LAB MANUAL

Machine Learning

Department of AI

Eng. Mai AL- Romaishi

2024-2025

Estimated Completion Time

2.0 hours

Student Name			
Department			
Date			
Lab Grade		**Obtained grade**	
Instructor’s signature			

ML Crash Course – Part I

- **Objectives of this lab:**
  1- **Introduction to Python**
     - **What is Python?**
     - **Using of Python**
     - **What can Python do?**
     - **Why Python?**
     - **Python Language advantages**

  2- **Install Python with Anaconda & VS-Code**

  3- **First Python Program**

  4- **Python String**

  5- **Python Numbers**

  6- **Python Lists**

  7- **Python Sets**

  8- **Python Tubles**

  9- **Python Dictionary**

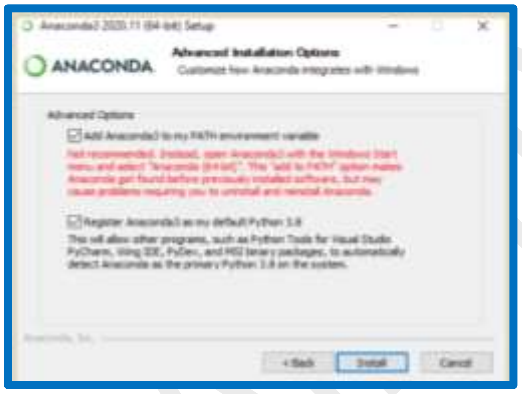
Introduction

- **Python** is a widely used general-purpose, high level programming language. It was created by Guido van Rossum in 1991 and further developed by the Python Software Foundation.
- It was designed with an emphasis on code readability, and its syntax allows programmers to express their concepts in fewer lines of code.
- Python is a programming language that lets you work quickly and integrate systems more efficiently.
- **It is used for:**
  1. Web development (server-side)
  2. Software development
  3. Mathematics
  4. System scripting
- **What can Python do?**
  1. Python can be used on a server to create web applications.
  2. Python can connect to database systems. It can also read and modify files.
  3. Python can be used to handle big data and perform complex mathematics.
- **Why Python?**
  1. Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc.).
  2. Python has a simple syntax similar to the English language.
  3. Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
  4. 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 Language advantages**
  1. Extensive support libraries(NumPy for numerical calculations, Pandas for data analytics etc.)
  2. Open source and community development
  3. Easy to learn
  4. User-friendly data structures
  5. High-level language
  6. Dynamically typed language(No need to mention data type based on value assigned, it takes data type)
  7. Portable across Operating systems

▪ Install Python with Anaconda

- **Anaconda:**
  1. A distribution of python
  2. Includes Python and many other libraries
  3. All-in-one install
- **Jupyter:**
  1. A development environment for writing code and markdown and display images
  2. Most popular IDE
  3. Best learning/teaching tool

- **https://www.anaconda.com/**
  1. Download the Anaconda Installer
  2. Make sure checking both boxes
  3. Search for Anaconda Navigator on your computer


Anaconda3 2020.11 (64-bit) Setup

Advanced Installation Options
Customize how Anaconda integrates with Windows

Advanced Options

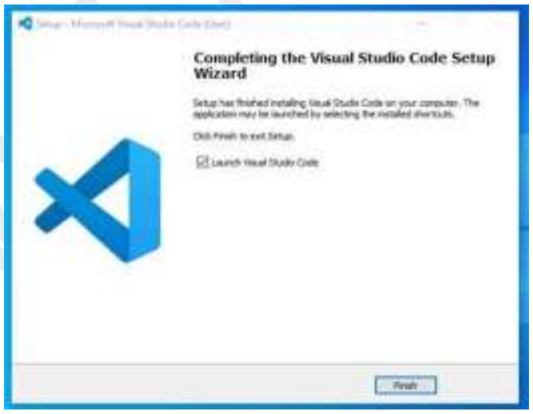
[✓] Add Anaconda3 to my PATH environment variable

Not recommended. Instead, open Anaconda3 with the Windows Start menu and select "Anaconda (64-bit)". This "add to PATH" option makes Anaconda get found before previously installed software, but may cause problems requiring you to uninstall and reinstall Anaconda.

[✓] Register Anaconda3 as my default Python 3.8

This will allow other programs, such as Python Tools for Visual Studio PyCharm, Wing IDE, PyDev, and MSI binary packages, to automatically detect Anaconda as the primary Python 3.8 on the system.

▪ Install Python with VSCode


Setup - Microsoft Visual Studio Code (User)

Completing the Visual Studio Code Setup Wizard

Setup has finished installing Visual Studio Code on your computer. The application may be launched by selecting the installed shortcuts.

Click Finish to exit Setup.

[✓] Launch Visual Studio Code

- **[Download Visual Studio Code - Mac, Linux, Windows]()**

First Python Program

- #This is a comment
  print("Hello, Python!")
- Comments starts with a #, and Python will ignore them

Python 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:

1- **Text Type**: str
2- **Numeric Types**: int, float, complex
3- **Sequence Types:** list, tuple, range
4- **Mapping Type:** dict
5- **Set Types:** set, frozenset
6- **Boolean Type:** bool
7- **Binary Types:** bytes, bytearray, memoryview

Python String

- In Python, Strings 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 1:**

```
EX= “Welcome to python”
print(EX)
print(EX[0])
print(EX[4:9])
print(EX[4:])
print(EX*3)
print(EX+' in Cyber')
```

- **Example 2:**

```
a=”Introduction to AI”
print(a[-7])
print(len(a))
print(a.upper())
print(a.lower())
print(a.strip()) #method removes any whitespace from the beginning or the end
print(a.replace("AI", "CYS"))#method replaces a string with another string
print("AI" in a)
print("CYS" not in a)
```

- **Example 3:**

```
age = 23
txt = "My name is Mai, I am " + age         # error
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

```
 age = 23
txt = "My name is Mai, and I am {}"
print(txt.format(age))
```

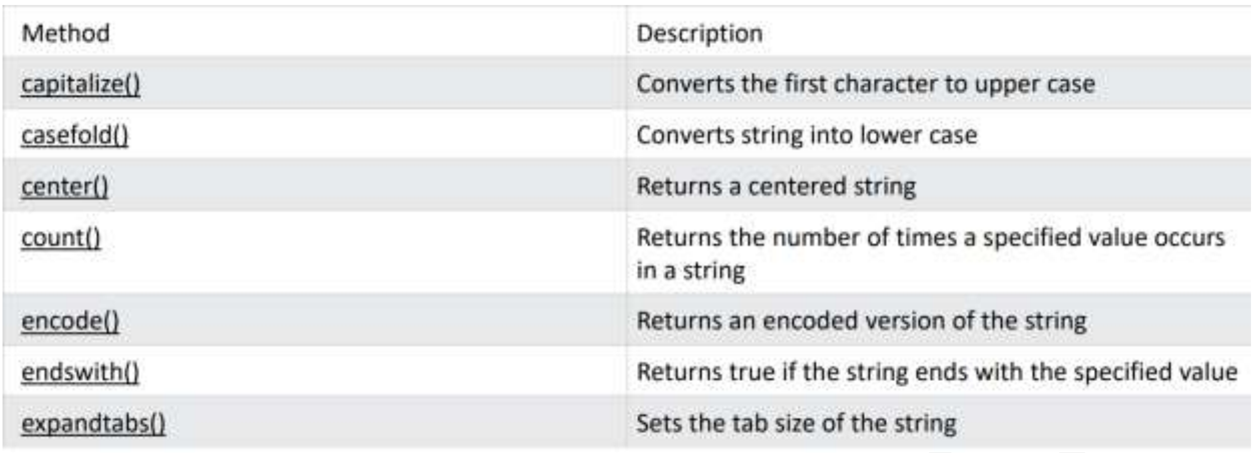
- **Example 4:**

```
quantity = 3

itemno = 567

price = 49.95

myorder = "I want {} pieces of item {} for {} dollars."

print(myorder.format(quantity, itemno, price))
```

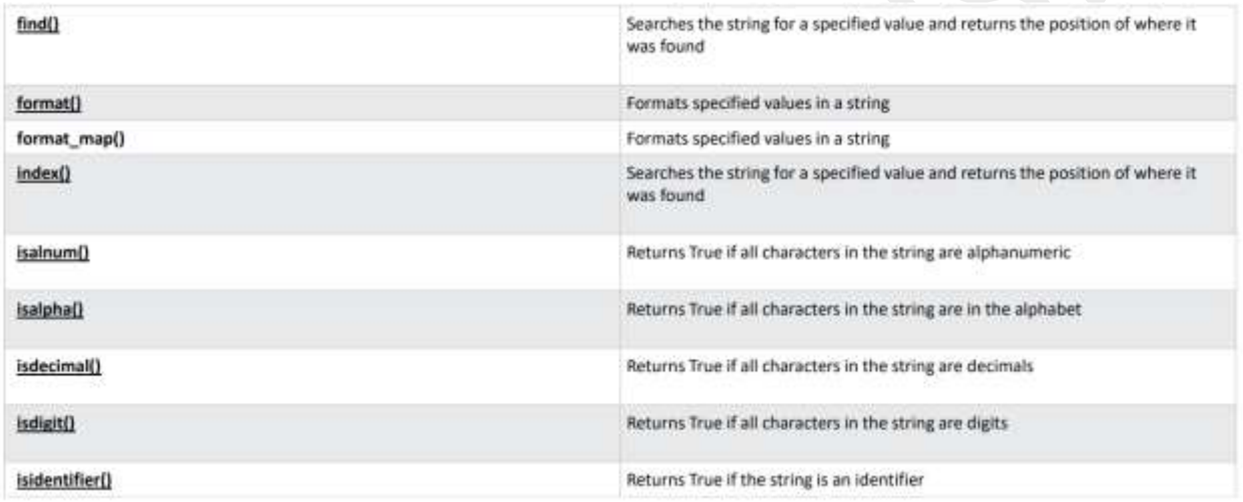
Or

```
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))
```

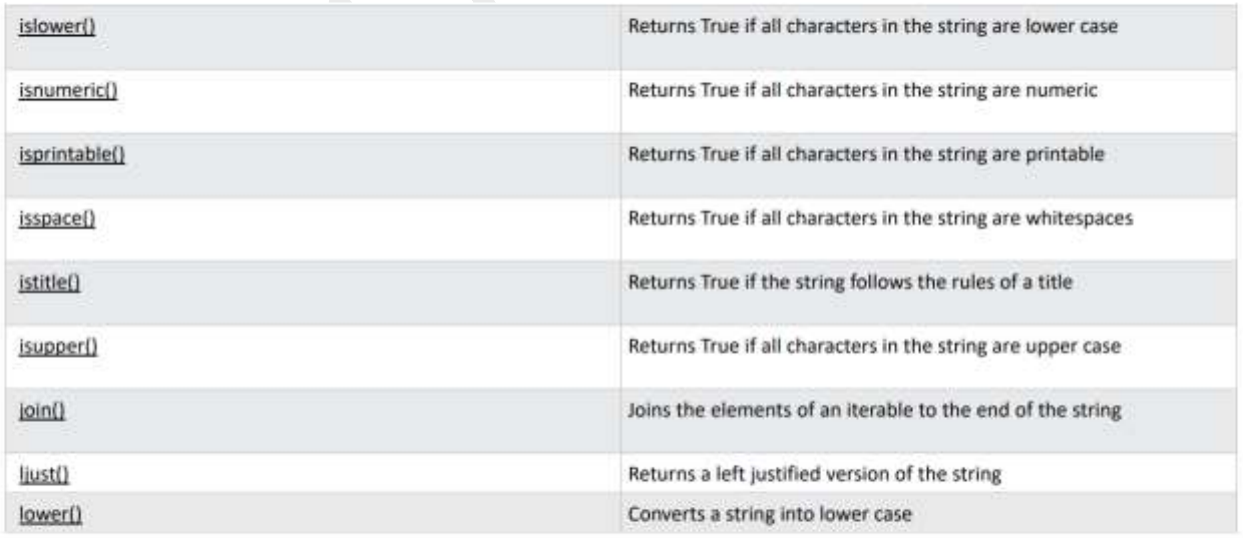
- **Follow Python String**



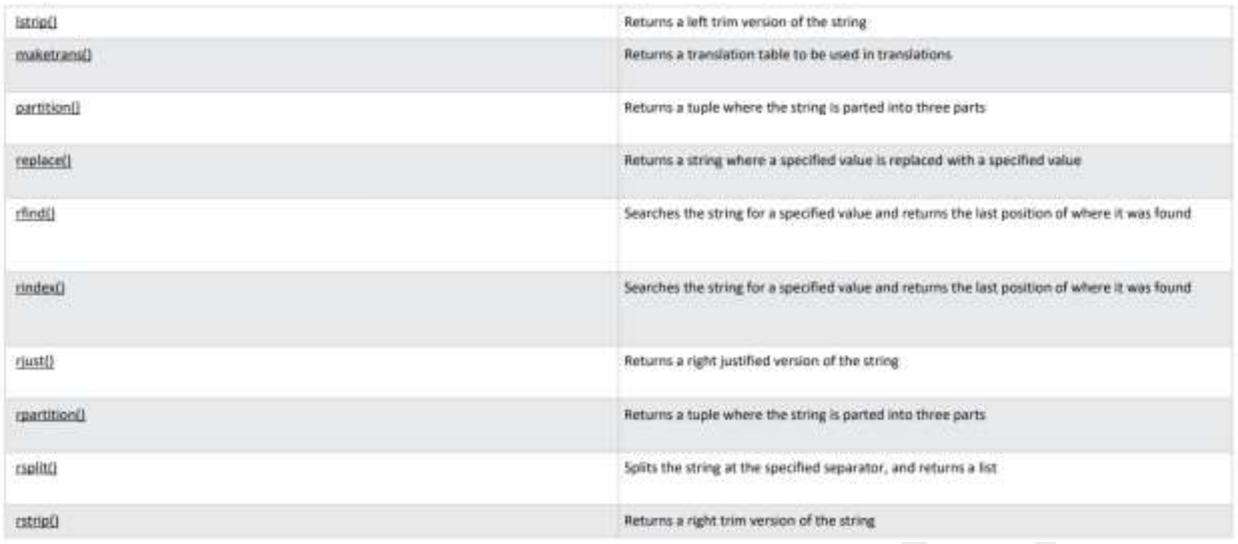
Method	Description
capitalize()	Converts the first character to upper case
casefold()	Converts string into lower case
center()	Returns a centered string
count()	Returns the number of times a specified value occurs in a string
encode()	Returns an encoded version of the string
endswith()	Returns true if the string ends with the specified value
expandtabs()	Sets the tab size of the string



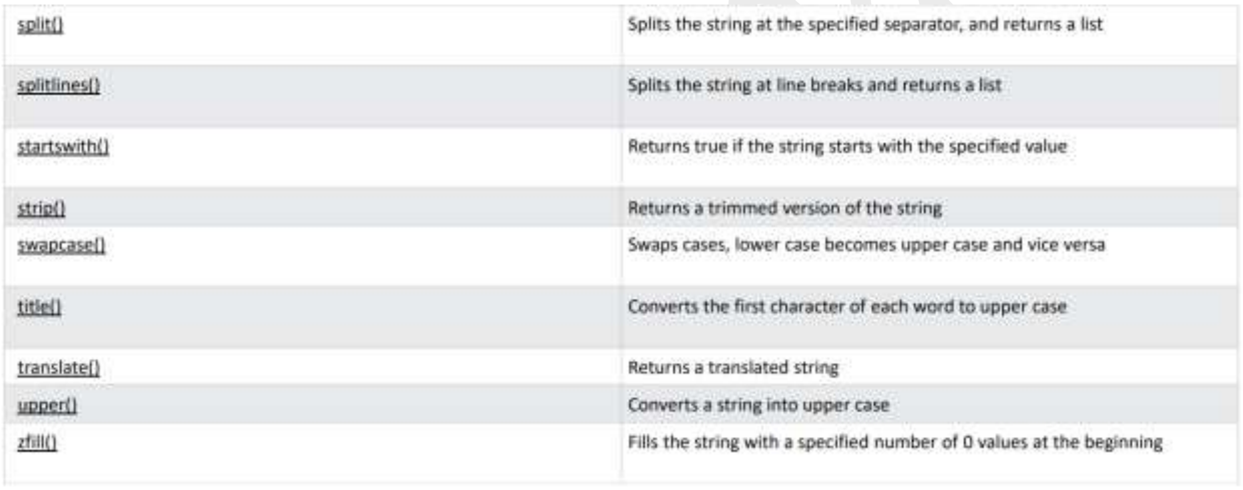
find()	Searches the string for a specified value and returns the position of where it was found
format()	Formats specified values in a string
format_map()	Formats specified values in a string
index()	Searches the string for a specified value and returns the position of where it was found
isalnum()	Returns True if all characters in the string are alphanumeric
isalpha()	Returns True if all characters in the string are in the alphabet
isdecimal()	Returns True if all characters in the string are decimals
isdigit()	Returns True if all characters in the string are digits
isidentifier()	Returns True if the string is an identifier



islower()	Returns True if all characters in the string are lower case
isnumeric()	Returns True if all characters in the string are numeric
isprintable()	Returns True if all characters in the string are printable
isspace()	Returns True if all characters in the string are whitespaces
istitle()	Returns True if the string follows the rules of a title
isupper()	Returns True if all characters in the string are upper case
join()	Joins the elements of an iterable to the end of the string
ljust()	Returns a left justified version of the string
lower()	Converts a string into lower case



lstrip()	Returns a left trim version of the string
maketrans()	Returns a translation table to be used in translations
partition()	Returns a tuple where the string is parted into three parts
replace()	Returns a string where a specified value is replaced with a specified value
rfind()	Searches the string for a specified value and returns the last position of where it was found
rindex()	Searches the string for a specified value and returns the last position of where it was found
rjust()	Returns a right justified version of the string
rpartition()	Returns a tuple where the string is parted into three parts
rsplit()	Splits the string at the specified separator, and returns a list
rstrip()	Returns a right trim version of the string



split()	Splits the string at the specified separator, and returns a list
splitlines()	Splits the string at line breaks and returns a list
startswith()	Returns true if the string starts with the specified value
strip()	Returns a trimmed version of the string
swapcase()	Swaps cases, lower case becomes upper case and vice versa
title()	Converts the first character of each word to upper case
translate()	Returns a translated string
upper()	Converts a string into upper case
zfill()	Fills the string with a specified number of 0 values at the beginning

- ## Multiline Strings:
- You can assign a multiline string to a variable by using three quotes
- **Example:**

```
 MS= """You can assign a multiline string to a variable by using three quotes ."""
print(MS)
```

Or three single quotes:

```
 MS= ''' You can assign a multiline string to a variable by using three quotes .'''
print(MS)
```

- ## Python Numbers
- Python Numbers There are three numeric types in Python:
  1-int
  2-float
  3-complex
- Variables of numeric types are created when you assign a value to them
- **Examples:**

```
#print("number")
 v1 = 10
 v2 = 9.89999
print(v1)
print(v2)
print(v1,v2)
print("v1=",v1)
```

- ## Python Lists
  - Lists are just like dynamic sized arrays, declared in other languages (vector in C++ and Array List in Java). Lists need not be homogeneous always which makes it a most powerful tool in Python. A single list may contain Data Types like Integers, Strings, as well as Objects. Lists are mutable, and hence, they can be altered even after their creation.
  - List in Python are ordered and have a definite count. The elements in a list are indexed according to a definite sequence and the indexing of a list is done with 0 being the first index. Each element in the list has its definite place in the list, which allows duplicating of elements in the list, with each element having its own distinct place and credibility.

▪ Follow Python Lists

- 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.
- **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.

- **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.

- **List Items - Data Types**

List items can be of any data type

- **Example 1:**

```
 #********************list ********************
 list1 = [1, 'Mai',78.152,88]
 print(list1)
print(list1[2:])
print(list1[1:3])
print(list1[3])
list1[3]=1000
print(list1[3])
print(list1)
```

- **Example 2:**

```
List2 = ["apple", "banana", "orange", "kiwi", "mango"]
print(List2 [-4:-1])
List2 [1]=99
List2 = ["Mai", "Yousif", "Ail"]
List2 [1:3] = ["Reem", "Ali"]
print(List2)
List2 [1:3] = ["Reem"]
print(List2)
List2.insert(2, “Lama")
print(List2)
```

- **Follow Example 2:**

```
List2.append("AAAA")
print(List2)
List2= ["apple", "banana", "cherry"]
t = ["mango", "pineapple", "papaya"]
List2.extend(t)
print(List2)
List2.remove("banana")
print(List2)
t.pop(1)
print(t)
```

```
t.pop()
 print(t)
List3= ["apple", "banana", "cherry"]
del List3 [0]
print(List3)
del List3
print(List3) # error
```

```
List4= ["apple", "banana", "cherry"]
List4.clear()
print(List4)
 #Print list Element
 for x in List4:
print(x)
 for i in range(len(List4)):
print(List4 [i])
 [print(x) for x in List4]
```

```
 List5= [100, 50, 65, 82, 23]
List5.sort()
print(List5)
List5.sort(reverse = True)
 print(List5)
List5= ["orange", "mango", "kiwi", "pineapple", "banana"]
 List5.sort()
print(List5)
```

```
List6 = ["apple", "banana", "cherry"]
mylist = List6.copy()
print(mylist)
 List6 = ["apple", "banana", "cherry"]
mylist = list(List6)
print(mylist)
L1 = ["a", "b", "c"]
```

```
L2 = [1, 2, 3]
L3 = L1 + L2
 print(L3)
```

▪ Python Tuples

- 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
- **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 1 :**

```
tuple1=(1, 'Mai',2,33.32,99, ’Yousif’)
print(tuple1)
print(tuple1[1])
print(tuple1[1:])
print(tuple1[-1])
print(tuple1[1:3])
print(tuple1[-3:-1]) #tuple1[2]=9 error we cannot edit
```

- **Example 2 :**

```
T2 = ("apple", "banana")
y = ("orange",)
T2 += y
print(T2)
#Remove
T2 = ("apple", "banana", "cherry")
del T2
```

▪ Python Set

- A set is a collection which is unordered, unchangeable, and unindexed.
- 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.
- **Duplicates Not Allowed**
  Sets cannot have two items with the same value.
- **Example :**

```
set1 = {"abc", 34, True, 40, "male"}
print(set1)
Set1.add(99)
print(set1)
s1 = {"apple", "banana", "cherry"}
s2 = {"pineapple", "mango", "papaya"}
s1.update(s2)
print(s1)
s 1 .pop()
print( s 1 )
s 1 .remove('apple')
print( s 1 )
s 1 .discard('apple')
print( s 1 )
s 1 .clear()
del s 1
print( s 1 )
```

Python Dictionary

- Dictionary in Python is an unordered collection of data values, used to store data values like a map, which unlike other Data Types that hold only single value as an element, Dictionary holds key : value pair. Key value is provided in the dictionary to make it more optimized.
- Note – Keys in a dictionary doesn't allows Polymorphism.
- **Ordered or Unordered?**
  As of Python version 3.7, dictionaries are ordered. In Python 3.6 and earlier, dictionaries are unordered.
- **Changeable**
  Dictionaries are changeable, meaning that we can change, add or remove items after the dictionary has been created.
- **Duplicates Not Allowed**
  Dictionaries cannot have two items with the same key
- **Example 1 :**
  ```
  d1 = { "brand": "Ford", "model": "Mustang", "year": 1964, "year": 2020 }
  print(d1)
  ```

- **Example 2 :**
  ```
  d2 = { "brand": "Ford", "electric": False, "year": 1964, "colors": ["red", "white",
  "blue"] } print(d2) x=d1['year']
  print(x)
  ```

- **Example 3 :**
  ```
  x=d1.get('year')
   print(x)
  x = d1.keys()
  car = { "brand": "Ford", "model": "Mustang", "year": 1964 } car["color"] = "white"
  print(x)
  x = car.values()
  print(x)
  x = car.items()
  print(x)
  d2 = { "brand": "Ford", "model": "Mustang", "year": 1964 }d2.update({"year":
  2020})
  print( d 2 )
  ```

- **Example 4 :**

```
dict1={}
list=['std','one']
dict1[1]=list
dict2={1:'one',2:'two',1:'three'}
dict3={'name':mai,'age':23}
print(dict1)
print(dict1[1])
print(dict2)
print(dict3)
print(dict2.keys())
print(dict2.values())
```

Exercises 1

1- **Write a program that asks the user to enter a string.**

- The program should then print the following:
  - a- The total number of characters in the string
  - b- The string repeated 10 times
  - c- The first character of the string (remember that string indices start at 0)
  - d- The first three characters of the string
  - e- The last three characters of the string
  - f- The string backwards
  - g- The seventh character of the string if the string is long enough and a message otherwise
  - h- The string with its first and last characters removed
  - i- The string in all caps
  - j- The string with every a replaced with an e
  - k- The string with every letter replaced by a space

Exercises 2

1- **Write a program that asks the user to enter a list of integers. Do the following:**

1. Print the total number of items in the list.
2. Print the last item in the list.
3. Print the list in reverse order.
4. Print Yes if the list contains a 5 and no otherwise.
5. Print the number fives in the list.
6. Remove the first and last items from the list, sort the remaining items, and print the result. Print how many integers in the list are less than 5

Exercises 3.

1- Write a program that generates a list of 20 random numbers between 1 and 100.

a- Print the list.
b- Print the average of the elements in the list.
c- Print the largest and smallest values in the list.
d- Print the second largest and second smallest entries in the list
e- Print how many even numbers are in the list.

Exercises 4.

Start with the list [8,9,10]. Do the following:

1- Set the second entry (index 1) to 17
2- Add 4, 5, and 6 to the end of the list
3- Remove the first entry from the list
4- Sort the list
5- Double the list
6- Insert 25 at index 3 The final list should equal [4,5,6,25,10,17,4,5,6,10,17]

Exercises 5

- Ask the user to enter a list containing numbers between 1 and 12. Then replace all of the entries in the list that are greater than 10 with 10.

Exercises 6.

- Ask the user to enter a list of strings. Create a new list that consists of those strings with their first characters removed.

Exercises 7.

- Create the following lists using a for loop.
  a- A list consisting of the integers 0 through 49
  b- A list containing the squares of the integers 1 through 50.
  c- The list ['a','bb','ccc','dddd', . . .] that ends with 26 copies of the letter z.

. Exercises 8

- Write a program that repeatedly asks the user to enter product names and prices. Store all of these in a dictionary whose keys are the product names and whose values are the prices. When the user is done entering products and prices, allow them to repeatedly enter a product name and print the corresponding price or a message if the product is not in the dictionary.

Exercises 9

- Using the dictionary created in the previous problem, allow the user to

Exercises 10

- For this problem, use the dictionary from the beginning of this chapter whose keys are month names and whose values are the number of days in the corresponding months.
  - a- Ask the user to enter a month name and use the dictionary to tell them how many days are in the month.
  - b- Print out all of the keys in alphabetical order.
  - c- Print out all of the months with 31 days.
  - d- Print out the (key-value) pairs sorted by the number of days in each month
  - e- Modify the program from part (a) and the dictionary so that the user does not have to know how to spell the month name exactly. That is, all they have to do is spell the first three letters of the month name correctly

ML Crash Course – Part II

- **Objectives of this lab:**

  10- **Control Flow Statements**

  11- **Functions in Python**

  12- **Variables in pathon**

Control Flow Statements

- The flow control statements are divided into three categories
- **Conditional statements**

  Conditional statements: In Python, condition statements act depending on whether a given condition is true or false

  . You can execute different blocks of codes depending on the outcome of a condition. Condition statements always evaluate to either True or False. There are four types of conditional statements.

  1-if statement

  2-if-else

  3-if-elif-else

  4-nested if-else

- **Examples:**

  #If-else example

```
x=int(input("plase ente number 1 \n"))
y=int(input("plase ente number 2 \n"))
if x>y:
 print("number 1 grate then number 2")
else:
 print("number 2 grate then number 1")
```

  #Example if – elif – else

```
grad=float(input("plase enter your gread \n "))
if grad<=100 and grad>=90:
 print("Excelent ")
elif grad<=90 and grad>=80:
print("V.Good ")
elif grad < 80 and grad >=65: print("Good ")
elif grad < 56 and grad >=50:
print("Accept ")
else:
print("fiald -_- ")
```

- **Loop statements.**

  In Python, iterative statements allow us to execute a block of code repeatedly as long as the condition is True. We also call it a loop statement. Python provides us the following two loop statement to perform some actions repeatedly

  1-for loop

  2-while loop

- **For Loop Examples:**

```
#ex1
n="welcom to python "
for i in n :
print(i)
#ex2
list1=[1,23,3,7,8,9,10,55]
for i in list1:
 print(i)
#ex3
for i in range(6):
print(i)
#ex4
for i in range(1,6):
 print(i)
#ex5
for i in range(1,30,3):#(initiall,size,increment)
print(i)
#ex6
sum=0
 for i in range(10):
 if i%2==0 :
sum=sum+i
print("sum",sum )
```

- **#While Examples**

```
i = 11
 while i <= 10:
if i%2==0:
print('event:',i )
else:
print("odd :",i) i += 1
 else:
print("i is no longer less than 10")
```

Functions in Python

A function in Python is an aggregation of related statements designed to perform a computational, logical, or evaluative task. The idea is to put some commonly or repeatedly done task together and make a function so that instead of writing the same code again and again for different inputs, we can call the function to reuse code contained in it over and over again. Functions can be both builtin or user-defined. It helps the program to be concise, non-repetitive, and organized.

Creating a Function:

In Python a function is defined using the def keyword Arguments: Information can be passed into functions as arguments.

Arguments

are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.

Calling a Function:

To call a function, use the function name followed by parenthesis:

- **Examples:**

```
 #ex1 function without parameter and can't return
def my_print():
print("welcom to our lab")
my_print()
#ex2 function with out parameter and can return
 def sum():
v1=int(input("plase enter number 1"))
v2= int(input("plase enter number 2"))
return v1+v2
 print(sum())
```

#ex3 function with parameter and can't return

```
def sum( v1,v2):
print( v1+v2)
 v1=int(input("please enter number 1"))
v2= int(input("please enter number 2"))
sum(v1,v2)
```

#ex4 function with parameter and can return

```
def sum( v1,v2):
return v1+v2
 v1=int(input("please enter number 1"))
v2= int(input("please enter number 2"))
print(sum(v1,v2))
```

- **Default arguments:**

A default argument is a parameter that assumes a default value if a value is not provided in the function call for that argument. The following example illustrates Default arguments.

- **Example:**

```
def myFun(x, y=50):
 print("x: ", x)
print("y: ", y)
myFun(10)
```

- **Keyword arguments:**
  The idea is to allow the caller to specify the argument name with values so that caller does not need to remember the order of parameters
- **Example :-**

```
def student(firstname, lastname):
print(firstname, lastname)
```

# Keyword arguments

```
student(firstname='Mai', lastname='Abduallah)
```

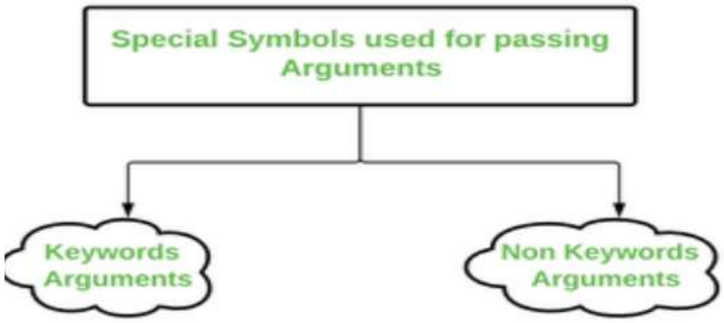
student(lastname='Yousif', firstname='Ahmed')

- **Variable-length arguments:**

  In Python, we can pass a variable number of arguments to a function using special symbols. There are two special symbols Special Symbols Used for passing arguments:-

  1- *args (Non-Keyword Arguments)

  2- **kwargs (Keyword Arguments)

- 

- ***args**

  The special syntax *args in function definitions in python is used to pass a variable number of arguments to a function. It is used to pass a non-key worded, variable-length argument list.

- **Example:**

```
def myFun(*argv):
 for arg in argv:
print(arg)
myFun('Hello', 'Welcome', 'to', 'AI')
```

- ****kwargs**

  The special syntax **kwargs in function definitions in python is used to pass a keyworded, variable-length argument list. We use the name kwargs with the double star. The reason is because the double star allows us to pass through keyword arguments (and any number of them).

- **Example:**

```
def my_function(**kwr):
print("Her last name is " + kwr["lname"])
my_function(fname = "Mai ", lname = "Abduallah")
```

- **Python lambda:**

  In Python, anonymous function means that a function is without a name. As we already know that def keyword is used to define the normal functions and the lambda keyword is used to create anonymous functions. It has the following syntax:

- **Syntax:**

  lambda arguments : expression

  - This function can have any number of arguments but only one expression, which is evaluated and returned.
  - One is free to use lambda functions wherever function objects are required.
  - You need to keep in your knowledge that lambda functions are syntactically restricted to a single expression.
  - It has various uses in particular fields of programming besides other types of expressions in functions.

- **Examples:**

  #Ex1

```
ex=lambda v1,v2:v1+v2
 print(ex(5,8))
```

#Ex2

```
 def testfunc(num):
return lambda x : x * num
 result1 = testfunc(10)
print(result1(9))
```

#Ex3

```
numbers_list = [2, 6, 8, 10, 11, 4, 12, 7, 13, 17, 0, 3, 21]
filtered_list = list(filter(lambda num: (num > 7), numbers_list))
 print(filtered_list)
```

- **The filter() function**

extracts elements from an iterable (list, tuple etc.) for which a function returns True.

- **Example :**

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
# returns True if number is even
def check_even(number):
    if number % 2 == 0:
        return True
   return False
# Extract elements from the numbers list for which check_even() returns True
even_numbers_iterator = filter(check_even, numbers)
# converting to list even_numbers = list(even_numbers_iterator)
print(even_numbers)
# Output: [2, 4, 6, 8, 10]
```

- Local variables :

Let’s say we have two functions like the ones below that each use a variable i:

```
def func1():
  for i in range(10):
  print(i)
def func2():
  I=100
 func1()

print(i)
```

- A problem that could arise here is that when we call func1, we might mess up the value of i in func2. In a large program it would be a nightmare trying to make sure that we don’t repeat variable names in different functions, and, fortunately, we don’t have to worry about this. When a variable is defined inside a function, it is local to that function, which means it essentially does not exist outside that function. This way each function can define its own variables and not have to worry about if those variable names are used in other functions.

- Global variables

Global variables On the other hand, sometimes you actually do want the same variable to be available to multiple functions. Such a variable is called a global variable. You have to be careful using global variables, especially in larger programs, but a few global variables used judiciously are fine in smaller programs. Here is a short example:

```
- def reset():
    global time_left
    time_left = 0
 def print_time():
   print(time_left)
reset()
#time_left=30
# print_time()
```

- In that program we have a variable time_left that we would like multiple functions to have access to. If a function wants to change the value of that variable, we need to tell the function that time_left is a global variable. We use a global statement in the function to do this. On the other hand, if we just want to use the value of the global variable, we do not need a global statement.

Exercises 1

1- **Convert decimal number to octal using print() output formatting**

Exercises 2

1- **-Display float number with 2 decimal places using print() Expected Output:**

Display 458.541315 as 458.54

Exercises 3.

1- -Print the following pattern

```
1
1   2
1   2   3
1   2   3   4
1   2   3   4   5
```

Exercises 4.

-Print the following pattern using for loop:

```
5    4    3    2    1

4    3    2    1

3    2    1

2    1

1
```

Exercises 5

-Reverse the following list using for loop list1 = [10, 20, 30, 40, 50]

Exercises 6.

- Display numbers from -10 to -1 using for loop.

Exercises 7.

- Reverse a given integer number Given: 76542

  Expected output: 24567

. Exercises 8

- Write a program to display all prime numbers within a range

Exercises 9

- Create a function showEmployee() in such a way that it should accept employee name, and its salary and display both. If the salary is missing in the function call assign default value 9000 to salary.

Exercises 10

- Return the largest item from the given list.

Exercises 11

- The digital root of a number n is obtained as follows: Add up the digits n to get a new number. Add up the digits of that to get another new number. Keep doing this until you get a number that has only one digit. That number is the digital root. For example, if n = 45893, we add up the digits to get 4 + 5 + 8 + 9 + 3 = 29. We then add up the digits of 29 to get 2 + 9 = 11. We then add up the digits of 11 to get 1 + 1 = 2. Since 2 has only one digit, 2 is our digital root.

Exercises 12

- Write a function called matches that takes two strings as arguments and returns how many matches there are between the strings. A match is where the two strings have the same character at the same index. For instance, 'python' and 'path' match in the first, third, and fourth characters, so the function should return 3.

ML Crash Course – Part III

- **Objectives of this lab:**

  13- Python Arrays

  14- Python Lists

  15- Python NumPy

  16- Python List Vs NumPy Arrays Comparison

Python Arrays

An array is a collection of items stored at contiguous memory locations. The idea is to store multiple items of the same type together. This makes it easier to calculate the position of each element by simply adding an offset to a base value, i.e., the memory location of the first element of the array (generally denoted by the name of the array). For simplicity, we can think of an array a fleet of stairs where on each step is placed a value (let's say one of your friends). Here, you can identify the location of any of your friends by simply knowing the count of the step they are on. Array can be handled in Python by a module named array. They can be useful when we have to manipulate only a specific data type value. A user can treat lists as arrays. However, user cannot constraint the type of elements stored in a list. If you create arrays using the array module, all elements of the array must be of the same type.

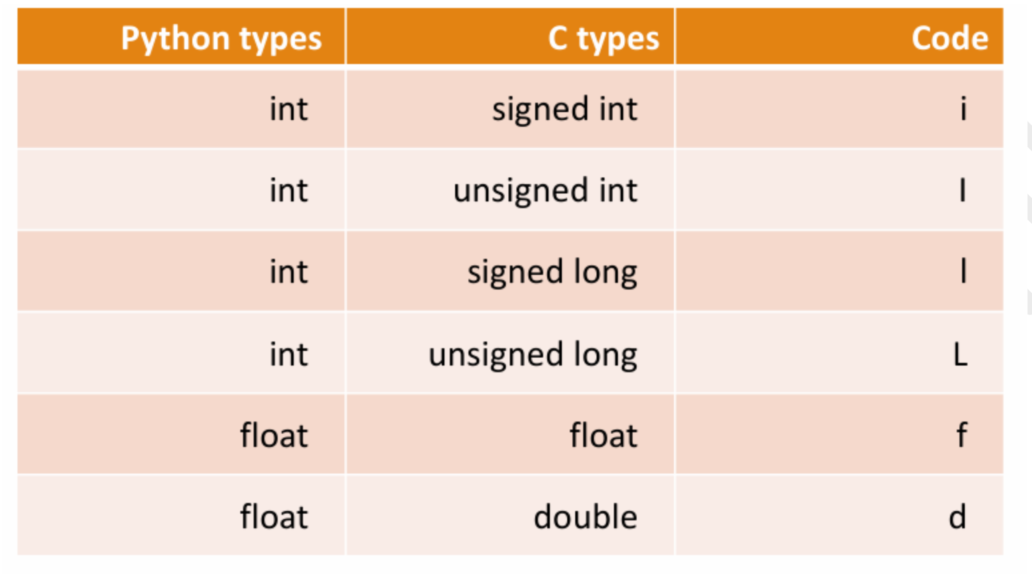
Creating an Array:

Array in Python can be created by importing array module. array(data_type, value_list) is used to create an array with data type and value list specified in its arguments.

Examples:

```
import array as arr
# creating an array with integer type
a = arr.array('i', [1, 2, 3])
# printing original array print("The new created array is : ", end=" ")
 for i in range(0, 3):
 print(a[i], end=" ")
print()
# creating an array with float type
 b = arr.array('d', [2.5, 3.2, 3.3])
 # printing original array print("The new created array is : ", end=" ")
 for i in range(0, 3):
print(b[i], end=" ")
```

Commonly used type codes are listed as follows:



Python types	C types	Code
int	signed int	i
int	unsigned int	I
int	signed long	l
int	unsigned long	L
float	float	f
float	double	d

Adding Elements to an Array:

Elements can be added to the Array by using built-in insert() function. Insert is used to insert one or more data elements into an array. Based on the requirement, a new element can be added at the beginning, end, or any given index of array. append() is also used to add the value mentioned in its arguments at the end of the array.

Example :

```
for i in range(0, 3):
 print(a[i], end=" ")
print()
# inserting array using
# insert() function a.insert(1, 4)
 print("Array after insertion : ", end=" ")
```

```
for i in (a):
print(i, end=" ")
print()
 # array with float type
 b = arr.array('d', [2.5, 3.2, 3.3])
print("Array before insertion : ", end=" ")
for i in range(0, 3):
 print(b[i], end=" ")
 print()
 # adding an element using append()
b.append(4.4)
 print("Array after insertion : ", end=" ")
for i in (b):
print(i, end=" ")
 print()
```

Removing Elements from the Array:

Elements can be removed from the array by using built-in remove() function but an Error arises if element doesn’t exist in the set. Remove() method only removes one element at a time, to remove range of elements, iterator is used. pop() function can also be used to remove and return an element from the array, but by default it removes only the last element of the array, to remove element from a specific position of the array, index of the element is passed as an argument to the pop() method. Note– Remove method in List will only remove the first occurrence of the searched element.

Example:

```
import array
 # initializing array with array values
 # initializes array with signed integers
 arr = array.array('i', [1, 2, 3, 1, 5])
 # printing original array
print("The new created array is : ", end="")
for i in range(0, 5): print(arr[i], end=" ")
print()
# using pop() to remove element at 2nd position
 print("The popped element is : ", end="")
 print(arr.pop(2))
# printing array after popping print("The array after popping is : ", end="")
for i in range(0, 4):
print(arr[i], end=" ")
print("\r")
# using remove() to rem ove 1st occurrence of 1
arr.remove(1)
# printing array after removing print("The array after removing is : ", end="")
for i in range(0, 3):
print(arr[i], end=" ")
```

Python Lists

Why uses a list?

-The list data structure is very flexible It has many unique inbuilt functionalities like pop(), append(), etc which makes it easier, where the data keeps changing.-Also, the list can contain duplicate elements i.e two or more items can have the same values.-Lists are Heterogeneous i.e, different kinds of objects/elements can be added-As Lists are mutable it is used in applications where the values of the items change frequently.

Example :

```
list1=[1,2,3,4,5,6,7]
 list2=['Mai','Abduallah']
 for x in list1:
print(x,end=‘ ')
#print(x,sep=‘ ')
list1.append(9)
# add value to end
print(list1[0:], sep=‘ ')
 list3=list1.copy()
print(list1[0:], sep=‘ ')
list2.extend(list1)
print(list2[0:], sep=‘ ')
list1.insert(3,4)
#post,value
print(list1[0:], sep=‘ ')
```

Example :

```
 list1=[1,2,3,4,5,6,7]
 list2=['Mai','Abduallah']
for x in list1:
print(x,end=' ')
#print(x,sep=' ')
list1.append(9)
# add value to end
print(list1[0:], sep=' ')
list3=list1.copy()
print(list1[0:], sep=' ')
list2.extend(list1)
print(list2[0:], sep=' ')
list1.insert(3,4)
#post,value
print(list1[0:], sep=' ')
```

Example:

```
 l1=[1,2,3,5,7,8,9]
 l1.pop(3)
#index
print(l1)
l1.remove(9)
 #value
print(l1)
l1.reverse()
 print(l1)
list.clear()
print(l1)
```

Example:

```
 L2=[7,5,1,4,1,2,9]
L2.sort()
 L2.sort(reverse=True)
#Optional. reverse=True will sort the list
#descending. Default is reverse=False
print(L2)
```

Python NumPy

Is a general-purpose array processing package which provides tools for handling the n dimensional arrays. It provides various computing tools such as comprehensive mathematical functions, linear algebra routines. NumPy provides both the flexibility of Python and the speed of well-optimized compiled C code. It's easy to use syntax makes it highly accessible and productive for programmers from any background.

Arrays in Numpy:

Array in Numpy is a table of elements (usually numbers), all of the same type, indexed by a tuple of positive integers. In Numpy, number of dimensions of the array is called rank of the array.A tuple of integers giving the size of the array along each dimension is known as shape of the array. An array class in Numpy is called as ndarray. Elements in Numpy arrays are accessed by using square brackets and can be initialized by using nested Python Lists.

Example:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
#arr = np.array((1, 2, 3, 4, 5))
print(arr)
```

Example:

```
import numpy as np
a = np.array(42)
 b = np.array([1, 2, 3, 4, 5])
c = np.array([[1, 2, 3], [4, 5, 6]])
d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])
print(a)
print(a.ndim)
print(b)
 print(b.ndim)
 print(c)
print(c.ndim)
 print(d)
print(d.ndim)
```

Example

```
#index
import numpy as np
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
print('5th element on 2nd dim: ', arr[1, 4])
```

Example:

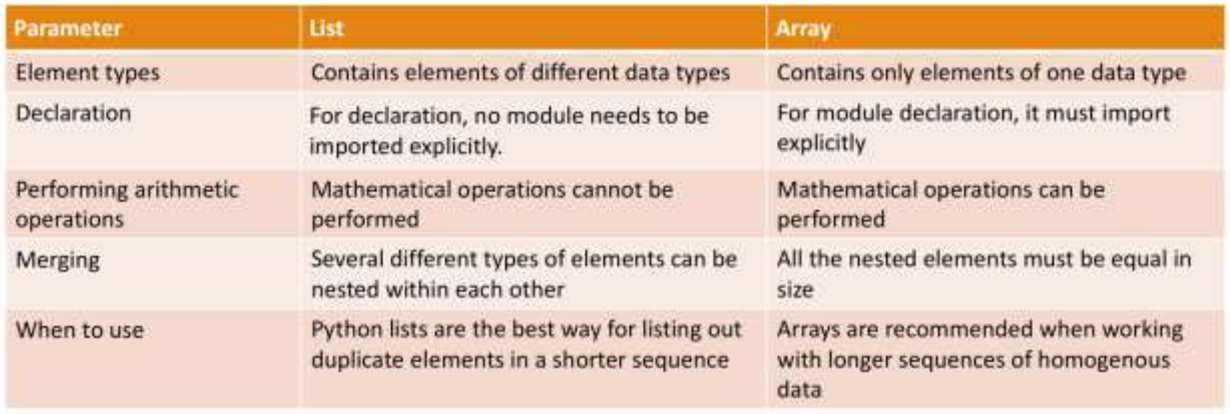
```
#Slicing arrays
#Ex1
 import numpy as np
 arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[1:5])
```

```
 #Ex2
import numpy as np
 arr = np.array([1, 2, 3, 4, 5, 6, 7])
 print(arr[-3:-1])
```

```
 #Ex3
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[1:5:2])
#[start:last:step]
```

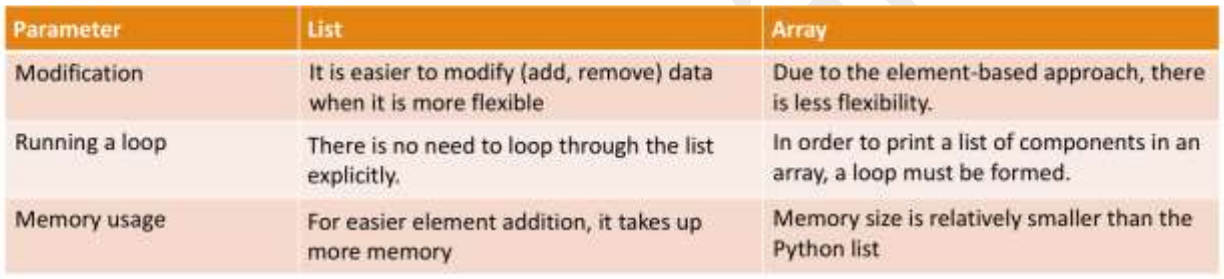
```
 #Ex4
 import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7])
 print(arr[::2])
```

Python List Vs NumPy Arrays Comparison



Parameter	List	Array
Element types	Contains elements of different data types	Contains only elements of one data type
Declaration	For declaration, no module needs to be imported explicitly.	For module declaration, it must import explicitly
Performing arithmetic operations	Mathematical operations cannot be performed	Mathematical operations can be performed
Merging	Several different types of elements can be nested within each other	All the nested elements must be equal in size
When to use	Python lists are the best way for listing out duplicate elements in a shorter sequence	Arrays are recommended when working with longer sequences of homogenous data

Python List Vs NumPy Arrays Comparison



Parameter	List	Array
Modification	It is easier to modify (add, remove) data when it is more flexible	Due to the element-based approach, there is less flexibility.
Running a loop	There is no need to loop through the list explicitly.	In order to print a list of components in an array, a loop must be formed.
Memory usage	For easier element addition, it takes up more memory	Memory size is relatively smaller than the Python list

Exercises

1- Given a Python list of numbers. Turn every item of a list into its square
2- Remove empty strings from the list of strings
3- Add item 7000 after 6000 in the following Python List
4- Given a Python list, find value 20 in the list, and if it is present, replace it with 200. Only update the first occurrence of a value
5- Given a Python list, remove all occurrence of 20 from the list
6- Create a 5X2 integer array from a range between 100 to 200 such that the difference between each element is 10
7- Following is the provided numPy array. return array of items in the third column from all rows
8- Return array of odd rows and even columns from below numpy array
9- Delete the second column from a given array and insert the following new column in its place

ML Crash Course – Part IV

- **Objectives of this lab:**

  17- Simple Linear Regression

  18- What is Linear Regression

  19- Rules of dataset

  20- Best Places to find dataset

  21- Python Example

Linear Regression

Linear regression is a fundamental statistical and machine learning technique used to model the relationship between a dependent variable and one or more independent variables. Here’s a breakdown of its key aspects:

Key Concepts

1. Dependent and Independent Variables:

- Dependent Variable (Target): The variable you want to predict (e.g., house prices).
- Independent Variables (Features): The input variables used to make predictions (e.g., size of the house, number of bedrooms).

2. Linear Relationship:

- Assumes a linear relationship between the dependent and independent variables. The relationship can be represented by a straight line in a two-dimensional space or a hyperplane in higher dimensions.

3. Equation:

- The standard form of a linear regression model is:

$$\mathbf{y=\beta 0+\beta 1x1+\beta 2x2+...+\beta nxn+\epsilon y}$$

where:

- **yyy is the dependent variable.**
- **β0\beta_0β0 is the intercept.**
- **β1,β2,...,βn\beta_1, \beta_2, ..., \beta_nβ1,β2,...,βn are the coefficients of the independent variables.**
- **x1,x2,...,xnx_1, x_2, ..., x_nx1,x2,...,xn are the independent variables.**
- **ϵ\epsilonϵ is the error term.**

4. Objective:

- The goal of linear regression is to find the best-fitting line (or hyperplane) that minimizes the difference between the predicted values and the actual values. This is often done using methods like Ordinary Least Squares (OLS).

5. Types of Linear Regression:

- Simple Linear Regression: Involves one independent variable.
- Multiple Linear Regression: Involves two or more independent variables.

Applications

- Predicting sales based on advertising spend.
- Estimating property prices based on various features.
- Forecasting trends in time series data.

Advantages and Disadvantages

- **Advantages:**
  - Easy to understand and interpret.
  - Computationally efficient.
  - Works well when the relationship is approximately linear.
- **Disadvantages:**
  - Assumes linearity, which may not hold for all datasets.
  - Sensitive to outliers.
  - Can perform poorly if the features are not relevant.

Rules of dataset

1- One case-One Row

2- One Varible-One Column

3- Create a code book

4-Save as CSV file

Best Places to find dataset

1- Kaggle

2- Googl dataset search

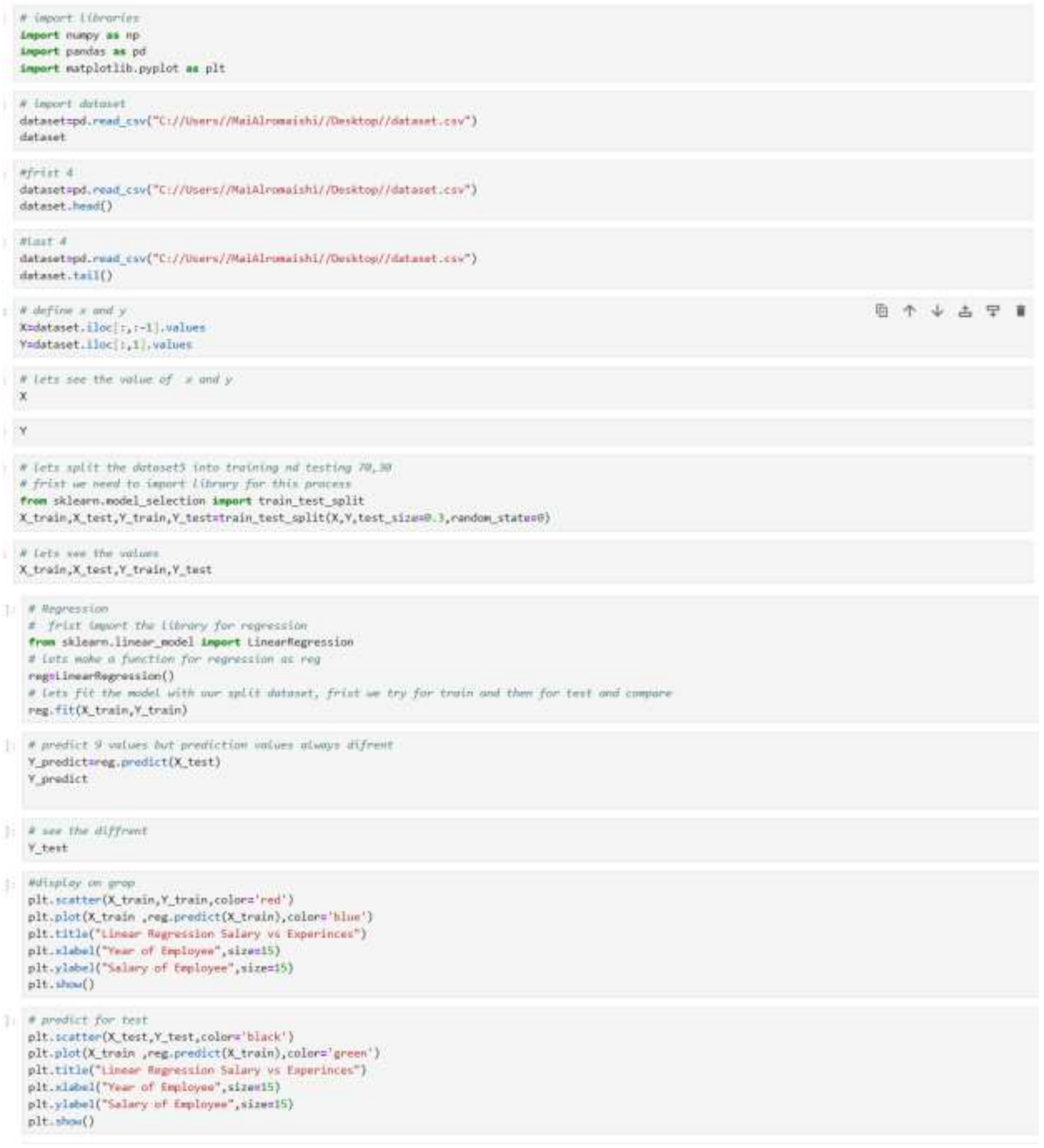
3-538

4-Githup

5-DataGov

6-Data.Nasa.Gov

Example



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

```
# import dataset
dataset=pd.read_csv("C://Users//MaiAlromaishi//Desktop//dataset.csv")
dataset
```

```
#frist 4
dataset=pd.read_csv("C://Users//MaiAlromaishi//Desktop//dataset.csv")
dataset.head()
```

```
#last 4
dataset=pd.read_csv("C://Users//MaiAlromaishi//Desktop//dataset.csv")
dataset.tail()
```

```
# define x and y
X=dataset.iloc[:,:-1].values
Y=dataset.iloc[:,1].values
```

```
# lets see the value of  x and y
X
```

```
Y
```

```
# lets split the dataset5 into training nd testing 70,30
# frist we need to import library for this process
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.3,random_state=0)
```

```
# lets see the values
X_train,X_test,Y_train,Y_test
```

```
# Regression
#  frist import the library for regression
from sklearn.linear_model import LinearRegression
# lets make a function for regression as reg
reg=LinearRegression()
# lets fit the model with our split dataset, frist we try for train and then for test and compare
reg.fit(X_train,Y_train)
```

```
# predict 9 values but prediction values always difrent
Y_predict=reg.predict(X_test)
Y_predict
```

```
# see the diffrent
Y_test
```

```
#display on grop
plt.scatter(X_train,Y_train,color='red')
plt.plot(X_train ,reg.predict(X_train),color='blue')
plt.title("Linear Regression Salary vs Experinces")
plt.xlabel("Year of Employee",size=15)
plt.ylabel("Salary of Employee",size=15)
plt.show()
```

```
# predict for test
plt.scatter(X_test,Y_test,color='black')
plt.plot(X_train ,reg.predict(X_train),color='green')
plt.title("Linear Regression Salary vs Experinces")
plt.xlabel("Year of Employee",size=15)
plt.ylabel("Salary of Employee",size=15)
plt.show()
```

ML Crash Course – Part V

- **Objectives of this lab:**

    22- **Multiple linear regression**

    23- **What is Multiple Linear Regression**

    24- **Python Example**

Multiple linear regression

Multiple linear regression is an extension of simple linear regression that models the relationship between one dependent variable and two or more independent variables. It is widely used for predictive analysis in various fields such as finance, economics, and social sciences.

Key Concepts

1. **Dependent Variable:**
   - The variable you want to predict (e.g., sales revenue).
2. **Independent Variables:**
   - Multiple input variables that influence the dependent variable (e.g., advertising spend, price, and location).
3. **Equation:**
   - The model can be represented by the equation:

     $$\mathbf{y=\beta 0+\beta 1x1+\beta 2x2+...+\beta nxn+\epsilon y}$$

   - **Where:**
     - yyy = dependent variable
     - β0\beta_0β0 = intercept
     - β1,β2,...,βn\beta_1, \beta_2, ..., \beta_nβ1,β2,...,βn = coefficients for the independent variables
     - x1,x2,...,xnx_1, x_2, ..., x_nx1,x2,...,xn = independent variables
     - ϵ\epsilonϵ = error term

4. Objective:

- To find the coefficients that minimize the difference between the predicted values and the actual values. This is typically done using Ordinary Least Squares (OLS) method.

5. Assumptions:

- Linearity: The relationship between the independent and dependent variables is linear.
- Independence: Observations are independent of each other.
- Homoscedasticity: Constant variance of errors.
- Normality: The residuals (errors) should be normally distributed.

6. Applications

- Predicting housing prices based on multiple features (e.g., size, location, number of rooms).
- Analyzing the impact of various factors on employee performance.
- Forecasting sales based on multiple marketing channels.

Advantages and Disadvantages

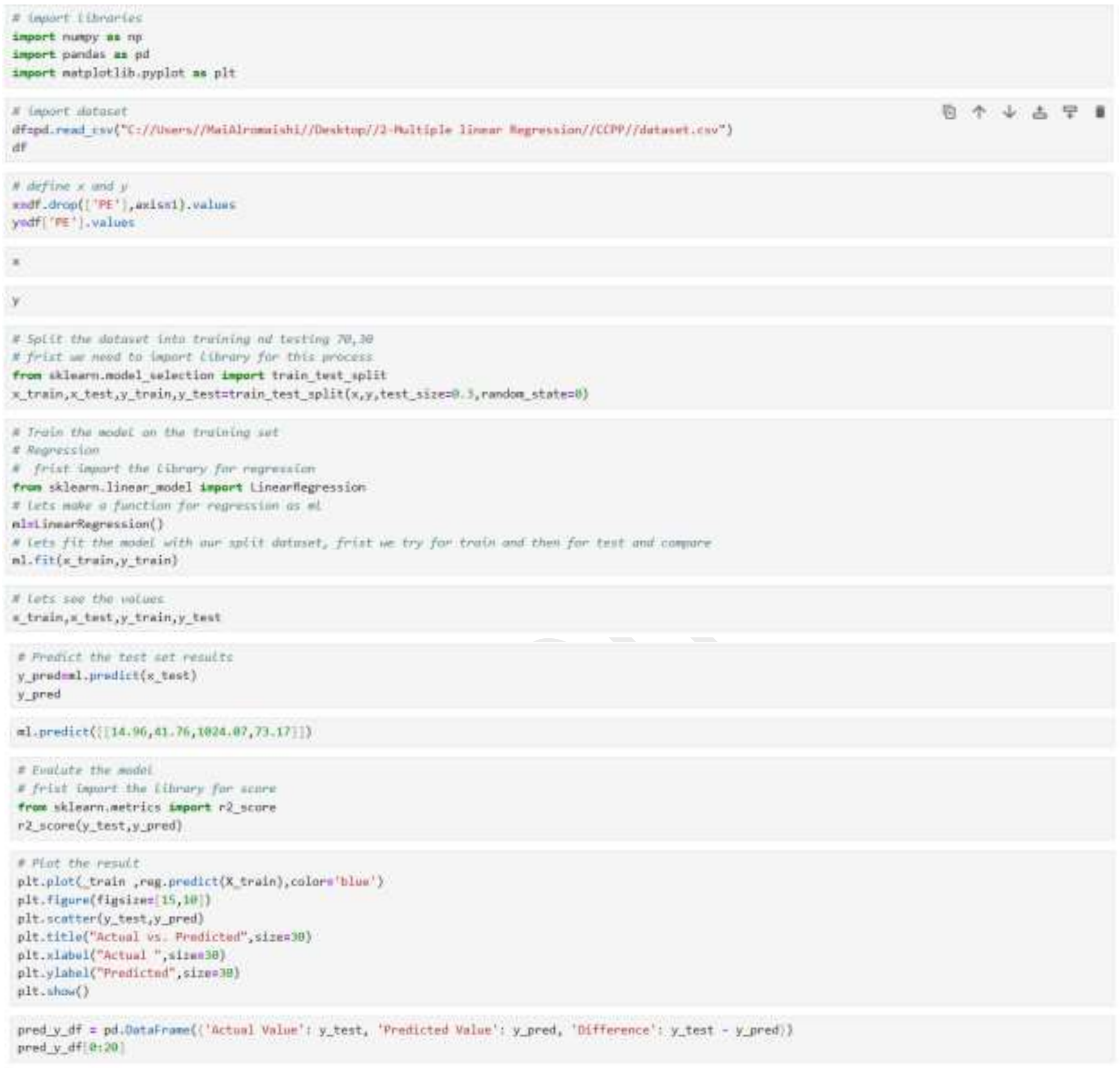
7. Advantages:

- Can handle multiple predictors, providing a more comprehensive analysis.
- Allows for understanding the impact of each variable on the dependent variable.

8. Disadvantages:

- Complexity increases with the number of variables, making interpretation harder.
- Sensitive to multicollinearity (when independent variables are highly correlated).
- Assumes a linear relationship, which may not always be valid.

Example



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

```
# import dataset
df=pd.read_csv("C://Users//MaiAlromaishi//Desktop//2-Multiple linear Regression//CCPP//dataset.csv")
df
```

```
# define x and y
x=df.drop(['PE'],axis=1).values
y=df['PE'].values
```

```
x
```

```
y
```

```
# Split the dataset into training nd testing 70,30
# frist we need to import library for this process
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=0)
```

```
# Train the model on the training set
# Regression
#  frist import the library for regression
from sklearn.linear_model import LinearRegression
# lets make a function for regression as ml
ml=LinearRegression()
# lets fit the model with our split dataset, frist we try for train and then for test and compare
ml.fit(x_train,y_train)
```

```
# lets see the values
x_train,x_test,y_train,y_test
```

```
# Predict the test set results
y_pred=ml.predict(x_test)
y_pred
```

```
ml.predict([[14.96,41.76,1024.07,73.17]])
```

```
# Evaluate the model
# frist import the library for score
from sklearn.metrics import r2_score
r2_score(y_test,y_pred)
```

```
# Plot the result
plt.plot(_train ,reg.predict(X_train),color='blue')
plt.figure(figsize=[15,10])
plt.scatter(y_test,y_pred)
plt.title("Actual vs. Predicted",size=30)
plt.xlabel("Actual ",size=30)
plt.ylabel("Predicted",size=30)
plt.show()
```

```
pred_y_df = pd.DataFrame({'Actual Value': y_test, 'Predicted Value': y_pred, 'Difference': y_test - y_pred})
pred_y_df[0:20]
```

ML Crash Course – Part V

- **Objectives of this lab:**

  25- **Logistic regression**

  26- **What is Logistic Regression**

  27- **Python Example**

Logistic regression

Logistic regression is a statistical method used for binary classification problems in machine learning. It models the probability that a given input belongs to a particular category. Unlike linear regression, which predicts continuous values, logistic regression predicts discrete outcomes.

Key Concepts

1. **Binary Outcome**:
   - The dependent variable is categorical with two possible outcomes (e.g., success/failure, yes/no, 0/1).
2. **Logistic Function**:
   - Logistic regression uses the logistic function (also known as the sigmoid function) to model the probability:

$$P(Y=1|X) = \frac{1}{1+e^{-(\beta_0+\beta_1 x_1+\beta_2 x_2+\ldots+\beta_n x_n)}}$$

   - Where:
     - P(Y=1∣X)P(Y=1|X)P(Y=1∣X) is the probability that the dependent variable YYY equals 1 given the independent variables XXX.
     - β0\beta_0β0 is the intercept, and β1,β2,...,βn\beta_1, \beta_2, ..., \beta_nβ1,β2,...,βn are the coefficients for the independent variables x1,x2,...,xnx_1, x_2, ..., x_nx1,x2,...,xn.
3. **Decision Boundary**:
   - The output of the logistic function is a probability ranging from 0 to 1. A threshold (commonly 0.5) is used to classify the output as one of the two categories.

4. **Loss Function**:

   - Logistic regression uses a loss function called binary cross-entropy (or log loss) to measure the difference between predicted probabilities and actual outcomes.

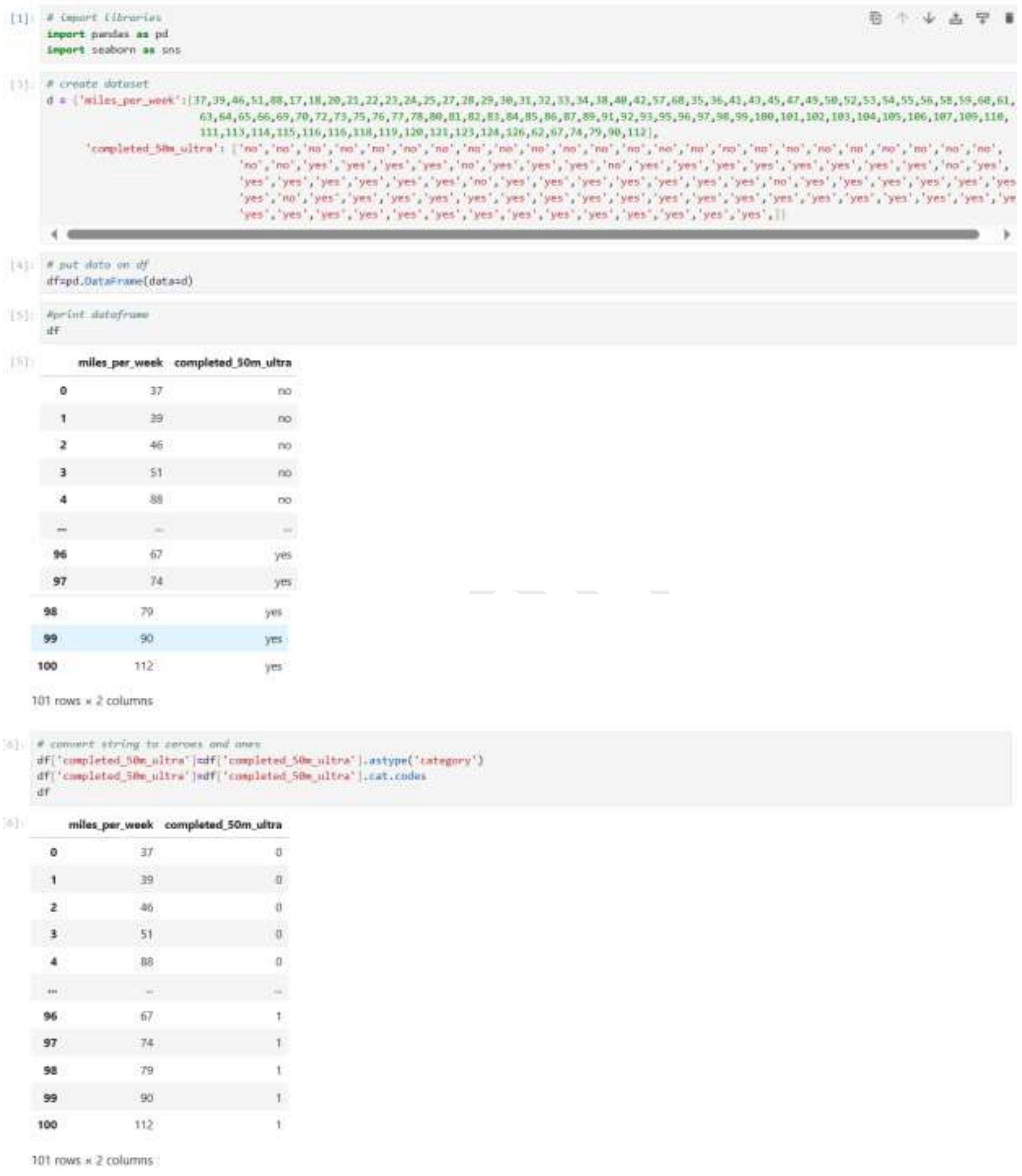
Applications

- Medical diagnosis (e.g., predicting the presence or absence of a disease).
- Customer churn prediction (e.g., whether a customer will leave a service).
- Credit scoring (e.g., classifying borrowers as low or high risk).

Advantages and Disadvantages

- **Advantages**:
  - Simple and easy to interpret.
  - Efficient for binary classification problems.
  - Can handle non-linear relationships by using polynomial or interaction terms.
- **Disadvantages**:
  - Assumes a linear relationship between the independent variables and the log-odds of the dependent variable.
  - Sensitive to outliers.
  - Not suitable for complex relationships without additional techniques.

Example



```
[1]: # Import Libraries
     import pandas as pd
     import seaborn as sns
```

```
[3]: # create dataset
     d = {'miles_per_week':[37,39,46,51,88,17,18,20,21,22,23,24,25,27,28,29,30,31,32,33,34,38,40,42,57,68,35,36,41,43,45,47,49,50,52,53,54,55,56,58,59,60,61,
                           63,64,65,66,69,70,72,73,75,76,77,78,80,81,82,83,84,85,86,87,89,91,92,93,95,96,97,98,99,100,101,102,103,104,105,106,107,109,110,
                           111,113,114,115,116,116,118,119,120,121,123,124,126,62,67,74,79,90,112],
          'completed_50m_ultra': ['no','no','no','no','no','no','no','no','no','no','no','no','no','no','no','no','no','no','no','no','no','no','no','no',
                                  'no','no','yes','yes','yes','yes','no','yes','yes','yes','no','yes','yes','yes','yes','yes','yes','yes','yes','no','yes',
                                  'yes','yes','yes','yes','yes','yes','no','yes','yes','yes','yes','yes','yes','yes','no','yes','yes','yes','yes','yes','yes
                                  'yes','no','yes','yes','yes','yes','yes','yes','yes','yes','yes','yes','yes','yes','yes','yes','yes','yes','yes','yes','ye
                                  'yes','yes','yes','yes','yes','yes','yes','yes','yes','yes','yes','yes','yes','yes',]}
```

```
[4]: # put data on df
     df=pd.DataFrame(data=d)
```

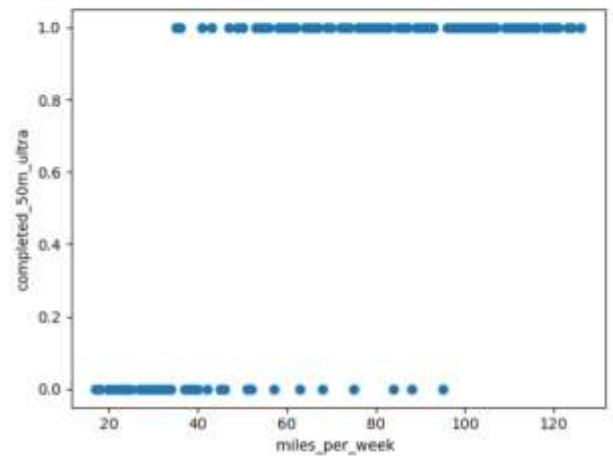
```
[5]: #print dataframe
     df
```

```
[5]:      miles_per_week  completed_50m_ultra
     0                37                   no
     1                39                   no
     2                46                   no
     3                51                   no
     4                88                   no
     ...             ...                  ...
     96               67                  yes
     97               74                  yes
     98               79                  yes
     99               90                  yes
     100             112                  yes

     101 rows × 2 columns
```

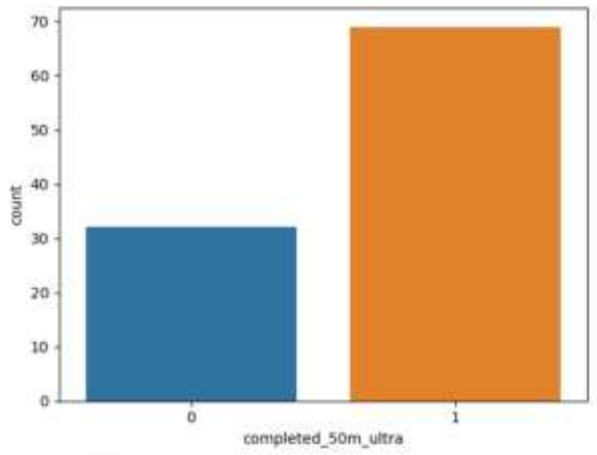
```
[6]: # convert string to zeroes and ones
     df['completed_50m_ultra']=df['completed_50m_ultra'].astype('category')
     df['completed_50m_ultra']=df['completed_50m_ultra'].cat.codes
     df
```

```
[6]:      miles_per_week  completed_50m_ultra
     0                37                    0
     1                39                    0
     2                46                    0
     3                51                    0
     4                88                    0
     ...             ...                  ...
     96               67                    1
     97               74                    1
     98               79                    1
     99               90                    1
     100             112                    1

     101 rows × 2 columns
```

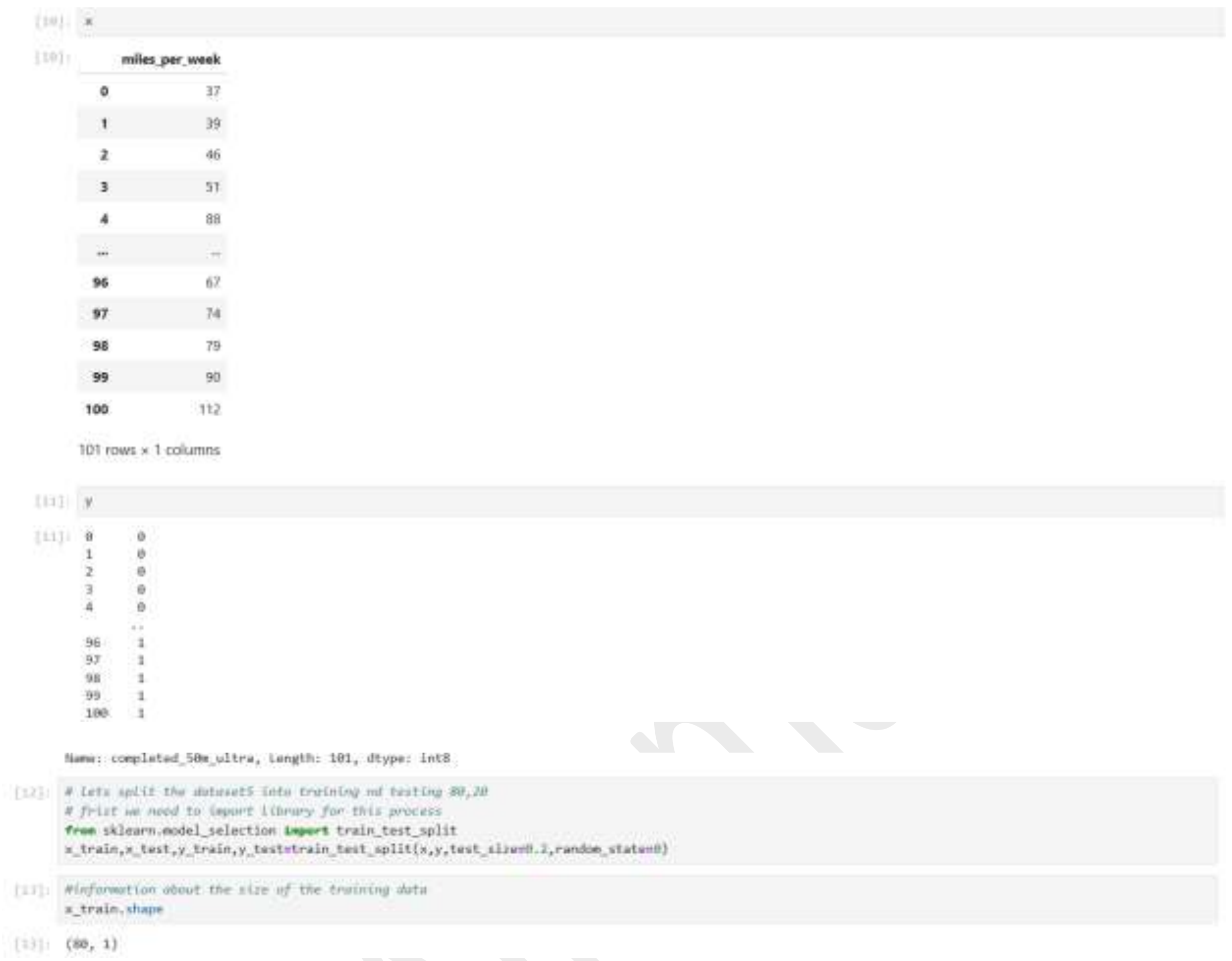
```
[7]: #plot the dataset
     from  matplotlib import pyplot as plt
     plt.scatter(df.miles_per_week,df.completed_50m_ultra)
     plt.xlabel("miles_per_week ")
     plt.ylabel("completed_50m_ultra")

[7]: Text(0, 0.5, 'completed_50m_ultra')
```



```
[8]: #to see how many completed_50m_ultra
     sns.countplot(x='completed_50m_ultra',data=df)

[8]: <Axes: xlabel='completed_50m_ultra', ylabel='count'>
```



```
[9]: # define x and y
     x=df.iloc[:,0:1]
     y=df.iloc[:,1]

[10]: x
```



```
[10]: x
```

[10]:

	miles_per_week
0	37
1	39
2	46
3	51
4	88
...	...
96	67
97	74
98	79
99	90
100	112

101 rows × 1 columns

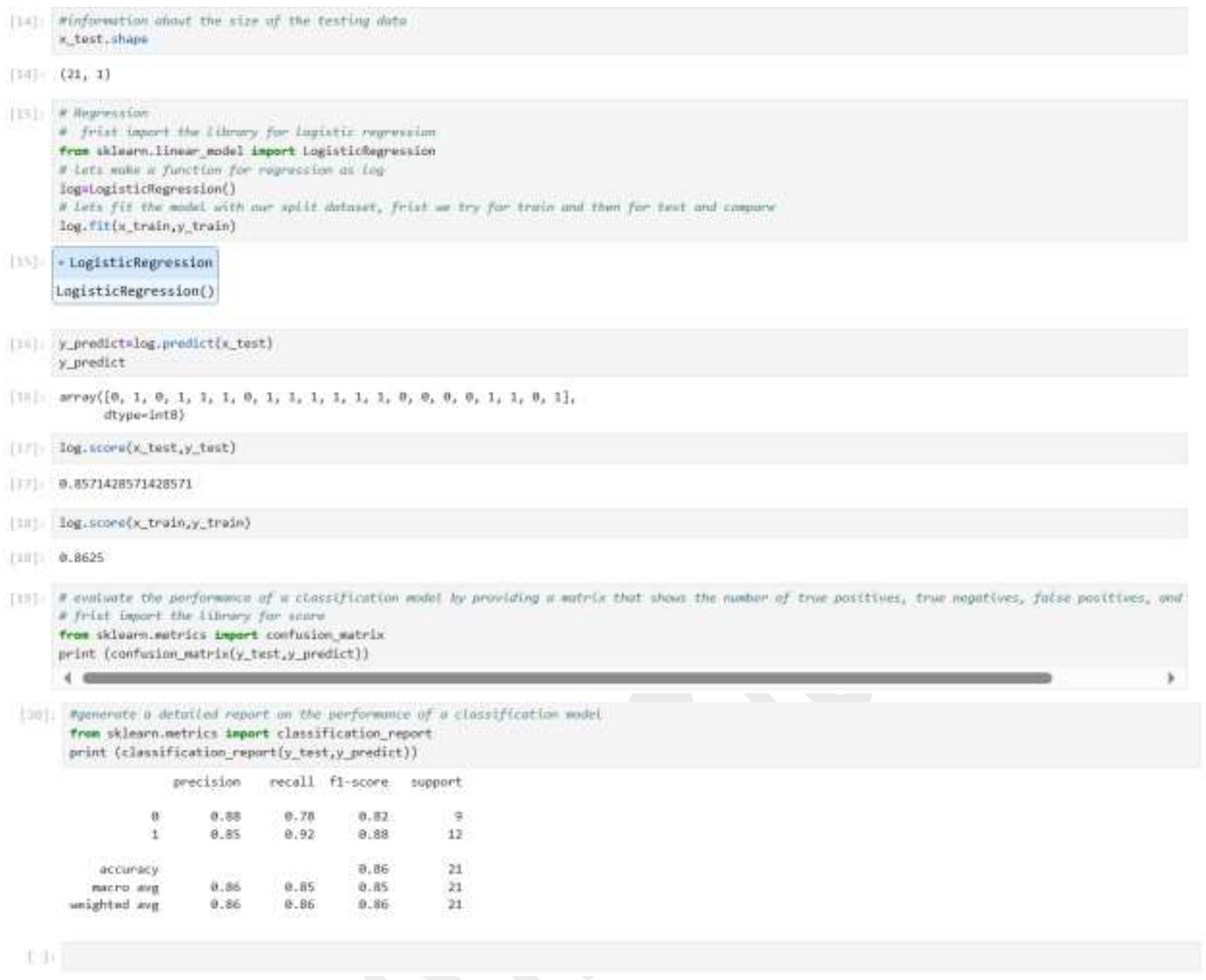
```
[11]: y
```

```
[11]: 0      0
      1      0
      2      0
      3      0
      4      0
            ..
      96     1
      97     1
      98     1
      99     1
      100    1

Name: completed_50m_ultra, Length: 101, dtype: int8
```

```
[12]: # Lets split the dataset5 into training nd testing 80,20
      # frist we need to import library for this process
      from sklearn.model_selection import train_test_split
      x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=0)
```

```
[13]: #information about the size of the training data
      x_train.shape
```

```
[13]: (80, 1)
```



```
[14]: #information about the size of the testing data
      x_test.shape

[14]: (21, 1)

[15]: # Regression
      #  frist import the library for logistic regression
      from sklearn.linear_model import LogisticRegression
      # lets make a function for regression as log
      log=LogisticRegression()
      # lets fit the model with our split dataset, frist we try for train and then for test and compare
      log.fit(x_train,y_train)

[15]: ▾ LogisticRegression
      LogisticRegression()

[16]: y_predict=log.predict(x_test)
      y_predict

[16]: array([0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1],
            dtype=int8)

[17]: log.score(x_test,y_test)

[17]: 0.8571428571428571

[18]: log.score(x_train,y_train)

[18]: 0.8625

[19]: # evaluate the performance of a classification model by providing a matrix that shows the number of true positives, true negatives, false positives, and
      # frist import the library for score
      from sklearn.metrics import confusion_matrix
      print (confusion_matrix(y_test,y_predict))

[20]: #generate a detailed report on the performance of a classification model
      from sklearn.metrics import classification_report
      print (classification_report(y_test,y_predict))

              precision    recall  f1-score   support

           0       0.88      0.78      0.82         9
           1       0.85      0.92      0.88        12

    accuracy                           0.86        21
   macro avg       0.86      0.85      0.85        21
weighted avg       0.86      0.86      0.86        21

[ ]:
```

ML Crash Course – Part VI

- **Objectives of this lab:**

  28- **Neural Networks (NN)**

  29- **What are Neural Networks**

  30- **Python Example**

Neural Networks

Neural Networks (NN) are a class of algorithms inspired by the structure and function of the human brain. They are widely used in machine learning for tasks such as classification, regression, and pattern recognition. Here's a breakdown of the key concepts:

Key Concepts

1. **Structure**:
    - **Neurons**: The basic units of a neural network, analogous to biological neurons. Each neuron receives inputs, processes them, and produces an output.
    - **Layers**:
        - **Input Layer**: The first layer that receives input data.
        - **Hidden Layers**: Intermediate layers that perform computations and extract features. There can be one or more hidden layers.
        - **Output Layer**: The final layer that produces the output of the network.
2. **Activation Functions**:
    - Functions applied to the output of neurons to introduce non-linearity into the model. Common activation functions include:
        - **Sigmoid**: Outputs values between 0 and 1.
        - **ReLU (Rectified Linear Unit)**: Outputs the input directly if positive; otherwise, it outputs zero.
        - **Softmax**: Used in multi-class classification to provide probabilities for each class.

3. **Forward Propagation**:
   - The process of passing input data through the network to obtain an output. Each neuron's output is calculated based on its input and the weights assigned to those inputs.
4. **Backpropagation**:
   - An algorithm used to train neural networks. It involves calculating the gradient of the loss function (which measures prediction error) with respect to each weight by the chain rule, allowing weights to be updated to minimize error.
5. **Loss Function**:
   - A function that measures the difference between the predicted output and the actual output. Common loss functions include mean squared error (for regression) and cross-entropy (for classification).

Types of Neural Networks

1. **Feedforward Neural Networks**: The simplest type, where connections between nodes do not form cycles. Data moves in one direction—from input to output.
2. **Convolutional Neural Networks (CNNs)**: Primarily used for image processing, they utilize convolutional layers to automatically detect features like edges and textures.
3. **Recurrent Neural Networks (RNNs)**: Designed for sequential data, such as time series or natural language, they have connections that loop back to allow information persistence.
4. **Generative Adversarial Networks (GANs)**: Comprises two networks—a generator and a discriminator—that compete against each other to produce new data samples.

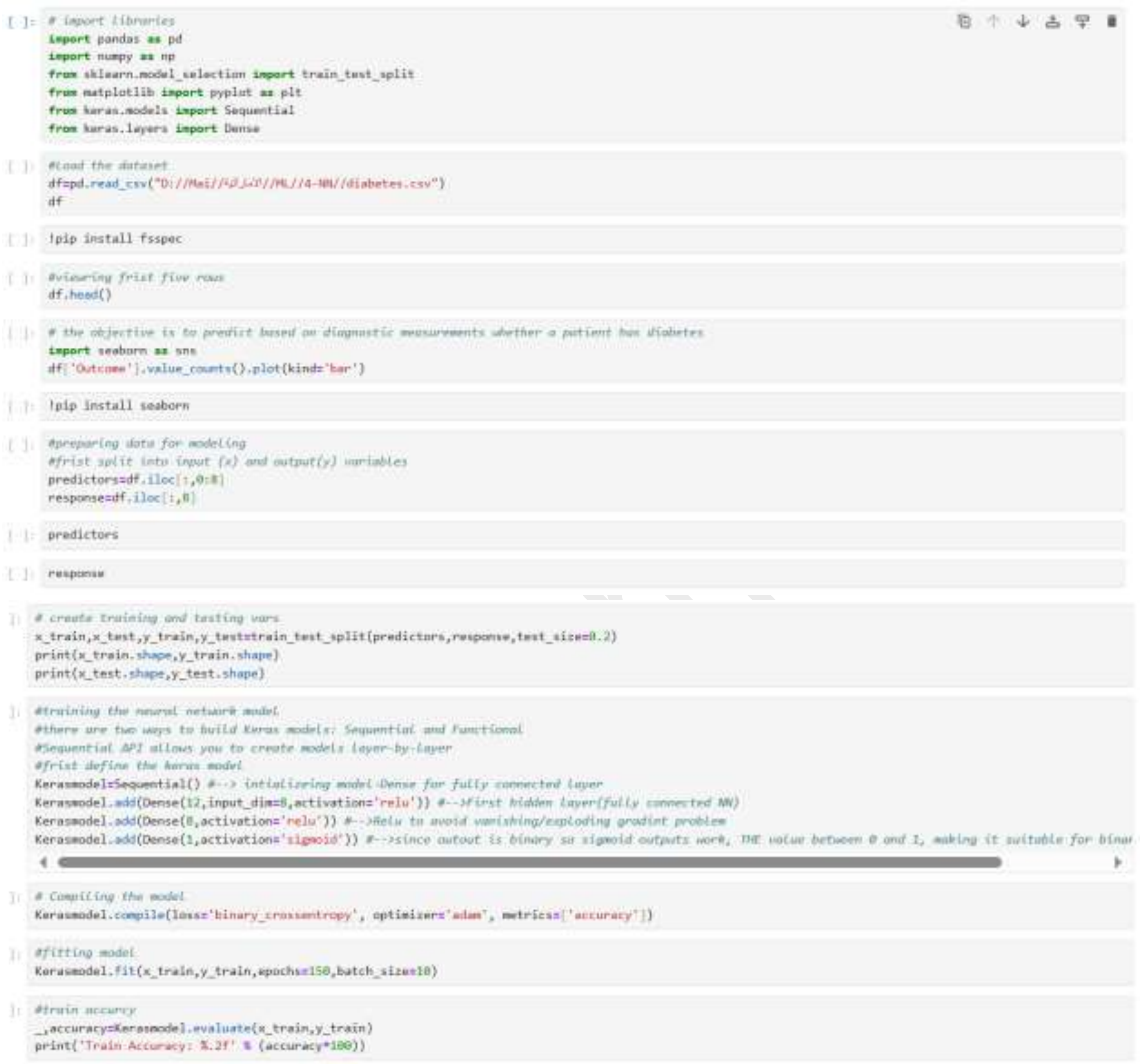
Applications

- Image and speech recognition
- Natural language processing (NLP)
- Game playing (e.g., AlphaGo)
- Autonomous vehicles
- Financial forecasting

Advantages and Disadvantages

- **Advantages**:
  - Capable of modeling complex relationships and patterns.
  - Can learn from large amounts of data.
  - Highly flexible and adaptable to various tasks.
- **Disadvantages**:
  - Requires a substantial amount of data and computational resources.
  - Can be prone to overfitting if not properly regularized.
  - Often considered a "black box," making interpretation of results challenging.

Example



```
[ ]: # import libraries
     import pandas as pd
     import numpy as np
     from sklearn.model_selection import train_test_split
     from matplotlib import pyplot as plt
     from keras.models import Sequential
     from keras.layers import Dense

[ ]: #Load the dataset
     df=pd.read_csv("D://Mai//[illegible]//ML//4-NN//diabetes.csv")
     df

[ ]: !pip install fsspec

[ ]: #viewing frist five rows
     df.head()

[ ]: # the objective is to predict based on diagnostic measurements whether a patient has diabetes
     import seaborn as sns
     df['Outcome'].value_counts().plot(kind='bar')

[ ]: !pip install seaborn

[ ]: #preparing data for modeling
     #frist split into input (x) and output(y) variables
     predictors=df.iloc[:,0:8]
     response=df.iloc[:,8]

[ ]: predictors

[ ]: response

]: # create training and testing vars
   x_train,x_test,y_train,y_test=train_test_split(predictors,response,test_size=0.2)
   print(x_train.shape,y_train.shape)
   print(x_test.shape,y_test.shape)

]: #training the neural network model
   #there are two ways to build Keras models: Sequential and Functional
   #Sequential API allows you to create models layer-by-layer
   #frist define the keras model
   Kerasmodel=Sequential() #--> intializing model Dense for fully connected layer
   Kerasmodel.add(Dense(12,input_dim=8,activation='relu')) #-->First hidden layer(fully connected NN)
   Kerasmodel.add(Dense(8,activation='relu')) #-->Relu to avoid vanishing/exploding gradint problem
   Kerasmodel.add(Dense(1,activation='sigmoid')) #-->since outout is binary so sigmoid outputs work, THE value between 0 and 1, making it suitable for binar

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

]: #fitting model
   Kerasmodel.fit(x_train,y_train,epochs=150,batch_size=10)

]: #train accurcy
   _,accuracy=Kerasmodel.evaluate(x_train,y_train)
   print('Train Accuracy: %.2f' % (accuracy*100))
```

ML Crash Course – Part VII

- **Objectives of this lab:**

  31- **Decision Trees (DT)**

  32- **What is Decision Tree**

  33- **Python Example**

Decision Trees

Decision Trees (DT) are a popular machine learning algorithm used for both classification and regression tasks. They model decisions and their possible consequences, visually representing decisions in a tree-like structure. Here's an overview of key concepts related to decision trees:

Key Concepts

1. **Structure**:
   - **Nodes**: Each internal node represents a feature (attribute) of the dataset, and each branch represents a decision rule.
   - **Leaves**: Terminal nodes that represent the outcome or prediction (e.g., class labels for classification tasks).
2. **Splitting**:
   - The process of dividing the dataset into subsets based on the value of a chosen feature. The goal is to create splits that maximize the separation of classes (for classification) or minimize prediction error (for regression).
3. **Impurity Measures**:
   - Decision trees use metrics to evaluate the quality of splits. Common measures include:
     - **Gini Impurity**: Measures the impurity of a node; lower values indicate better classification.
     - **Entropy**: Used in information gain to determine the effectiveness of a split.
     - **Mean Squared Error (MSE)**: Used for regression trees to measure the variance in the target variable.

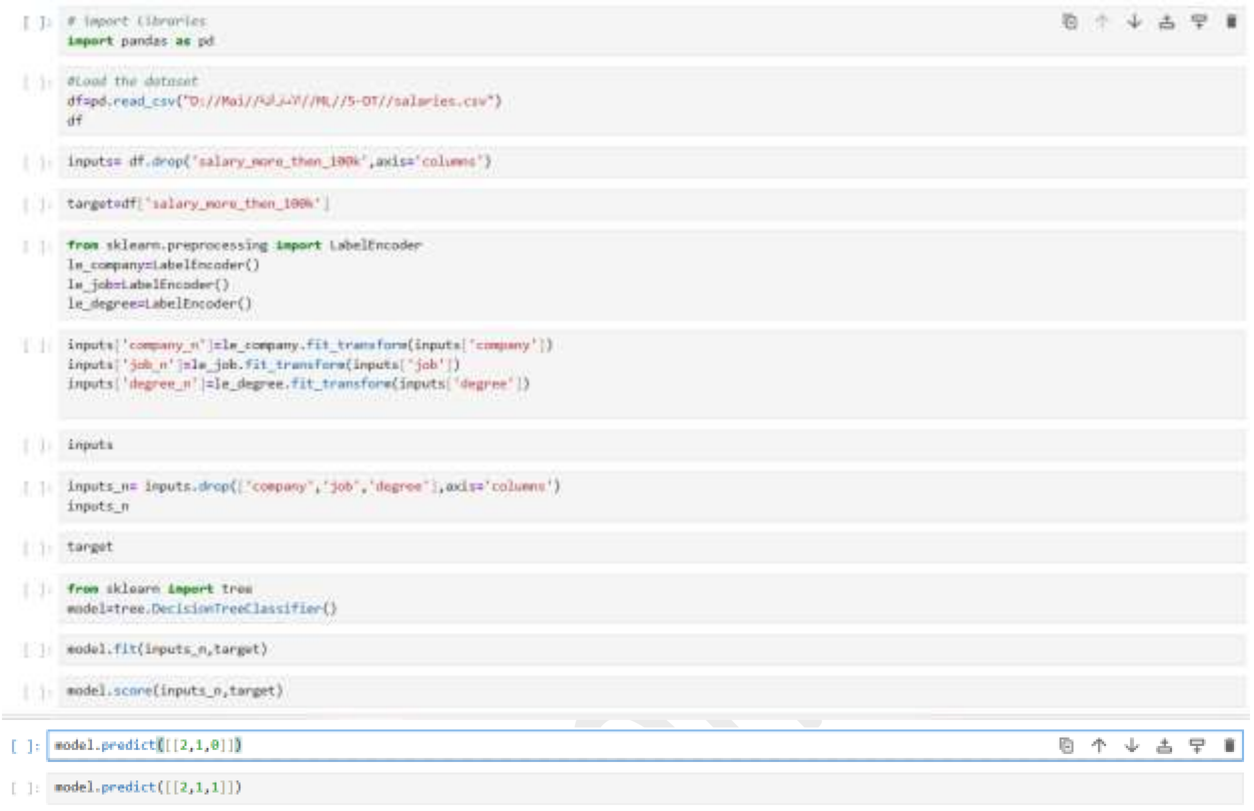
4. **Pruning**:
   - The process of removing branches that have little importance to reduce overfitting. Pruning helps to simplify the model and improve its generalization to unseen data.

5. **Advantages and Disadvantages**:
   - **Advantages**:
     - Easy to interpret and visualize.
     - Requires little data preprocessing (e.g., no need for normalization).
     - Handles both numerical and categorical data.
   - **Disadvantages**:
     - Prone to overfitting, especially with complex trees.
     - Sensitive to noisy data and outliers.
     - Can be biased if one class dominates the dataset.

Applications

- Customer segmentation
- Fraud detection
- Credit scoring
- Medical diagnosis
- Predictive maintenance

Examples



```
[ ]: # import libraries
     import pandas as pd

[ ]: #Load the dataset
     df=pd.read_csv("D://Mai//[illegible]//ML//5-DT//salaries.csv")
     df

[ ]: inputs= df.drop('salary_more_then_100k',axis='columns')

[ ]: target=df['salary_more_then_100k']

[ ]: from sklearn.preprocessing import LabelEncoder
     le_company=LabelEncoder()
     le_job=LabelEncoder()
     le_degree=LabelEncoder()

[ ]: inputs['company_n']=le_company.fit_transform(inputs['company'])
     inputs['job_n']=le_job.fit_transform(inputs['job'])
     inputs['degree_n']=le_degree.fit_transform(inputs['degree'])

[ ]: inputs

[ ]: inputs_n= inputs.drop(['company','job','degree'],axis='columns')
     inputs_n

[ ]: target

[ ]: from sklearn import tree
     model=tree.DecisionTreeClassifier()

[ ]: model.fit(inputs_n,target)

[ ]: model.score(inputs_n,target)

[ ]: model.predict([[2,1,0]])

[ ]: model.predict([[2,1,1]])
```

ML Crash Course – Part VIII

- **Objectives of this lab:**

34- **Decision Trees (DT) Classification**

35- **What is Decision Tree Classification?**

36- **Python Example**

Decision Trees (DT) Classification

DT Classification refers to the use of Decision Trees as a method for classification tasks in machine learning. Decision trees are intuitive and powerful models that represent decisions and their possible consequences in a tree-like structure.

Key Concepts

1. **Structure:**
   - Root Node: Represents the entire dataset and splits into branches based on decision rules.
   - Internal Nodes: Represent features used to make decisions.
   - Leaves: Terminal nodes that represent the predicted class labels.
2. **Splitting Criteria:**
   - Decision trees use various criteria to determine how to split nodes. Common splitting criteria include:
     - Gini Impurity: Measures the impurity of a node; lower values are preferred.
     - Entropy: Measures the level of disorder or uncertainty; used to calculate information gain.
     - Information Gain: The reduction in entropy or impurity after a split.
3. **Training the Model:**
   - The decision tree is built by recursively splitting the data based on the selected feature and splitting criteria until a stopping condition is met (e.g., maximum depth, minimum samples per leaf).

4. **Pruning:**
    - After the tree is built, pruning may be applied to remove branches that have little predictive power to avoid overfitting.
5. **Prediction:**
    - New instances are classified by traversing the tree from the root to a leaf node based on the feature values of the instance.

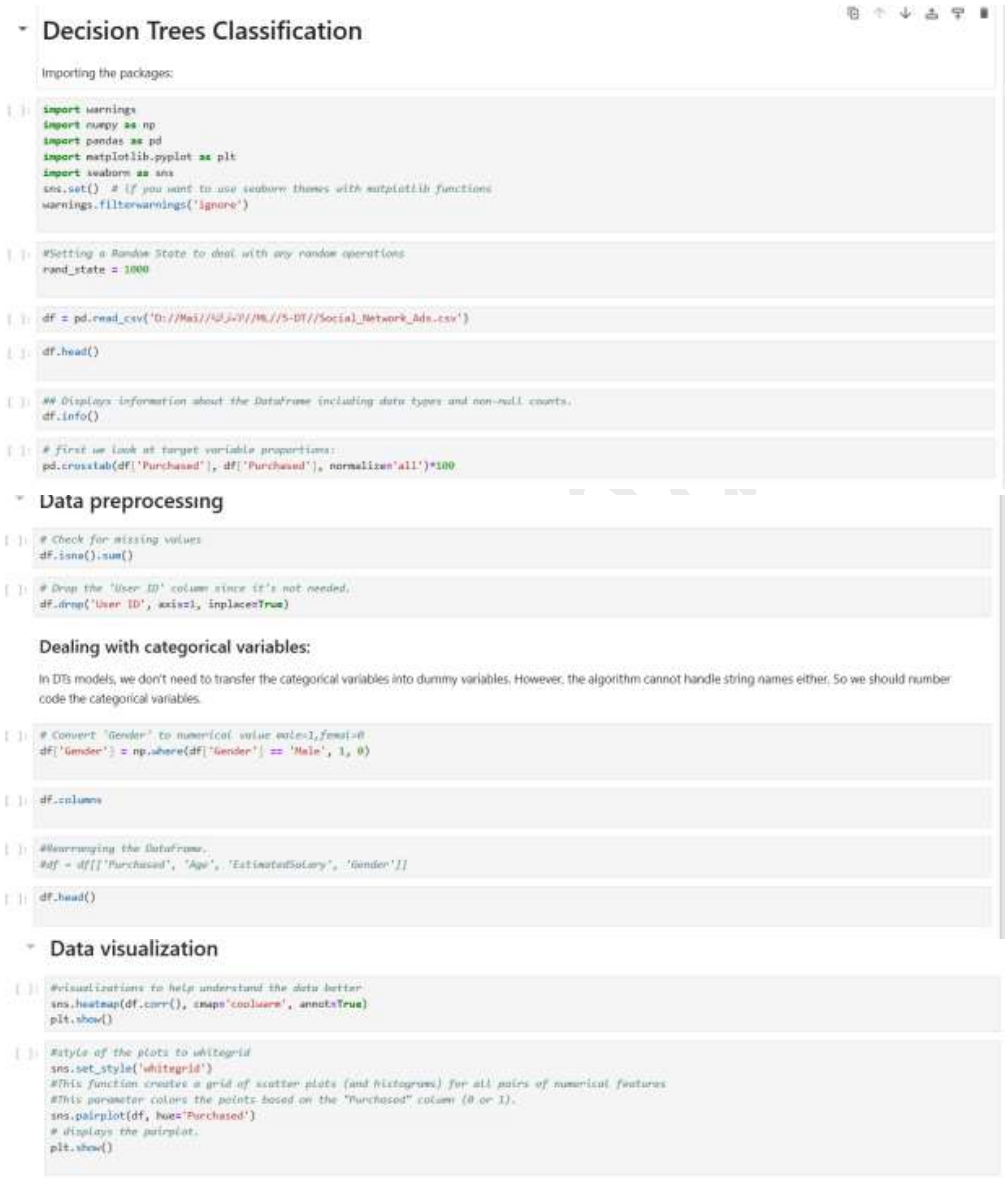
Advantages and Disadvantages

- **Advantages:**
    - Easy to interpret and visualize, making them user-friendly.
    - Requires little data preprocessing (e.g., no need for feature scaling).
    - Can handle both numerical and categorical data.
- **Disadvantages:**
    - Prone to overfitting, especially with deep trees.
    - Sensitive to noise and outliers in the data.
    - May not generalize well if the tree is too complex.

Applications

- Medical diagnosis (e.g., predicting the presence of diseases).
- Customer segmentation (e.g., classifying customers based on behavior).
- Fraud detection (e.g., identifying fraudulent transactions).

Example



Decision Trees Classification

Importing the packages:

```
import warnings
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()  # if you want to use seaborn themes with matplotlib functions
warnings.filterwarnings('ignore')
```

```
#Setting a Random State to deal with any random operations
rand_state = 1000
```

```
df = pd.read_csv('D://Mai//[illegible]//ML//5-DT//Social_Network_Ads.csv')
```

```
df.head()
```

```
## Displays information about the DataFrame including data types and non-null counts.
df.info()
```

```
# first we look at target variable proportions:
pd.crosstab(df['Purchased'], df['Purchased'], normalize='all')*100
```

Data preprocessing

```
# Check for missing values
df.isna().sum()
```

```
# Drop the 'User ID' column since it's not needed.
df.drop('User ID', axis=1, inplace=True)
```

Dealing with categorical variables:

In DTs models, we don't need to transfer the categorical variables into dummy variables. However, the algorithm cannot handle string names either. So we should number code the categorical variables.

```
# Convert 'Gender' to numerical value male=1,femal=0
df['Gender'] = np.where(df['Gender'] == 'Male', 1, 0)
```

```
df.columns
```

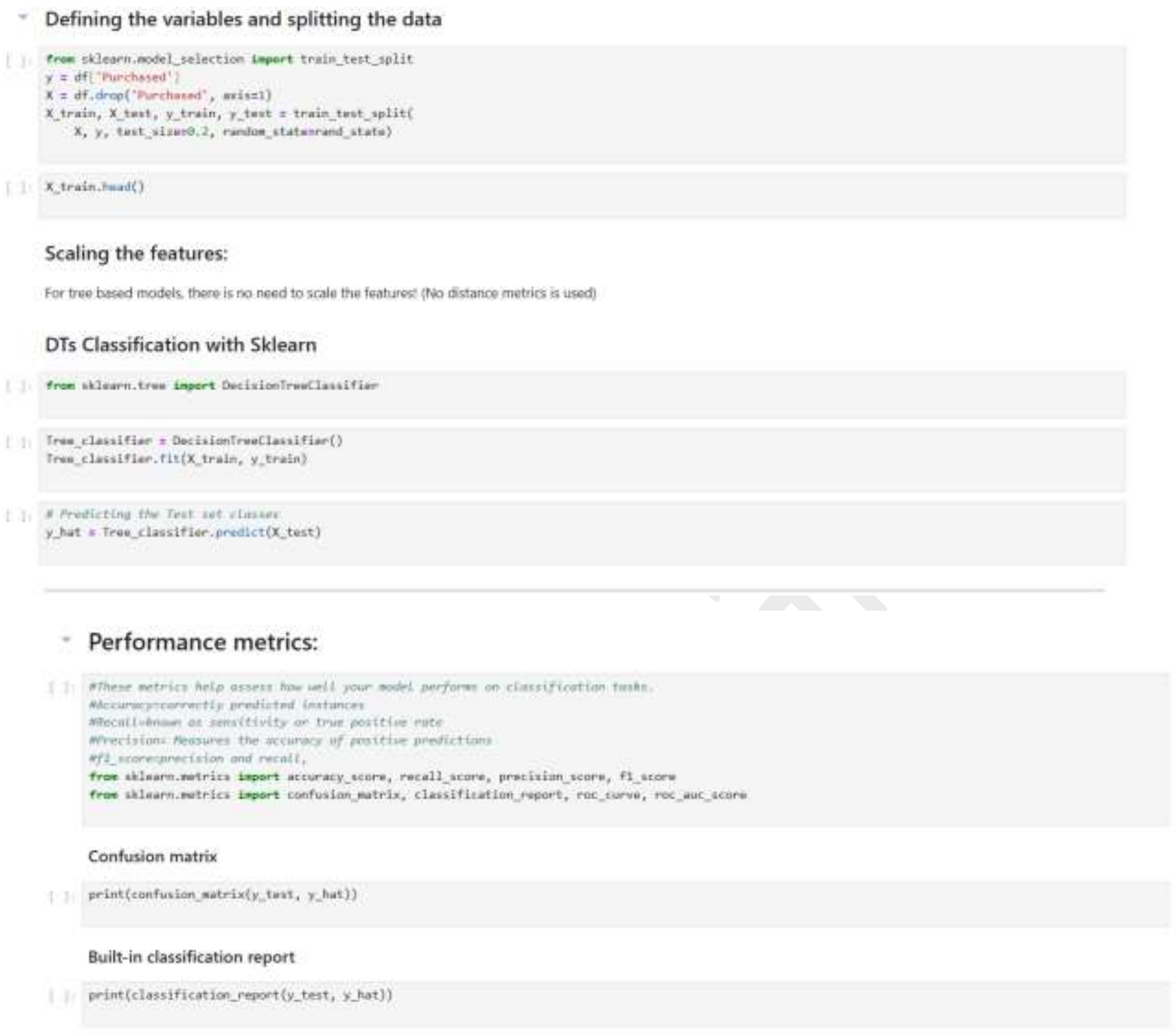
```
#Rearranging the DataFrame.
#df = df[['Purchased', 'Age', 'EstimatedSalary', 'Gender']]
```

```
df.head()
```

Data visualization

```
#visualizations to help understand the data better
sns.heatmap(df.corr(), cmap='coolwarm', annot=True)
plt.show()
```

```
#style of the plots to whitegrid
sns.set_style('whitegrid')
#This function creates a grid of scatter plots (and histograms) for all pairs of numerical features
#This parameter colors the points based on the "Purchased" column (0 or 1).
sns.pairplot(df, hue='Purchased')
# displays the pairplot.
plt.show()
```



Defining the variables and splitting the data

```
from sklearn.model_selection import train_test_split
y = df['Purchased']
X = df.drop('Purchased', axis=1)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=rand_state)
```

```
X_train.head()
```

Scaling the features:

For tree based models, there is no need to scale the features! (No distance metrics is used)

DTs Classification with Sklearn

```
from sklearn.tree import DecisionTreeClassifier
```

```
Tree_classifier = DecisionTreeClassifier()
Tree_classifier.fit(X_train, y_train)
```

```
# Predicting the Test set classes
y_hat = Tree_classifier.predict(X_test)
```

Performance metrics:

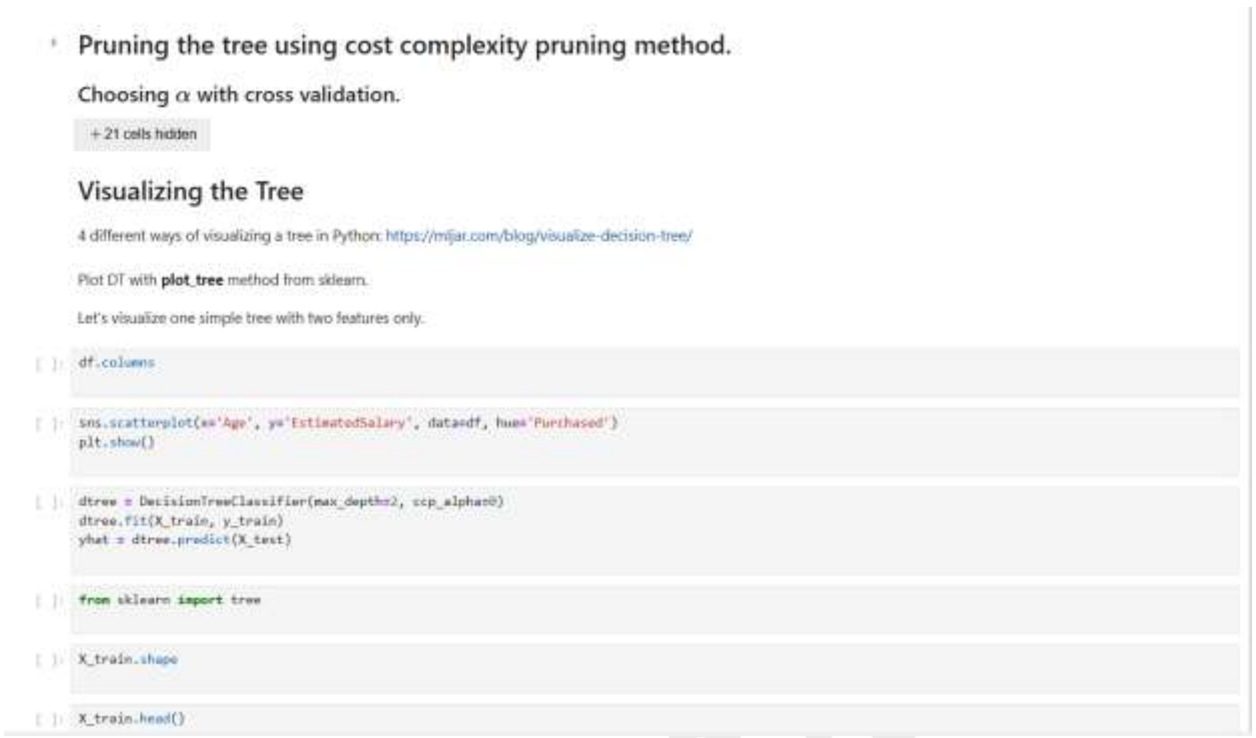
```
#These metrics help assess how well your model performs on classification tasks.
#Accuracy:correctly predicted instances
#Recall:known as sensitivity or true positive rate
#Precision: Measures the accuracy of positive predictions
#f1_score:precision and recall,
from sklearn.metrics import accuracy_score, recall_score, precision_score, f1_score
from sklearn.metrics import confusion_matrix, classification_report, roc_curve, roc_auc_score
```

Confusion matrix

```
print(confusion_matrix(y_test, y_hat))
```

Built-in classification report

```
print(classification_report(y_test, y_hat))
```



Pruning the tree using cost complexity pruning method.

Choosing α with cross validation.

+ 21 cells hidden

Visualizing the Tree

4 different ways of visualizing a tree in Python: https://mljar.com/blog/visualize-decision-tree/

Plot DT with **plot_tree** method from sklearn.

Let's visualize one simple tree with two features only.

```
[ ]: df.columns
```

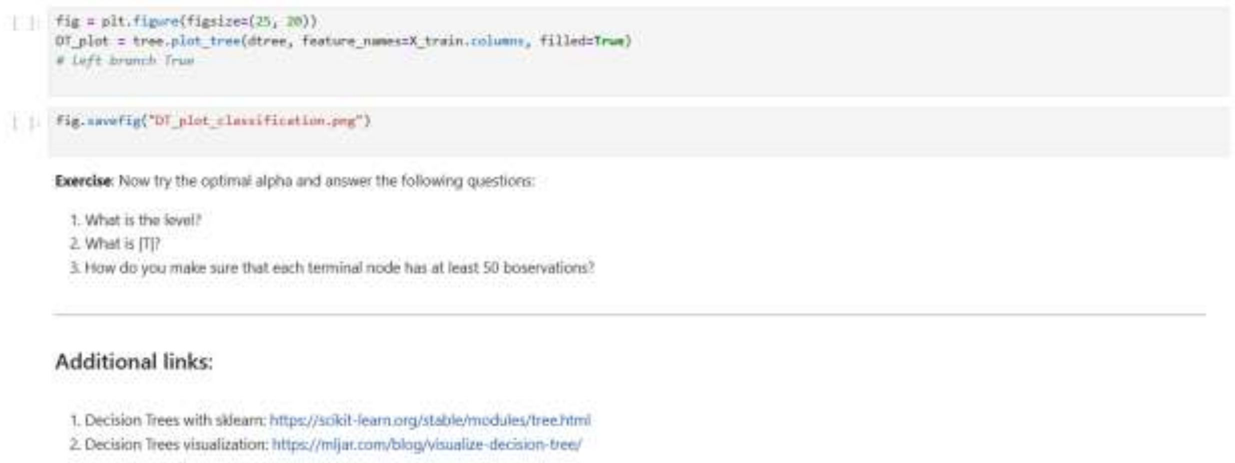
```
[ ]: sns.scatterplot(x='Age', y='EstimatedSalary', data=df, hue='Purchased')
     plt.show()
```

```
[ ]: dtree = DecisionTreeClassifier(max_depth=2, ccp_alpha=0)
     dtree.fit(X_train, y_train)
     yhat = dtree.predict(X_test)
```

```
[ ]: from sklearn import tree
```

```
[ ]: X_train.shape
```

```
[ ]: X_train.head()
```



```
[ ]: fig = plt.figure(figsize=(25, 20))
     DT_plot = tree.plot_tree(dtree, feature_names=X_train.columns, filled=True)
     # Left branch True
```

```
[ ]: fig.savefig("DT_plot_classification.png")
```

Exercise: Now try the optimal alpha and answer the following questions:

1. What is the level?
2. What is |T|?
3. How do you make sure that each terminal node has at least 50 boservations?

Additional links:

1. Decision Trees with sklearn: https://scikit-learn.org/stable/modules/tree.html
2. Decision Trees visualization: https://mljar.com/blog/visualize-decision-tree/

ML Crash Course – Part IX

- **Objectives of this lab:**

  37- **Decision Trees (DT) Regression**

  38- **What is Decision Tree Regression ?**

  39- **Python Example**

DT Regression

DT Regression refers to the use of Decision Trees for regression tasks in machine learning. Unlike classification trees, which predict categorical outcomes, regression trees are designed to predict continuous numerical values. Here's an overview of key concepts related to decision tree regression:

Key Concepts

1. Structure:

- Similar to decision tree classification, the structure consists of nodes and leaves:
  - Root Node: Represents the entire dataset.
  - Internal Nodes: Split the data based on decision rules.
  - Leaves: Represent the predicted continuous values.

2. Splitting Criteria:

- The objective is to minimize the variance in the target variable within each split. Common criteria include:
  - Mean Squared Error (MSE): Measures the average squared difference between actual and predicted values.
  - Mean Absolute Error (MAE): Measures the average absolute difference between actual and predicted values.

3. Training the Model:

- The decision tree is built by recursively splitting the dataset into subsets based on feature values until a stopping condition is met (e.g., maximum depth, minimum samples per leaf).

4. Prediction:

- For a given input, the tree traverses from the root to a leaf node, where the predicted value is the average of the target values of the training samples that reached that leaf.

5. Pruning:

- Pruning may be applied to reduce the complexity of the tree and prevent overfitting, similar to classification trees.

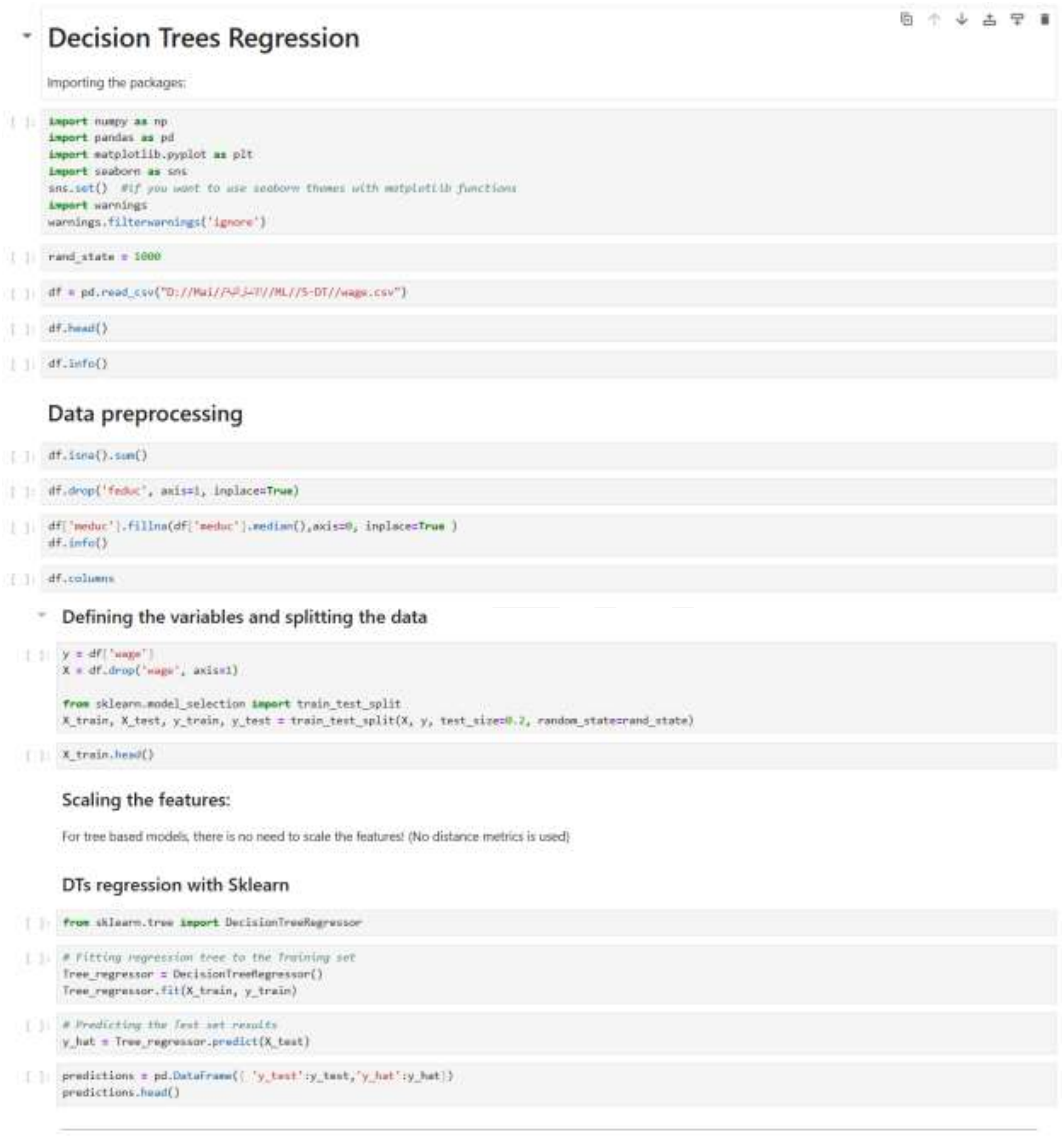
Advantages and Disadvantages

- **Advantages:**
  - Easy to interpret and visualize, making them user-friendly.
  - Works well with both numerical and categorical data.
  - Captures non-linear relationships without requiring complex transformations.
- **Disadvantages:**
  - Prone to overfitting if the tree is too deep.
  - Sensitive to small variations in the data, which can lead to different trees.
  - May not perform well if the data is sparse or noisy.

Applications

- Predicting house prices based on various features (e.g., size, location).
- Estimating sales revenue based on marketing spend.
- Forecasting demand in supply chain management.

Example



Decision Trees Regression

Importing the packages:

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()  #if you want to use seaborn themes with matplotlib functions
import warnings
warnings.filterwarnings('ignore')
```

```python
rand_state = 1000
```

```python
df = pd.read_csv("D://Mai//[illegible]//ML//5-DT//wage.csv")
```

```python
df.head()
```

```python
df.info()
```

Data preprocessing

```python
df.isna().sum()
```

```python
df.drop('feduc', axis=1, inplace=True)
```

```python
df['meduc'].fillna(df['meduc'].median(),axis=0, inplace=True )
df.info()
```

```python
df.columns
```

Defining the variables and splitting the data

```python
y = df['wage']
X = df.drop('wage', axis=1)

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=rand_state)
```

```python
X_train.head()
```

Scaling the features:

For tree based models, there is no need to scale the features! (No distance metrics is used)

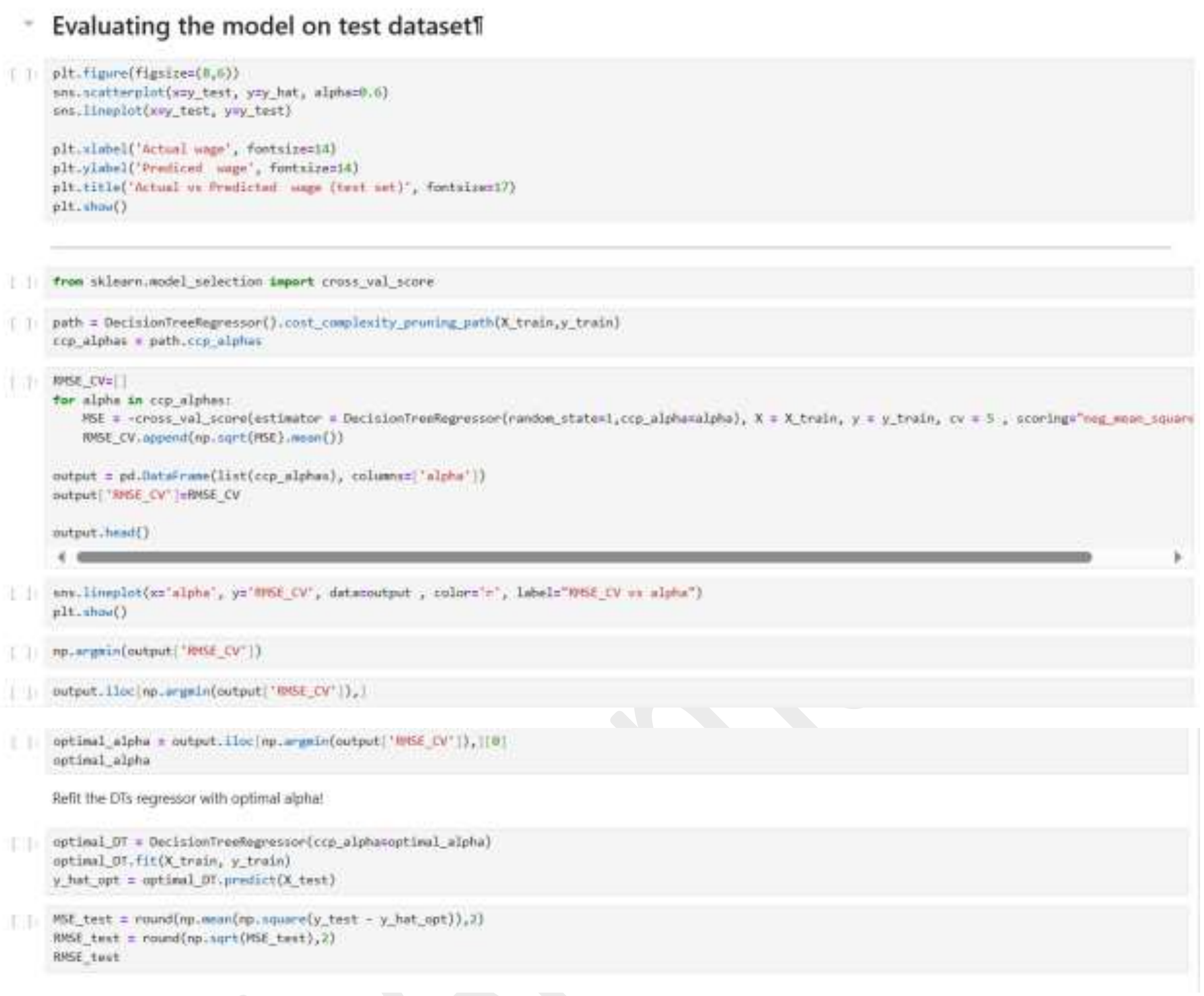
DTs regression with Sklearn

```python
from sklearn.tree import DecisionTreeRegressor
```

```python
# Fitting regression tree to the Training set
Tree_regressor = DecisionTreeRegressor()
Tree_regressor.fit(X_train, y_train)
```

```python
# Predicting the Test set results
y_hat = Tree_regressor.predict(X_test)
```

```python
predictions = pd.DataFrame({ 'y_test':y_test,'y_hat':y_hat})
predictions.head()
```



Evaluating the model on test dataset¶

```
plt.figure(figsize=(8,6))
sns.scatterplot(x=y_test, y=y_hat, alpha=0.6)
sns.lineplot(x=y_test, y=y_test)

plt.xlabel('Actual wage', fontsize=14)
plt.ylabel('Prediced  wage', fontsize=14)
plt.title('Actual vs Predicted  wage (test set)', fontsize=17)
plt.show()
```

```
from sklearn.model_selection import cross_val_score
```

```
path = DecisionTreeRegressor().cost_complexity_pruning_path(X_train,y_train)
ccp_alphas = path.ccp_alphas
```

```
RMSE_CV=[]
for alpha in ccp_alphas:
    MSE = -cross_val_score(estimator = DecisionTreeRegressor(random_state=1,ccp_alpha=alpha), X = X_train, y = y_train, cv = 5 , scoring="neg_mean_square
    RMSE_CV.append(np.sqrt(MSE).mean())

output = pd.DataFrame(list(ccp_alphas), columns=['alpha'])
output['RMSE_CV']=RMSE_CV

output.head()
```

```
sns.lineplot(x='alpha', y='RMSE_CV', data=output , color='r', label="RMSE_CV vs alpha")
plt.show()
```

```
np.argmin(output['RMSE_CV'])
```

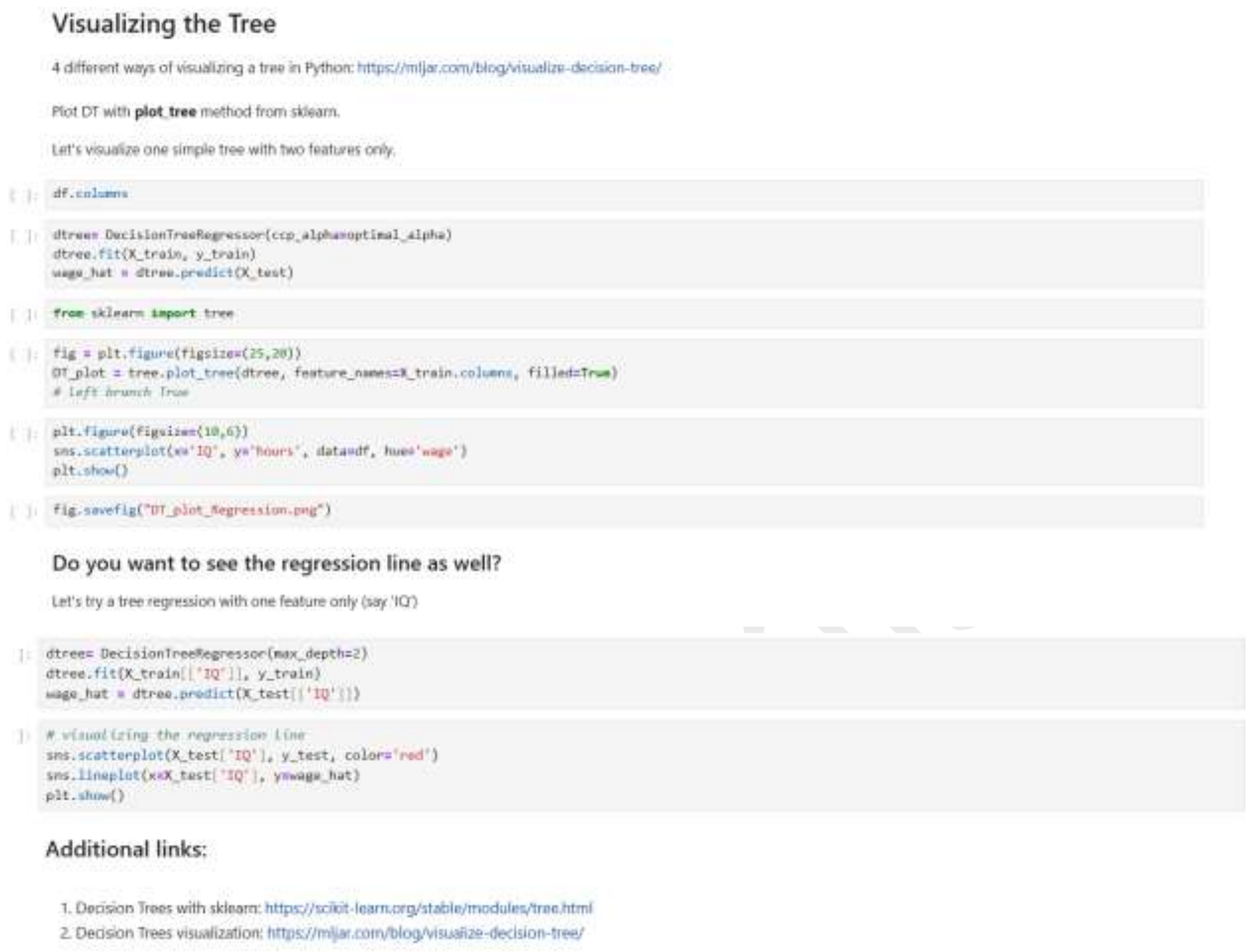
```
output.iloc[np.argmin(output['RMSE_CV']),]
```

```
optimal_alpha = output.iloc[np.argmin(output['RMSE_CV']),][0]
optimal_alpha
```

Refit the DTs regressor with optimal alpha!

```
optimal_DT = DecisionTreeRegressor(ccp_alpha=optimal_alpha)
optimal_DT.fit(X_train, y_train)
y_hat_opt = optimal_DT.predict(X_test)
```

```
MSE_test = round(np.mean(np.square(y_test - y_hat_opt)),2)
RMSE_test = round(np.sqrt(MSE_test),2)
RMSE_test
```



Visualizing the Tree

4 different ways of visualizing a tree in Python: https://mljar.com/blog/visualize-decision-tree/

Plot DT with **plot_tree** method from sklearn.

Let's visualize one simple tree with two features only.

```
[ ]: df.columns
```

```
[ ]: dtree= DecisionTreeRegressor(ccp_alpha=optimal_alpha)
     dtree.fit(X_train, y_train)
     wage_hat = dtree.predict(X_test)
```

```
[ ]: from sklearn import tree
```

```
[ ]: fig = plt.figure(figsize=(25,20))
     DT_plot = tree.plot_tree(dtree, feature_names=X_train.columns, filled=True)
     # left branch True
```

```
[ ]: plt.figure(figsize=(10,6))
     sns.scatterplot(x='IQ', y='hours', data=df, hue='wage')
     plt.show()
```

```
[ ]: fig.savefig("DT_plot_Regression.png")
```

Do you want to see the regression line as well?

Let's try a tree regression with one feature only (say 'IQ')

```
]: dtree= DecisionTreeRegressor(max_depth=2)
   dtree.fit(X_train[['IQ']], y_train)
   wage_hat = dtree.predict(X_test[['IQ']])
```

```
]: # visualizing the regression line
   sns.scatterplot(X_test['IQ'], y_test, color='red')
   sns.lineplot(x=X_test['IQ'], y=wage_hat)
   plt.show()
```

Additional links:

1. Decision Trees with sklearn: https://scikit-learn.org/stable/modules/tree.html
2. Decision Trees visualization: https://mljar.com/blog/visualize-decision-tree/

ML Crash Course – Part X

- **Objectives of this lab:**

    40- Convolutional Neural Networks (CNNs)

    41- What are Convolutional Neural Networks?

    42- Python Example

Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a specialized type of neural network primarily used for processing structured grid data, such as images. They are particularly effective in tasks like image recognition, classification, and object detection. Here's an overview of CNNs:

Key Concepts

1. **Architecture**:
    - **Input Layer**: Accepts the raw pixel values of the image.
    - **Convolutional Layers**: Apply convolution operations to extract features from the input. Each convolutional layer uses filters (kernels) that slide over the input to create feature maps.
    - **Activation Function**: Typically, a non-linear function like ReLU (Rectified Linear Unit) is applied after convolution to introduce non-linearity.
    - **Pooling Layers**: Reduce the dimensionality of feature maps while retaining important information. Common pooling methods include max pooling and average pooling.
    - **Fully Connected Layers**: After several convolutional and pooling layers, the final feature maps are flattened and passed through one or more fully connected layers to produce the output.
2. **Convolution Operation**:
    - The core operation of CNNs, where a filter is applied to the input image to produce a feature map. It captures spatial hierarchies in the data.

3. **Pooling**:
    - Pooling layers downsample the feature maps, reducing their spatial dimensions and computational load while preserving essential features. This helps in making the network invariant to small translations in the input.
4. **Training**:
    - CNNs are trained using backpropagation and a loss function (e.g., cross-entropy for classification tasks) to minimize the difference between predicted and actual labels.
5. **Transfer Learning**:
    - CNNs can leverage pre-trained models (e.g., VGG, ResNet) on large datasets and fine-tune them for specific tasks, which is especially useful when data is limited.

Advantages and Disadvantages

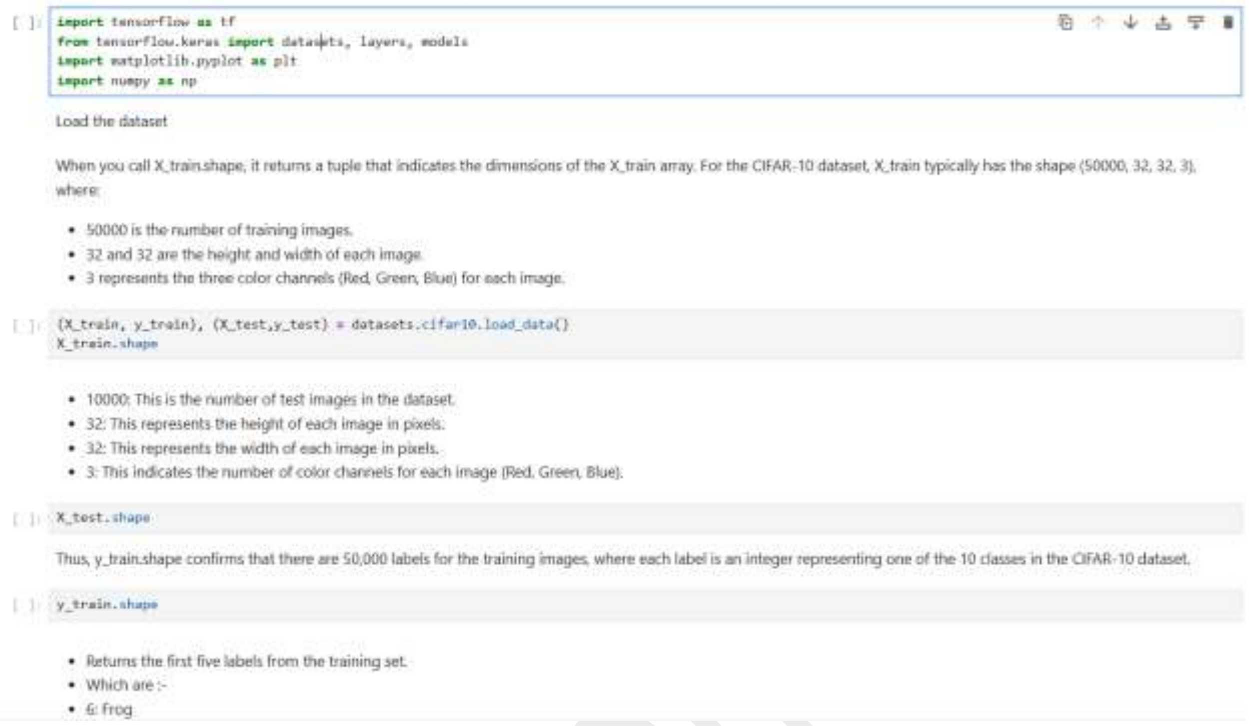
- **Advantages**:
    - Highly effective for image data due to their ability to automatically learn spatial hierarchies.
    - Reduce the need for manual feature extraction, allowing for end-to-end learning.
    - Robust to variations in the input data due to pooling layers.
- **Disadvantages**:
    - Require substantial computational resources (e.g., GPUs) for training.
    - Can be prone to overfitting, especially with limited training data.

    - May require careful tuning of hyperparameters (e.g., learning rate, number of layers).

Applications

- Image and video analysis (e.g., facial recognition, object detection).
- Medical image analysis (e.g., detecting tumors in radiology images).
- Autonomous vehicles (e.g., recognizing road signs and pedestrians).
- Natural language processing (for tasks like text classification and sentiment analysis).

Example



```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
import numpy as np
```

Load the dataset

When you call X_train.shape, it returns a tuple that indicates the dimensions of the X_train array. For the CIFAR-10 dataset, X_train typically has the shape (50000, 32, 32, 3), where:

- 50000 is the number of training images.
- 32 and 32 are the height and width of each image.
- 3 represents the three color channels (Red, Green, Blue) for each image.

```
(X_train, y_train), (X_test,y_test) = datasets.cifar10.load_data()
X_train.shape
```

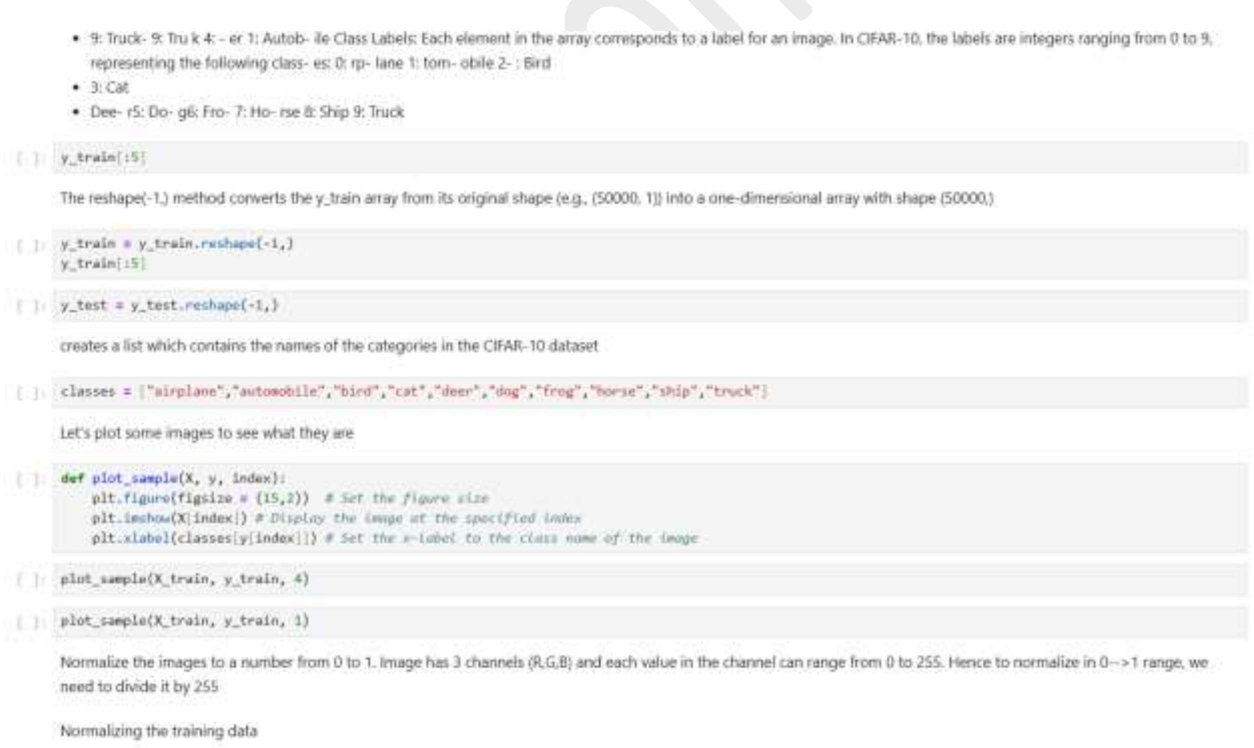
- 10000: This is the number of test images in the dataset.
- 32: This represents the height of each image in pixels.
- 32: This represents the width of each image in pixels.
- 3: This indicates the number of color channels for each image (Red, Green, Blue).

```
X_test.shape
```

Thus, y_train.shape confirms that there are 50,000 labels for the training images, where each label is an integer representing one of the 10 classes in the CIFAR-10 dataset.

```
y_train.shape
```

- Returns the first five labels from the training set.
- Which are :-
- 6: Frog



- 9: Truck- 9: Tru k 4: - er 1: Autob- ile Class Labels: Each element in the array corresponds to a label for an image. In CIFAR-10, the labels are integers ranging from 0 to 9, representing the following class- es: 0: rp- lane 1: tom- obile 2- : Bird
- 3: Cat
- Dee- r5: Do- g6: Fro- 7: Ho- rse 8: Ship 9: Truck

```
y_train[:5]
```

The reshape(-1,) method converts the y_train array from its original shape (e.g., (50000, 1)) into a one-dimensional array with shape (50000,)

```
y_train = y_train.reshape(-1,)
y_train[:5]
```

```
y_test = y_test.reshape(-1,)
```

creates a list which contains the names of the categories in the CIFAR-10 dataset

```
classes = ["airplane","automobile","bird","cat","deer","dog","frog","horse","ship","truck"]
```

Let's plot some images to see what they are

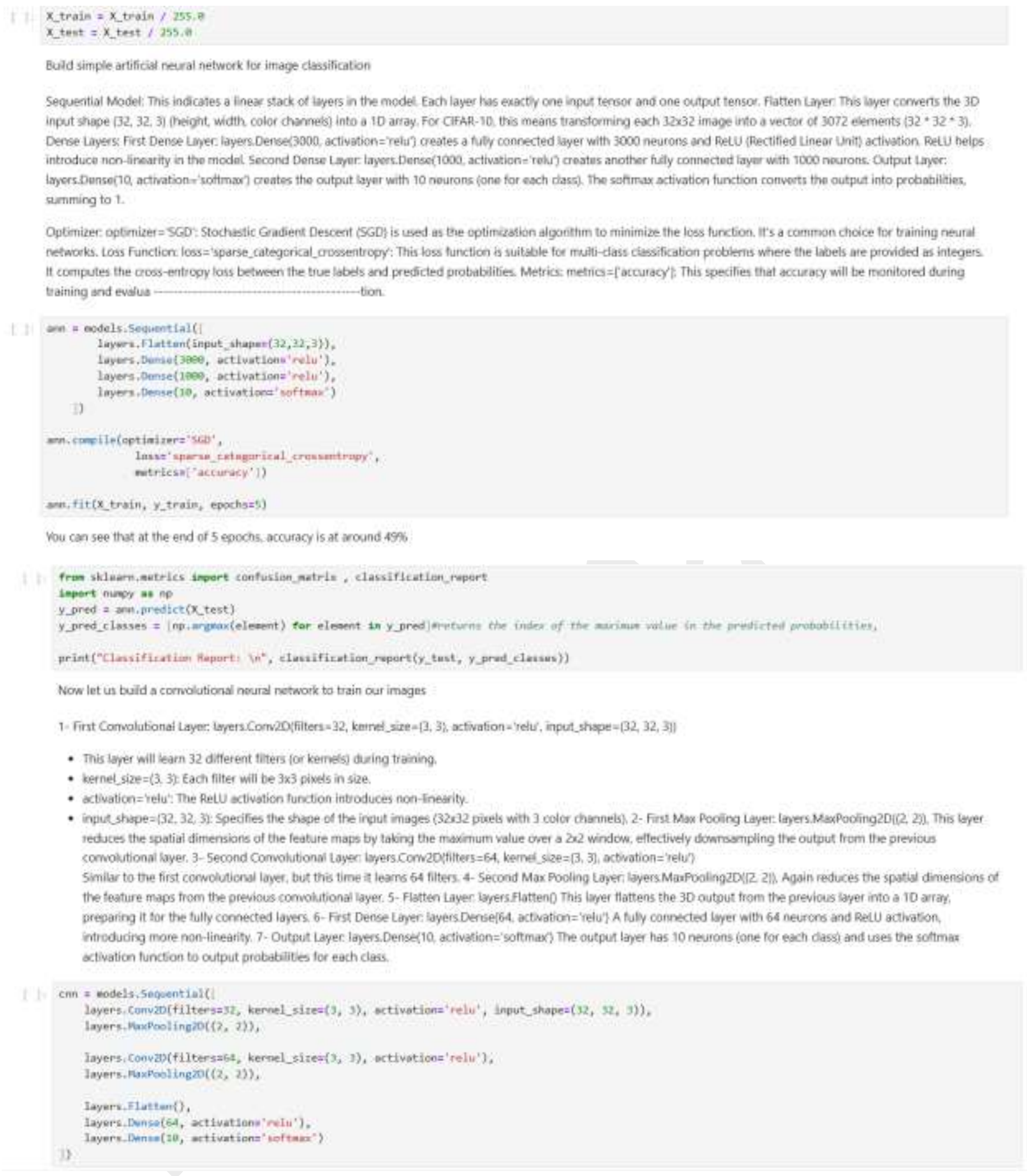
```
def plot_sample(X, y, index):
    plt.figure(figsize = (15,2))  # Set the figure size
    plt.imshow(X[index]) # Display the image at the specified index
    plt.xlabel(classes[y[index]]) # Set the x-label to the class name of the image
```

```
plot_sample(X_train, y_train, 4)
```

```
plot_sample(X_train, y_train, 1)
```

Normalize the images to a number from 0 to 1. Image has 3 channels (R,G,B) and each value in the channel can range from 0 to 255. Hence to normalize in 0-->1 range, we need to divide it by 255

Normalizing the training data



```
X_train = X_train / 255.0
X_test = X_test / 255.0
```

Build simple artificial neural network for image classification

Sequential Model: This indicates a linear stack of layers in the model. Each layer has exactly one input tensor and one output tensor. Flatten Layer: This layer converts the 3D input shape (32, 32, 3) (height, width, color channels) into a 1D array. For CIFAR-10, this means transforming each 32x32 image into a vector of 3072 elements (32 * 32 * 3). Dense Layers: First Dense Layer: layers.Dense(3000, activation='relu') creates a fully connected layer with 3000 neurons and ReLU (Rectified Linear Unit) activation. ReLU helps introduce non-linearity in the model. Second Dense Layer: layers.Dense(1000, activation='relu') creates another fully connected layer with 1000 neurons. Output Layer: layers.Dense(10, activation='softmax') creates the output layer with 10 neurons (one for each class). The softmax activation function converts the output into probabilities, summing to 1.

Optimizer: optimizer='SGD': Stochastic Gradient Descent (SGD) is used as the optimization algorithm to minimize the loss function. It's a common choice for training neural networks. Loss Function: loss='sparse_categorical_crossentropy': This loss function is suitable for multi-class classification problems where the labels are provided as integers. It computes the cross-entropy loss between the true labels and predicted probabilities. Metrics: metrics=['accuracy']: This specifies that accuracy will be monitored during training and evalua --tion.

```
ann = models.Sequential([
        layers.Flatten(input_shape=(32,32,3)),
        layers.Dense(3000, activation='relu'),
        layers.Dense(1000, activation='relu'),
        layers.Dense(10, activation='softmax')
    ])

ann.compile(optimizer='SGD',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

ann.fit(X_train, y_train, epochs=5)
```

You can see that at the end of 5 epochs, accuracy is at around 49%

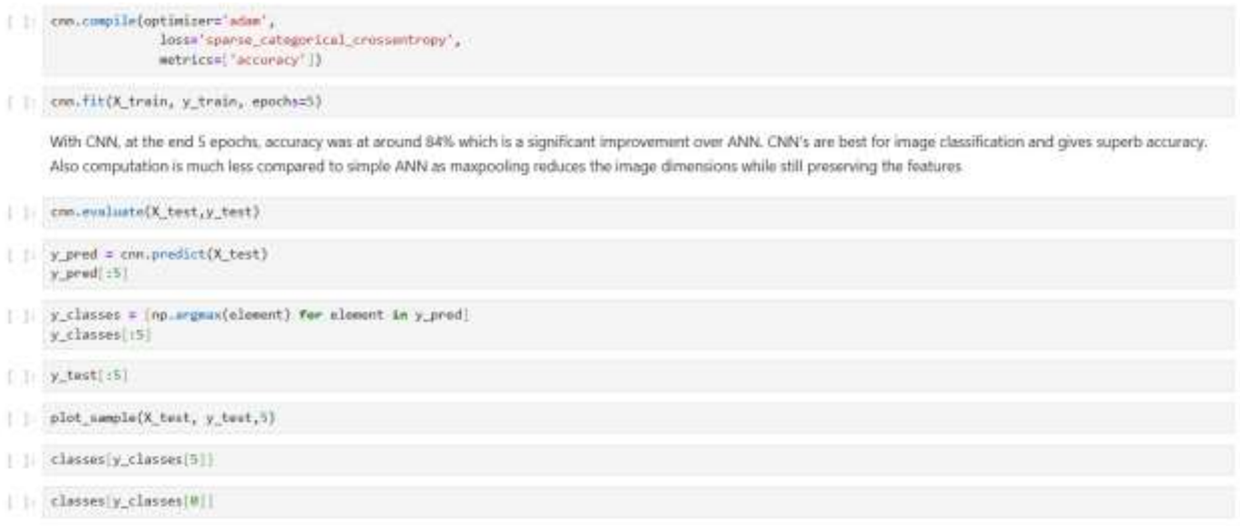
```
from sklearn.metrics import confusion_matrix , classification_report
import numpy as np
y_pred = ann.predict(X_test)
y_pred_classes = [np.argmax(element) for element in y_pred]#returns the index of the maximum value in the predicted probabilities,

print("Classification Report: \n", classification_report(y_test, y_pred_classes))
```

Now let us build a convolutional neural network to train our images

1- First Convolutional Layer: layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu', input_shape=(32, 32, 3))

- This layer will learn 32 different filters (or kernels) during training.
- kernel_size=(3, 3): Each filter will be 3x3 pixels in size.
- activation='relu': The ReLU activation function introduces non-linearity.
- input_shape=(32, 32, 3): Specifies the shape of the input images (32x32 pixels with 3 color channels). 2- First Max Pooling Layer: layers.MaxPooling2D((2, 2)). This layer reduces the spatial dimensions of the feature maps by taking the maximum value over a 2x2 window, effectively downsampling the output from the previous convolutional layer. 3- Second Convolutional Layer: layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu')
  Similar to the first convolutional layer, but this time it learns 64 filters. 4- Second Max Pooling Layer: layers.MaxPooling2D((2, 2)). Again reduces the spatial dimensions of the feature maps from the previous convolutional layer. 5- Flatten Layer: layers.Flatten() This layer flattens the 3D output from the previous layer into a 1D array, preparing it for the fully connected layers. 6- First Dense Layer: layers.Dense(64, activation='relu') A fully connected layer with 64 neurons and ReLU activation, introducing more non-linearity. 7- Output Layer: layers.Dense(10, activation='softmax') The output layer has 10 neurons (one for each class) and uses the softmax activation function to output probabilities for each class.

```
cnn = models.Sequential([
    layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu', input_shape=(32, 32, 3)),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')
])
```



```
[ ]: cnn.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
```

```
[ ]: cnn.fit(X_train, y_train, epochs=5)
```

With CNN, at the end 5 epochs, accuracy was at around 84% which is a significant improvement over ANN. CNN's are best for image classification and gives superb accuracy. Also computation is much less compared to simple ANN as maxpooling reduces the image dimensions while still preserving the features

```
[ ]: cnn.evaluate(X_test,y_test)
```

```
[ ]: y_pred = cnn.predict(X_test)
     y_pred[:5]
```

```
[ ]: y_classes = [np.argmax(element) for element in y_pred]
     y_classes[:5]
```

```
[ ]: y_test[:5]
```

```
[ ]: plot_sample(X_test, y_test,3)
```

```
[ ]: classes[y_classes[5]]
```

```
[ ]: classes[y_classes[0]]
```

Believe in Yourself &Keep Pushing Forward ☺