

# Python Mastery for AI Enthusiasts: From Basics to Artificial

## **Preface**

Greetings, AI Enthusiasts!

I'm Michael, a Mechanical Engineer with a B.Sc, an MBA holder, and a passionate enthusiast for the worlds of Artificial Intelligence (AI) and Cybersecurity. I'm excited share with you some of Python knowledge that may help you enter to any aspect that you thinking about.

The overarching mission behind this endeavor is to democratize access to the field of AI, making it accessible to all, regardless of your background or expertise. This handbook span across general topics and terminology.

Throughout this handbook, you'll embark on a journey that not only uncovers Python's fundamental principles but also delves into its advanced facets. We'll dissect Python code, demystifying its intricacies with explanations. By the end of this expedition, you'll be equipped with the tools of Python's Artificial Intelligence domain, ready to embark on your own AI adventures or just coding in python environment.

This handbook will not explain each function or method deeply. If you have questions, curiosities, or simply wish to connect, please don't hesitate to reach out to me via my [LinkedIn page](#). I'm do my best to support your AI and/or Python journey, and I encourage you to join our ever-growing community.

For a deeper dive into Python code and syntax, I recommend visiting [pynative.com](https://pynative.com), a valuable resource to enhance your learning. When it comes to exploring individual functions or methods, you have the power of the internet at your fingertips, and don't forget our AI assistant, ChatGPT (or equivalent), is just a conversation away for quick insights.

Remember, in the realm of AI your potential is limitless. Wishing you the best of luck and happy coding.

— Michael

For contributions: [click here](#)

## Contents

Module 1 – General topics.....	4
Initial quick research for basic functions .....	4
Storing data in python .....	4
Calculating in python .....	4
Conditions and decision making .....	5
Repeating a task - Loops .....	5
Grouping tasks - Functions.....	7
Pre-defined functions - Operations on numbers and strings .....	7
Lists .....	8
Tuples.....	8
Module 2 – Python, advanced .....	9
File handling.....	9
Importing python modules .....	10
Random module.....	10
Statistic module .....	10
time module.....	11
Exceptions in python.....	11
Errors and exceptions .....	11
Exception handling.....	11
Finally and raise .....	12
Regular expressions – RegEx.....	12
Database connection .....	13
GUI development - tkinter .....	15
Object oriented programming .....	16
OOP's and related concepts.....	16

# Python Mastery for AI Enthusiasts: From Basics to Artificial

Classes and objects .....	16
Multi-threading in python.....	17
Introduction to multithreading - Threading module .....	17
Threading example .....	18
Module 3 – My first code example .....	18
Brief words.....	18
Sample code:.....	19
explanation: .....	21
Module 4 - AI in python .....	23
Introduction to machine learning .....	23
Simple linear regression.....	24
Multiple linear regression .....	26
Sample code for multi linear regression .....	27
Decision tree .....	28
Classification .....	29
Logistic regression.....	30
Support vector machine.....	32
Clustering .....	33
Neural network and deep learning .....	34
Generative Adversarial Networks (GAN's).....	35
Module 3.5 - SUM it, and understanding the used case's .....	23

## Module 1 – General topics

### Initial quick research for basic functions

Apply quick research for the following python functions: print(), #, continue, pass, break, print, len, True, False. (there are more, but its enough for now)

### Storing data in python

1. In python we can store different data categories with same syntax, we don't need to declare if the variable is a string or float number or integer. Examples: myVar = 30 | myVar = 25.34. | myStr = "Hello" | myStr="30". The quotes defines string, the number defines integer, and the dot defines float number. Python defines those without any special syntax.  
A variable whose value could be either True or False called a Boolean type variable.
2. In order to enter some string or integer as user input, we will use input built-in function.  
Name=input("enter your name:") – the data that we will input will be string.  
Print(name) – prints our entered name.
3. In order to label our input as integer or float numbers, we use int\float built-in functions:  
Age=int(input("enter your age:"))  
Temp=float(input("enter temperature"))  
Pay attention: if we will enter 23.4 as our age we will get an error. We will discuss it later.

### Calculating in python

1. Arithmetic operators:  
 $+$ ,  $-$ ,  $/$ ,  $*$ ,  $\%$ ,  $//$ .
2. Comparison operators  
 $==$ ,  $!=$ ,  $<>$ ,  $>$ ,  $<$ , etc
3. Logical operators  
AND, OR, NOT – those will return TRUE by their built-in condition  
Examples:  
a=True  
b=False  
print("a and b is", a and b) #print false  
print("a or b is", a or b) #print true  
print("not b is", not b) #print true
4. Walrus operator (new addition in python3)  
#using assignment operator  
var=False  
print(var)  
Syntax can be simplified:  
#using walrus operator  
print(var:=False)

## Conditions and decision making

1. Decision making statements:
  - a. If
  - b. If else
  - c. elif – if we want to check multiple conditions we can use it.

If condition 1:

Statements

elif condition 2:

Statements

elif condition 2:

Statements

else condition 4:

Statements

elif will always come after an if statement, it cannot exist individually.

elif statements will be checked in the order in which they appear.

## Repeating a task - Loops

### #range function

for num in range(1,101) – range taking start and end value (1 to 100 for this example). Range(100) and range(0,100) give the same result. We can use **step** for range function, default step is 1. Range(0,10,**2**)

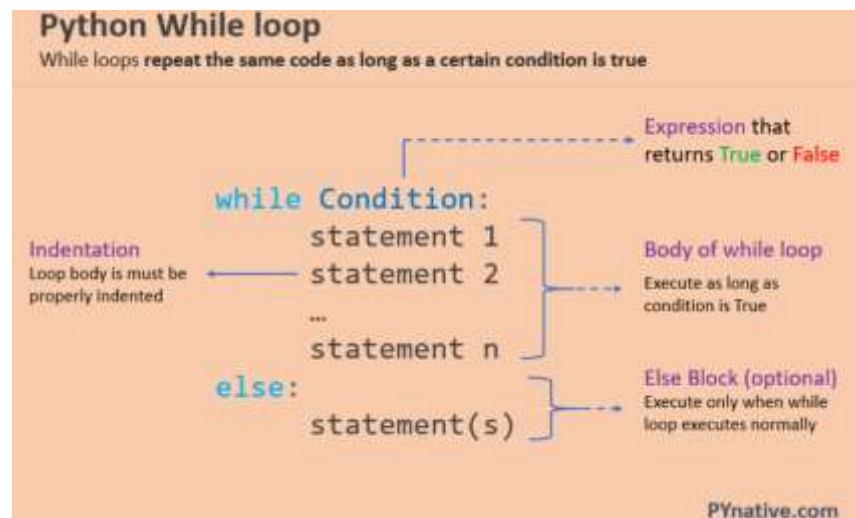
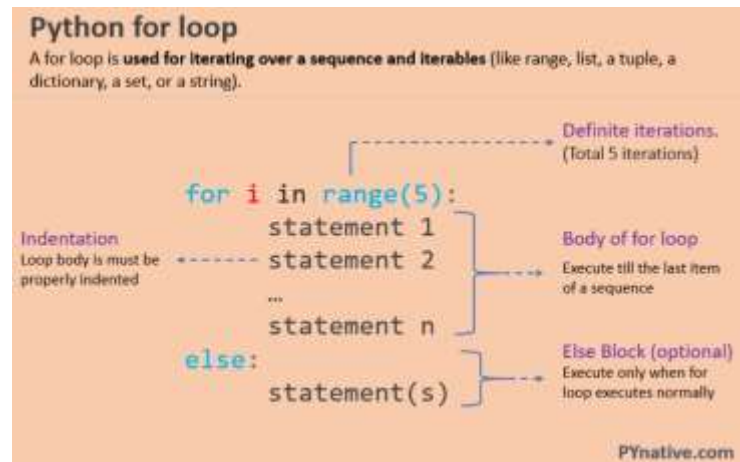
### #while/for loops

From my point of view, python "for" loop is a little annoying, comparing to other computer languages. Yet, after you will understand the general idea you will see that there is not really an issue there.

While the "for" loop isn't getting any "parameter" to count in, the loop is iterate over a sequence such as list, tuple, string, objects (range for example), etc.

General idea for choosing between 'for' or 'while' loop: 'for'- having fixed number of operations, 'while' otherwise. Keep in your mind - you can use the 'break' , 'continue' , and the 'if-else' conditions at this loops.

# Python Mastery for AI Enthusiasts: From Basics to Artificial



### #Examples:

```
for num in range(10):  
    print(num)
```

will print the numbers between 0 to 9.

```
1 numbers = [1, 2, 3, 4, 5]  
2 for i in numbers:  
3     square = i ** 2  
4     print("Square of:", i, "is:", square)
```

will goes over the numbers list and exponent the number by 2.

```
numbers = [4, 2, 5, 7, 0]  
for i, v in enumerate(numbers):  
    print('Numbers[' + i + '] = ' + v)
```

prints "numbers[i] = -the numbers itself-".

```
name = "Jessa"  
i = 0  
res = len(name) - 1  
while i <= res:  
    print(name[i])  
    i = i + 1
```

this will print the "Jessa" letter vertically.

## Grouping tasks - Functions

#Functions enable us to perform similar addition operations without writing the same add operation repeatedly. Following by the scheme/syntax:

```
def functionName (input parameters)

    statements

    return something

calling the function: funcationName(input parameters)
```

#Lambda functions – anonymous functions that don't need names. Lambda needs three thing: Lambda keyword, parameter (or bound variable), dunction body.

```
Adder = lambda x, y: x + y

Print (Adder(1,2))
```

## Pre-defined functions - Operations on numbers and strings

1. We can use prepared module that in order to make mathematics actions at our code.

```
Import math
Math.pow(4,3) – first number raised to the power of the seconed
Math.floor(5.2) – output 5
Math.ceil (5.2) – output 6
Math.fabs (-10) – absolute value returned
Math.log (5) – natural logarithm of 5
Math.log(5,7) – log of 5 number to the base of 7
Math.sqrt(25) – output float number 5.0
```

2. Strings functions

Str.functionName() – returns the string in sentence case, with the first letter chaptalize and the rest are small. Some examples:

```
str = "here I am"
# Capitalize the string
capitalized_str = str.capitalize()
print(capitalized_str) # This will return "Here i am"
# Count the number of occurrences of the letter 'e' in the string
count_e = str.count('e')
print(count_e) # This will return 2
# Find the index of the first occurrence of the letter 'h' in the string
index_h = str.find('h')
print(index_h) # This will return 0
```

---

---

```
str2 = " "
inter = ("I", "am", "awesome")
a = str2.join(inter)
```

```
print(a) # This will print "I am awesome"
```

## Lists

Lists can store multiple values of different data types like string, numbers, etc.

```
my_list = [4.3, "hi", 6, "Me", 78]
# Access the element at index 2 (which is "6")
print(my_list[2]) # This will display "6"
# Slice the list to create 'a', containing elements at indices 1, 2, and 3
a = my_list[1:4]
print(a)
# Slice the list to create 'b', containing all elements
b = my_list[:]

print(b)
# Slice the list to create 'c' from index 0 to 5
c = my_list[:5]
print(c)
# Update the element at index 3 (which is "Me") to 56
my_list[3] = 56
print(my_list)
# Append 44 to the end of the list
my_list.append(44) # List will be [4.3, "hi", 6, 56, 78, 44]
print(my_list)
# Delete the element at index 0 (which is 4.3)
del my_list[0] # List will be ["hi", 6, 56, 78, 44]
print(my_list)
# Remove the first occurrence of 6
my_list.remove(6) # List will be ["hi", 56, 78, 44]
print(my_list)
```

## Tuples

Tuples are similar to list, with one change – we cannot change the items inside the tuple. Second change is the definition of tuple parentheses (round brackets)

```
my_tuple = (2, 4, 5, 6, "hi", 7)
# Access the element at index 1 (which is 4)
print(my_tuple[1]) # This will print '4'
# Slice the tuple from index 1 to 4 (inclusive)
print(my_tuple[1:4]) # This will print (4, 5, 6, 'hi')
# Printing the entire tuple
print(my_tuple) # This will print (2, 4, 5, 6, 'hi', 7)
# Create 'c' by slicing the tuple from index 0 to 5
c = my_tuple[:5]
# Create 'd' by slicing the tuple from index 3 to the end
d = my_tuple[3:]
# Delete the entire tuple
del my_tuple # This deletes the entire tupleKeys and their value
```



# Python Mastery for AI Enthusiasts: From Basics to Artificial

Items in a dictionary are separated by a comma. Note that the last item doesn't have a comma following it. Keys and values are separated by a colon (:). Items in a dictionary are enclosed within curly brackets.

```
dict1 = {
    'name': 'xyz',
    'age': 25,
    'hobby': 'dancing'
}

# Access the value associated with the 'age' key (which is 25)
print(dict1['age'])
# Update the value associated with the 'name' key to 'lionel'
dict1['name'] = 'lionel'
# Add a new key-value pair 'profession': 'pilot'
dict1['profession'] = 'pilot'
# Remove the item with the key 'name'
del dict1['name']
# Remove the entire dictionary
del dict1
# Clear the dictionary (it will be empty)
dict1 = {}
```

## Module 2 – Python, advanced

### File handling

The built-in methods and dictionaries in python can handle two major types of files – text files (.txt) and Binary files (.bin). In python file processing takes place in the following order:

- a. Open a file that returns a file handle (File Object)
- b. Use the handle to perform read or write action
- c. Close the file

To open a file we use its built-in `open()` function. Following the following syntax

```
File object = open(file_name, access_mode)

#Access mode can be r,r+,w,w+,a,a+ (google it)
```

To read a file we will read it with 'r' mode. There are three functions which we can use to read the files in python.

- a. `Read()` – read the file to the end
- b. `Readline()` – read one entire line from the file
- c. `Readlines()` – return all the lines in a file in the format of a list where each element is a line in the file.

# Python Mastery for AI Enthusiasts: From Basics to Artificial

The `write()` method can write a string or sequence of bytes to a file. Any information inside the file will be overwritten.

To write a sequence without overwriting the data, we can use append mode.

## Importing python modules

Modules dividing the project into different smaller based on some features or components. File named as 'example.py' called as a module.

- a. We need to use the `import` keyword along with the desired module name.

Syntax: `import module_name`

- b. We can use the functions inside a module using a dot.

Syntax: `module_name.functionName()`

## Random module

```
import random
print('random number =>', random.random()) //float number between 0.0 to 1.0
print(random.randint(3,9)) //integer number between 3 to 9
print(random.randrange(start,stop, step – not mandatory) // generate integer
List = [10,6,4,11,1]
print(random.shuffle(List)) //print the list shuffled in order
//we can define weights to probability for numbers with choices function
Print(random.choices(List, weights=(10,50,90,20,30), k=5 {function returned k sized list})
```

## Statistic module

```
import statistics
```

```
Scores = [6,7,2,5,7,8,2,5,2,8]
a=statistics.mean(Scores) // Mean – the average of a set of numbers.
b=statistics.median(Scores) // Median – middle number, or midpoint of the data
b1=statistics.median_low(Scores) // value from the data that is low from the actual median
b2=statistics.median_high(Scores) // value from the data that is high from the actual median
c=statistics.mode(Scores) // mode – the value which occurs frequently
```

variance refers to the average of the square differences from the mean. In other words – how varied the data is?

```
Grades = [70,90,50,85,65,83,94]
Grades_mean = statistics.mean(grades)
Grades_variance = statistics.variance(Grades_mean)
deviation how much variation from the mean exists.
Stdevgrades = statistics.stdev(Grades)
```

# Python Mastery for AI Enthusiasts: From Basics to Artificial

## time module

```
import time

current_time = time.time() //set the current time to the parameter since epoch (1970's)

local_time = time.ctime(current_time) //set the day,date, clock time, year to local param
```

## Exceptions in python

Python program terminates as soon as it encounters an unhandled error. There are two types of errors:

- Syntax errors
- Logical errors (exceptions)

When we knowing and understanding the error, we can handle it in order to prevent crashing.

## Errors and exceptions

According to exceptions, python generates an exception that could be handled. It basically prevents the program getting crashed. Our code might not have any syntax error, still it can lead to exceptions when executed. Python tries to be very clear with the exceptions and line number, we just need to read it.

- zeroDivisionError – divide by zero
- ImportError – import module that isn't existis
- IndexError – if there is a list of 10 number and we want to reach 11st number
- IndentationError – indentation (spaces at the beginning of a code line) not specified properly.
- ValueError – arguments (the value that is sent to the function) have invalid values.
- TypeError – incorrect type of function or operation is applied to an object
- NameError -
- Exception – base class for all exception. If we're not sure witch exception can occur we can use it.

## Exception handling

By handling exception, we can provide an alternative flow of execution to avoid crushing. When exception occurs the python interpreter stops the current process. The "try" block execute the program, while the "except" block handle the exceptions.

Try:

```
#code lines...
Except NameError as e:
    #optional blocks
    #handling of exception (if required)
    Print("some error occurred")
```

# Python Mastery for AI Enthusiasts: From Basics to Artificial

## Finally and raise

The "finally" block contains code that will execute even if an exception occurs. Here's the syntax:

try:

```
# Code lines...
```

```
print(ourvar)
```

except:

```
# Optional blocks
```

```
# Handling of exception (if required)
```

```
print("ourvar isn't defined")
```

finally:

```
# Other code lines... (always executed)
```

```
print("Output is printed successfully")
```

## # Example: Raise exceptions

```
def print_five_items(data):
```

```
    if len(data) != 5: # Check if the number of elements in the data is not equal to 5
```

```
        raise ValueError("The argument must have five elements") # Raise a ValueError if the condition is not met
```

```
    for item in data:
```

```
        print(item)
```

```
data = [10, "apple", 3.14, "banana", True]
```

```
print_five_items(data)
```

## Regular expressions – RegEx

Regex is an expression which contains a sequence of characters that define a search pattern. They are extremely useful when extracting info from text, log files, spreadsheets, and documents.

`\d` - matches any decimal digit (like 0 to 9)

`\D` - match any non decimal digit (not 0 to 9)

`\s` - match any white space character.

`\S` - match any not white space character

`\w` - match any alphanumeric character (set of all numbers and letters in both lower and uppercase)

`\W` - match any non-alphanumeric character

```
import re
```

```
String="hello my name is Michael"
```

```
Print(re.findall(r"Michael",String))
```

```
String2="Hello I am 2 schnitzel and 49 pastrams"
```

```
Print(re.findall(r"\d",String)) \ return '2' , '4' , '9'
```

```
Print(re.findall(r"\d+",String)) \ return '2' , '49'
```

# Python Mastery for AI Enthusiasts: From Basics to Artificial

```
If re.search("awesome","isn't this is awesome?")
    Print("awesome detected") \\ print awesome because the word dedcted in string
Result=re.split(r"s","awesome")
    Print(Result) \\ output will be ['awe' , 'ome']
```

## Database connection

MySQL is an open source relational database management system based on Structured Query Language. The server handles all of the database infrastructure (or commands). Commands sent to MySQL server via the MySQL client. The process will be like:

- MySQL creates database, defining the relationship of each table.
- Clients can make requests by typing specific SQL statement on MySQL.
- The server application will respond with the requested info and it will appear on client side

download MySQL

download MySQL.connector //check with the code `import mysql.connector` is running without issues. If yes than you successfully installed those.

```
Import mysql.connector as mysql
# via 'connect()' method we need three parameters 'host', 'user', passwd:
db=mysql.connect(
    host="ip address when MySQL installed"
    user="your username"
    passwd="your password")
print(db) #it will print a connection object if everything is fine
```

```
import mysql.connector
from mysql.connector import Error #show us when we failed to connect database or if any
other database error occurred while working with the database
```

`connection.is_connected()` #method of the MySQL connection class through which we can verify our python application connected to MySQL.

`Connection.cursor()` #returns a cursor object. Using cursor object give us the possibility to execute SQL queries. The object can execute operation such as SQL statements.

`Cursor.close()` #close the cursor object. Once we close the cursor object we can not execute any SQL statements.

`Connection.close` #closing the database connection

### Tables

In order to store information on the database we need to create a table. It also required to select our database first and then created tables inside that database. Rows are called tuples, Columns called attributes.

```
Import mysql.connector
db = mysql.connector.connect(
    host="ip_address"
    user="your username"
    passwd="your password"
```

```
        database="your databse name"
    )
    Cursor = db.cursor()
    Cursor.execute("CREATE TABLE people(name varchar(50), age int, City varchar(50))")
```

If code running without errors, then we created a table. Now we can run SELECT query in SQL server to display the table with: `SELECT * FROM database_name.dbo.people`

*CRUD operations*

C=create #inserting record

R=read #reading the records

U=update #update record with new value

D=delete #delete record from databse

```
Import mysql.connector
db_connection = mysql.connector.connect( #build our MySQL connection
    host="ip_address"
    user="your usernanem"
    passwd="your password"
    database="your databse name"
)
My_database = db_connection.cursor()
Sql_statements = "SELECT * FROM My_database) #stored MySQL SELECT query in var
My_database.execute(sql_statement) #executrion of our MySQL Query
Output = my_database.fetchall() #return all the result's sets as a list of tuples
for x in output #print out the tuples one by one, from Output.
    print(x)
```

MySQL statement to update a table data row: `UPDATE TABLE_NAME SET parameter_name="new value" where parameter2_name = "new value"`

MySQL statement to delete a table data row: `DELETE FROM TABLE_NAME where parameter_name="existing value"`

```
Import mysql.connector
db_connection = mysql.connector.connect( //build our MySQL connection
    host="ip_address"
    user="your usernanem"
    passwd="your password"
    database="your databse name"
)
My_database = db_connection.cursor()
Sql_query = "DELETE FROM TABLE_NAME where paramet_name="exsiting_ value""
My_database.execute(sql_query)
Db_connections.commit() // don't forget using commit() function! Otherwise,
changes will not be saved.
```

```
print("Rows deleted successfully")
```

## GUI development - tkinter

Cursor – indicated the places where the system will accept input next.

Selection – refers to a list of items to which a user will apply an operation.

Adjustment handle – serves as the indicator of a drag and drop operation.

We're going to have a top-level window object that acts like container where we put all of our widgets in.

```
import tkinter as tk
# Create a top-level window object as the main application window
win = tk.Tk()
# Set the window's size
win.geometry("200x200")
# Create a button widget
btn = tk.Button(win, text="Click Me")
# Pack the button using the pack() geometry manager
btn.pack()
# Start the main event loop to run the application
win.mainloop()
```

We can also create labels and checkbuttons as the following:

```
import tkinter as tk
# Create the main application window
root = tk.Tk()
root.geometry("400x400")
# Create a label and display it
label1 = tk.Label(root, text="Hi, welcome to GUI")
label1.pack()
# Create and display checkbuttons
c = tk.Checkbutton(root, text="Python")
c.pack()
c1 = tk.Checkbutton(root, text="C++")
c1.pack()
c2 = tk.Checkbutton(root, text="C")
c2.pack()
# Start the main event loop
root.mainloop()
```

The code displays three simple checkboxes with text beside them. It doesn't have functionality because there are no function triggers.

Checkbuttons have various methods you can use:

1. `invoke()` - Invokes the method inside the checkbutton.

2. `select()` - Turns on the checkbox.
3. `deselect()` - Turns off the checkbox.
4. `toggle()` - Toggles between different checkboxes.
5. `flash()` - Flashes between active and normal colors.

## Object oriented programming

### OOP's and related concepts

In order to evade spaghetti code, we can use object-oriented programming (OOP). OOP's is about breaking the problem down into rational smaller pieces – associated with the concept of class, objects, inheritance, polymorphism, abstraction, encapsulation, etc.

### Classes and objects

A class is a blueprint for creating objects, and it encapsulates the details and behaviors shared by multiple objects. For example, consider an Animal class, which can contain subclasses like Dog, Cat, and Elephant. The Dog class can further include breeds like Poodle, Bulldog, and Amstaph. Each subclass and breed can have specific attributes and behaviors, defining their unique characteristics.

Example for creating simple class:

```
Class MyClass:
```

```
    a=20
```

```
    b=40
```

```
print(MyClass.40)
```

In this example, we define a class called MyClass with attributes 'a' and 'b'. We can access these attributes using the class name.

an object is an instantiation of a class.no memory or storage is allocated. Please examine the following example:

```
class Crow:
```

```
    species = "bird"
```

```
    def __init__(self, name, age):
```

```
        self.name = name
```

```
        self.age = age
```

```
blu = Crow("Blu", 10)
```

```
woo = Crow("Woo", 15)
```

```
print("Blu is a {}".format(blu.__class__.species))
```

```
print("Woo is also a {}".format(woo.__class__.species))
```

```
print("{} is {} years old".format(blu.name, blu.age))
```

```
print("{} is {} years old".format(woo.name, woo.age))
```

In this example, we create a Crow class with class attribute species. We also define an `__init__` method to initialize object-specific attributes like name and age. We then create two Crow objects, 'blu' and 'woo', and print their characteristics.



# Python Mastery for AI Enthusiasts: From Basics to Artificial

Constructors:

`__class__` allows you to access class attributes directly.

`__init__` is an initializer method that runs when an object is created, allowing you to set object-specific attributes.



## Multi-threading in python

Threads are units of execution, each with its independent set of instructions. A thread encompasses a starting point, a sequence of execution, and yields a result. Consider a scenario, such as playing a game like FIFA. Although it's a single process, multiple threads run in parallel, each responsible for different aspects like commentary, user input, player movements, and more.

The capability of a process to execute multiple threads concurrently within an application is termed "multithreading." In Python, when we run a program, a primary thread is created. During the process, additional threads can be initiated, which are referred to as secondary threads.

It's important to note that multithreading is only viable when threads do not depend on each other. In such cases, they can run in parallel, enhancing the overall performance and responsiveness of the application.

### Introduction to multithreading - Threading module

#### Thread Methods and Concepts:

- `threading.activeCount()`: Returns the number of currently active thread objects. This is equivalent to the length of the list returned by `enumerate()`.
- `threading.currentThread()`: Returns the current thread object, allowing you to access information about the currently executing thread.
- `threading.enumerate()`: This method is similar to `activeCount()` and provides a list of all currently alive thread objects.
- `threading.main_thread()`: Returns the main thread object. The main thread is the one that the Python interpreter started when your program began.
- `threading.stack_size([size])`: This method returns the stack size used when creating a new thread. It can also be used to set the stack size for new threads.

#### Thread Synchronization and Control:

- Condition: A condition variable allows one thread (Thread X) to wait for another thread (Thread Y) to complete a certain "condition" before proceeding.
- Event: Events are used to synchronize multiple threads. Threads can wait until a specific event is set or cleared by other threads.
- Timer: Timers can be used to delay the execution of a thread until a specified time has passed.
- Barrier: A barrier is a synchronization primitive that requires a specified number of threads to arrive before they can all continue execution together.

#### Thread Class Methods:

# Python Mastery for AI Enthusiasts: From Basics to Artificial

- `run()`: This method acts as the entry point for the thread's execution logic. You should override it in your custom thread class to define what the thread should do.
- `start()`: The `start()` method is used to initiate the thread and begin executing its `run()` method. It creates a new thread of execution.
- `isAlive()`: This method checks whether the thread is still executing or has terminated.
- `getName()`: Returns the name of the thread.
- `setName(name)`: Sets the name of the thread. This can help identify threads more easily when debugging or monitoring their behavior.

These methods and concepts are part of Python's threading module, which provides powerful tools for working with threads and controlling their behavior in a multi-threaded application.

## Threading example

```
import threading
def print_hello(num):
    print("You are an employee number:", num)
# Create a thread and specify the target function and its arguments
t1 = threading.Thread(target=print_hello, args=(10,))
# Start the thread
t1.start()
# Wait for the thread to complete using join
t1.join()
# Print "end" when the thread has finished
print("End")
```

### Some notes:

- comma after 10 to create a single-item tuple)
- We create a Thread object `t1` and specify the target function `print_hello` and its arguments `(10,)`
- (note the comma after 10 to create a single-item tuple).
- We use `t1.join()` to wait for the thread to complete.

## Module 3 – My first code example

After gaining an understanding of the mentioned subjects, let's delve into them by exploring my **first** Python code. Later on. **This initial project involved to be automated on-going process having the possibility to command and communicate with Elmo MAS/3AXIS servo kit, store and process the data, with process monitoring via AI predictors. SO!! Everyone can do anything!! Keep it in your mind!**

### Brief words

With designed workflow, my application will need to perform the following tasks:

- Receiving a data stream via UDP protocol. This data stream originates from an external servo motor, which transmits its encoder values along with current and voltage consumption information.
- Assigning a timestamp to each data object within the specified data stream.

# Python Mastery for AI Enthusiasts: From Basics to Artificial

- Storing this timestamped data in the main Excel spreadsheet, which comprises multiple tabs for organized data storage.
- Generating real-time plots to visualize this data stream.
- Conducting data manipulation to facilitate various AI-related tasks, which I'll elaborate on later.

Code will execute the following operations:

- Receive a data stream based on user input, consisting of numeric values representing encoder data and information on current and voltage consumption. This user input is referred to as "sensor\_type."
- Apply timestamps to each data entry within the specified data stream.
- Save this timestamped data into the primary Excel sheet with its multiple tabs.
- Generate real-time plots to visualize the data stream.

More detailed explanations for each of these code blocks in subsequent subjects.

Sample code:

```
import pandas as pd
import time
import matplotlib.pyplot as plt

# Create empty DataFrames to store data for each sensor type
data = pd.DataFrame(columns=["ID", "sensor_type", "value", "timestamp"])
current_data = pd.DataFrame(columns=["ID", "value", "timestamp"])
voltage_data = pd.DataFrame(columns=["ID", "value", "timestamp"])
encoder_data = pd.DataFrame(columns=["ID", "value", "timestamp"])

# Mapping of sensor type abbreviations to full names
sensor_type_mapping = {
    "c": "current",
    "v": "voltage",
    "e": "encoder"
}

# Function to process and add data to DataFrame's
def process_input(user_input):
    parts = user_input.split()
    if len(parts) % 2 != 0:
        print("Invalid input format. Please provide pairs of [sensor_type] [float_number].")
        return

    timestamp = time.strftime("%Y-%m-%d %H:%M:%S")

    for i in range(0, len(parts), 2):
        sensor_type = parts[i]
        value = float(parts[i + 1])
```

```
if sensor_type not in sensor_type_mapping:
    print("Invalid sensor type. Please use 'c' for current, 'v' for voltage, or 'e' for encoder.")
    return

full_sensor_type = sensor_type_mapping[sensor_type]
data.loc[len(data)] = [len(data) + 1, full_sensor_type, value, timestamp]

if full_sensor_type == "current":
    current_data.loc[len(current_data)] = [len(current_data) + 1, value, timestamp]
elif full_sensor_type == "voltage":
    voltage_data.loc[len(voltage_data)] = [len(voltage_data) + 1, value, timestamp]
elif full_sensor_type == "encoder":
    encoder_data.loc[len(encoder_data)] = [len(encoder_data) + 1, value, timestamp]

print("Data added successfully.")

# Initialize a figure and subplots
fig, (ay_current, ay_voltage, ay_encoder) = plt.subplots(3, 1, sharex=True)
fig.suptitle("Real-Time Sensor Data")

# plot updating function
def update_plots(i):
    ay_current.clear()
    ay_voltage.clear()
    ay_encoder.clear()

    ay_current.plot(current_data["timestamp"], current_data["value"], marker='o', linestyle='-',
color='b')
    ay_current.set_ylabel("Current")

    ay_voltage.plot(voltage_data["timestamp"], voltage_data["value"], marker='o', linestyle='-',
color='r')
    ay_voltage.set_ylabel("Voltage")

    ay_encoder.plot(encoder_data["timestamp"], encoder_data["value"], marker='o', linestyle='-',
color='g')
    ay_encoder.set_ylabel("Encoder")

    plt.xticks(rotation=45)

# Main loop
while True:
    user_input = input("Enter sensor data (e.g., 'c 2.5 v 10.3 e 100'): ")

    if user_input.lower() == "exit":
        # Save the data to an Excel file with tabs
        with pd.ExcelWriter("sensor_data_with_tabs.xlsx") as writer:
            data.to_excel(writer, sheet_name="main", index=False)
            current_data.to_excel(writer, sheet_name="current", index=False)
            voltage_data.to_excel(writer, sheet_name="voltage", index=False)
```

# Python Mastery for AI Enthusiasts: From Basics to Artificial

```
encoder_data.to_excel(writer, sheet_name="encoder", index=False)

print("Data saved to 'sensor_data_with_tabs.xlsx'. Exiting...")
#exit
    break

process_input(user_input)

# Update the plots using the function that i created
update_plots(0)

plt.pause(0.5) # Pause to give time for the plot to update
```

## explanation:

1. creating empty 'Data Frame' using the help of 'pandas' extension. CMD pip install pandas OR pip3 install pandas in the terminal or command line. Data frame contain four columns (ID, sensor\_type aka data\_type, input value, timestamp).
  - o data frame is 2 dimension labeled data structure with potentially different types.Also, additional three datasets prepared for each sensor\_type for future implentatio into our excel file with additional three tabs.

```
# Create empty DataFrames to store data for each sensor type
data = pd.DataFrame(columns=["ID", "sensor_type", "value", "timestamp"])
current data = pd.DataFrame(columns=["ID", "value", "timestamp"])
voltage data = pd.DataFrame(columns=["ID", "value", "timestamp"])
encoder data = pd.DataFrame(columns=["ID", "value", "timestamp"])
```

2. user input defined as "c" for "current" , "v" for "voltage" , "e" for "encoder". The next "sensor\_type\_mapping {}" object define. Reminder: after that object defined, we can call him by his name.

```
# Mapping of sensor type abbreviations to full names
sensor_type_mapping = {
    "c": "current",
    "v": "voltage",
    "e": "encoder"
}
```

3. the following function defined as the core of the code – processing the user input that I decide he will be as following "c VALUE v VALUE e VALUE". Function name defined as process\_input and its getting our user input.

'Parts' separate the user input with split() function.

- o The split() method splits a string into a list
- o The len() function returns the number of items (length) in an object.
- o If one of the parts contain cVALUE with no backspace than % 2 reminder will be greater than zero.
- o We can see it in operation here:

```
Enter sensor data (e.g., 'c 2.5 v 10.3 e 100'): c100
Invalid Input Format. Please provide pairs of (sensor_type) (Plot_number).
Enter sensor data (e.g., 'c 2.5 v 10.3 e 100'): v200 v 100
Invalid Input Format. Please provide pairs of (sensor_type) (Plot_number).
Enter sensor data (e.g., 'c 2.5 v 10.3 e 100'): e 200 v 100
Data added successfully.
Enter sensor data (e.g., 'c 2.5 v 10.3 e 100'): exit
Data saved to 'sensor_data_with_tabs.xlsx'. Exiting...

Process finished with ctrl code 0
```

'time.strftime' giving the timestamp the appropriate time.

# Python Mastery for AI Enthusiasts: From Basics to Artificial

Next 'for' condition insert the letter to 'sensor\_type' and the VALUE into 'float number'. Additional logic condition preformed that letter are inputted correct (c, v, e only). I map the digits into full phrases.

- data.loc allows the return of specifies rows for the dataframes that mentioned earlier. When its completed we can see printed output onscreen.

```
def process_input(user_input):
    parts = user_input.split()
    if len(parts) % 2 != 0:
        print("Invalid input format. Please provide pairs of [sensor_type] [float_number].")
        return

    timestamp = time.strftime("%Y-%m-%d %H:%M:%S")

    for i in range(0, len(parts), 2):
        sensor_type = parts[i]
        value = float(parts[i + 1])

        if sensor_type not in sensor_type_mapping:
            print("Invalid sensor type. Please use 'c' for current, 'v' for voltage, or 'e' for encoder.")
            return

        full_sensor_type = sensor_type_mapping[sensor_type]
        data.loc[len(data)] = [len(data) + 1, full_sensor_type, value, timestamp]

        if full_sensor_type == "current":
            current_data.loc[len(current_data)] = [len(current_data) + 1, value, timestamp]
        elif full_sensor_type == "voltage":
            voltage_data.loc[len(voltage_data)] = [len(voltage_data) + 1, value, timestamp]
        elif full_sensor_type == "encoder":
            encoder_data.loc[len(encoder_data)] = [len(encoder_data) + 1, value, timestamp]

    print("Data added successfully.")
```

4. initiate figures with common x-axis and separate three y-axis's.
5. creating a function for updating the plot named 'updating\_plots'.
6. Main loop:
  - Insert the input into 'user\_input'
  - 'exit' – saves the data and exit the program.
  - Save data to excel after 'exit' execute . Pay attention to the names. For example: data.toExcel means that our section 3 data.loc will kept into excel.
  - If 'exit isn't execute than our section 3 'process\_input' function start to act.
  - After the input processed , our section 5 'updating\_plots' function start to act.
  - Small pause for refreshing and updating.
  - Loop continue.

# Python Mastery for AI Enthusiasts: From Basics to Artificial

```
while True:
    user_input = input("Enter sensor data (e.g., 'c 2.5 v 10.3 e 100'): ")

    if user_input.lower() == "exit":
        # Save the data to our Excel file with tabs
        with pd.ExcelWriter("sensor_data_with_tabs.xlsx") as writer:
            data.to_excel(writer, sheet_name="main", index=False)
            current_data.to_excel(writer, sheet_name="current", index=False)
            voltage_data.to_excel(writer, sheet_name="voltage", index=False)
            encoder_data.to_excel(writer, sheet_name="encoder", index=False)

        print("Data saved to 'sensor_data_with_tabs.xlsx'. Exiting...")
        #exit
        break

    process_input(user_input)

    # Update the plots using the function that i created
    update_plots(0)

    plt.pause(0.5) # Pause to give time for the plot to update
```

## Module 4 - AI in python

### Introduction to machine learning

Machine learning is a subfield of artificial intelligence that focuses on creating algorithms that enable computers to learn from and make predictions or decisions based on data.

General explanation: Clean the data, select proper algorithm for prediction model, train your model to understand patterns within the data.

We going to learn and explore examples of ML, python AI libraries (scipy, scikit), Apply the appropriate form of regression to a data set technique, apply an appropriate classification method, use the correct clustering algorithms on different data sets, explain how recommendation system work and more.

- Regression\estimation techniques used for predicting a continues value (for example: Co2 emission from car engine, house price based on several characteristics).
- A classification technique used for predicting the class OR category of a case (for example: if a cell is benign OR infectious. whether OR not a customer will churn).
- Clustering groups of cases, for example, can be used for customer segmentation OR find similar patients\customer.
- Association technique is used for finding items OR events that often occur (for example: grocery items that are usually bough by particular customer).
- Anomaly detection is used to discover abnormal and unusual cases (for example: credit card fraud detection).



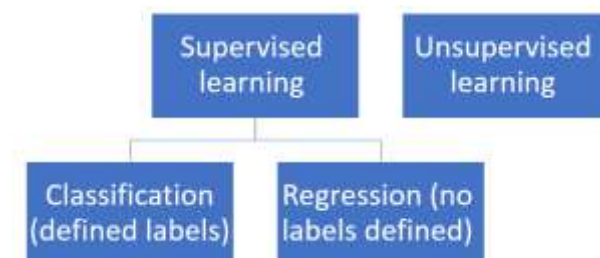
# Python Mastery for AI Enthusiasts: From Basics to Artificial

- Sequence mining used for predicting the next event (for example: website click stream)
- Dimension reduction is used to reduce the size of data.
- Recommendation system associated people references with other who have similar characteristics.

Definition that will make us understand the differences between artificial intelligent (AI), machine learning (ML), Deep learning (DL):

- Artificial intelligent: Tries to make computer smarter. general field within the scope of Computer vision, Language processing, Creativity and Summarization.
- Machine learning: branch of AI. Teaches the computer to solve problem by looking at hundred's examples, learning from it, and solve the problem. ("the autonomous acquisition of knowledge through the use of computer programs)
- Deep learning: specific field of ML. computer can actually learn and make intelligent decisions

We have two main ways to execute a task – supervised or unsupervised.



- SUPERVISE: LEARNING WITH THE HELP OF TEACHER. Supervise means to observe and direct the execution of a task. We teach (load\train) the model with knowledge so that we can have it predict future instances. How specific? We train the model with some data from labeled dataset (we already know the class of each data). there are two types of supervised learning techniques. The classification is the process of predicting a discrete class label OR category. The regression is the process of predicting a continues value (predict Co2 emission of a new car engine). **[we have ML algorithm for classification and regression] , [supervised learning requires at least one input attribute]**
- UNSUPERVISED: LEARNING FROM ENVIROMENT. We do not supervise the model, we will let the model work on his own and discover information may not visible to the human eye. The unsupervised algorithm trains on the dataset and draws conclusion on unlabeled data – we will use this when we know little to none about the data, or the outcomes that are to be expected (for example: density estimation, dimension reduction, market basket analysis, **clustering**). Con: we have fewer evaluation methods that can be used to ensure that the outcome of the model is accurate. Also the unsupervised learning created a less controllable environment

## Simple linear regression

General idea is to use historical data and make a model via regression. This is fundamental ml technique for modeling and relating between TWO variables. Independent (predictor) and dependent (target).

Simple example:



# Python Mastery for AI Enthusiasts: From Basics to Artificial

```
from sklearn.linear_model import LinearRegression
import numpy as np

# Sample data
X = np.array([1, 2, 3, 4, 5]).reshape(-1, 1)
y = np.array([2, 4, 5, 4, 5])

# Create a Linear Regression model
model = LinearRegression()

# Fit the model to your data
model.fit(X, y)

# Make predictions
new_data = np.array([6]).reshape(-1, 1)
predictions = model.predict(new_data)
```

we will try to learn about Linear, Non-linear, simple and multiple regression. Also, how to evaluate a regression model and calculate his accuracy. In regression there are two types of var's – dependent (X axis) and one or more independent (Y axis). The dependent is our goal\target to predict – **important thing to understand is that our X value need to be continuous and cannot be discrete value.**

Simple regression -  $Y=a+bX$  – one independent var predict one dependent var. it can be linear or non-linear. Linearity depend if there is two or more Y vars in order to predict X.

Residual error – the distance between the line to his actual point. The mean (MSE) of all residual error shows how the line fits the whole dataset.

How can we estimate the accuracy of model?

1. You train the model on the entire dataset then you test it using a portion of the dataset
2. For better results – we can use test\split approach. Portion of the data is use for building model, and the rest use for testing. [create a mask to select random rows using np.random.rand() function]

Training accuracy - is the percentage of correct predictions that our model makes when using the test dataset. High training accuracy isn't necessarily a good thing (over fitted data – capture noise and produce a non-generalized model).

Out of sample accuracy – the percentage of correct predictions that the model makes on data that model has not been trained on. We want it high of course.

## Evaluation metrics of model

Each of these metrics can be used for evaluation. The choosing dependents on the type of model, your dataset, and domain of knowledge.

- Mean absolute error – average error, the easiest to apply and understand.
- Mean squared error – mean of squared error. Increasing larger errors on compression to smaller ones.
- Root mean squared error – most popular. Easy to relates its information.

# Python Mastery for AI Enthusiasts: From Basics to Artificial

- R squared – isn't an error. It represents how close the data values fitted to regression line. We want it high.

In order to build a model we will do the following steps:

1. Import needed package (such as matplotlib, pandas, pylab, numpy,
2. Download\export\storage the data .csv.
3. Reading the data – `pd.read_csv("name.csv")`
4. Data explorations –
5. Creating train and test dataset – creating mask `msk` – `np.random.rand(df)`
6. Modeling – using sklearn package to model data.
7. Evaluation

## Multiple linear regression

This specific regression is extension for the simple linear regression and now we will having several predictors. Of course its more realistic and powerful.

Simple example:

```
from sklearn.linear_model import LinearRegression
import numpy as np

# Sample data with multiple features
X = np.array([[1, 2], [2, 3], [3, 4], [4, 5], [5, 6]])
y = np.array([3, 6, 7, 6, 10])

# Create a Multiple Linear Regression model
model = LinearRegression()

# Fit the model to your data
model.fit(X, y)

# Make predictions
new_data = np.array([[6, 7]])
predictions = model.predict(new_data)
```

at this example we're using X1,X2 to predict y.

It can be used when we would like to identify the strength of the effect that the independent var have one the dependent vat. Second it can be use to predict the impact of change.

How to find the values of the coefficients:

1. Under 10K rows – ordinary least squares.
2. Optimization algorithm (such as gradient descent) to find the best parameters.

Should we use all the fields in out dataset? No. Adding too many independent variables without any theoretical justification may result in an overfit model.

**REMEMBER!** Multiply regression is a specific type of linear regression. Therefore, a linear relationship between dependent vars and each of our independent var should exists.

How can I check linearity? You can use scatter plots and then visually check for linearity. If relationship isn't linear than we need to use non-linear regression.

# Python Mastery for AI Enthusiasts: From Basics to Artificial

## Sample code for multi linear regression

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

from sklearn.metrics import mean_squared_error, r2_score

# Step 1: Import the Excel file

excel_file = pd.ExcelFile("samples3.xlsx") # Replace with your file path

# Step 2: Load data from each sheet with specific columns

position_df = excel_file.parse("position_values", usecols=["ID", "value1", "dx", "x1", "dx1", "x11", "fft"])

position_df = position_df.drop(columns=["dx", "dx1", "x11"]) # Exclude irrelevant data

speed_df = excel_file.parse("speed_values", usecols=["ID", "value2", "dv", "v1", "fft"])

speed_df = speed_df.drop(columns=["dv", "v1"]) # Exclude irrelevant data

current_df = excel_file.parse("current_values", usecols=["ID", "value3"])

# Step 3: Merge the data frames based on the 'ID' column

merged_df = position_df.merge(speed_df, on="ID")

merged_df = merged_df.merge(current_df, on="ID")

# Step 4: Data Preprocessing

X = merged_df.drop(columns=["value3"]) # Exclude the dependent variable

y = merged_df["value3"] # Dependent variable

# Step 5: Split the data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 6: Build a Multiple Regression Model

model = LinearRegression()

model.fit(X_train, y_train)

# Step 7: Evaluate the Model

y_pred = model.predict(X_test)
```

# Python Mastery for AI Enthusiasts: From Basics to Artificial

```
# Calculate out-of-sample accuracy (Root Mean Squared Error)
```

```
rmse = mean_squared_error(y_test, y_pred, squared=False)
```

```
# Calculate R-squared
```

```
r_squared = r2_score(y_test, y_pred)
```

```
# Residual errors
```

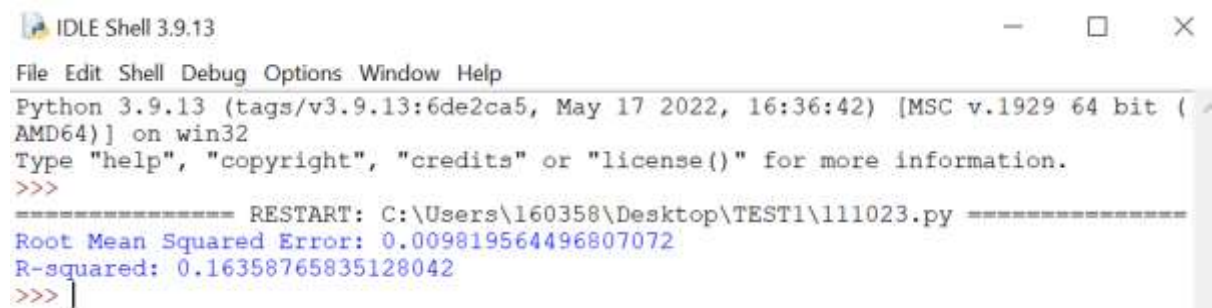
```
residuals = y_test - y_pred
```

```
# Print the evaluation metrics
```

```
print(f"Root Mean Squared Error: {rmse}")
```

```
print(f"R-squared: {r_squared}")
```

output:



```
IDLE Shell 3.9.13
File Edit Shell Debug Options Window Help
Python 3.9.13 (tags/v3.9.13:6de2ca5, May 17 2022, 16:36:42) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\Users\160358\Desktop\TEST1\111023.py =====
Root Mean Squared Error: 0.009819564496807072
R-squared: 0.16358765835128042
>>> |
```

A positive R-squared indicates that the model is explaining a portion of the variance in the data, and the lower RMSE suggests improved accuracy.

While these results are an improvement, there is still room for further optimization

## Decision tree

Decision tree is a popular algorithm for both regression and classification tasks. This mimics the human decision-making process and are easy to interpret.

It's a flow chart & structure in which internal nodes represent tests on an attribute, each branch represents the outcome of the test, and each leaf node represents a class label. It can also be used for classification.

# Python Mastery for AI Enthusiasts: From Basics to Artificial

```
from sklearn.tree import DecisionTreeRegressor
import numpy as np

# Sample data
X = np.array([1, 2, 3, 4, 5]).reshape(-1, 1)
y = np.array([2, 4, 6, 4, 6])

# Create a Decision Tree model for regression
model = DecisionTreeRegressor()

# Fit the model to your data
model.fit(X, y)

# Make predictions
new_data = np.array([6]).reshape(-1, 1)
predictions = model.predict(new_data)
```

In this example, we are using a decision tree for regression to predict y based on a single feature x.

## Classification

While regression was about predicting continues values, classifications brings us the concept of discret classes – opening up new possibilities. The goal is to assign items to PREDIFINED categories.

Simple example:

```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Sample data
X, y = your_classification_data

# Split your data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Create a Decision Tree Classifier model
model = DecisionTreeClassifier()

# Fit the model to your training data
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
```

In this example we are implanting several new things. First of all we're splitting the data into test and training (80%-20%). We're using decision tree classifier to classify data into predefined categories. Finally, we're trying to evaluate the accuracy of the prediction (will explained later).

Classification is a supervised learning approach that categorized or classifying some unknow items into a discrete set of classes. Classification attempts to learn the relationship between a set of feature variables and a target variable.

# Python Mastery for AI Enthusiasts: From Basics to Artificial

How it works? Given a training data points along with the target labels classification determines the class label for an unlabeled test case,

Classification algorithms: K-Nearest-Neighbor, Decision trees, Logistic regression, neural networks, Support-Vector-Machines.

## KNN

find value of K in KNN algorithm to predict the segmentation of customer (for example). In other words KNN label new points into an existing bunch of labeled points. It works as follows:

1. Pick a value of K
2. Calculate the distance of unknown case from all cases (Euclidean distance)
3. Select the K observations in the training data that "nearest" to the unknown data point.
4. Predict the response of the unknown data point using the most popular response value from the k nearest neighbor.

How to select the correct K? choose K=1 and then use the training part for refining the K.

How to compute the similarity between cases? Manhattan distance to calculate the distance between two points (Euclidean distance)

## Evaluation matrices for classifiers

TBD

## Logistic regression

Another classification algorithm used, when the dependent variable is binary (1,0 \ yes, no). despite his name this is not regression type technique, it's specifically designed for classification tasks.

Example:

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Sample data
X, y = your_classification_data

# Create a logistic Regression model
model = LogisticRegression()

# Fit the model to your data
model.fit(X, y)

# Make predictions
y_pred = model.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
```

Scikit-learn provides several options for configuring logistic regression models:

## Regularization:

# Python Mastery for AI Enthusiasts: From Basics to Artificial

- **penalty:** You can choose between 'l1' (Lasso) and 'l2' (Ridge) regularization, or you can set it to 'none' for no regularization.
- **C:** This is the inverse of the regularization strength, with smaller values resulting in stronger regularization.

## Solver:

You can choose the solver algorithm for optimization, such as 'lbfgs,' 'liblinear,' 'newton-cg,' or 'sag.' The choice of solver affects the algorithm used to find the optimal coefficients.

## Class Weights:

You can assign different weights to classes to address class imbalance issues. For example, you can set it to 'balanced' to automatically adjust the weights.

## Multiclass Classification:

Scikit-learn's logistic regression can be used for multiclass classification using the 'multinomial' option for the 'multi\_class' parameter.

## Types of Independent Variables:

In Python's scikit-learn, you can use various types of independent variables, including:

- **NumPy Arrays:** You can use NumPy arrays for both numeric and categorical data.
- **Pandas DataFrames:** If you're working with tabular data, Pandas DataFrames are commonly used for logistic regression.
- **Categorical Data Encoding:** Scikit-learn provides tools for encoding categorical variables using techniques like one-hot encoding.

## When to Choose Logistic Regression:

Logistic regression is a good choice when:

- **Binary Classification:** You have a binary classification problem where the outcome is one of two classes (e.g., yes/no, spam/not spam).
- **Interpretability:** You want a model that provides interpretable results, as logistic regression coefficients can be directly linked to the influence of features on the prediction.
- **Feature Importance:** You want to understand which features have the most impact on the classification decision.
- **Linear Relationship:** You believe that the relationship between features and the log-odds of the outcome is approximately linear. If not, you may need to consider more complex models like decision trees or neural networks.
- **Speed and Efficiency:** Logistic regression is relatively fast and can handle large datasets efficiently, making it a good choice for many real-world problems.
- **Scalability:** It can be a useful choice in cases where you need to quickly prototype a solution and later explore more complex models if necessary.

## Support vector machine

SVM is additional tools for classification and regression tasks. They work by finding a hyperplane that maximize the separation between different classes. SVM's really valuable for complex classifcaion problems where other methods might struggles.

```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Sample data
X, y = your_classification_data

# Create a Support Vector Machine model
model = SVC()

# Fit the model to your data
model.fit(X, y)

# Make predictions
y_pred = model.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
```

Options in scikit-learn's Support Vector Machine (SVM):

Scikit-learn's SVM implementation provides several options for configuring your SVM model:

- Kernel Function: You can choose the kernel function to be used, such as 'linear,' 'poly,' 'rbf' (Radial basis function), or 'sigmoid.' The choice of kernel influences the decision boundary shape.
- Regularization (C): This parameter controls the trade-off between maximizing the margin and minimizing the classification error. Smaller values of C result in a wider margin but may allow some misclassifications, while larger values of C reduce the margin but aim to classify all training examples correctly.
- Class Weights: You can assign different weights to classes to handle class imbalance. This is useful when one class has significantly fewer samples than the other(s).
- Support Vectors: After training the model, you can access the support vectors, which are the data points closest to the decision boundary. These vectors are crucial for making predictions.
- Multiclass Classification: SVMs can be extended to multiclass classification using techniques like one-vs-one (OVO) or one-vs-the-rest (OvR).

### Types of Independent Variables:

- Numeric Variables: You can use numeric features for SVM, such as age, income, or temperature.
- Categorical Variables: Categorical features can be used, but you'll need to perform suitable encoding, such as one-hot encoding.
- Scaling: It's often recommended to scale or normalize numeric features to ensure that they have a similar impact on the decision boundary. Scikit-learn provides tools like StandardScaler for this purpose.

### When to Choose Support Vector Machine (SVM):



# Python Mastery for AI Enthusiasts: From Basics to Artificial

- Binary or Multiclass Classification: SVM is versatile and can be used for both binary and multiclass classification tasks.
- Large or Small Data: It works well with both small and large datasets, thanks to its ability to maximize the margin between classes.
- Non-linear Separation: SVM can handle non-linear decision boundaries effectively, especially with the use of kernel functions.
- Regularization: You want to control overfitting by adjusting the regularization parameter (C).
- Feature Space Interpretation: You want a model that provides clear decision boundaries and allows for the interpretation of support vectors.
- High-Dimensional Data: SVM can be applied to high-dimensional feature spaces, making it suitable for text classification and image analysis.
- Data Separability: When your data is well-separated

## Clustering

Type of unsupervised learning process. The idea is to extract patterns and structures from data without labeled outcomes. The algorithm identify natural grouping or clusters within data.

```
from sklearn.cluster import KMeans
import numpy as np

# Create a dataset (X) - array-like data
X = np.array([[1, 2], [9, 8], [1.5, 1.8], [9, 8], [1, 0.8], [9, 1]])

# Create a K-Means clustering model with 2 clusters
model = KMeans(n_clusters=2)

# Fit the model to the data
model.fit(X)

# Get the cluster labels for each data point
labels = model.labels_

# Get the cluster centers
cluster_centers = model.cluster_centers_
```

In scikit-learn, K-Means clustering provides several options for configuration:

- Number of Clusters (n\_clusters): You specify the number of clusters you want to form.
- Initialization (init): You can choose different initialization methods for the cluster centroids, such as 'k-means++' (smart initialization), 'random' (randomly selected initial centroids), or your custom array of initial centroids.
- Random Initialization Repetitions (n\_init): The number of times the algorithm will be run with different centroid seeds. The final result will be the best output from n\_init runs.
- Convergence Criteria (tol): Tolerance to declare convergence. It's the relative change in the within-cluster sum of squares that controls the convergence.
- Other Parameters: There are other parameters to fine-tune, such as the maximum number of iterations (max\_iter), the method for calculating the initial centroids (algorithm), and more.

Types of Independent Variables:

- **Numeric Variables:** Most clustering algorithms work with numeric variables, such as features in a dataset.
- **Categorical Variables:** For clustering categorical data, you may need to perform suitable encoding or transformation, such as one-hot encoding or binary encoding, to convert them into a numeric format.
- **Mixed Data:** Some clustering algorithms can handle datasets with a mix of numerical and categorical variables. However, data preprocessing is often required to make these variables compatible with the chosen algorithm.

## When to Choose Clustering:

Choose clustering when you want to uncover hidden structures or groupings within your data, which can lead to insights, data simplification, and more effective decision-making. The choice of clustering algorithm and approach depends on the nature of your data and the specific goals of your analysis.

- **Data Segmentation:** Clustering is used when you want to segment data into groups with similar characteristics. For example, customer segmentation in marketing.
- **Anomaly Detection:** Clustering can help detect anomalies or outliers in data by identifying data points that don't fit well into any cluster.
- **Pattern Recognition:** Clustering can reveal patterns or structures in data when you're not sure what to look for.
- **Recommendation Systems:** It can be used in recommendation systems to group users or items with similar preferences.
- **Dimensionality Reduction:** Clustering can be used as a dimensionality reduction technique when you want to reduce the number of features in a dataset.
- **Exploratory Data Analysis:** Clustering is valuable in data exploration, allowing you to understand the underlying structure of your data.
- **Initial Data Exploration:** It can serve as an initial step to explore your data before applying more sophisticated machine learning techniques.

## Neural network and deep learning

Taking inspire from the human brain neural networking, neural network is a class of ML. deep learning is subset of neural network that can create complex models. In Python, you can use libraries like TensorFlow, Keras, and PyTorch to build and train neural networks. Here's an example using TensorFlow and Keras for a simple feedforward neural network:

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

# Create a sequential model
model = keras.Sequential([
    layers.Input(shape=(input_dim,)), # Define input shape
    layers.Dense(128, activation='relu'), # Add a hidden layer
    layers.Dense(1, activation='sigmoid') # Add an output layer
])

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
model.fit(X_train, y_train, epochs=10, batch_size=32)
```

# Python Mastery for AI Enthusiasts: From Basics to Artificial

The options in neural networks vary depending on the library used, but common configuration options include:

- Activation Functions: You can choose from various activation functions for hidden and output layers, such as 'relu,' 'sigmoid,' 'tanh,' or custom functions.
- Optimizers: Options like 'adam,' 'sgd,' or 'rmsprop' control the optimization algorithm used for training.
- Loss Functions: The choice of loss function depends on the type of problem, e.g., 'binary\_crossentropy' for binary classification, 'categorical\_crossentropy' for multiclass classification, and 'mean\_squared\_error' for regression.
- Layers and Architecture: You can define the architecture of the neural network by adding layers and specifying their configurations.

Deep learning involves neural network with many layers. Here is an example of neural network deep learning:

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

model = keras.Sequential([
    layers.Input(shape=(input_dim,)),
    layers.Dense(128, activation='relu'),
    layers.Dense(128, activation='relu'),
    layers.Dense(64, activation='relu'),
    layers.Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam', loss='binary_crossentropy',
              model.fit(X_train, y_train, epochs=10, batch_size=32))
```

Deep learning is chosen in the following scenarios:

- Complex Patterns: When the problem involves recognizing complex patterns or features, such as in image or speech recognition.
- Large Datasets: For tasks with large datasets where deep learning models can automatically learn intricate patterns.
- Sequential Data: When dealing with sequential data, like time series, text, or audio.
- State-of-the-Art Performance: When state-of-the-art performance is required in domains such as computer vision, natural language processing, and speech processing.
- Custom Architectures: When you need the flexibility to design custom neural network architectures.
- Transfer Learning: For tasks where pre-trained models can be fine-tuned for a specific task, saving time and resources.
- Parallel Processing: Deep learning frameworks are designed for efficient utilization of GPUs and TPUs, making them suitable for large-scale parallel processing.

However, deep learning models are computationally intensive and require significant amounts of data, making them overkill for smaller or simpler tasks. Consider traditional machine learning techniques for those scenarios.

## Generative Adversarial Networks (GAN's)

GANs as well relate to deep learning and neural network. are a class of machine learning models designed to generate new data that resembles existing data. They have applications in image generation, style transfer, etc.

when to Choose Neural Networks and GANs:

# Python Mastery for AI Enthusiasts: From Basics to Artificial

- For a wide range of tasks, such as image classification, natural language processing, and regression.
- When you have a labeled dataset and want to learn patterns in the data.
- When you need to perform supervised or unsupervised learning on structured or unstructured data.

Use GANs:

For generative tasks like image, text, or music generation.

- When you want to create data that follows a similar distribution to your training data.
- To create synthetic data for augmentation and improve the performance of other machine learning models.
- When you want to explore creative AI applications, art, or content generation.

## Module 4.5 - SUM it, and understanding the used case's

Method	Common tasks
Linear regression	Sales forecasting
	Predicting equipment failure time
Multiple regression	Same as linear regression
Classification	Spam detection
	Sentiment analysis
	Disease diagnose
Logistic regression	Binary classification tasks
	Customer churn prediction
SVMs	Binary and multiclass classifications
	Image and text classifications
clustering	Customer segmentation
	Recommendations system
	Anomaly detection
Neural networks (RNN, LSTM, CNN)	Image recognition
	Natural language processing
	Speech recognition
	Predictive maintenance (time-series data)
GANs	Image and video generation
	Data augmentation (growth)
	Synthetic data creation
	Predictive maintenance (data generation)

For example, in order to create predictive maintenance, the best practice for my opinion is the combine the following methods:

- In order to predict the failure of equipment we will use auto-regressive models with time-series forecasting techniques. Neural network is effective for analyzing time series sensor data and predicting failures.
- Using labeled data with known past failures will take us into supervised learning methods like Decision trees or Random forests.
- Anomaly detection techniques such as One class SVM will help generate early messages.
- Those are shortly explained. Of course, we can use with much more such as Bayesian Networks, GANs for training predictive models, isolation forest for anomaly detection and much more.

# Python Mastery for AI Enthusiasts: From Basics to Artificial

## Simple example predictive maintenance code outline:

*# Import necessary libraries*

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import IsolationForest # For anomaly detection
from statsmodels.tsa.ar_model import AutoReg # For auto-regressive model
```

*# Load your dataset (replace 'your\_data.csv' with your actual data file)*

```
data = pd.read_csv('your_data.csv')
```

*# Data Preprocessing*

```
X = data[['Sensor1', 'Sensor2', 'Sensor3']] # Features
y = data['Failure'] # Target variable
```

*# Split the data into training and testing sets*

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

*# Train a Random Forest Classifier (you may tune hyperparameters as needed)*

```
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
rf_classifier.fit(X_train, y_train)
```

*# Make predictions on the test data*

```
y_pred = rf_classifier.predict(X_test)
```

*# Evaluate the model*

```
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)
```

```
print("Random Forest Classifier Results:")
```

```
print(f"Accuracy: {accuracy}")
```

```
print(f"Confusion Matrix:\n{conf_matrix}")
```

```
print(f"Classification Report:\n{classification_rep}")
```

*# Decision Tree Classifier*

```
dt_classifier = DecisionTreeClassifier(random_state=42)
dt_classifier.fit(X_train, y_train)
```

*# Make predictions using Decision Tree*

```
y_pred_dt = dt_classifier.predict(X_test)
```

*# Evaluate the Decision Tree model*

```
accuracy_dt = accuracy_score(y_test, y_pred_dt)
```

```
conf_matrix_dt = confusion_matrix(y_test, y_pred_dt)
```

*#By examining the values in the confusion matrix, you can calculate various  
#performance metrics like accuracy, precision, recall, and F1 score, which  
# help you evaluate the effectiveness of the Decision Tree classifier in  
#predicting equipment failures.*

```
classification_rep_dt = classification_report(y_test, y_pred_dt)
```

*#The classification report provides a more detailed summary of the  
#classifiers performance, including metrics such as precision,  
# recall, F1 score, and support for each class*

# Python Mastery for AI Enthusiasts: From Basics to Artificial

```
print("\nDecision Tree Classifier Results:")
print(f"Accuracy: {accuracy_dt}")
print(f"Confusion Matrix:\n{conf_matrix_dt}")
print(f"Classification Report:\n{classification_rep_dt}")

# Anomaly Detection (Isolation Forest)
anomaly_detector = IsolationForest(contamination=0.1, random_state=42)
anomaly_detector.fit(X_train)
#or you can use one-class SVM:
#anomaly_detector = OneClassSVM(nu=0.1)
#anomaly_detector.fit(X_train)

# Detect anomalies in the test data
anomalies = anomaly_detector.predict(X_test)

# Auto-Regressive Model
# Assuming 'time_series_data' is a DataFrame with time series data
lags = 15 # Number of lag values
ar_model = AutoReg(time_series_data, lags=lags)
ar_model_fit = ar_model.fit()

# Predict failures using the Auto-Regressive model
failures_predicted = ar_model_fit.predict(start=lags, end=len(time_series_data) - 1, dynamic=False)
```

**T-H-E     E-N-D**

*I really hope that it was helpful for you!*

*In order to improve and help others, please share your insights with me.*

GOOD LUCK!!

*Best Regards,*

*M.*

For contributions: [click here](#)