters:

```
inertias = []

for i in range(1,11):
    kmeans = KMeans(n_clusters=i)
    kmeans.fit(data)
    inertias.append(kmeans.inertia_)

plt.plot(range(1,11), inertias, marker='o')
plt.title('Elbow method')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.show()
```

Result:



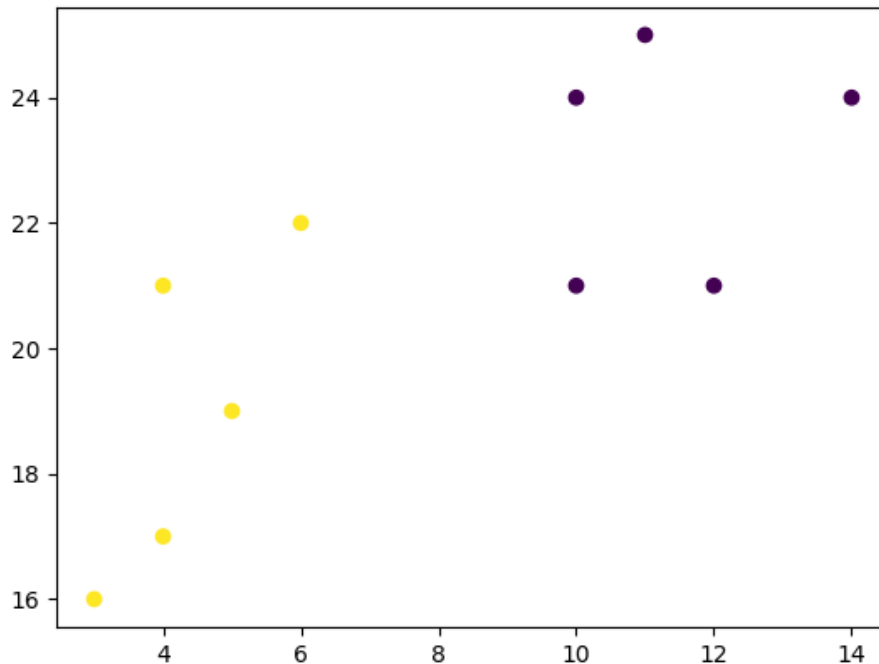
We can see that the "elbow" on the graph above (where the inertia becomes more linear) is at  $K=2$ . We can then fit our K-means algorithm one more time and plot the different clusters assigned to the data:

```
kmeans = KMeans(n_clusters=2)
kmeans.fit(data)

plt.scatter(x, y, c=kmeans.labels_)
plt.show()
```

Result:





## Machine Learning - Bootstrap Aggregation (Bagging)

### Bagging

Methods such as Decision Trees, can be prone to overfitting on the training set which can lead to wrong predictions on new data.

Bootstrap Aggregation (bagging) is an ensembling method that attempts to resolve overfitting for classification or regression problems. Bagging aims to improve the accuracy and performance of machine learning algorithms. It does this by taking random subsets of an original dataset, with replacement, and fits either a classifier (for classification) or regressor (for regression) to each subset. The predictions for each subset are then aggregated through majority vote for classification or averaging for regression, increasing prediction accuracy.

### Evaluating a Base Classifier

To see how bagging can improve model performance, we must start by evaluating how the base classifier performs on the dataset. If you do not know what decision trees are review the lesson on decision trees before moving forward, as bagging is an continuation of the concept.

We will be looking to identify different classes of wines found in Sklearn's wine dataset.

Let's start by importing the necessary modules.

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier
```

Next we need to load in the data and store it into X (input features) and y (target). The parameter `as_frame` is set equal to `True` so we do not lose the feature names when loading the data. (sklearn version older than 0.23 must skip the `as_frame` argument as it is not supported)

```
data = datasets.load_wine(as_frame = True)
```

```
X = data.data
y = data.target
```

In order to properly evaluate our model on unseen data, we need to split X and y into train and test sets. For information on splitting data, see the Train/Test lesson.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25,
random_state = 22)
```

With our data prepared, we can now instantiate a base classifier and fit it to the training data.

```
dtree = DecisionTreeClassifier(random_state = 22)
dtree.fit(X_train,y_train)
```

Result:

```
DecisionTreeClassifier(random_state=22)
```

We can now predict the class of wine the unseen test set and evaluate the model performance.

```
y_pred = dtree.predict(X_test)
```

```
print("Train data accuracy:",accuracy_score(y_true = y_train, y_pred =
dtree.predict(X_train)))
print("Test data accuracy:",accuracy_score(y_true = y_test, y_pred = y_pred))
```

Result:

```
Train data accuracy: 1.0
Test data accuracy: 0.8222222222222222
```

## Example

Import the necessary data and evaluate base classifier performance.

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier

data = datasets.load_wine(as_frame = True)

X = data.data
y = data.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25,
random_state = 22)

dtree = DecisionTreeClassifier(random_state = 22)
dtree.fit(X_train,y_train)

y_pred = dtree.predict(X_test)

print("Train data accuracy:",accuracy_score(y_true = y_train, y_pred =
dtree.predict(X_train)))
print("Test data accuracy:",accuracy_score(y_true = y_test, y_pred = y_pred))
```

[Run example »](#)

The base classifier performs reasonably well on the dataset achieving 82% accuracy on the test dataset with the current parameters (Different results may occur if you do not have the random\_state parameter set).

Now that we have a baseline accuracy for the test dataset, we can see how the Bagging Classifier out performs a single Decision Tree Classifier.

## Creating a Bagging Classifier

For bagging we need to set the parameter n\_estimators, this is the number of base classifiers that our model is going to aggregate together.

For this sample dataset the number of estimators is relatively low, it is often the case that much larger ranges are explored. Hyperparameter tuning is usually done with a grid search, but for now we will use a select set of values for the number of estimators.

We start by importing the necessary model.

```
from sklearn.ensemble import BaggingClassifier
```

Now lets create a range of values that represent the number of estimators we want to use in each ensemble.

```
estimator_range = [2,4,6,8,10,12,14,16]
```

To see how the Bagging Classifier performs with differing values of `n_estimators` we need a way to iterate over the range of values and store the results from each ensemble. To do this we will create a for loop, storing the models and scores in separate lists for later vizualizations.

Note: The default parameter for the base classifier in `BaggingClassifier` is the `DecisionTreeClassifier` therefore we do not need to set it when instantiating the bagging model.

```
models = []  
scores = []
```

for `n_estimators` in `estimator_range`:

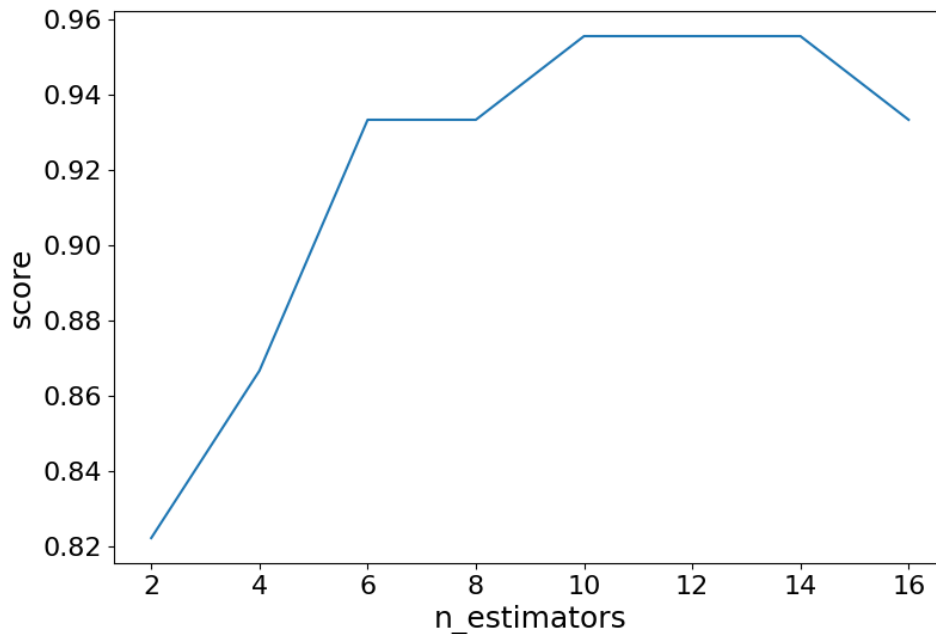
```
    # Create bagging classifier  
    clf = BaggingClassifier(n_estimators = n_estimators, random_state = 22)  
  
    # Fit the model  
    clf.fit(X_train, y_train)  
  
    # Append the model and score to their respective list  
    models.append(clf)  
    scores.append(accuracy_score(y_true = y_test, y_pred = clf.predict(X_test)))
```

With the models and scores stored, we can now visualize the improvement in model performance.

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

```
# Generate the plot of scores against number of estimators  
plt.figure(figsize=(9,6))  
plt.plot(estimator_range, scores)  
  
# Adjust labels and font (to make visable)  
plt.xlabel("n_estimators", fontsize = 18)  
plt.ylabel("score", fontsize = 18)  
plt.tick_params(labelsize = 16)
```

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



Example

Import the necessary data and evaluate the BaggingClassifier performance.

```
import matplotlib.pyplot as plt  
from sklearn import datasets  
from sklearn.model_selection import train_test_split  
from sklearn.metrics import accuracy_score  
from sklearn.ensemble import BaggingClassifier
```

```
data = datasets.load_wine(as_frame = True)
```

```
X = data.data  
y = data.target
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25,  
random_state = 22)
```

```
estimator_range = [2,4,6,8,10,12,14,16]
```

```
models = []
```

```
scores = []

for n_estimators in estimator_range:

    # Create bagging classifier
    clf = BaggingClassifier(n_estimators = n_estimators, random_state = 22)

    # Fit the model
    clf.fit(X_train, y_train)

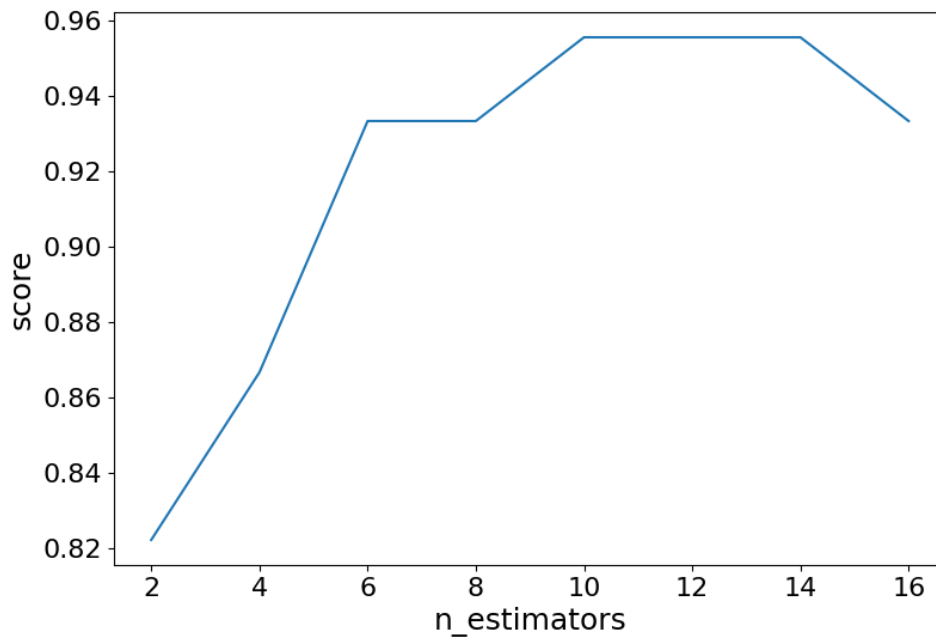
    # Append the model and score to their respective list
    models.append(clf)
    scores.append(accuracy_score(y_true = y_test, y_pred = clf.predict(X_test)))

# Generate the plot of scores against number of estimators
plt.figure(figsize=(9,6))
plt.plot(estimator_range, scores)

# Adjust labels and font (to make visible)
plt.xlabel("n_estimators", fontsize = 18)
plt.ylabel("score", fontsize = 18)
plt.tick_params(labelsize = 16)

# Visualize plot
plt.show()

Result
```



[Run example »](#)

## Results Explained

By iterating through different values for the number of estimators we can see an increase in model performance from 82.2% to 95.5%. After 14 estimators the accuracy begins to drop, again if you set a different `random_state` the values you see will vary. That is why it is best practice to use cross validation to ensure stable results.

In this case, we see a 13.3% increase in accuracy when it comes to identifying the type of the wine.

## Another Form of Evaluation

As bootstrapping chooses random subsets of observations to create classifiers, there are observations that are left out in the selection process. These "out-of-bag" observations can then be used to evaluate the model, similarly to that of a test set. Keep in mind, that out-of-bag estimation can overestimate error in binary classification problems and should only be used as a compliment to other metrics.

We saw in the last exercise that 12 estimators yielded the highest accuracy, so we will use that to create our model. This time setting the parameter `oob_score` to `true` to evaluate the model with out-of-bag score.

### Example

Create a model with out-of-bag metric.

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.ensemble import BaggingClassifier

data = datasets.load_wine(as_frame = True)

X = data.data
y = data.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25,
random_state = 22)

oob_model = BaggingClassifier(n_estimators = 12, oob_score = True, random_state
= 22)

oob_model.fit(X_train, y_train)

print(oob_model.oob_score_)
```

[Run example »](#)

Since the samples used in OOB and the test set are different, and the dataset is relatively small, there is a difference in the accuracy. It is rare that they would be exactly the same, again OOB should be used quick means for estimating error, but is not the only evaluation metric.

### Generating Decision Trees from Bagging Classifier

As was seen in the [Decision Tree](#) lesson, it is possible to graph the decision tree the model created. It is also possible to see the individual decision trees that went into the aggregated classifier. This helps us to gain a more intuitive understanding on how the bagging model arrives at its predictions.

Note: This is only functional with smaller datasets, where the trees are relatively shallow and narrow making them easy to visualize.



We will need to import plot\_tree function from sklearn.tree. The different trees can be graphed by changing the estimator you wish to visualize.

## Example

### Generate Decision Trees from Bagging Classifier

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import plot_tree
```

```
X = data.data
y = data.target
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25,
random_state = 22)
```

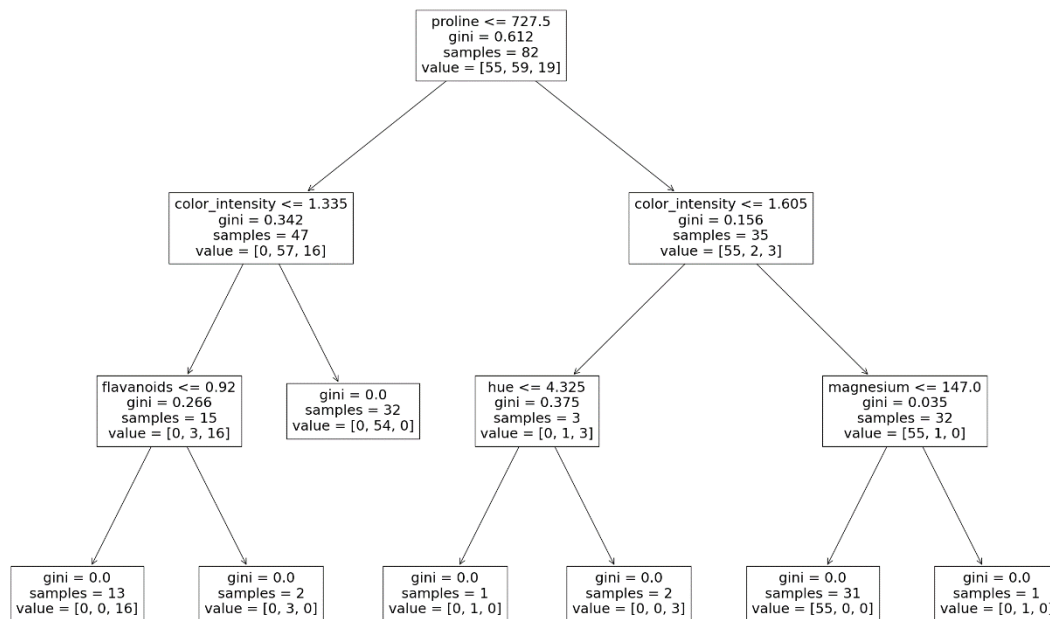
```
clf = BaggingClassifier(n_estimators = 12, oob_score = True, random_state = 22)
```

```
clf.fit(X_train, y_train)
```

```
plt.figure(figsize=(30, 20))
```

```
plot_tree(clf.estimators_[0], feature_names = X.columns)
```

## Result



[Run example »](#)

## Cross Validation

When adjusting models we are aiming to increase overall model performance on unseen data. Hyperparameter tuning can lead to much better performance on test sets. However, optimizing parameters to the test set can lead information leakage causing the model to perform worse on unseen data. To correct for this we can perform cross validation.

To better understand CV, we will be performing different methods on the iris dataset. Let us first load in and separate the data.

```
from sklearn import datasets
```

```
X, y = datasets.load_iris(return_X_y=True)
```

There are many methods to cross validation, we will start by looking at k-fold cross validation.

## K-Fold

The training data used in the model is split, into k number of smaller sets, to be used to validate the model. The model is then trained on k-1 folds of training set. The remaining fold is then used as a validation set to evaluate the model.

As we will be trying to classify different species of iris flowers we will need to import a classifier model, for this exercise we will be using a `DecisionTreeClassifier`. We will also need to import CV modules from sklearn.

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import KFold, cross_val_score
```

With the data loaded we can now create and fit a model for evaluation.

```
clf = DecisionTreeClassifier(random_state=42)
```

Now let's evaluate our model and see how it performs on each *k*-fold.

```
k_folds = KFold(n_splits = 5)
```

```
scores = cross_val_score(clf, X, y, cv = k_folds)
```

It is also good practice to see how CV performed overall by averaging the scores for all folds.

#### Example

Run k-fold CV:

```
from sklearn import datasets
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import KFold, cross_val_score

X, y = datasets.load_iris(return_X_y=True)

clf = DecisionTreeClassifier(random_state=42)

k_folds = KFold(n_splits = 5)

scores = cross_val_score(clf, X, y, cv = k_folds)

print("Cross Validation Scores: ", scores)
print("Average CV Score: ", scores.mean())
print("Number of CV Scores used in Average: ", len(scores))
```

#### Stratified K-Fold

In cases where classes are imbalanced we need a way to account for the imbalance in both the train and validation sets. To do so we can stratify the target classes, meaning that both sets will have an equal proportion of all classes.

#### Example

```
from sklearn import datasets
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import StratifiedKFold, cross_val_score

X, y = datasets.load_iris(return_X_y=True)

clf = DecisionTreeClassifier(random_state=42)

sk_folds = StratifiedKFold(n_splits = 5)

scores = cross_val_score(clf, X, y, cv = sk_folds)

print("Cross Validation Scores: ", scores)
print("Average CV Score: ", scores.mean())
print("Number of CV Scores used in Average: ", len(scores))
```

### [Run example »](#)

While the number of folds is the same, the average CV increases from the basic k-fold when making sure there is stratified classes.

## Leave-One-Out (LOO)

Instead of selecting the number of splits in the training data set like k-fold LeaveOneOut, utilize 1 observation to validate and n-1 observations to train. This method is an exhaustive technique.

### Example

Run LOO CV:

```
from sklearn import datasets
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import LeaveOneOut, cross_val_score

X, y = datasets.load_iris(return_X_y=True)

clf = DecisionTreeClassifier(random_state=42)

loo = LeaveOneOut()

scores = cross_val_score(clf, X, y, cv = loo)

print("Cross Validation Scores: ", scores)
print("Average CV Score: ", scores.mean())
print("Number of CV Scores used in Average: ", len(scores))
```

### [Run example »](#)

We can observe that the number of cross validation scores performed is equal to the number of observations in the dataset. In this case there are 150 observations in the iris dataset.

The average CV score is 94%.

## Leave-P-Out (LPO)

Leave-P-Out is simply a nuanced difference to the Leave-One-Out idea, in that we can select the number of  $p$  to use in our validation set.

### Example

Run LPO CV:

```
from sklearn import datasets
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import LeavePOut, cross_val_score

X, y = datasets.load_iris(return_X_y=True)

clf = DecisionTreeClassifier(random_state=42)

lpo = LeavePOut(p=2)

scores = cross_val_score(clf, X, y, cv = lpo)

print("Cross Validation Scores: ", scores)
print("Average CV Score: ", scores.mean())
print("Number of CV Scores used in Average: ", len(scores))
```

[Run example »](#)

As we can see this is an exhaustive method with many more scores being calculated than Leave-One-Out, even with a  $p = 2$ , yet it achieves roughly the same average CV score.

### Shuffle Split

Unlike KFold, ShuffleSplit leaves out a percentage of the data, not to be used in the train or validation sets. To do so we must decide what the train and test sizes are, as well as the number of splits.

### Example

Run Shuffle Split CV:

```
from sklearn import datasets
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import ShuffleSplit, cross_val_score
```

```

X, y = datasets.load_iris(return_X_y=True)

clf = DecisionTreeClassifier(random_state=42)

ss = ShuffleSplit(train_size=0.6, test_size=0.3, n_splits = 5)

scores = cross_val_score(clf, X, y, cv = ss)

print("Cross Validation Scores: ", scores)
print("Average CV Score: ", scores.mean())
print("Number of CV Scores used in Average: ", len(scores))

```

## AUC - ROC Curve

In classification, there are many different evaluation metrics. The most popular is **accuracy**, which measures how often the model is correct. This is a great metric because it is easy to understand and getting the most correct guesses is often desired. There are some cases where you might consider using another evaluation metric.

Another common metric is **AUC**, area under the receiver operating characteristic (**ROC**) curve. The Receiver operating characteristic curve plots the true positive (**TP**) rate versus the false positive (**FP**) rate at different classification thresholds. The thresholds are different probability cutoffs that separate the two classes in binary classification. It uses probability to tell us how well a model separates the classes.

## Imbalanced Data

Suppose we have an imbalanced data set where the majority of our data is of one value. We can obtain high accuracy for the model by predicting the majority class.

### Example

```

import numpy as np
from sklearn.metrics import accuracy_score, confusion_matrix, roc_auc_score,
roc_curve

n = 10000
ratio = .95
n_0 = int((1-ratio) * n)
n_1 = int(ratio * n)

y = np.array([0] * n_0 + [1] * n_1)
# below are the probabilities obtained from a hypothetical model that always

```

```
predicts the majority class
# probability of predicting class 1 is going to be 100%
y_proba = np.array([1]*n)
y_pred = y_proba > .5
```

```
print(f'accuracy score: {accuracy_score(y, y_pred)}')
cf_mat = confusion_matrix(y, y_pred)
print('Confusion matrix')
print(cf_mat)
print(f'class 0 accuracy: {cf_mat[0][0]/n_0}')
print(f'class 1 accuracy: {cf_mat[1][1]/n_1}')
```

[Run example »](#)

Although we obtain a very high accuracy, the model provided no information about the data so it's not useful. We accurately predict class 1 100% of the time while inaccurately predict class 0 0% of the time. At the expense of accuracy, it might be better to have a model that can somewhat separate the two classes.

#### Example

```
# below are the probabilities obtained from a hypothetical model that doesn't
always predict the mode
y_proba_2 = np.array(
    np.random.uniform(0, .7, n_0).tolist() +
    np.random.uniform(.3, 1, n_1).tolist()
)
y_pred_2 = y_proba_2 > .5

print(f'accuracy score: {accuracy_score(y, y_pred_2)}')
cf_mat = confusion_matrix(y, y_pred_2)
print('Confusion matrix')
print(cf_mat)
print(f'class 0 accuracy: {cf_mat[0][0]/n_0}')
print(f'class 1 accuracy: {cf_mat[1][1]/n_1}')
```

[Run example »](#)

For the second set of predictions, we do not have as high of an accuracy score as the first but the accuracy for each class is more balanced. Using accuracy as an evaluation metric we would rate the first model higher than the second even though it doesn't tell us anything about the data.

In cases like this, using another evaluation metric like AUC would be preferred.

```
import matplotlib.pyplot as plt

def plot_roc_curve(true_y, y_prob):
    """
    plots the roc curve based of the probabilities
    """

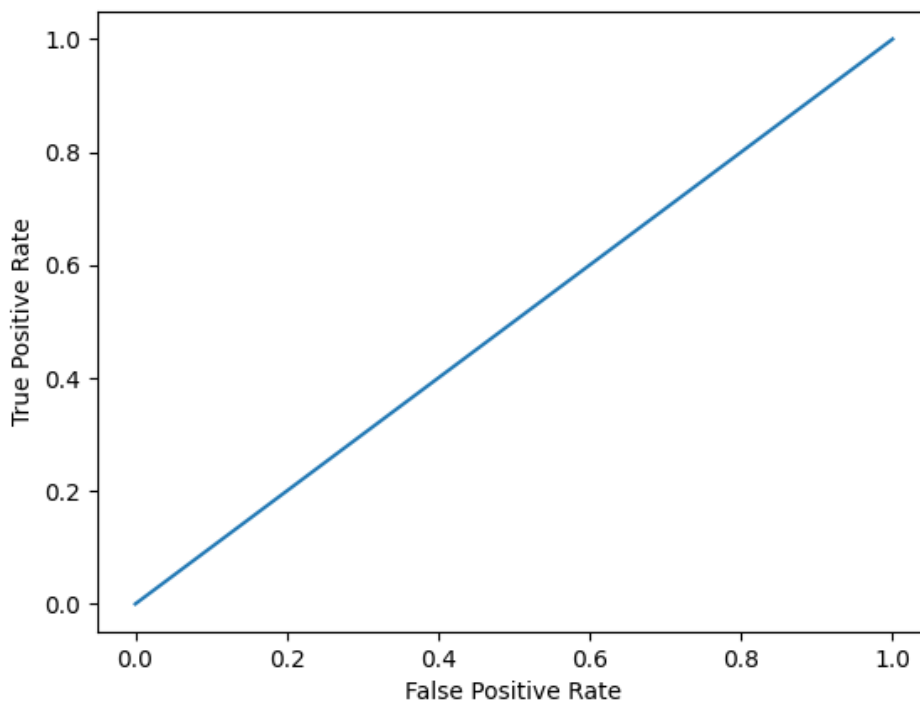
    fpr, tpr, thresholds = roc_curve(true_y, y_prob)
    plt.plot(fpr, tpr)
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
```

Example

Model 1:

```
plot_roc_curve(y, y_proba)
print(f'model 1 AUC score: {roc_auc_score(y, y_proba)}')
```

Result



model 1 AUC score: 0.5

[Run example »](#)

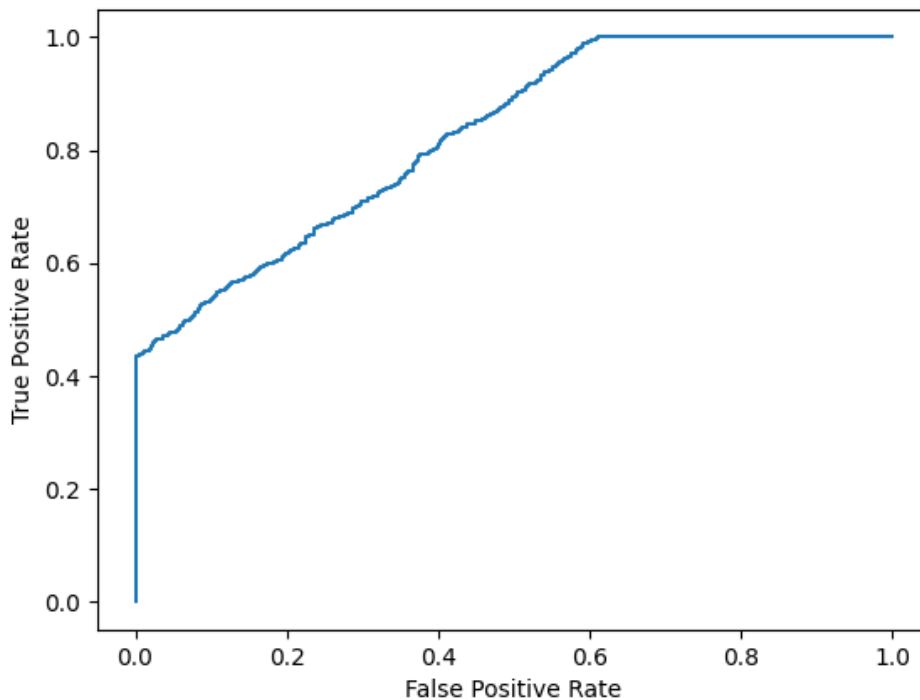
Example



Model 2:

```
plot_roc_curve(y, y_proba_2)
print(f'model 2 AUC score: {roc_auc_score(y, y_proba_2)}')
```

Result



model 2 AUC score: 0.8270551578947367

[Run example »](#)

An AUC score of around .5 would mean that the model is unable to make a distinction between the two classes and the curve would look like a line with a slope of 1. An AUC score closer to 1 means that the model has the ability to separate the two classes and the curve would come closer to the top left corner of the graph.

## Probabilities

Because AUC is a metric that utilizes probabilities of the class predictions, we can be more confident in a model that has a higher AUC score than one with a lower score even if they have similar accuracies.

In the data below, we have two sets of probabilities from hypothetical models. The first has probabilities that are not as "confident" when predicting the two classes

(the probabilities are close to .5). The second has probabilities that are more "confident" when predicting the two classes (the probabilities are close to the extremes of 0 or 1).

Example

```
import numpy as np
```

```
n = 10000
y = np.array([0] * n + [1] * n)
#
y_prob_1 = np.array(
    np.random.uniform(.25, .5, n//2).tolist() +
    np.random.uniform(.3, .7, n).tolist() +
    np.random.uniform(.5, .75, n//2).tolist()
)
y_prob_2 = np.array(
    np.random.uniform(0, .4, n//2).tolist() +
    np.random.uniform(.3, .7, n).tolist() +
    np.random.uniform(.6, 1, n//2).tolist()
)

print(f'model 1 accuracy score: {accuracy_score(y, y_prob_1>.5)}')
print(f'model 2 accuracy score: {accuracy_score(y, y_prob_2>.5)}')

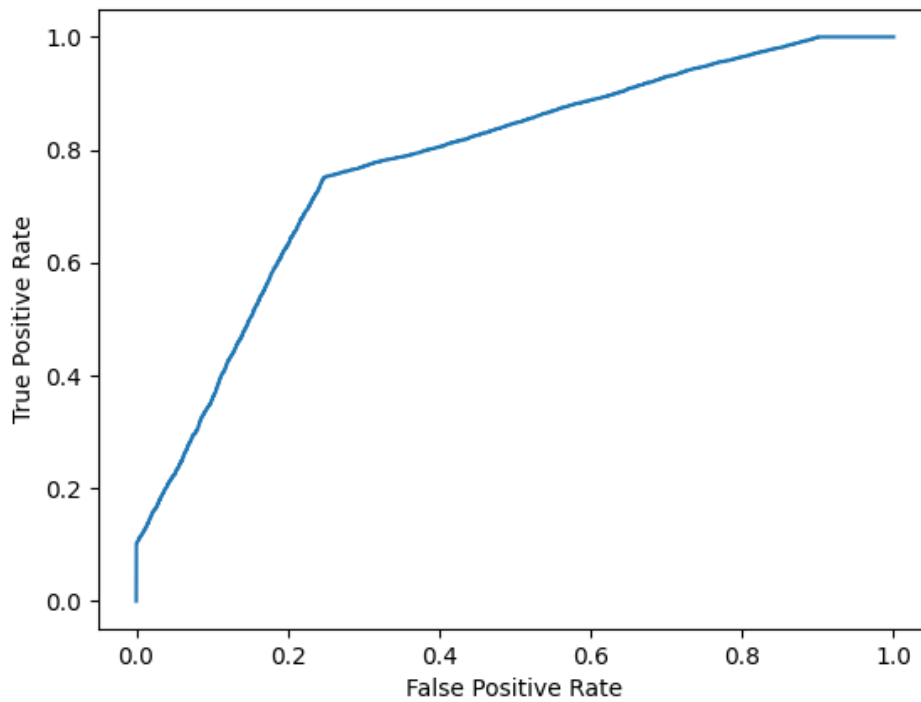
print(f'model 1 AUC score: {roc_auc_score(y, y_prob_1)}')
print(f'model 2 AUC score: {roc_auc_score(y, y_prob_2)}')
```

Example

Plot model 1:

```
plot_roc_curve(y, y_prob_1)
```

Result



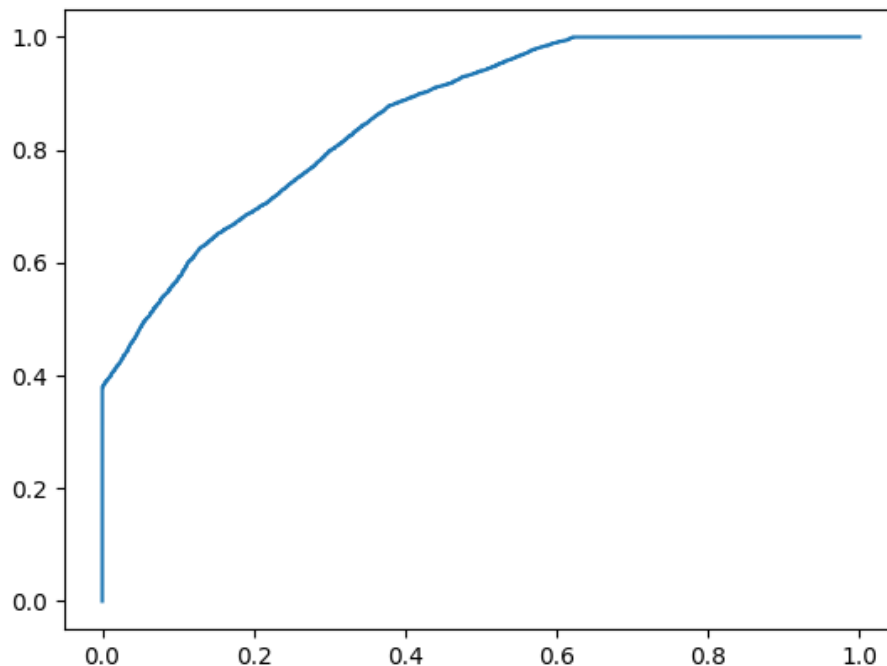
[Run example »](#)

Example

Plot model 2:

```
fpr, tpr, thresholds = roc_curve(y, y_prob_2)
plt.plot(fpr, tpr)
```

Result



## K-nearest neighbors (KNN)

### KNN

KNN is a simple, supervised machine learning (ML) algorithm that can be used for classification or regression tasks - and is also frequently used in missing value imputation. It is based on the idea that the observations closest to a given data point are the most "similar" observations in a data set, and we can therefore classify unforeseen points based on the values of the closest existing points. By choosing  $K$ , the user can select the number of nearby observations to use in the algorithm.

Here, we will show you how to implement the KNN algorithm for classification, and show how different values of  $K$  affect the results.

### How does it work?

$K$  is the number of nearest neighbors to use. For classification, a majority vote is used to determine which class a new observation should fall into. Larger values of  $K$  are often more robust to outliers and produce more stable decision boundaries than very small values ( $K=3$  would be better than  $K=1$ , which might produce undesirable results).

### Example

Start by visualizing some data points:

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

```
x = [4, 5, 10, 4, 3, 11, 14, 8, 10, 12]
```

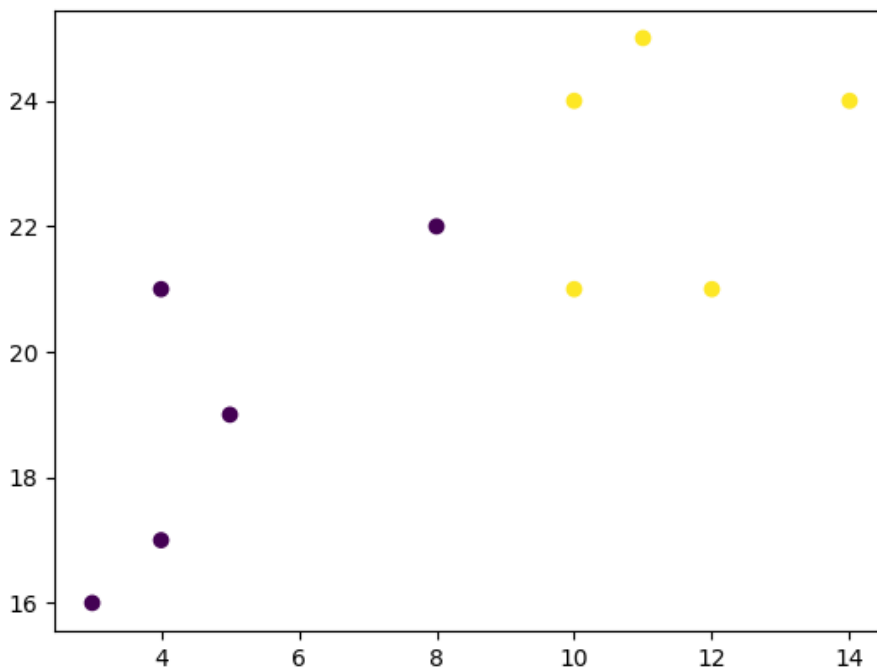
```
y = [21, 19, 24, 17, 16, 25, 24, 22, 21, 21]
```

```
classes = [0, 0, 1, 0, 0, 1, 1, 0, 1, 1]
```

```
plt.scatter(x, y, c=classes)
```

```
plt.show()
```

Result



[Run example »](#)

Now we fit the KNN algorithm with K=1:

```
from sklearn.neighbors import KNeighborsClassifier
```

```
data = list(zip(x, y))
```

```
knn = KNeighborsClassifier(n_neighbors=1)
```

```
knn.fit(data, classes)
```

And use it to classify a new data point:

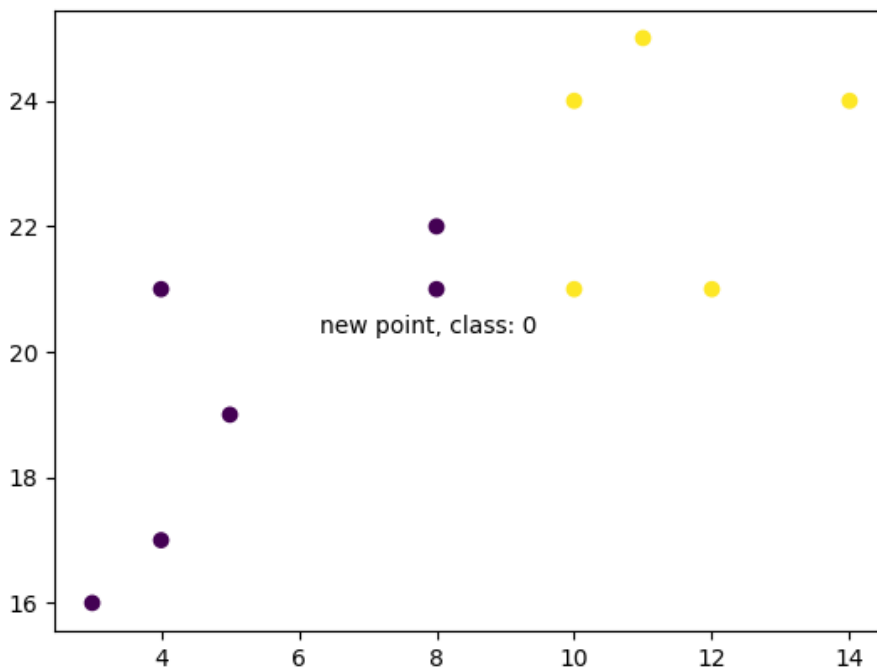
Example

```
new_x = 8
new_y = 21
new_point = [(new_x, new_y)]

prediction = knn.predict(new_point)

plt.scatter(x + [new_x], y + [new_y], c=classes + [prediction[0]])
plt.text(x=new_x-1.7, y=new_y-0.7, s=f"new point, class: {prediction[0]}")
plt.show()
```

Result



[Run example »](#)

Now we do the same thing, but with a higher K value which changes the prediction:

Example

```

knn = KNeighborsClassifier(n_neighbors=5)

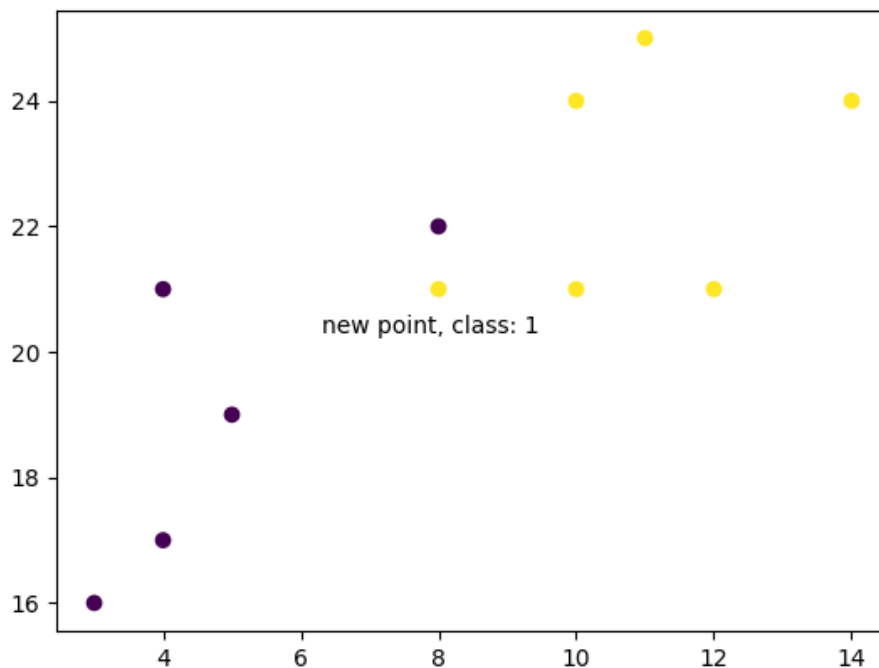
knn.fit(data, classes)

prediction = knn.predict(new_point)

plt.scatter(x + [new_x], y + [new_y], c=classes + [prediction[0]])
plt.text(x=new_x-1.7, y=new_y-0.7, s=f"new point, class: {prediction[0]}")
plt.show()

```

Result



[Run example »](#)

Example Explained

Import the modules you need.

You can learn about the Matplotlib module in our ["Matplotlib Tutorial"](#).

scikit-learn is a popular library for machine learning in Python.

```

import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier

```

Create arrays that resemble variables in a dataset. We have two input features (x and y) and then a target class (class). The input features that are pre-labeled with our target class will be used to predict the class of new data. Note that while we only use two input features here, this method will work with any number of variables:

```
x = [4, 5, 10, 4, 3, 11, 14, 8, 10, 12]
y = [21, 19, 24, 17, 16, 25, 24, 22, 21, 21]
classes = [0, 0, 1, 0, 0, 1, 1, 0, 1, 1]
```

Turn the input features into a set of points:

```
data = list(zip(x, y))
print(data)
```

Result:

```
[(4, 21), (5, 19), (10, 24), (4, 17), (3, 16), (11, 25), (14, 24), (8, 22), (10, 21),
(12, 21)]
```

Using the input features and target class, we fit a KNN model on the model using 1 nearest neighbor:

```
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(data, classes)
```

Then, we can use the same KNN object to predict the class of new, unforeseen data points. First we create new x and y features, and then call `knn.predict()` on the new data point to get a class of 0 or 1:

```
new_x = 8
new_y = 21
new_point = [(new_x, new_y)]
prediction = knn.predict(new_point)
print(prediction)
```

Result:

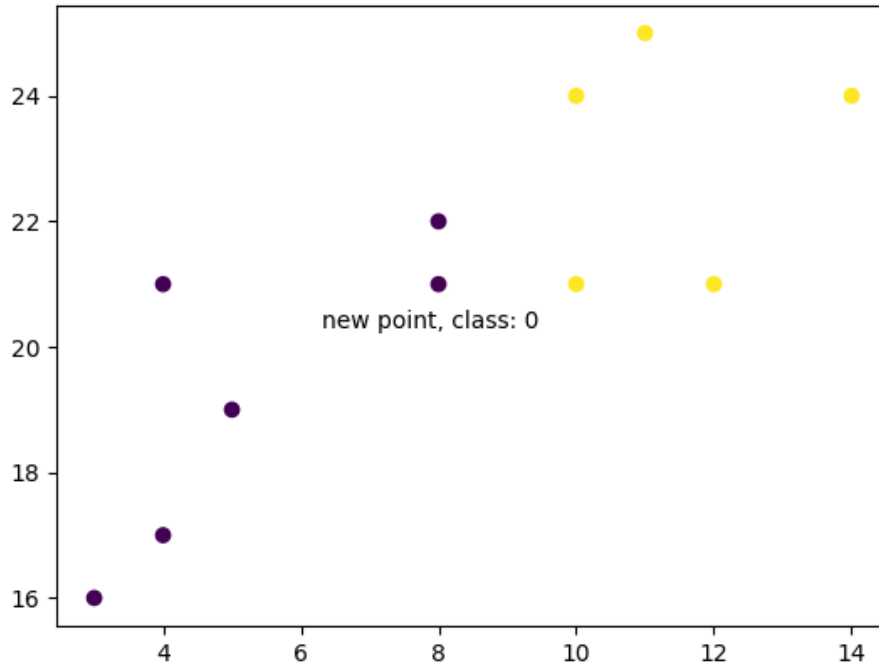
```
[0]
```

When we plot all the data along with the new point and class, we can see it's been labeled blue with the 1 class. The text annotation is just to highlight the location of the new point:

```
plt.scatter(x + [new_x], y + [new_y], c=classes + [prediction[0]])
plt.text(x=new_x-1.7, y=new_y-0.7, s=f"new point, class: {prediction[0]}")
plt.show()
```



Result:



However, when we change the number of neighbors to 5, the number of points used to classify our new point changes. As a result, so does the classification of the new point:

```
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(data, classes)
prediction = knn.predict(new_point)
print(prediction)
```

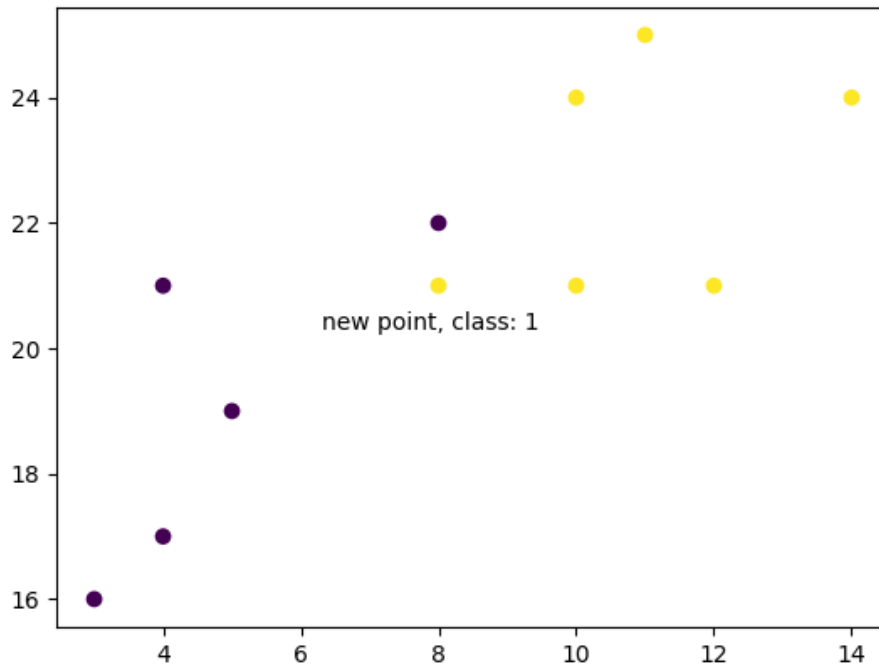
Result:

```
[1]
```

When we plot the class of the new point along with the older points, we note that the color has changed based on the associated class label:

```
plt.scatter(x + [new_x], y + [new_y], c=classes + [prediction[0]])
plt.text(x=new_x-1.7, y=new_y-0.7, s=f"new point, class: {prediction[0]}")
plt.show()
```

Result:



## Python MySQL

### MySQL Database

To be able to experiment with the code examples in this tutorial, you should have MySQL installed on your computer.

You can download a free MySQL database at <https://www.mysql.com/downloads/>.

### Install MySQL Driver

Python needs a MySQL driver to access the MySQL database.

In this tutorial we will use the driver "MySQL Connector".

We recommend that you use PIP to install "MySQL Connector".

PIP is most likely already installed in your Python environment.

Navigate your command line to the location of PIP, and type the following:

Download and install "MySQL Connector":

```
C:\Users\Your Name\AppData\Local\Programs\Python\Python36-32\Scripts>python -m pip install mysql-connector-python
```

Now you have downloaded and installed a MySQL driver.

## Test MySQL Connector

To test if the installation was successful, or if you already have "MySQL Connector" installed, create a Python page with the following content:

demo\_mysql\_test.py:

```
import mysql.connector
```

[Run example »](#)

If the above code was executed with no errors, "MySQL Connector" is installed and ready to be used.

## Create Connection

Start by creating a connection to the database.

Use the username and password from your MySQL database:

demo\_mysql\_connection.py:

```
import mysql.connector
```

```
mydb = mysql.connector.connect(  
    host="localhost",  
    user="yourusername",  
    password="yourpassword"  
)
```

```
print(mydb)
```

## Creating a Database

To create a database in MySQL, use the "CREATE DATABASE" statement:

Example

create a database named "mydatabase":

```
import mysql.connector
```

```
mydb = mysql.connector.connect(  
    host="localhost",  
    user="yourusername",  
    password="yourpassword"  
)
```

```
mycursor = mydb.cursor()
```

```
mycursor.execute("CREATE DATABASE mydatabase")
```

[Run example »](#)

If the above code was executed with no errors, you have successfully created a database.

## Check if Database Exists

You can check if a database exist by listing all databases in your system by using the "SHOW DATABASES" statement:

### Example

Return a list of your system's databases:

```
import mysql.connector
```

```
mydb = mysql.connector.connect(  
    host="localhost",  
    user="yourusername",  
    password="yourpassword"  
)
```

```
mycursor = mydb.cursor()
```

```
mycursor.execute("SHOW DATABASES")
```

```
for x in mycursor:  
    print(x)
```

[Run example »](#)

Or you can try to access the database when making the connection:

Example

Try connecting to the database "mydatabase":

```
import mysql.connector

mydb = mysql.connector.connect(
    host="localhost",
    user="yourusername",
    password="yourpassword",
    database="mydatabase"
)
```

## Python MySQL Create Table

Creating a Table

To create a table in MySQL, use the "CREATE TABLE" statement.

Make sure you define the name of the database when you create the connection

Example

Create a table named "customers":

```
import mysql.connector

mydb = mysql.connector.connect(
    host="localhost",
    user="yourusername",
    password="yourpassword",
    database="mydatabase"
)

mycursor = mydb.cursor()

mycursor.execute("CREATE TABLE customers (name VARCHAR(255), address VARCHAR(255))")
```

[Run example »](#)

If the above code was executed with no errors, you have now successfully created a table.

### Check if Table Exists

You can check if a table exist by listing all tables in your database with the "SHOW TABLES" statement:

### Example

Return a list of your system's databases:

```
import mysql.connector

mydb = mysql.connector.connect(
    host="localhost",
    user="yourusername",
    password="yourpassword",
    database="mydatabase"
)

mycursor = mydb.cursor()

mycursor.execute("SHOW TABLES")

for x in mycursor:
    print(x)
```

### Primary Key

When creating a table, you should also create a column with a unique key for each record.

This can be done by defining a PRIMARY KEY.

We use the statement "INT AUTO\_INCREMENT PRIMARY KEY" which will insert a unique number for each record. Starting at 1, and increased by one for each record.

### Example

Create primary key when creating the table:

```
import mysql.connector

mydb = mysql.connector.connect(
    host="localhost",
    user="yourusername",
    password="yourpassword",
    database="mydatabase"
)

mycursor = mydb.cursor()

mycursor.execute("CREATE TABLE customers (id INT AUTO_INCREMENT
PRIMARY KEY, name VARCHAR(255), address VARCHAR(255))")
```

[Run example »](#)

If the table already exists, use the ALTER TABLE keyword:

Example

Create primary key on an existing table:

```
import mysql.connector

mydb = mysql.connector.connect(
    host="localhost",
    user="yourusername",
    password="yourpassword",
    database="mydatabase"
)

mycursor = mydb.cursor()

mycursor.execute("ALTER TABLE customers ADD COLUMN id INT
AUTO_INCREMENT PRIMARY KEY")
```

Machine learning and artificial intelligence (AI) have the potential to revolutionize the way cellular networks operate and provide services to users. These technologies can enable networks to become more efficient, reliable, and intelligent, leading to improved user experiences and better performance. In this review, we will discuss some of the key ways in which machine learning and AI can be used in future cellular networks.

1. **Network Optimization:** Machine learning and AI can be used to optimize network performance by analyzing network data and identifying patterns that can be used to improve network efficiency. For example, AI algorithms can be used to predict traffic patterns, identify network bottlenecks, and adjust network resources accordingly. This can lead to improved network reliability, faster data transfer speeds, and better overall network performance.
2. **Predictive Maintenance:** Machine learning and AI can be used to predict when network equipment will fail, allowing for proactive maintenance and avoiding costly downtime. This can be achieved by analyzing network data and identifying patterns that indicate a potential failure. For example, AI algorithms can be used to predict when a cell tower will need maintenance, allowing for preventative maintenance to be scheduled before a failure occurs.
3. **Self-Organizing Networks:** Machine learning and AI can be used to enable networks to self-organize and adapt to changing conditions. For example, AI algorithms can be used to dynamically adjust network resources based on traffic patterns and network usage. This can lead to more efficient network operations, improved network performance, and a better user experience.
4. **User Experience Optimization:** Machine learning and AI can be used to optimize the user experience by analyzing user data and identifying patterns that can be used to improve the quality of service. For example, AI algorithms can be used to predict when a user is likely to experience poor network coverage, allowing for proactive steps to be taken to improve the user experience.

Overall, the use of machine learning and AI in future cellular networks has the potential to significantly improve network performance and the user experience. As these technologies continue to evolve, we can expect to see more advanced applications and use cases in the future.

## Types of Machine Learning

**Machine learning is a subset of AI, which enables the machine to automatically learn from data, improve performance from past experiences, and make predictions.** Machine learning contains a set of algorithms that work on a huge amount of data. Data is fed to these algorithms to train them, and on the basis of training, they build the model & perform a specific task.



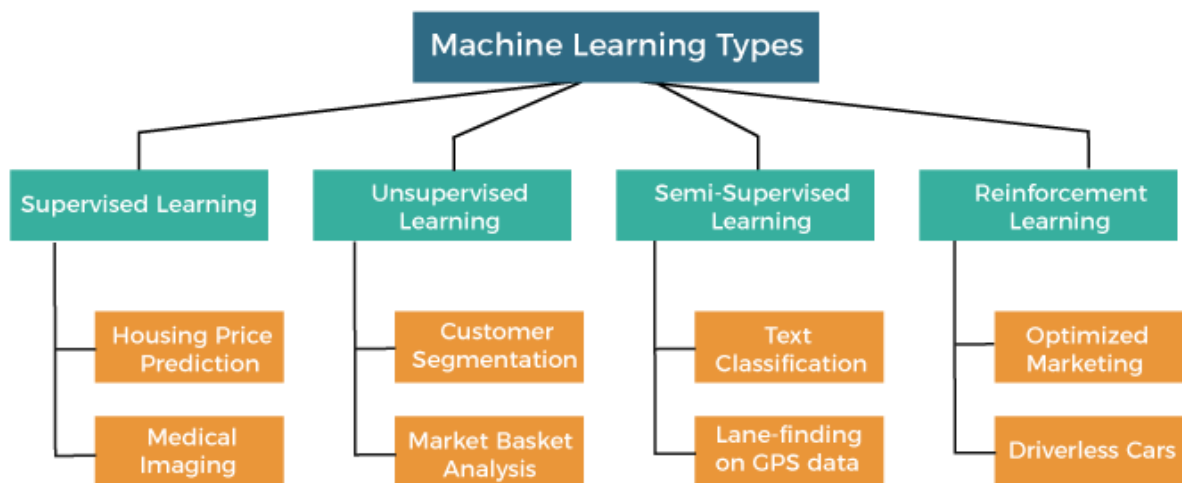
# Types of Machine Learning



These ML algorithms help to solve different business problems like Regression, Classification, Forecasting, Clustering, and Associations, etc.

Based on the methods and way of learning, machine learning is divided into mainly four types, which are:

1. Supervised Machine Learning
2. Unsupervised Machine Learning
3. Semi-Supervised Machine Learning
4. Reinforcement Learning



In this topic, we will provide a detailed description of the types of Machine Learning along with their respective algorithms:

## 1. Supervised Machine Learning

As its name suggests, **Supervised machine learning** is based on supervision. It means in the supervised learning technique, we train the machines using the "labelled" dataset, and based on the training, the machine predicts the output. Here, the labelled data specifies that some of the inputs are already mapped to the output. More precisely, we can say; first, we train the machine with the input and corresponding output, and then we ask the machine to predict the output using the test dataset.

Let's understand supervised learning with an example. Suppose we have an input dataset of cats and dog images. So, first, we will provide the training to the machine to understand the images, such as the **shape & size of the tail of cat and dog, Shape of eyes, colour, height (dogs are taller, cats are smaller), etc.** After completion of training, we input the picture of a cat and ask the machine to identify the object and predict the output. Now, the machine is well trained, so it will check all the features of the object, such as height, shape, colour, eyes, ears, tail, etc., and find that it's a cat. So, it will put it in the Cat category. This is the process of how the machine identifies the objects in Supervised Learning.

**The main goal of the supervised learning technique is to map the input variable(x) with the output variable(y).** Some real-world applications of supervised learning are **Risk Assessment, Fraud Detection, Spam filtering**, etc.

### Categories of Supervised Machine Learning

Supervised machine learning can be classified into two types of problems, which are given below:

- **Classification**
- **Regression**

#### a) Classification

Classification algorithms are used to solve the classification problems in which the output variable is categorical, such as **"Yes" or No, Male or Female, Red or Blue, etc.** The classification algorithms predict the categories present in the dataset. Some real-world examples of classification algorithms are **Spam Detection, Email filtering, etc.**

Some popular classification algorithms are given below:

- **Random Forest Algorithm**
- **Decision Tree Algorithm**
- **Logistic Regression Algorithm**
- **Support Vector Machine Algorithm**

## b) Regression

Regression algorithms are used to solve regression problems in which there is a linear relationship between input and output variables. These are used to predict continuous output variables, such as market trends, weather prediction, etc.

Some popular Regression algorithms are given below:

- **Simple Linear Regression Algorithm**
- **Multivariate Regression Algorithm**
- **Decision Tree Algorithm**
- **Lasso Regression**

## Advantages and Disadvantages of Supervised Learning

### Advantages:

- Since supervised learning work with the labelled dataset so we can have an exact idea about the classes of objects.
- These algorithms are helpful in predicting the output on the basis of prior experience.

### Disadvantages:

- These algorithms are not able to solve complex tasks.
- It may predict the wrong output if the test data is different from the training data.
- It requires lots of computational time to train the algorithm.

## Applications of Supervised Learning

Some common applications of Supervised Learning are given below:

- **Image Segmentation:**  
Supervised Learning algorithms are used in image segmentation. In this process, image classification is performed on different image data with pre-defined labels.
- **Medical Diagnosis:**  
Supervised algorithms are also used in the medical field for diagnosis purposes. It is done by using medical images and past labelled data with labels for disease conditions. With such a process, the machine can identify a disease for the new patients.
- **Fraud Detection** - Supervised Learning classification algorithms are used for identifying fraud transactions, fraud customers, etc. It is done by using historic data to identify the patterns that can lead to possible fraud.
- **Spam detection** - In spam detection & filtering, classification algorithms are used. These algorithms classify an email as spam or not spam. The spam emails are sent to the spam folder.
- **Speech Recognition** - Supervised learning algorithms are also used in speech recognition. The algorithm is trained with voice data, and various identifications can be done using the same, such as voice-activated passwords, voice commands, etc.

## 2. Unsupervised Machine Learning

**Unsupervised learning** is different from the Supervised learning technique; as its name suggests, there is no need for supervision. It means, in

unsupervised machine learning, the machine is trained using the unlabeled dataset, and the machine predicts the output without any supervision.

In unsupervised learning, the models are trained with the data that is neither classified nor labelled, and the model acts on that data without any supervision.

**The main aim of the unsupervised learning algorithm is to group or categories the unsorted dataset according to the similarities, patterns, and differences.** Machines are instructed to find the hidden patterns from the input dataset.

Let's take an example to understand it more precisely; suppose there is a basket of fruit images, and we input it into the machine learning model. The images are totally unknown to the model, and the task of the machine is to find the patterns and categories of the objects.

So, now the machine will discover its patterns and differences, such as colour difference, shape difference, and predict the output when it is tested with the test dataset.

## Categories of Unsupervised Machine Learning

Unsupervised Learning can be further classified into two types, which are given below:

- **Clustering**
- **Association**

### 1) Clustering

The clustering technique is used when we want to find the inherent groups from the data. It is a way to group the objects into a cluster such that the objects with the most similarities remain in one group and have fewer or no similarities with the objects of other groups. An example of the clustering algorithm is grouping the customers by their purchasing behaviour.

Some of the popular clustering algorithms are given below:

- **K-Means Clustering algorithm**

- **Mean-shift algorithm**
- **DBSCAN Algorithm**
- **Principal Component Analysis**
- **Independent Component Analysis**

## 2) Association

Association rule learning is an unsupervised learning technique, which finds interesting relations among variables within a large dataset. The main aim of this learning algorithm is to find the dependency of one data item on another data item and map those variables accordingly so that it can generate maximum profit. This algorithm is mainly applied in **Market Basket analysis, Web usage mining, continuous production**, etc.

Some popular algorithms of Association rule learning are **Apriori Algorithm, Eclat, FP-growth algorithm**.

## Advantages and Disadvantages of Unsupervised Learning Algorithm

### Advantages:

- These algorithms can be used for complicated tasks compared to the supervised ones because these algorithms work on the unlabeled dataset.
- Unsupervised algorithms are preferable for various tasks as getting the unlabeled dataset is easier as compared to the labelled dataset.

### Disadvantages:

- The output of an unsupervised algorithm can be less accurate as the dataset is not labelled, and algorithms are not trained with the exact output in prior.
- Working with Unsupervised learning is more difficult as it works with the unlabelled dataset that does not map with the output.

## Applications of Unsupervised Learning

- **Network Analysis:** Unsupervised learning is used for identifying plagiarism and copyright in document network analysis of text data for scholarly articles.
- **Recommendation Systems:** Recommendation systems widely use unsupervised learning techniques for building recommendation applications for different web applications and e-commerce websites.
- **Anomaly Detection:** Anomaly detection is a popular application of unsupervised learning, which can identify unusual data points within the dataset. It is used to discover fraudulent transactions.
- **Singular Value Decomposition:** Singular Value Decomposition or SVD is used to extract particular information from the database. For example, extracting information of each user located at a particular location.

### 3. Semi-Supervised Learning

**Semi-Supervised learning is a type of Machine Learning algorithm that lies between Supervised and Unsupervised machine learning.** It represents the intermediate ground between Supervised (With Labelled training data) and Unsupervised learning (with no labelled training data) algorithms and uses the combination of labelled and unlabeled datasets during the training period.

Although Semi-supervised learning is the middle ground between supervised and unsupervised learning and operates on the data that consists of a few labels, it mostly consists of unlabeled data. As labels are costly, but for corporate purposes, they may have few labels. It is completely different from supervised and unsupervised learning as they are based on the presence & absence of labels.

**To overcome the drawbacks of supervised learning and unsupervised learning algorithms, the concept of Semi-supervised learning is introduced.** The main aim of **semi-supervised learning** is to effectively use all the available data, rather than only labelled data like in supervised learning. Initially, similar data is clustered along with an unsupervised learning algorithm, and further, it helps to label the unlabeled data into

labelled data. It is because labelled data is a comparatively more expensive acquisition than unlabeled data.

We can imagine these algorithms with an example. Supervised learning is where a student is under the supervision of an instructor at home and college. Further, if that student is self-analysing the same concept without any help from the instructor, it comes under unsupervised learning. Under semi-supervised learning, the student has to revise himself after analyzing the same concept under the guidance of an instructor at college.

## Advantages and disadvantages of Semi-supervised Learning

### Advantages:

- It is simple and easy to understand the algorithm.
- It is highly efficient.
- It is used to solve drawbacks of Supervised and Unsupervised Learning algorithms.

### Disadvantages:

- Iterations results may not be stable.
- We cannot apply these algorithms to network-level data.
- Accuracy is low.

## 4. Reinforcement Learning

**Reinforcement learning works on a feedback-based process, in which an AI agent (A software component) automatically explore its surrounding by hitting & trail, taking action, learning from experiences, and improving its performance.** Agent gets rewarded for each good action and get punished for each bad action; hence the goal of reinforcement learning agent is to maximize the rewards.

In reinforcement learning, there is no labelled data like supervised learning, and agents learn from their experiences only.



The **reinforcement learning** process is similar to a human being; for example, a child learns various things by experiences in his day-to-day life. An example of reinforcement learning is to play a game, where the Game is the environment, moves of an agent at each step define states, and the goal of the agent is to get a high score. Agent receives feedback in terms of punishment and rewards.

Due to its way of working, reinforcement learning is employed in different fields such as **Game theory, Operation Research, Information theory, multi-agent systems.**

A reinforcement learning problem can be formalized using **Markov Decision Process(MDP)**. In MDP, the agent constantly interacts with the environment and performs actions; at each action, the environment responds and generates a new state.

## Categories of Reinforcement Learning

Reinforcement learning is categorized mainly into two types of methods/algorithms:

- **Positive Reinforcement Learning:** Positive reinforcement learning specifies increasing the tendency that the required behaviour would occur again by adding something. It enhances the strength of the behaviour of the agent and positively impacts it.
- **Negative Reinforcement Learning:** Negative reinforcement learning works exactly opposite to the positive RL. It increases the tendency that the specific behaviour would occur again by avoiding the negative condition.

## Real-world Use cases of Reinforcement Learning

- **Video Games:**  
RL algorithms are much popular in gaming applications. It is used to gain super-human performance. Some popular games that use RL algorithms are **AlphaGO** and **AlphaGO Zero**.
- **Resource Management:**  
The "Resource Management with Deep Reinforcement Learning"

paper showed that how to use RL in computer to automatically learn and schedule resources to wait for different jobs in order to minimize average job slowdown.

- **Robotics:**

RL is widely being used in Robotics applications. Robots are used in the industrial and manufacturing area, and these robots are made more powerful with reinforcement learning. There are different industries that have their vision of building intelligent robots using AI and Machine learning technology.

- **Text Mining**

Text-mining, one of the great applications of NLP, is now being implemented with the help of Reinforcement Learning by Salesforce company.

## Advantages and Disadvantages of Reinforcement Learning

### Advantages

- It helps in solving complex real-world problems which are difficult to be solved by general techniques.
- The learning model of RL is similar to the learning of human beings; hence most accurate results can be found.
- Helps in achieving long term results.

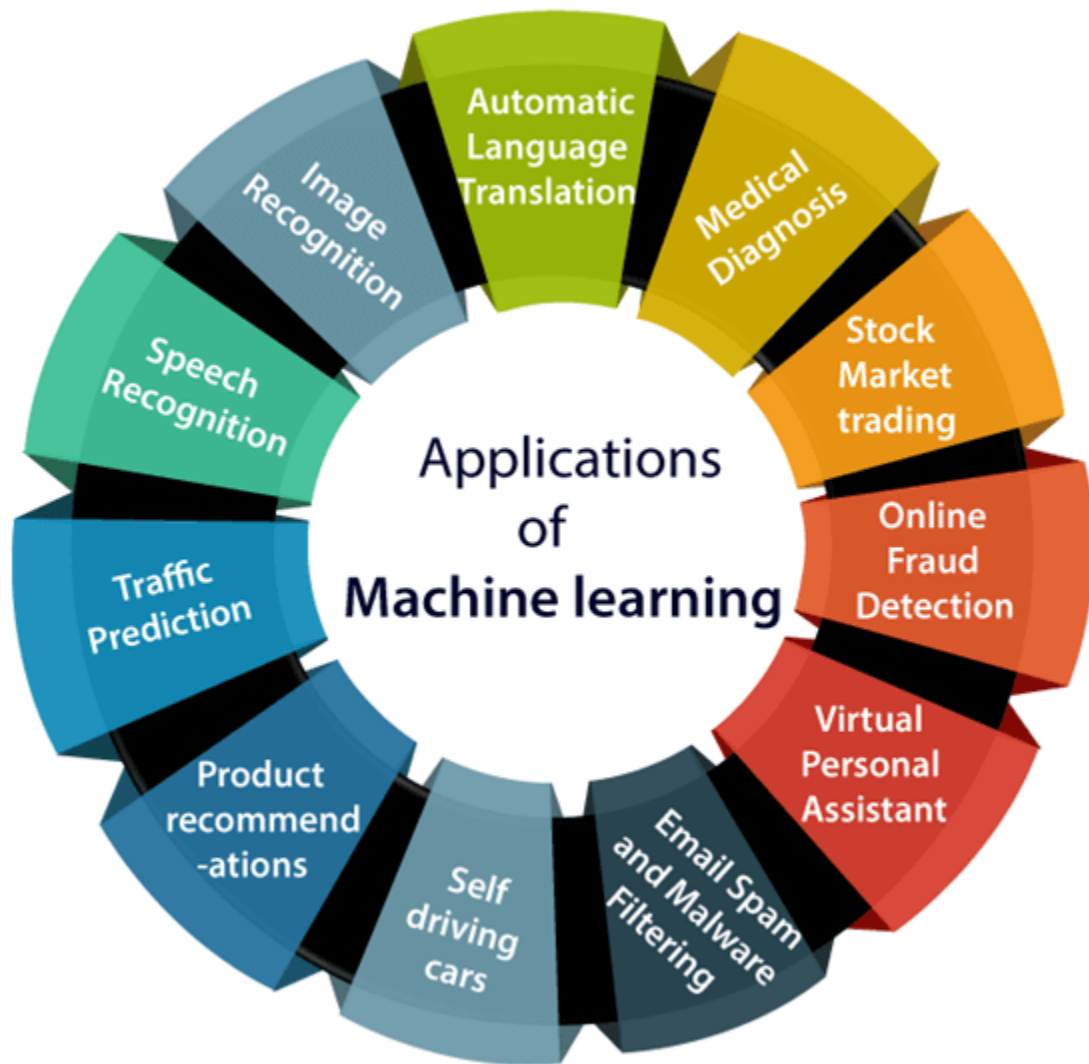
### Disadvantage

- RL algorithms are not preferred for simple problems.
- RL algorithms require huge data and computations.
- Too much reinforcement learning can lead to an overload of states which can weaken the results.

The curse of dimensionality limits reinforcement learning for real physical systems.

# Applications of Machine learning

Machine learning is a buzzword for today's technology, and it is growing very rapidly day by day. We are using machine learning in our daily life even without knowing it such as Google Maps, Google assistant, Alexa, etc. Below are some most trending real-world applications of Machine Learning:



## 1. Image Recognition:

Image recognition is one of the most common applications of machine learning. It is used to identify objects, persons, places, digital images, etc. The popular use case of image recognition and face detection is, **Automatic friend tagging suggestion**:

Facebook provides us a feature of auto friend tagging suggestion. Whenever we upload a photo with our Facebook friends, then we automatically get a tagging suggestion with name, and the technology behind this is machine learning's **face detection** and **recognition algorithm**.

It is based on the Facebook project named "**Deep Face**," which is responsible for face recognition and person identification in the picture.

## 2. Speech Recognition

While using Google, we get an option of "**Search by voice**," it comes under speech recognition, and it's a popular application of machine learning.

Speech recognition is a process of converting voice instructions into text, and it is also known as "**Speech to text**", or "**Computer speech recognition**." At present, machine learning algorithms are widely used by various applications of speech recognition. **Google assistant**, **Siri**, **Cortana**, and **Alexa** are using speech recognition technology to follow the voice instructions.

## 3. Traffic prediction:

If we want to visit a new place, we take help of Google Maps, which shows us the correct path with the shortest route and predicts the traffic conditions.

It predicts the traffic conditions such as whether traffic is cleared, slow-moving, or heavily congested with the help of two ways:

- **Real Time location** of the vehicle from Google Map app and sensors
- **Average time has taken** on past days at the same time.

Everyone who is using Google Map is helping this app to make it better. It takes information from the user and sends back to its database to improve the performance.

## 4. Product recommendations:

Machine learning is widely used by various e-commerce and entertainment companies such as **Amazon**, **Netflix**, etc., for product recommendation to the user. Whenever we search for some product on Amazon, then we started getting an advertisement for the same product while internet surfing on the same browser and this is because of machine learning.

Google understands the user interest using various machine learning algorithms and suggests the product as per customer interest.

As similar, when we use Netflix, we find some recommendations for entertainment series, movies, etc., and this is also done with the help of machine learning.

## 5. Self-driving cars:

One of the most exciting applications of machine learning is self-driving cars. Machine learning plays a significant role in self-driving cars. Tesla, the most popular car manufacturing company is working on self-driving car. It is using unsupervised learning method to train the car models to detect people and objects while driving.

## 6. Email Spam and Malware Filtering:

Whenever we receive a new email, it is filtered automatically as important, normal, and spam. We always receive an important mail in our inbox with the important symbol and spam emails in our spam box, and the technology behind this is Machine learning. Below are some spam filters used by Gmail:

- Content Filter
- Header filter
- General blacklists filter
- Rules-based filters
- Permission filters

Some machine learning algorithms such as **Multi-Layer Perceptron**, **Decision tree**, and **Naïve Bayes classifier** are used for email spam filtering and malware detection.

## 7. Virtual Personal Assistant:

We have various virtual personal assistants such as **Google assistant, Alexa, Cortana, Siri**. As the name suggests, they help us in finding the information using our voice instruction. These assistants can help us in various ways just by our voice instructions such as Play music, call someone, Open an email, Scheduling an appointment, etc.

These virtual assistants use machine learning algorithms as an important part.

These assistant record our voice instructions, send it over the server on a cloud, and decode it using ML algorithms and act accordingly.

## 8. Online Fraud Detection:

Machine learning is making our online transaction safe and secure by detecting fraud transaction. Whenever we perform some online transaction, there may be various ways that a fraudulent transaction can take place such as **fake accounts, fake ids**, and **steal money** in the middle of a transaction. So to detect this, **Feed Forward Neural network** helps us by checking whether it is a genuine transaction or a fraud transaction.

For each genuine transaction, the output is converted into some hash values, and these values become the input for the next round. For each genuine transaction, there is a specific pattern which gets change for the fraud transaction hence, it detects it and makes our online transactions more secure.

## 9. Stock Market trading:

Machine learning is widely used in stock market trading. In the stock market, there is always a risk of up and downs in shares, so for this machine learning's **long short term memory neural network** is used for the prediction of stock market trends.

## 10. Medical Diagnosis:

In medical science, machine learning is used for diseases diagnoses. With this, medical technology is growing very fast and able to build 3D models that can predict the exact position of lesions in the brain.

It helps in finding brain tumors and other brain-related diseases easily.

## 11. Automatic Language Translation:

Nowadays, if we visit a new place and we are not aware of the language then it is not a problem at all, as for this also machine learning helps us by converting the text into our known languages. Google's GNMT (Google Neural Machine Translation) provide this feature, which is a Neural Machine Learning that translates the text into our familiar language, and it called as automatic translation.

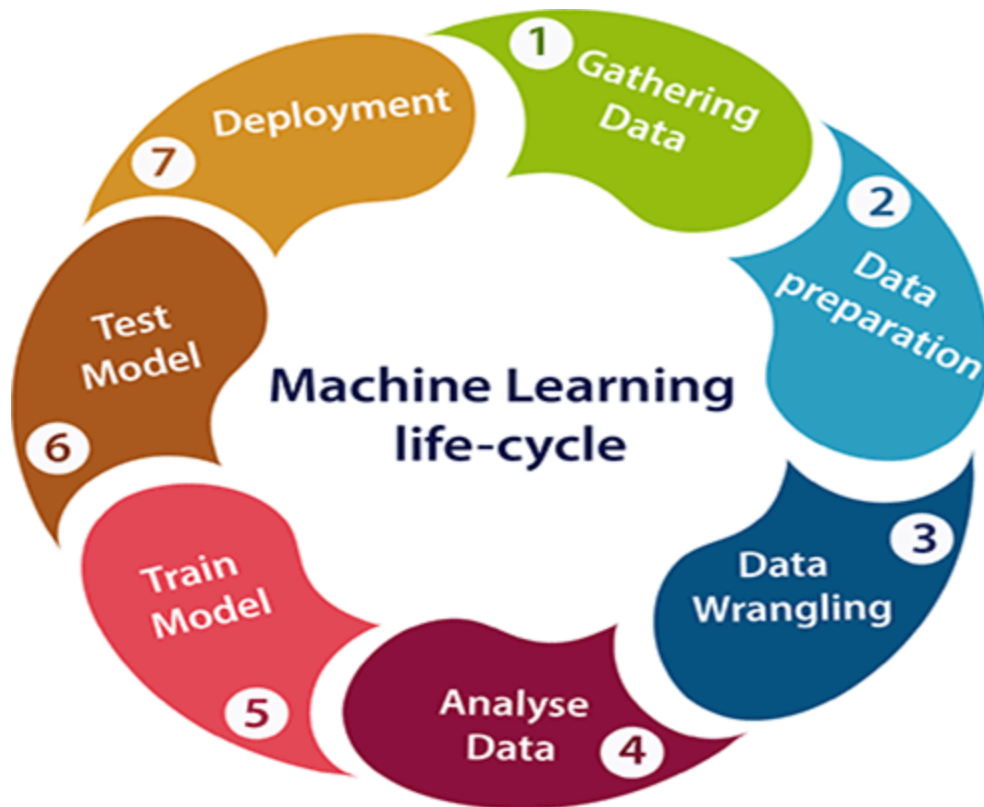
The technology behind the automatic translation is a sequence to sequence learning algorithm, which is used with image recognition and translates the text from one language to another language.

## Machine learning Life cycle

Machine learning has given the computer systems the abilities to automatically learn without being explicitly programmed. But how does a machine learning system work? So, it can be described using the life cycle of machine learning. Machine learning life cycle is a cyclic process to build an efficient machine learning project. The main purpose of the life cycle is to find a solution to the problem or project.

Machine learning life cycle involves seven major steps, which are given below:

- **Gathering Data**
- **Data preparation**
- **Data Wrangling**
- **Analyse Data**
- **Train the model**
- **Test the model**
- **Deployment**



The most important thing in the complete process is to understand the problem and to know the purpose of the problem. Therefore, before starting the life cycle, we need to understand the problem because the good result depends on the better understanding of the problem.

In the complete life cycle process, to solve a problem, we create a machine learning system called "model", and this model is created by providing "training". But to train a model, we need data, hence, life cycle starts by collecting data.

## 1. Gathering Data:

Data Gathering is the first step of the machine learning life cycle. The goal of this step is to identify and obtain all data-related problems.

In this step, we need to identify the different data sources, as data can be collected from various sources such as **files, database, internet, or mobile devices**. It is one of the most important steps of the life cycle. The quantity and quality of the collected data will determine the efficiency of the output. The more will be the data, the more accurate will be the prediction.



This step includes the below tasks:

- **Identify various data sources**
- **Collect data**
- **Integrate the data obtained from different sources**

By performing the above task, we get a coherent set of data, also called as a **dataset**. It will be used in further steps.

---

## 2. Data preparation

After collecting the data, we need to prepare it for further steps. Data preparation is a step where we put our data into a suitable place and prepare it to use in our machine learning training.

In this step, first, we put all data together, and then randomize the ordering of data.

This step can be further divided into two processes:

- **Data exploration:**  
It is used to understand the nature of data that we have to work with. We need to understand the characteristics, format, and quality of data.  
A better understanding of data leads to an effective outcome. In this, we find Correlations, general trends, and outliers.
  - **Data pre-processing:**  
Now the next step is preprocessing of data for its analysis.
- 

## 3. Data Wrangling

Data wrangling is the process of cleaning and converting raw data into a useable format. It is the process of cleaning the data, selecting the variable

to use, and transforming the data in a proper format to make it more suitable for analysis in the next step. It is one of the most important steps of the complete process. Cleaning of data is required to address the quality issues.

It is not necessary that data we have collected is always of our use as some of the data may not be useful. In real-world applications, collected data may have various issues, including:

- **Missing Values**
- **Duplicate data**
- **Invalid data**
- **Noise**

So, we use various filtering techniques to clean the data.

It is mandatory to detect and remove the above issues because it can negatively affect the quality of the outcome.

---

## 4. Data Analysis

Now the cleaned and prepared data is passed on to the analysis step. This step involves:

- **Selection of analytical techniques**
- **Building models**
- **Review the result**

The aim of this step is to build a machine learning model to analyze the data using various analytical techniques and review the outcome. It starts with the determination of the type of the problems, where we select the machine learning techniques such as **Classification, Regression, Cluster analysis, Association**, etc. then build the model using prepared data, and evaluate the model.

Hence, in this step, we take the data and use machine learning algorithms to build the model.

## 5. Train Model

Now the next step is to train the model, in this step we train our model to improve its performance for better outcome of the problem.

We use datasets to train the model using various machine learning algorithms. Training a model is required so that it can understand the various patterns, rules, and, features.

## 6. Test Model

Once our machine learning model has been trained on a given dataset, then we test the model. In this step, we check for the accuracy of our model by providing a test dataset to it.

Testing the model determines the percentage accuracy of the model as per the requirement of project or problem.

## 7. Deployment

The last step of machine learning life cycle is deployment, where we deploy the model in the real-world system.

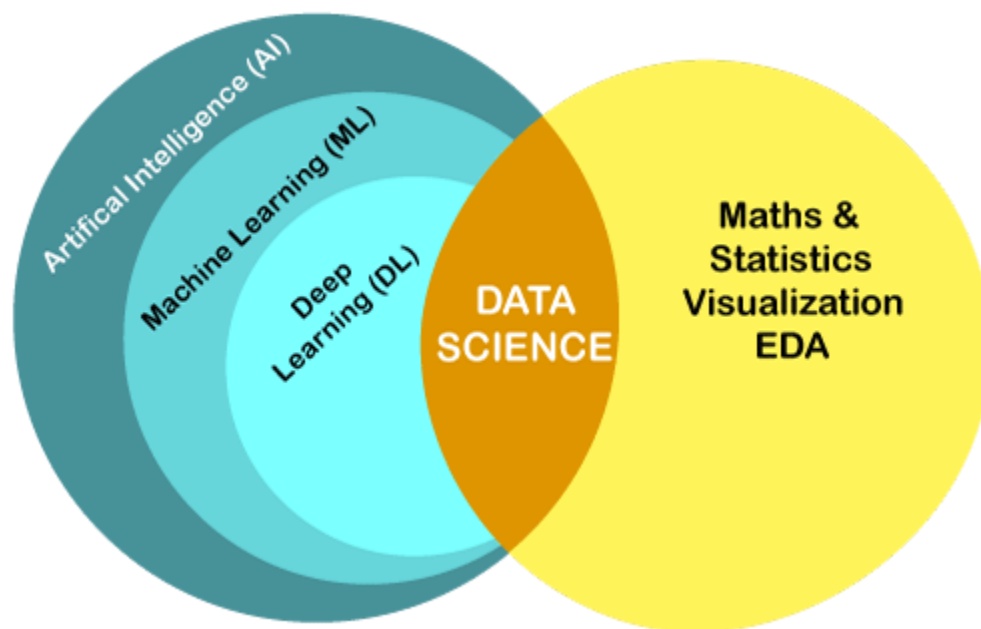
If the above-prepared model is producing an accurate result as per our requirement with acceptable speed, then we deploy the model in the real system. But before deploying the project, we will check whether it is improving its performance using available data or not. The deployment phase is similar to making the final report for a project.

## Difference Between Data Science and Machine Learning

Data Science is the study of *data cleansing, preparation, and analysis*, while machine learning is a branch of AI and subfield of data science. Data Science and Machine Learning are the two popular modern technologies, and they are growing with an immoderate rate. But these two buzzwords,

along with artificial intelligence and deep learning are very confusing term, so it is important to understand how they are different from each other. In this topic, we will understand the difference between Data Science and Machine Learning only, and how they relate to each other.

Data Science and [Machine Learning](#) are closely related to each other but have different functionalities and different goals. At a glance, *[Data Science](#) is a field to study the approaches to find insights from the raw data. Whereas, Machine Learning is a technique used by the group of data scientists to enable the machines to learn automatically from the past data.* To understand the difference in-depth, let's first have a brief introduction to these two technologies.



**Note: Data Science and Machine Learning are closely related to each other but cannot be treated as synonyms.**

## What is Data Science?

Data science, as its name suggests, is all about the data. Hence, we can define it as, **"A field of deep study of data that includes extracting useful insights from the data, and processing that information using different tools, statistical models, and Machine learning algorithms."** It

is a concept that is used to handle big data that includes data cleaning, data preparation, data analysis, and data visualization.

A data scientist collects the raw data from various sources, prepares and pre-processes the data, and applies machine learning algorithms, predictive analysis to extract useful insights from the collected data.

For example, Netflix uses data science techniques to understand user interest by mining the data and viewing patterns of its users.

### **Skills Required to become Data Scientist**

- An excellent programming knowledge of **Python, R, SAS, or Scala**.
- Experience in SQL database Coding.
- Knowledge of Machine Learning Algorithms.
- Deep Knowledge of Statistics concepts.
- Data Mining, cleaning, and Visualizing skills.
- Skills to use Big data tools such as Hadoop.

## **What is Machine Learning?**

Machine learning is a part of artificial intelligence and the subfield of Data Science. It is a growing technology that enables machines to learn from past data and perform a given task automatically. It can be defined as:

Machine Learning allows the computers to learn from the past experiences by its own, it uses statistical methods to improve the performance and predict the output without being explicitly programmed.

The popular applications of ML are ***Email spam filtering, product recommendations, online fraud detection, etc.***

### **Skills Needed for the Machine Learning Engineer:**

- Understanding and implementation of Machine Learning Algorithms.
- Natural Language Processing.
- Good Programming knowledge of Python or R.

- Knowledge of Statistics and probability concepts.
- Knowledge of data modeling and data evaluation.

## Where is Machine Learning used in Data Science?

The use of machine learning in data science can be understood by the development process or life cycle of Data Science. The different steps that occur in Data science lifecycle are as follows:



1. **Business Requirements:** In this step, we try to understand the requirement for the business problem for which we want to use it. Suppose we want to create a recommendation system, and the business requirement is to increase sales.

2. **Data Acquisition:** In this step, the data is acquired to solve the given problem. For the recommendation system, we can get the ratings provided by the user for different products, comments, purchase history, etc.
3. **Data Processing:** In this step, the raw data acquired from the previous step is transformed into a suitable format, so that it can be easily used by the further steps.
4. **Data Exploration:** It is a step where we understand the patterns of the data, and try to find out the useful insights from the data.
5. **Modeling:** The data modeling is a step where machine learning algorithms are used. So, this step includes the whole machine learning process. The machine learning process involves importing the data, data cleaning, building a model, training the model, testing the model, and improving the model's efficiency.
6. **Deployment & Optimization:** This is the last step where the model is deployed on an actual project, and the performance of the model is checked.

## Comparison Between Data Science and Machine Learning

The below table describes the basic differences between Data Science and ML:

Data Science	Machine Learning
It deals with understanding and finding hidden patterns or useful insights from the data, which helps to take smarter business decisions.	It is a subfield of data science that enables the machine to learn from the past data and experiences automatically.
It is used for discovering insights from the data.	It is used for making predictions and classifying the results for new data points.

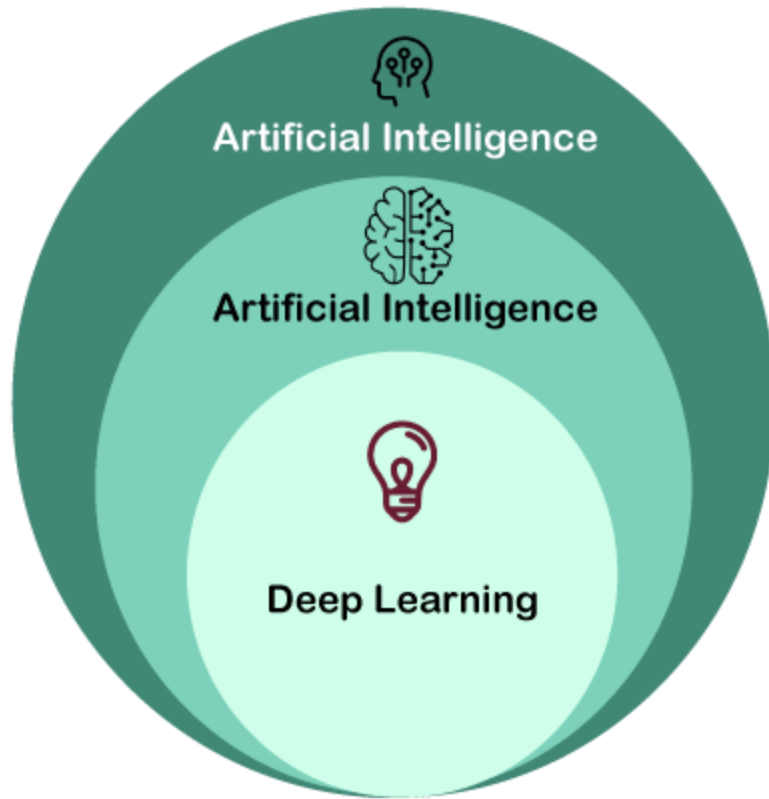
It is a broad term that includes various steps to create a model for a given problem and deploy the model.	It is used in the data modeling step of the data science as a complete process.
A data scientist needs to have skills to use big data tools like Hadoop, Hive and Pig, statistics, programming in Python, R, or Scala.	Machine Learning Engineer needs to have skills such as computer science fundamentals, programming skills in Python or R, statistics and probability concepts, etc.
It can work with raw, structured, and unstructured data.	It mostly requires structured data to work on.
Data scientists spent lots of time in handling the data, cleansing the data, and understanding its patterns.	ML engineers spend a lot of time for managing the complexities that occur during the implementation of algorithms and mathematical concepts behind that.

## Difference between Machine Learning and Deep Learning

Machine Learning and Deep Learning are the two main concepts of Data Science and the subsets of Artificial Intelligence. Most of the people think the machine learning, deep learning, and as well as artificial intelligence as the same buzzwords. But in actuality, all these terms are different but related to each other.

In this topic, we will learn how machine learning is different from deep learning. But before learning the differences, let's first have a brief introduction of [machine learning](#) and [deep learning](#).





## What is Machine Learning?

Machine learning is a part of artificial intelligence and growing technology that enables machines to learn from past data and perform a given task automatically.

Machine Learning allows the computers to learn from the experiences by its own, use statistical methods to improve the performance and predict the output without being explicitly programmed.

The popular applications of ML are Email spam filtering, product recommendations, online fraud detection, etc.

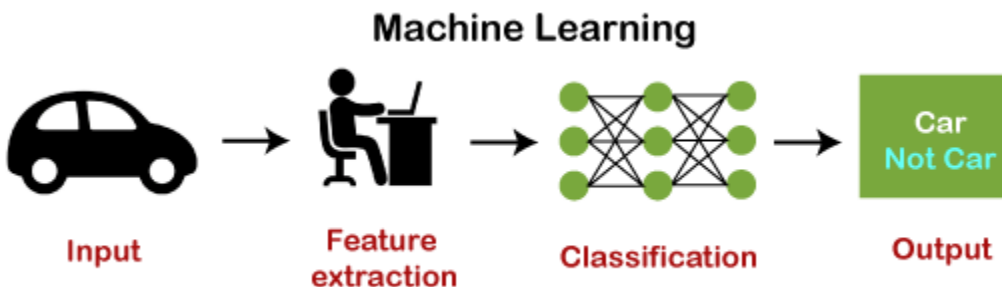
Some useful ML algorithms are:

- **Decision Tree algorithm**
- **Naïve Bayes**
- **Random Forest**

- **K-means clustering**
- **KNN algorithm**
- **Apriori Algorithm, etc.**

## How does Machine Learning work?

The working of machine learning models can be understood by the example of identifying the image of a cat or dog. To identify this, the ML model takes images of both cat and dog as input, extracts the different features of images such as shape, height, nose, eyes, etc., applies the classification algorithm, and predict the output. Consider the below image:



## What is Deep Learning?

*Deep Learning is the subset of machine learning or can be said as a special kind of machine learning.* It works technically in the same way as machine learning does, but with different capabilities and approaches. It is inspired by the functionality of human brain cells, which are called neurons, and leads to the concept of artificial neural networks. It is also called a deep neural network or deep neural learning.

In deep learning, models use different layers to learn and discover insights from the data.

Some popular applications of deep learning are self-driving cars, language translation, natural language processing, etc.

Some popular deep learning models are:

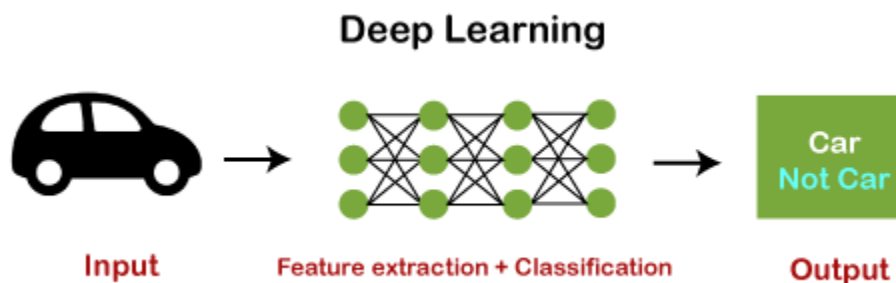
- **Convolutional Neural Network**
- **Recurrent Neural Network**

- **Autoencoders**
- **Classic Neural Networks, etc.**

## How Deep Learning Works?

We can understand the working of deep learning with the same example of identifying cat vs. dog. The deep learning model takes the images as the input and feed it directly to the algorithms without requiring any manual feature extraction step. The images pass to the different layers of the artificial neural network and predict the final output.

Consider the below image:



## Key comparisons between Machine Learning and Deep Learning

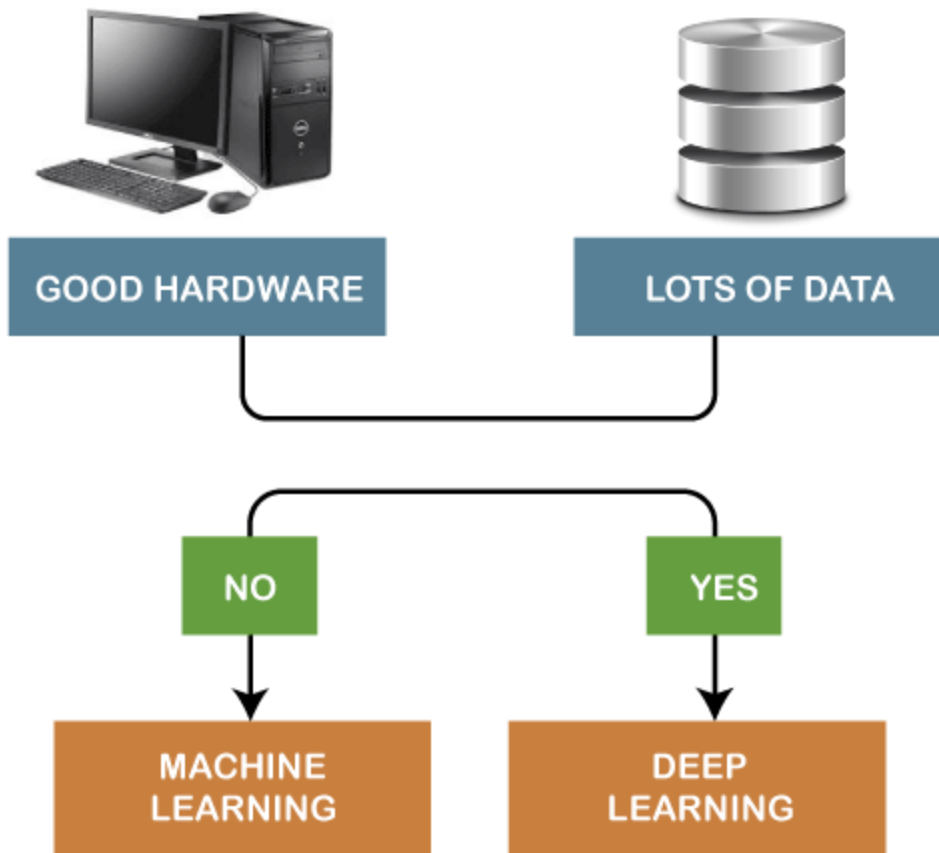
Let's understand the key differences between these two terms based on different parameters:

Parameter	Machine Learning	Deep Learning
<b>Data Dependency</b>	Although machine learning depends on the huge amount of data, it can work with a smaller amount of data.	Deep Learning algorithms highly depend on large amount of data, so we need to feed a large amount of data for good performance.
<b>Execution time</b>	Machine learning algorithm takes less time to train the model than deep learning, but it takes a long-time duration to test the model.	Deep Learning takes a long execution time to train the model, but less time to test the model.

<b>Hardware Dependencies</b>	Since machine learning models do not need much amount of data, so they can work on low-end machines.	The deep learning model needs a huge amount of data to work efficiently, so they need GPU's and hence the high-end machines.
<b>Feature Engineering</b>	Machine learning models need a step of feature extraction by the expert, and then it proceeds further.	Deep learning is the enhanced version of machine learning, so it does not need to develop the feature extractor for each problem; instead, it tries to learn high-level features from the data on its own.
<b>Problem-solving approach</b>	To solve a given problem, the traditional ML model breaks the problem in sub-parts, and after solving each part, produces the final result.	The problem-solving approach of a deep learning model is different from the traditional ML model, as it takes input for a given problem, and produce the end result. Hence it follows the end-to-end approach.
<b>Interpretation of result</b>	The interpretation of the result for a given problem is easy. As when we work with machine learning, we can interpret the result easily, it means why this result occur, what was the process.	The interpretation of the result for a given problem is very difficult. As when we work with the deep learning model, we may get a better result for a given problem than the machine learning model, but we cannot find why this particular outcome occurred, and the reasoning.
<b>Type of data</b>	Machine learning models mostly require data in a structured form.	Deep Learning models can work with structured and unstructured data both as they rely on the layers of the Artificial neural network.
<b>Suitable for</b>	Machine learning models are suitable for solving simple or bit-complex problems.	Deep learning models are suitable for solving complex problems.

## Which one to select among ML and Deep Learning?

As we have seen the brief introduction of ML and DL with some comparisons, now why and which one needs to be chosen to solve a particular problem. So, it can be understood by the given flowchart:



Hence, if you have lots of data and high hardware capabilities, go with deep learning. But if you don't have any of them, choose the ML model to solve your problem.

**Conclusion:** In conclusion, we can say that deep learning is machine learning with more capabilities and a different working approach. And selecting any of them to solve a particular problem is depend on the amount of data and complexity of the problem.

**References:**