

**Deccan Education Society's**  
**Navinchandra Mehta Institute of**  
**Technology and Development**

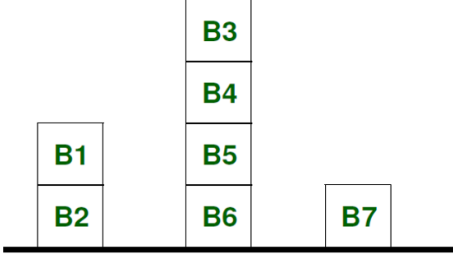
**C E R T I F I C A T E**

This is to certify that **Pooja Yadav** of M.C.A. Semester II with Roll No. **C22136** has completed **All Practical's** of **Artificial Intelligence and Machine Learning Lab** under my supervision in this college during the year 2022 -2023.

CO	R1 (Attendance)	R2 (Performance during lab session)	R3 (Innovation in problem solving technique)	R4 (Mock Viva)	R5 (Variation in implementation of learnt topics on projects)
CO1					
CO2					
CO3					
CO4					

Practical-in-charge

Head of Department  
MCA Department  
(NMITD)

Sr. No	Artificial Intelligence Section	Date	Sign
1.	<b>Logic Programming with Prolog – Representing Family Relationships</b>		
a)	Implement family relationships in Prolog as a Family KB using predicates: child, father, mother, male, female, parent, grandfather using Prolog. Make your own assumptions with respect to the needed atomic and conditional sentences. Demonstrate the program by establishing various types of queries pertaining to family relationships.	13-06-2023	
2.	<b>Problem Solving with Prolog</b>		
a)	<p><b>Blocks World:</b> Describe the “Blocks World” scene shown below to Prolog such that the following can be determined through Prolog queries:</p> <ul style="list-style-type: none"> <li>Block 3 is above Block 5</li> <li>Block 1 is to the left of Block 7</li> <li>Block 4 is to the right of Block 2</li> </ul> 	14-06-2023	
b)	<b>Map Coloring Problem:</b> Illustrate the solving of the popular constraint satisfaction problem known as Map Coloring problem using Prolog.	15-06-2023	
c)	<b>Water-Jug Puzzle:</b> Illustrate the solving of a Water-Jug puzzle using prolog. There are two jugs, one with 4 liter capacity and one with 3 liter capacity. There are no measurement markings on them. You can fill them, empty them or pour water from one jug to another. Initially both are empty. The final state should be such that you get 1 liter of water in one of the jug.	16-06-2023	

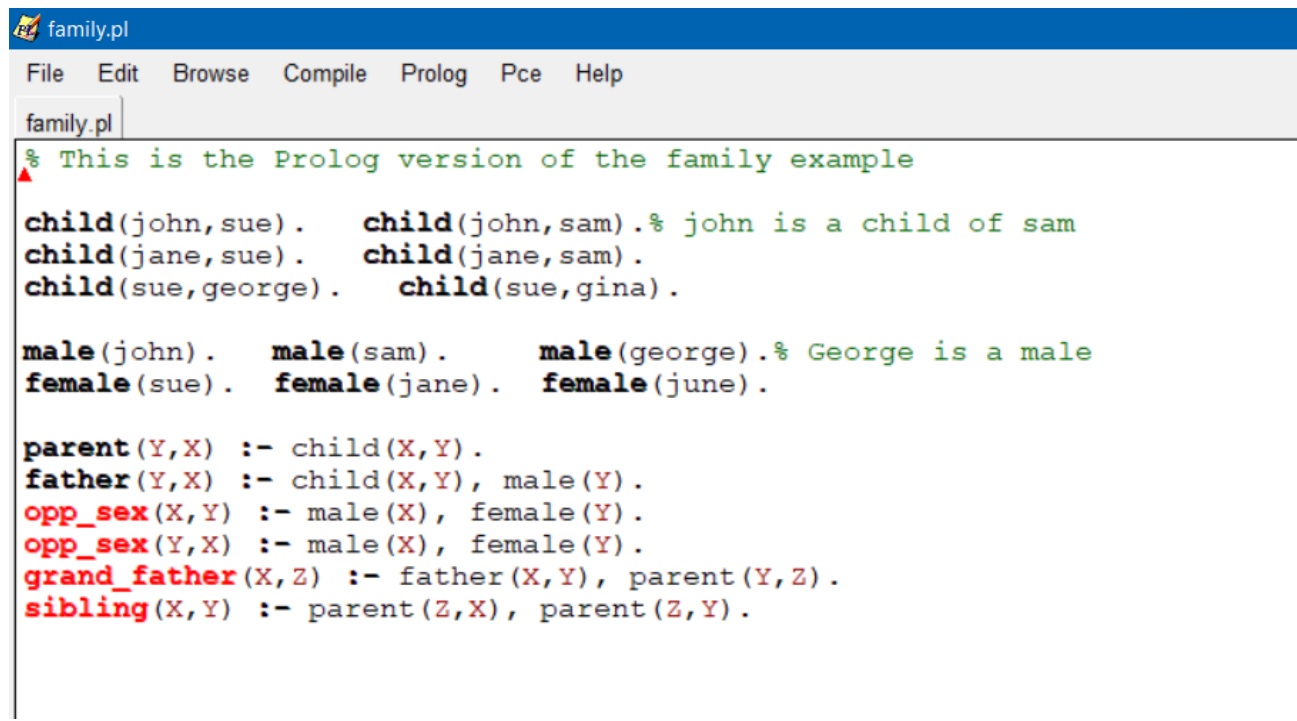
**INDEX – AI & ML Lab**

<b>Sr. No</b>	<b>Machine Learning Section</b>	<b>Date</b>	<b>Sign</b>
<b>1.</b>	<b>Coding in Basic Python and Python Packages for ML</b>	17-05-2023	
<b>2.</b>	<b>Prediction Using a Linear Regression Model</b>	19-05-2023	
<b>3.</b>	<b>Binary Classification using a Logistic Regression Model</b>	25-05-2023	
<b>4.</b>	<b>Multi-Class Classification using k-Nearest Neighbors</b>	06-06-2023	
<b>5.</b>	<b>Linear and Non-Linear SVM Classification</b>	07-06-2023	
<b>6.</b>	<b>Decision Tree Learning for Classification</b>	08-06-2023	
<b>7.</b>	<b>Ensemble Learning with AdaBoost Classifier</b>	09-06-2023	
<b>8.</b>	<b>Clustering with k-Means Algorithm</b>	12-06-2023	

## Artificial Intelligence

**1.Logic Programming with Prolog – Representing Family Relationships a.**  
**Implement family relationships in Prolog as a Family KB using predicates: child, father, mother, male, female, parent, grandfather using Prolog. Make your own assumptions with respect to the needed atomic and conditional sentences. Demonstrate the program by establishing various types of queries pertaining to family relationships.**

### Code:-


A screenshot of a Prolog IDE window titled 'family.pl'. The window has a menu bar with 'File', 'Edit', 'Browse', 'Compile', 'Prolog', 'Pce', and 'Help'. Below the menu bar is a text area containing Prolog code. The code defines a family knowledge base with predicates for child, male, female, parent, father, grand\_father, and sibling. Comments are used to explain some of the facts and rules.

```
family.pl
File Edit Browse Compile Prolog Pce Help
family.pl
% This is the Prolog version of the family example
child(john,sue).    child(john,sam).% john is a child of sam
child(jane,sue).    child(jane,sam).
child(sue,george).  child(sue,gina).

male(john).    male(sam).    male(george).% George is a male
female(sue).   female(jane).  female(june).

parent(Y,X) :- child(X,Y).
father(Y,X) :- child(X,Y), male(Y).
opp_sex(X,Y) :- male(X), female(Y).
opp_sex(Y,X) :- male(X), female(Y).
grand_father(X,Z) :- father(X,Y), parent(Y,Z).
sibling(X,Y) :- parent(Z,X), parent(Z,Y).
```

### Output:-

 SWI-Prolog (Multi-threaded, version 9.0.4)

File Edit Settings Run Debug Help

Welcome to SWI-Prolog (threaded, 32 bits, version 9.0.4)  
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.  
Please run `?- license.` for legal details.

For online help and background, visit <https://www.swi-prolog.org>  
For built-in help, use `?- help(Topic).` or `?- apropos(Word).`

```
?-
% c:/users/hp/downloads/prolog programs/family compiled 0.00 sec, -2 clauses
?-
|   child(john,Who).
Who = sue .

?- child(Who,george).
Who = sue.

?- parent(john,sue).
false.

?- parent(sue,john).
true .

?- grandfather(george,john).
Correct to: "grand_father(george,john)"?
Please answer 'y' or 'n'? yes
true .


?- grand_father(george,john).
true .

?- op_sex(george,jane).
Correct to: "opp_sex(george,jane)"?
Please answer 'y' or 'n'? yes
true .

?- opp_sex(george,john).
false.

?- opp_sex(george,john).
false.


?- sibling(jane,john).
true .
```

 SWI-Prolog (Multi-threaded, version 9.0.4)

File Edit Settings Run Debug Help

```
?- parent(sam,X).
X = john ,

?- parent(sam,X).
X = john ;
X = jane.
```

 SWI-Prolog (Multi-threaded, version 9.0.4)

File Edit Settings Run Debug Help

```
?- parent(X, john), female(X).  
X = sue ,
```

```
?- trace  
|  
true.
```

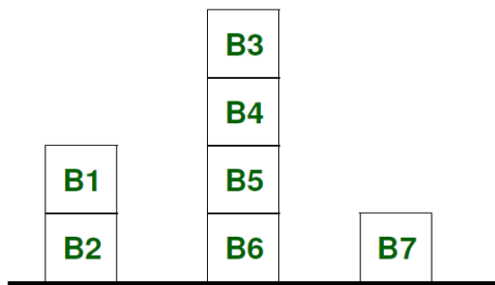
```
[trace] ?- parent(X, john), \+ female(X).  
  Call: (11) parent(_5004, john) ? creep  
  Call: (12) child(john, _5004) ? creep  
  Exit: (12) child(john, sue) ? creep  
  Exit: (11) parent(sue, john) ? creep  
  Call: (11) female(sue) ? creep  
  Exit: (11) female(sue) ? creep  
  Redo: (12) child(john, _5004) ? creep  
  Exit: (12) child(john, sam) ? creep  
  Exit: (11) parent(sam, john) ? creep  
  Call: (11) female(sam) ? creep  
  Fail: (11) female(sam) ? creep  
X = sam.
```

```
[trace] ?-  
|
```

## 2. Problem Solving with Prolog

a. **Blocks World:** Describe the “Blocks World” scene shown below to Prolog such that the following can be determined through Prolog queries:

- a. Block 3 is above Block 5
- b. Block 1 is to the left of Block 7
- c. Block 4 is to the right of Block 2



### Code:-

```
blocks.pl
File Edit Browse Compile Prolog Pce Help
blocks.pl
% on(X,Y) means that block X is directly on top of block Y.
on(b1,b2). on(b3,b4). on(b4,b5). on(b5,b6).

% just left(X,Y) means that blocks X and Y are on the table
% and that X is immediately to the left of Y.
just_left(b2,b6). just_left(b6,b7).

% above(X,Y) means that block X is somewhere above block Y
% in the pile where Y occurs.
above(X,Y) :- on(X,Y).
above(X,Y) :- on(X,Z), above(Z,Y).

% left(X,Y) means that block X is somewhere to the left
% of block Y but perhaps higher or lower than Y.
left(X,Y) :- just_left(X,Y).
left(X,Y) :- just_left(X,Z), left(Z,Y).
left(X,Y) :- on(X,Z), left(Z,Y). % leftmost is on something.
left(X,Y) :- on(Y,Z), left(X,Z). % rightmost is on something.

% right(X,Y) is the opposite of left(X,Y).
right(Y,X) :- left(X,Y).
```

**OUTPUT:-**

```
SWI-Prolog (Multi-threaded, version 9.0.4)
File Edit Settings Run Debug Help
Welcome to SWI-Prolog (threaded, 32 bits, version 9.0.4)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?-
% c:/users/hp/downloads/prolog programs/blocks compiled 0.02 sec, -2 clauses
?-
|   left(b2,Who).
Who = b6
Unknown action: , (h for help)
Action? ,

?- on(Who,b4).
Who = b3.

?- above(b3,Who).
Who = b4 ,

?- right(b6,Who).
Who = b2 ,

?- just_left(b3,Who).
false.

?- just_left(b2,Who).
Who = b6.

?- ls.
true.
```



## b. Map Coloring Problem: Illustrate the solving of the popular constraint satisfaction problem known as Map Coloring problem using Prolog.

### Code:-

```

mapcoloring.pl
File Edit Browse Compile Prolog Pce Help
mapcoloring.pl
% Solution for the map coloring problem as described in Figure 5.1.
% A map is depicted with 5 countries A, B, C, D, and E. The goal is to
% colour the countries on the map using just the colours red, white, and
% blue in such a way that no countries with a border between them have
% the same color.
% solution(A,B,C,D,E) holds if A,B,C,D,E are colours that solve the
% described map colouring problem. This is a particular example of the
% general class called 'constraint satisfaction problems'.

print_colors :- solution(A,B,C,D,E),nl,write('Country A is colored: '), write(A)
               ,nl,write('Country B is colored: '), write(B)
               ,nl,write('Country C is colored: '), write(C)
               ,nl,write('Country D is colored: '), write(D)
               ,nl,write('Country E is colored: '), write(E).

solution(A,B,C,D,E) :- color(A), color(B), color(C), color(D), color(E),
                       \+ A=B, \+ A=C, \+ A=D, \+ A=E, \+ B=C, \+ C=D, \+ D=E.

% The three colours are these:
color(red).
color(white).
color(blue).

```

### Output:-

```

SWI-Prolog (Multi-threaded, version 9.0.4)
File Edit Settings Run Debug Help
Welcome to SWI-Prolog (threaded, 32 bits, version 9.0.4)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?-
% c:/users/hp/downloads/prolog programs/mapcoloring compiled 0.00 sec, -2 clauses
?-
| color(A).
A = red .

?- color(B).
B = red .

?- color(C).
C = red .

?- color(D).
D = red .

?- color(E).
E = red .

?- color(red).
true.

?- color(orange).
false.

?- solution(A,B,C,D,E).
A = red,
B = D, D = white,
C = E, E = blue ,
?-

```

### C. Water-Jug Puzzle: Illustrate the solving of a Water-Jug puzzle using prolog.

There are two jugs, one with 4 litre capacity and one with 3 litre capacity. There are no measurement markings on them. You can fill them, empty them or pour water from one jug to another. Initially both are empty. The final state should be such that you get 1 litre of water in one of the jug.

#### Code:-

```
waterjug.pl
File Edit Browse Compile Prolog Pce Help
waterjug.pl
water_jug(X,Y):-X>4,Y<3, write('4L water jug overflowed.').nl.
water_jug(X,Y):-X<4,Y>3, write('3L water jug is overflowed.').nl.
water_jug(X,Y):-X>4,Y>3,write('Both water jugs overflowed.').nl.

water_jug(X,Y):- (X:=0,Y:=0,nl,write('4L:0 & 3L:3 (Action: Fill 3L jug.)'),YY is 3,water_jug(X,YY));
                (X:=0,Y:=0,nl,write('4L:4 & 3L:0 (Action: Fill 4L jug.)'),XX is 4, water_jug(XX,Y));
                (X:=2,Y:=0,nl,write('4L:2 & 3L:0 (Action: Goal State Reached...)'));
                (X:=4,Y:=0,nl,write('4L:1 & 3L:3 (Action: Pour water from 4L to 3L jug.)'),XX is X-3,YY is 3, water_jug(XX,YY));
                (X:=0,Y:=3,nl,write('4L:3 & 3L:0 (Action: Pour water from 3L to 4L jug.)'),XX is 3,YY is 0, water_jug(XX,YY));
                (X:=1,Y:=3,nl,write('4L:1 & 3L:0 (Action: Empty 3L jug.)'),YY is 0, water_jug(X,YY));
                (X:=3,Y:=0,nl,write('4L:3 & 3L:3 (Action: Fill 3L jug.)'),YY is 3, water_jug(X,YY));
                (X:=3,Y:=3,nl,write('4L:4 & 3L:2 (Action: Pour water from 3L to 4L jug until 4L jug is full.)'),XX is X+1,YY is Y-1,
water_jug(XX,YY));
                (X:=1,Y:=0,nl,write('4L:0 & 3L:1 (Action: Pour water from 4L to 3L jug.)'),XX is Y,YY is X, water_jug(XX,YY));
                (X:=0,Y:=1,nl,write('4L:4 & 3L:1 (Action: Fill 4L jug.)'),XX is 4, water_jug(XX,Y));
                (X:=4,Y:=1,nl,write('4L:2 & 3L:3 (Action: Pour water from 4L to 3L jug untill 3L jug is full.)'),XX is X-2,YY is Y+2,
water_jug(XX,YY));
                (X:=2,Y:=3,nl,write('4L:2 & 3L:0 (Action: Empty 3L jug.)'),YY is 0, water_jug(X,YY));
                (X:=4,Y:=2,nl,write('4L:0 & 3L:2 (Action: Empty 4L jug.)'),XX is 0, water_jug(XX,Y));
                (X:=0,Y:=2,nl,write('4L:2 & 3L:0 (Action: Pour wter from 3L jug to 4L jug.)'),XX is Y, YY is X, water_jug(XX,YY)).
```

#### Output:-

```
SWI-Prolog (Multi-threaded, version 9.0.4)
File Edit Settings Run Debug Help
Welcome to SWI-Prolog (threaded, 32 bits, version 9.0.4)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- cd("D:/C22093/Prolog Programs").
true.

?- ls.
% blocks.pl          family.pl          mapcoloring.pl    waterjug.pl
true.


?- [waterjug].
true.

?- water_jug(1,2).
false.

?- water_jug(4,4).
false.

?- water_jug(1,4).
3L water jug is overflowed.
true.

?- water_jug(4,1).
4L:2 & 3L:3 (Action: Pour water from 4L to 3L jug untill 3L jug is full.)
4L:2 & 3L:0 (Action: Empty 3L jug.)
4L:2 & 3L:0 (Action: Goal State Reached...)
true.
```

 SWI-Prolog (Multi-threaded, version 9.0.4)

File Edit Settings Run Debug Help

```
?- water_jug(3,1).  
false.  
  
?- water_jug(3,2).  
false.  
  
?- water_jug(2,3).  
  
4L:2 & 3L:0 (Action: Empty 3L jug.)  
4L:2 & 3L:0 (Action: Goal State Reached...)  
true .  
  
?- water_jug(3,3).  
  
4L:4 & 3L:2 (Action: Pour water from 3L to 4L jug until 4L jug is full.)  
4L:0 & 3L:2 (Action: Empty 4L jug.)  
4L:2 & 3L:0 (Action: Pour wter from 3L jug to 4L jug.)  
4L:2 & 3L:0 (Action: Goal State Reached...)  
true .  
  
?- water_jug(4,3).  
false.  
  
?- water_jug(4,4).  
false.  
  
?-
```

## Machine Learning

### 1. Coding in Basic Python and Python Packages for ML

#### 1. Introduction to Python

##### 1.1 Basic Datatypes in Python

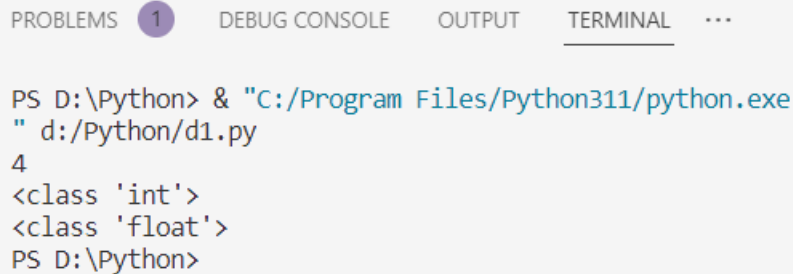
#### Numbers and Arithmetic Operations on Numbers

**Code:-**

```
# Number
x=4
print(x)
print(type(x))

y = 1.2
print(type(y))
```

**Output:-**



```
PROBLEMS 1 DEBUG CONSOLE OUTPUT TERMINAL ...

PS D:\Python> & "C:/Program Files/Python311/python.exe"
" d:/Python/d1.py
4
<class 'int'>
<class 'float'>
PS D:\Python>
```

**Code:-**

```
# Arithmetic operation in python

x=3
print(x)
print(x+1)
print(x-1)
print(x * 3)
print(x ** 2) # Exponentiation
x += 1 # x=x+1
print(x)
x *= 2
print(x)
```

```
y = 2.5
print(type(y))
print(y,y+1,y *4,y**6)

#Division

x = 88
y = 2
print(x/y) #---Division
print(x // y) #---integer result- fraction gets truncated
```

### **Output:-**



```
PROBLEMS 1 DEBUG CONSOLE OUTPUT TERMINAL SQL CONSOLE

PS D:\Python> & "C:/Program Files/Python311/python.exe" d:/Python/d2

3
4
2
9
9
4
8
<class 'float'>
2.5 3.5 10.0 244.140625
44.0
44
PS D:\Python>
```

### **Booleans and logic Operation on Booleans**

#### **Code:-**

```
t = True
f = False
print(type(t))
print(t and f)
print(t or f)
print(not t)
print(t != f)
```

**Output:-**

```
PROBLEMS 1 DEBUG CONSOLE OUTPUT TERMINAL SQL CONSOLE + \n\nPS D:\\Python> & "C:/Program Files/Python311/python.exe" d:/Python/d3.py  
<class 'bool'>  
False  
True  
False  
True  
PS D:\\Python>
```

**String and String Method in Python**

```
a = 'hell0'  
b = 'Diksha'  
print(a)  
print(len(b))  
hd = a + ' ' + b  
print(hd)  
hd1 = '%s %s %d' % (a,b,12)  
print(hd1)
```

**Output:-**

```
PROBLEMS 1 DEBUG CONSOLE OUTPUT TERMINAL SQL CONSOLE  
  
PS D:\\Python> & "C:/Program Files/Python311/python.exe" d:/Python/d4  
  
hell0  
6  
hell0 Diksha  
hell0 Diksha 12  
PS D:\\Python>
```

**Code:-**

```
# String Method  
d="Diksha Is Good Girl!!!!!!!! "   
c="diksha pathak"  
e="      "  
print(len(d))  
print(d.upper())  
print(d.lower())  
print(d.rstrip("!"))
```

```
print(d.replace("Diksha", "Shreya"))
print(d.split(" "))
a="introductionTopython"
print(a.capitalize())
print(len(a))
print(a.center(48))
print(d.endswith("!!"))
print(a.endswith("To",4,15))
print(d.find("iss"))
print(d.index("Is"))
print(a.isalnum())
print(a.isalpha())
print(c.islower())
print(d.isprintable())
print(a.isprintable())
print(e.isspace())
print(d.istitle())
print(d.swapcase())
print(d.title())
```

**Output:-**

PROBLEMS1DEBUG CONSOLEOUTPUTTERMINALSQL CONSOLE

```
Diksha Is Good Girl!!!!!!!
PS D:\Python> & "C:/Program Files/Python311/python.exe" d:/Python/d4
28
DIKSHA IS GOOD GIRL!!!!!!!
diksha is good girl!!!!!!!
Diksha Is Good Girl!!!!!!!
Shreya Is Good Girl!!!!!!!
['Diksha', 'Is', 'Good', 'Girl!!!!!!!', '']
Introductiontopython
20
        introductionTopython

False
False
-1
7
True
True
True
True
True
True
True
True
dIKSHA iS gOOD gIRL!!!!!!!
Diksha Is Good Girl!!!!!!!
PS D:\Python>
```

## **1.2 Container Types in Python:-**

**Lists:** A list in Python is equivalent to an array, but it is resizable and can contain elements of different types

### **Code:-**

```
ab=[2,4,5,6.3]
print(ab,ab[3])
print(ab[-1])
ab[3] = 'Pratik Sir'
print(ab)
ab.append('teaches AIML')
print(ab)
x = ab.pop()
print(x,ab)
```

### **Output:-**



```
PROBLEMS 1 DEBUG CONSOLE OUTPUT TERMINAL SQL CONSOLE
PS D:\Python> & "C:/Program Files/Python311/python.exe" d:/Python/d5

[2, 4, 5, 6.3] 6.3
6.3
[2, 4, 5, 'Pratik Sir']
[2, 4, 5, 'Pratik Sir', 'teaches AIML']
teaches AIML [2, 4, 5, 'Pratik Sir']
PS D:\Python>
```

**List Slicing:** Python provides a concise syntax to access sublists; this is known as slicing.

### **Code:-**

```
num = list(range(5))
print(type(num))
print(num)
print(num[2:4])
print(num[2:])
print(num[:4])
print(num[:])
```



```
print(num[:-1])
num[2:4] = [8,9]
print(num)
```

### Output:-

```
PROBLEMS 1 DEBUG CONSOLE OUTPUT TERMINAL SQL CONSOLE
PS D:\Python> & "C:/Program Files/Python311/python.exe" d:/Python/d6

<class 'list'>
[0, 1, 2, 3, 4]
[2, 3]
[2, 3, 4]
[0, 1, 2, 3]
[0, 1, 2, 3, 4]
[0, 1, 2, 3]
[0, 1, 8, 9, 4]
PS D:\Python>
```

## LOOPS:-

### Code:-

```
names = ["Diksha", "Shreya", "Priyanka", "Shweta"]
for name in names:
    print(name)
    if(name == "Shweta"):
        print("She is a good Girl")
```

### Output:-

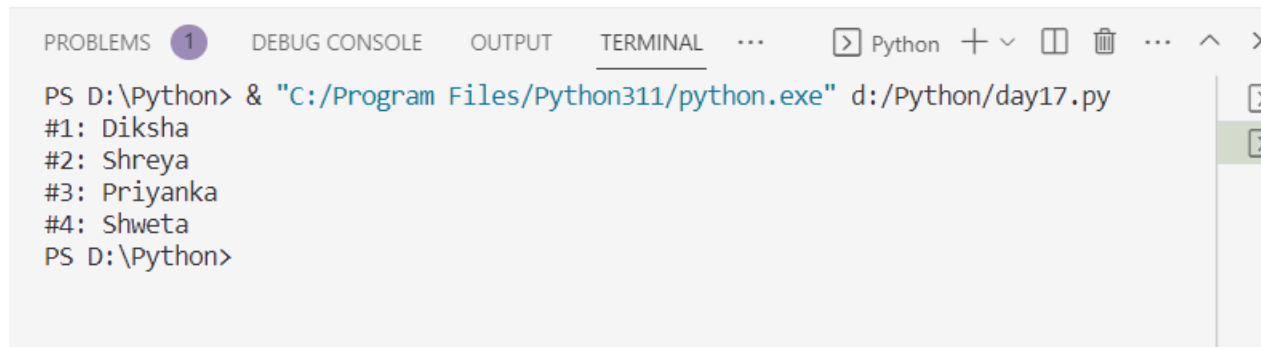
```
PROBLEMS 1 DEBUG CONSOLE OUTPUT TERMINAL ... Python + - [ ] [x] ...
PS D:\Python> & "C:/Program Files/Python311/python.exe" d:/Python/day17.py
Diksha
Shreya
Priyanka
Shweta
She is a good Girl
PS D:\Python>
```

**Enumerate function:** If you want to access the index of each element within the loop body you will have to use the built-in function `enumerate()`.

### Code:-

```
names = ["Diksha", "Shreya", "Priyanka", "Shweta"]
for idx, name in enumerate(names):
    print('#%d: %s' % (idx + 1, name))
```

### **Output:-**



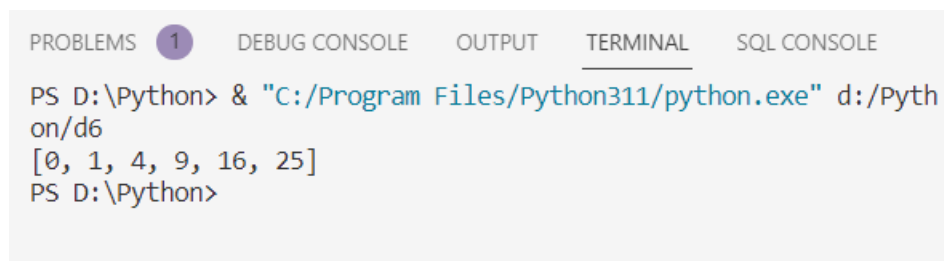
```
PROBLEMS 1 DEBUG CONSOLE OUTPUT TERMINAL ... Python + - [ ] [ ] ... ^ >
PS D:\Python> & "C:/Program Files/Python311/python.exe" d:/Python/day17.py
#1: Diksha
#2: Shreya
#3: Priyanka
#4: Shweta
PS D:\Python>
```

**List Comprehensions:** List comprehension provides a very concise syntax to construct a new list from scratch given an existing list of elements.

### **Code:-**

```
num = [0,1,2,3,4,5]
squares = []
for x in num:
    squares.append(x ** 2)
print(squares)
```

### **Output:-**



```
PROBLEMS 1 DEBUG CONSOLE OUTPUT TERMINAL SQL CONSOLE
PS D:\Python> & "C:/Program Files/Python311/python.exe" d:/Python/d6
[0, 1, 4, 9, 16, 25]
PS D:\Python>
```

### **Code:-**

```
num = [0,1,2,3,4,5]
squares = [X ** 2 for X in num]
print(squares)
```

**Output:-**

```
PROBLEMS 1 DEBUG CONSOLE OUTPUT TERMINAL SQL CONSOLE
PS D:\Python> & "C:/Program Files/Python311/python.exe" d:/Python/d6
[0, 1, 4, 9, 16, 25]
PS D:\Python>
```

**Code:-**

```
num = [0,1,2,3,4,5]
even_squares = [X ** 2 for X in num if X % 2 == 0]
print(even_squares)
```

**Output:-**

```
PROBLEMS 1 DEBUG CONSOLE OUTPUT TERMINAL SQL CONSOLE
PS D:\Python> & "C:/Program Files/Python311/python.exe" d:/Python/
d6
[0, 4, 16]
PS D:\Python>
```

**Dictionaries:** This is another Container type in Python. A dictionary stores (key, value) pairs, similar to a Map in Java or an object in Javascript.

**Code:-**

```
names = {'Diksha','Shreya','Priyanka'}
print(names)
print('Diksha' in names)
print('Shweta' in names)
names.add('Shweta')
print(names)
print(len(names))
names.remove('Diksha')
print(len(names))
```

**Output:-**

```
PROBLEMS 1 DEBUG CONSOLE OUTPUT TERMINAL SQL CONSOLE
PS D:\Python> & "C:/Program Files/Python311/python.exe" d:/Python/d7.py
{'Priyanka', 'Shreya', 'Diksha'}
True
False
{'Priyanka', 'Shreya', 'Diksha', 'Shweta'}
4
3
PS D:\Python>
```

## Looping over Sets

### Code:-

```
names = {'Diksha','Shreya','Priyanka'}
for idx, names in enumerate(names):
    print('#%d: %s' % (idx + 1, names))
```

### Output:-

```
PROBLEMS 1 DEBUG CONSOLE OUTPUT TERMINAL SQL CONSOLE
PS D:\Python> & "C:/Program Files/Python311/python.exe" d:/Python/d6
#1: Diksha
#2: Shreya
#3: Priyanka
PS D:\Python>
```

**Set Comprehensions:** Like lists and dictionaries, we can easily construct sets using set comprehensions.

```
from math import sqrt
num = {int(sqrt(x)) for x in range(30)}
print(num)
```

**output:-**

```
PROBLEMS 1 DEBUG CONSOLE OUTPUT TERMINAL SQL CONSOLE
PS D:\Python> & "C:/Program Files/Python311/python.exe" d:/Python/d6
{0, 1, 2, 3, 4, 5}
PS D:\Python>
```

**Tuples:** A tuple is an (immutable) ordered list of values. A tuple is in many ways similar to a list; One of the most important differences is that tuples can be used as keys in dictionaries and as elements of sets, while lists cannot.

```
d = {(x, x+1): x for x in range(10)}
t =(4,5)
print(d)
print(type(t))
print(d[t])
print(d[3,2])
```

**Output:-**

```
PS D:\Python> & "C:/Program Files/Python311/python.exe" d:/Python/d6
{(0, 1): 0, (1, 2): 1, (2, 3): 2, (3, 4): 3, (4, 5): 4, (5, 6): 5, (6, 7): 6, (7, 8): 7, (8, 9): 8, (9, 10): 9}
<class 'tuple'>
4
2
PS D:\Python>
```

### 1.3 Functions in Python

Python functions are defined using def keyword.

Code:-

```
def isGreater(a,b):
    if(a>b):
        print("first num is greater")
    else:
        print("second num is greater or equal")
a=int(input("enter num1 "))
```

```
b=int(input("enter num2 "))  
isGreater(a,b)
```

Output:-

```
PROBLEMS 1 DEBUG CONSOLE OUTPUT TERMINAL SQL CONSOLE  
PS D:\Python> & "C:/Program Files/Python311/python.exe" d:/Python/funday20  
enter num1 20  
enter num2 50  
second num is greater or equal  
PS D:\Python> █
```

Code:-

```
def hello(name, loud=False):  
    if loud:  
        print('Hello, %s!' % name.upper())  
    else:  
        print('Hello, %s' % name )  
hello('Diksha')  
hello('Shreya', loud=True)
```

Output:-

```
PROBLEMS 1 DEBUG CONSOLE OUTPUT TERMINAL SQL CONSOLE  
PS D:\Python> & "C:/Program Files/Python311/python.exe" d:/Python/d8.py  
Hello, Diksha  
Hello, SHREYA!  
PS D:\Python> █
```

## 1.4 Classes in Python

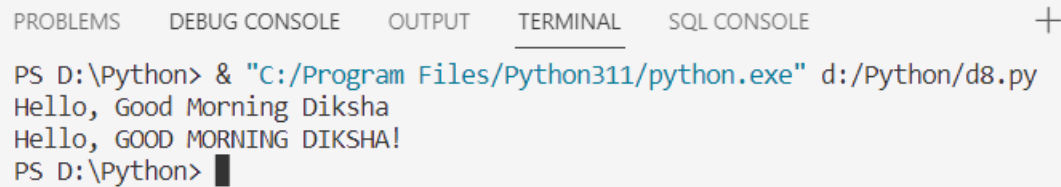
The syntax for defining classes in Python is straightforward:

Code:-

```
class Greeter(object):  
    def __init__(self,name):  
        self.name = name  
    def greet(self,loud=False):
```

```
if loud:
    print('Hello, %s!' % self.name.upper())
else:
    print('Hello, %s' % self.name )
g = Greeter('Good Morning Diksha')
g.greet()
g.greet(loud=True)
```

Output:-



```
PROBLEMS  DEBUG CONSOLE  OUTPUT  TERMINAL  SQL CONSOLE  +
PS D:\Python> & "C:/Program Files/Python311/python.exe" d:/Python/d8.py
Hello, Good Morning Diksha
Hello, GOOD MORNING DIKSHA!
PS D:\Python> █
```

## 2. Numpy

Numpy is the core library for scientific computing in Python. It provides a high-performance multi-dimensional array object, and tools for working with these arrays.

### 2.1 Arrays

A numpy array is a grid of values, all of the same type, and is indexed by a tuple of non-negative integers.

**rank:** The number of dimensions is the rank of the array.

**shape:** The shape of an array is a tuple of integers giving the size of the array along each dimension.

```
import pandas as pd
import numpy as np
print(pd.__version__)
print(np.__version__)
```

OP:-

PROBLEMS    DEBUG CONSOLE    OUTPUT    TERMINAL    SQL CONSOLE

```
PS D:\Python> & "C:/Program Files/Python311/python.exe" d:/Python/ai.py
1.5.3
1.24.2
PS D:\Python>
```

```
import numpy as np

a = np.array([1,2,3]) # create a rank 1 array
print(a)
print(type(a))
print(a.shape)
print(a.ndim) # rank or number of dimensions
print(a[0], a[1], a[2])
a[0] = 5
print(a)
```

## Output

```
PS D:\Python> & "C:/Program Files/Python311/python.exe" d:/Python/d1.py
[1 2 3]
<class 'numpy.ndarray'>
(3,)
1
1 2 3
[5 2 3]
PS D:\Python>
```

```
import numpy as np

b = np.array([[1,2,3],[4,5,6]]) # create an array of rank 2
print(b)
print(b.ndim) # rank 2 array
print(b.shape)
print(b[0,0],b[0,1],b[1,0])
```

## Output:-



```
PS D:\Python> & "C:/Program Files/Python311/python.exe" d:/Python/d1.py
[[1 2 3]
 [4 5 6]]
2
(2, 3)
1 2 4
PS D:\Python>
```

### CODE:-

*# differentiating between rank 1 and rank 2 numpy arrays*

```
import numpy as np
```

```
a = np.array([1,2,3,4,5,6]) # rank 1 array
```

```
b = np.array([[1,2,3,4,5,6]]) # rank 2 array
```

```
print('a.shape = ', a.shape, ' a.ndim = ', a.ndim)
```

```
print('b.shape = ', b.shape, ' b.ndim = ', b.ndim)
```

### Output:-

```
PS D:\Python> & "C:/Program Files/Python311/python.exe" d:/Python/d1.py
a.shape = (6,) a.ndim = 1
b.shape = (1, 6) b.ndim = 2
PS D:\Python>
```

NumPy also provides many functions to create arrays

```
import numpy as np
```

```
a = np.zeros((2,3)) # create an array of all zeros
```

```
print(a)
```

```
b = np.ones((1,2)) # create an array of all ones
```

```
print(b)
```

```
print(b.shape)
```

```
c = np.full((2,2), 7) # create an array of constants
```

```
print(c)
```

```
d = np.eye(3) # create a 3 x 3 identity matrix
```

```
print(d)
```

```
e = np.random.random((2,2)) # create an array filled with random values
print(e)
```

Output:

```
[[0. 0. 0.]
 [0. 0. 0.]]
[[1. 1.]]
(1, 2)
[[7 7]
 [7 7]]
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
[[0.97018897 0.26552665]
 [0.33742753 0.8838239 ]]
```

## 2.2 Array Indexing

Numpy offers several ways to index into arrays.

**Slicing:** Similar to Python lists, numpy arrays can be sliced. Since arrays may be multi-dimensional, you must specify a slice for each dimension of the array:

Code:-

```
import numpy as np

# Use slicing to pull out the subarray consisting of the first 2 rows
# and columns 1 and 2. So b will be a 2 x 2 array consisting of following elements:
#[[2,3],
# [6,7]]
a = np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]]) # create an array of rank 2 and shape (3,4)
b = a[:2, 1:3] # shape of b = (2,2); [[2,3], [6,7]]

print(a)
print(a.shape)
print(b)
print(b.shape)

print(a[0,1]) # prints 2
b[0,0] = 45
print(a[0,1])

print('a:', a)
```

```
print('b:',b)
```

output:-

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
(3, 4)
[[2 3]
 [6 7]]
(2, 2)
a: [[ 1  0  0  4]
     [ 5  0  0  8]
     [ 9 10 11 12]]
b: [[0 0]
     [0 0]]
```

When mixing integer indexing with slice indexing, the rank of the resulting array reduces.

Code:-

```
import numpy as np
a = np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]])

row_r1 = a[1, :] # Rank 1 view of the second row of a
print(row_r1, row_r1.shape)

row_r2 = a[1:2, :] # Rank 2 view of the second row of a
print(row_r2, row_r2.shape)

# We can make the same distinctions when accessing columns of an array
col_r1 = a[:, 1]
col_r2 = a[:, 1:2]

print(col_r1, col_r1.shape)
print(col_r2, col_r2.shape)
```

Output:-

```
[5 6 7 8] (4,)  
[[5 6 7 8]] (1, 4)  
[ 2  6 10] (3,)  
[[ 2]  
 [ 6]  
 [10]] (3, 1)
```

**Integer array indexing:** Allows you to construct arbitrary arrays using data from another array.

Code:-

```
import numpy as np  
a = np.array([[1,2], [3,4], [5,6]]) # a rank 2 array with shape (3,2)  
  
print(a[[0,1,2], [0,1,0]]) # Prints 1, 4 and 5  
print(a[0,0], a[1,1], a[2,0])  
  
print(a[[0,0],[1,1]])
```

Output:-

```
[1 4 5]  
1 4 5  
[2 2]
```

One useful trick with integer array indexing is selecting or mutating one element from each row of a matrix:

```
import numpy as np  
a = np.array([[1,2,3], [4,5,6], [7,8,9], [10,11,12]]) # a rank 2 array of shape (4,3)  
print(a)  
  
# Create an array of indices  
b = np.array([0, 2, 0, 1])  
  
print(a[np.arange(4),b]) # prints [1,6,7,11]  
  
a[np.arange(4),b] += 10  
print(a)
```

output:-

```
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]
[ 1  6  7 11]
[[11  2  3]
 [ 4  5 16]
 [17  8  9]
 [10 21 12]]
```

**Boolean array indexing:** It lets you pick out arbitrary elements of an array. It is used to select the elements of an array that satisfy some condition.

```
import numpy as np

a = np.array([[1,2], [3,4], [5,6]]) # a rank 2 array of shape (3,2)
bool_idx = (a > 2) # Find the elements of a that are bigger than 2;
                  # this returns a numpy array of the same shape as a,
                  # where each slot of bool_idx tells whether that element
                  # of a is > 2

print(bool_idx)
print(a[bool_idx])
print(a[a>2])
```

output:-

```
[[False False]
 [ True  True]
 [ True  True]]
[3 4 5 6]
[3 4 5 6]
```

## 2.3 Datatypes

Every numpy array is a grid of elements of the same type. Numpy provides a large set of numeric datatypes that we can use to construct arrays.

Numpy tries to guess a datatype when you create an array. However, an optional argument can specify the type you want.

```
import numpy as np
```

```
x = np.array([1, 2]) # Let numpy choose the datatype
print(x.dtype)

x = np.array([1.0, 2.0]) # Let numpy choose the datatype
print(x.dtype)

x = np.array([1.0, 2.0], dtype=np.int64) # Force a particular datatype
print(x.dtype)
```

Output:-

```
int64
float64
int64
```

## 2.4 Array Maths

Basic mathematical functions operate element wise on arrays.

```
import numpy as np

x = np.array([[1,2], [3,4]], dtype=np.float64)
y = np.array([[5,6], [7,8]], dtype=np.float64)

# Elementwise addition
print(x + y)
print(np.add(x,y))

# Elementwise subtraction
print(x - y)
print(np.subtract(x,y))

# Elementwise multiplication
print(x * y)
print(np.multiply(x,y))

# Elementwise division
print(x / y)
print(np.divide(x,y))

# Elementwise square root
print(np.sqrt(x))
```

Output:-

```
[[ 6.  8.]
 [10. 12.]]
[[ 6.  8.]
 [10. 12.]]
[[-4. -4.]
 [-4. -4.]]
[[-4. -4.]
 [-4. -4.]]
[[ 5. 12.]
 [21. 32.]]
[[ 5. 12.]
 [21. 32.]]
[[0.2      0.33333333]
 [0.42857143 0.5      ]]
[[0.2      0.33333333]
 [0.42857143 0.5      ]]
[[1.      1.41421356]
 [1.73205081 2.      ]]
```

To do matrix or vector multiplication, we need to use **dot** function. It is useful to compute the inner dot product of two vectors, to multiply a vector by a matrix, and to multiply two matrices.

**dot** is available both as a function in the numpy module as well as an instance method in the array object:

```
import numpy as np

# matrices
x = np.array([[1,2], [3,4]])
y = np.array([[5,6], [7,8]])

# vectors
v = np.array([9,10])
w = np.array([11,12])

# Inner product of two vectors
print(v.dot(w)) # method 1
print(np.dot(v,w)) # method 2

# Matrix/vector multiplication
```

```
print(x.dot(v)) # shape ??  
print(np.dot(x,v))  
  
# Matrix/Matric multiplication  
print(x.dot(y))  
print(np.dot(x,y))
```

### Output:

```
219  
219  
[29 67]  
[29 67]  
[[19 22]  
 [43 50]]  
[[19 22]  
 [43 50]]
```

some other mathematical functions in numpy

```
import numpy as np  
  
x = np.array([[1,2], [3,4]])  
print(np.sum(x))  
print(np.sum(x, axis=0)) # add elements in each column  
print(np.sum(x, axis=1)) # add elements in each row
```

### Output:-

```
10  
[4 6]  
[3 7]
```

Reshaping arrays - one example is transpose operation

```
import numpy as np  
x = np.array([[1,2],[3,4]])  
print(x)  
print(x.T)  
  
v = np.array([1,2,3])  
print(v)
```



```
print(v.T)
```

output:-

```
[[1 2]
 [3 4]]
[[1 3]
 [2 4]]
[1 2 3]
[1 2 3]
```

## 2.5 Broadcasting

Broadcasting is a mechanism that allows numpy to work with arrays of different shapes when performing mathematical operations.

Typically, the smaller tensor will be **broadcasted** to match the shape of the larger tensor.

Broadcasting consists of two steps:

1. Axes (called **broadcast axes**) are added to the smaller tensor to match the **ndim** of the larger tensor.
2. The smaller tensor is repeated alongside these new axes to match the full shape of the larger tensor

Code:-

```
import numpy as np

x = np.array([[1,2,3], [4,5,6], [7,8,9], [10,11,12]])
v = np.array([1,0,1]) # vv = [[1,0,1],[1,0,1],[1,0,1],[1,0,1]]
y = np.empty_like(x)

for i in range(4):
    y[i,:] = x[i,:] + v

print(y)
print(x + v) # implicitly v will be broadcasted to become a matrix of shape (4,3)

z = x + 5 # scalar 5 is broadcasted
print(z)
```

output:-

```
[[ 2  2  4]
 [ 5  5  7]
 [ 8  8 10]
 [11 11 13]]
[[ 2  2  4]
 [ 5  5  7]
 [ 8  8 10]
 [11 11 13]]
[[ 6  7  8]
 [ 9 10 11]
 [12 13 14]
 [15 16 17]]
```

### 3 Scipy

Numpy provides a high-performance multidimensional array and basic tools to compute and manipulate these arrays.

SciPy builds on this, and provides a large number of functions that operate on numpy arrays and are useful for different types of scientific and engineering applications.

## 3.1 Image Operations

SciPy provides some basic functions to work with images. For example, it has functions to read images into numpy arrays. To write numpy arrays as images onto your disk and resize images.

## 3.2 Distance between Points

The function **scipy.spatial.distance.pdist** computes the distance between all pairs of points given a set.

```
import numpy as np
from scipy.spatial.distance import pdist, squareform

x = np.array([[0,1], [1,0], [2,0]])
print(x)

d = squareform(pdist(x, 'euclidean'))
print(d)
```

output:-

```
[[0 1]
 [1 0]
 [2 0]]
[[0.          1.41421356  2.23606798]
 [1.41421356  0.          1.          ]
 [2.23606798  1.          0.          ]]
```

## 4 Pandas

pandas is a Python library for data analysis. It is built around a data structure called as the DataFrame.

Code:-

```
import pandas as pd

data = {"Name": ["John", "Anna", "Aamir", "Madhuri"],
        "Location": ["New York", "Paris", "Mumbai", "Pune"],
        "Age": [52, 25, 55, 45]}

data_pandas = pd.DataFrame(data)
display(data_pandas)
display(data_pandas[data_pandas.Age > 30])
```

Output:-

	Name	Location	Age
0	John	New York	52
1	Anna	Paris	25
2	Aamir	Mumbai	55
3	Madhuri	Pune	45

	Name	Location	Age
0	John	New York	52
2	Aamir	Mumbai	55
3	Madhuri	Pune	45

## 4. Matplotlib

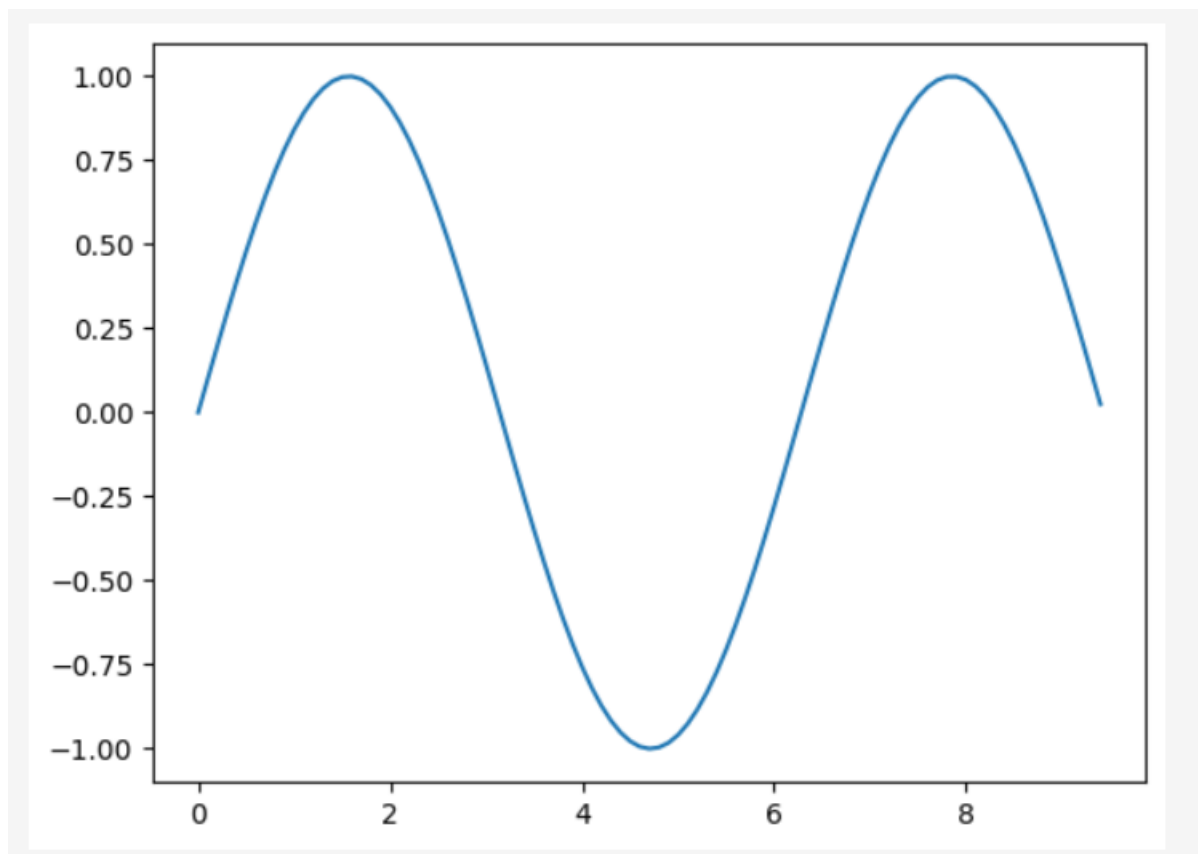
Matplotlib is a plotting library. It provides functions to visualize data and mathematical functions.

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

# compute x and y coordinates for points on a sine curve
x = np.arange(0, 3 * np.pi, 0.1)
y = np.sin(x)

# Plot the points using matplotlib
plt.plot(x, y)
```

Output:-



Adding multiple plots together with title, legend etc

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

```
# compute x and y coordinates for points on a sine curve
```

```
x = np.arange(0, 3 * np.pi, 0.1)
```

```
y_sin = np.sin(x)
```

```
y_cos = np.cos(x)
```

```
plt.plot(x, y_sin)
```

```
plt.plot(x, y_cos)
```

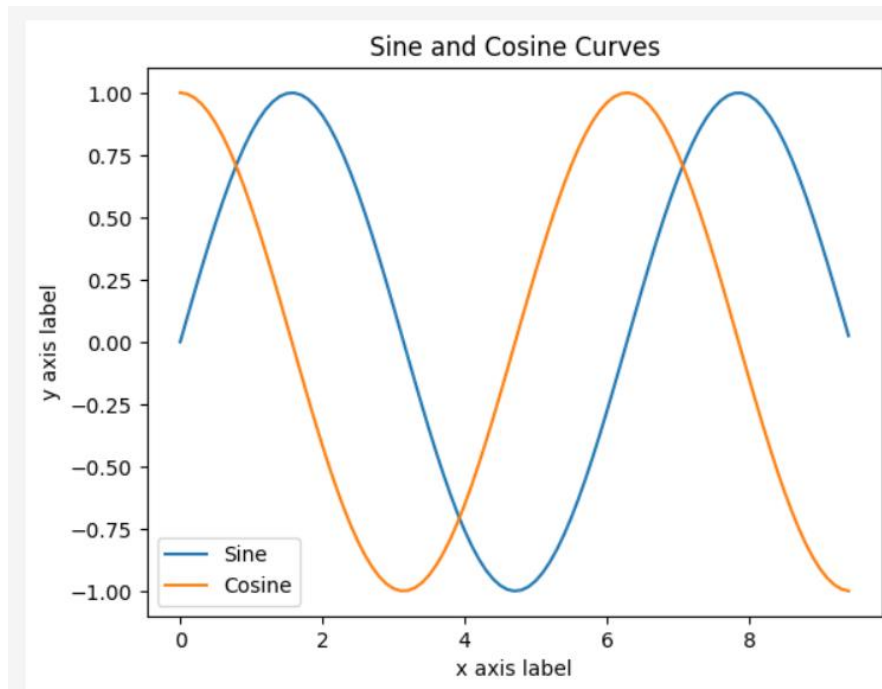
```
plt.xlabel('x axis label')
```

```
plt.ylabel('y axis label')
```

```
plt.title('Sine and Cosine Curves')
```

```
plt.legend(['Sine', 'Cosine'])
```

## Output:-



## Subplots

We can plot different things in the same figure using the **subplot** function

```
import numpy as np
```

```
import matplotlib.pyplot as plt
```

```
# compute x and y coordinates for points on a sine curve
```

```
x = np.arange(0, 3 * np.pi, 0.1)
```

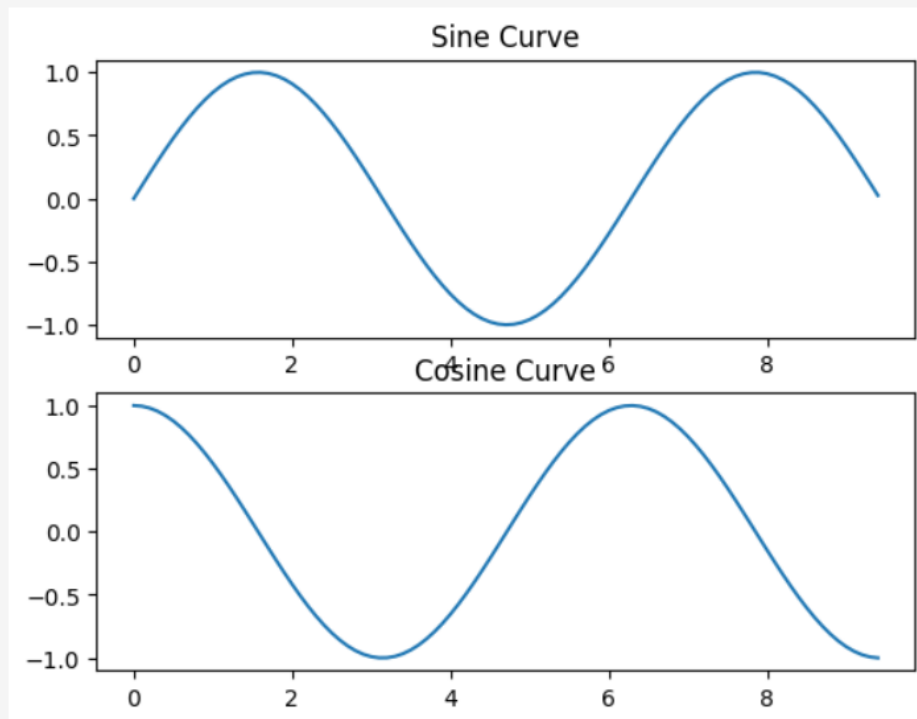
```
y_sin = np.sin(x)
y_cos = np.cos(x)

# first plot
plt.subplot(2,1,1) # setting up a subplot grid of height 2 and width 1, and first plot is active
plt.plot(x,y_sin)
plt.title('Sine Curve')

plt.subplot(2,1,2)
plt.plot(x, y_cos)
plt.title('Cosine Curve')
```

### Output:-

Text(0.5, 1.0, 'Cosine Curve')



## 2. Prediction Using a Linear Regression Model

**Problem Statement:** Use Scikit Learn to implement a Linear Regression Model that predicts average per capita life satisfaction index for a Country/Region given its GDP per capita value. Make use of OECD Better Life Index data along with IMF's GDP per capita data to train the Linear Regression Model.

Finally predict life satisfaction value for a region/country whose OECD BLI value is not in the training data on the basis of its GDP per capita.

*## Code to understand how 'pivot' function of Pandas work*

```
import pandas as pd
import numpy as np
df = pd.DataFrame({'foo':['one', 'one', 'one','two','two','two'],
                   'bar': ['A', 'B', 'C', 'A','B','C'],
                   'baz':[1,2,3,4,5,6],
                   'zoo':['x','y','z','q','w','t']})
print('Original Dataframe')
print(df)
print('\n')
print('Reshaped Dataframe after pivot')
print(df.pivot(index='foo', columns='bar', values='baz'))
print('\n')
```

Output:-

```
Original Dataframe
   foo bar  baz zoo
0  one  A    1   x
1  one  B    2   y
2  one  C    3   z
3  two  A    4   q
4  two  B    5   w
5  two  C    6   t
```

```
Reshaped Dataframe after pivot
bar  A  B  C
foo
one   1  2  3
two   4  5  6
```

The below function merges the OECD's life satisfaction data and the IMF's GDP per capita data

Code:-

```
def prepare_country_stats(oecd_bli, gdp_per_capita):
    oecd_bli = oecd_bli[oecd_bli["INEQUALITY"] == "TOT"]
    oecd_bli = oecd_bli.pivot(index="Country", columns="Indicator", values="Value")
    gdp_per_capita.rename(columns={"2015": "GDP per capita"}, inplace=True)
    gdp_per_capita.set_index("Country", inplace=True)
    full_country_stats = pd.merge(left=oecd_bli, right=gdp_per_capita, left_index=True, right_index=True)
    full_country_stats.sort_values(by="GDP per capita", inplace=True)
    remove_indices = [0,1,6,8,33,34,35]
    keep_indices = list(set(range(36)) - set(remove_indices))
    return full_country_stats[["GDP per capita", "Life satisfaction"]].iloc[keep_indices]
```

```
import os
datapath = os.path.join("datasets", "lifesat", "")
```

```
# To plot pretty figures directly within Jupyter
import matplotlib as mpl
mpl.rc('axes', labelsz=14)
mpl.rc('xtick', labelsz=12)
mpl.rc('ytick', labelsz=12)
```

## Step 1:

Download the OECD life satisfaction csv and GDP per capita csv from the respective web sources.

```
# Download the data
import urllib.request
DOWNLOAD_ROOT = "https://raw.githubusercontent.com/ageron/handson-ml2/master/"
os.makedirs(datapath, exist_ok=True)
for filename in ("oecd_bli_2015.csv", "gdp_per_capita.csv"):
    print("Downloading", filename)
    url = DOWNLOAD_ROOT + "datasets/lifesat/" + filename
    urllib.request.urlretrieve(url, datapath + filename)
```

```
Downloading oecd_bli_2015.csv
Downloading gdp_per_capita.csv
```

**Step 2:** Load the csv files for OECD BLI Index data and IMF's GDP per capita data into respective pandas data frames.



Merge it into a single dataframe after pivoting the data having Country column as index.

Visualize the correlation between GDP per capita with Life Satisfaction index using scatter plot.

Finally build, train and use the Scikit Learn's Linear Regression Model using the data.

```
# Code Example
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import sklearn.linear_model

# Load the data
oecd_bli = pd.read_csv(datapath + "oecd_bli_2015.csv", thousands=',')
gdp_per_capita = pd.read_csv(datapath + "gdp_per_capita.csv", thousands=',', delimiter='\t',
                             encoding='latin1', na_values="n/a")

# Prepare the data
country_stats = prepare_country_stats(oecd_bli, gdp_per_capita)
X = np.c_[country_stats["GDP per capita"]]
y = np.c_[country_stats["Life satisfaction"]]

print(country_stats.head())

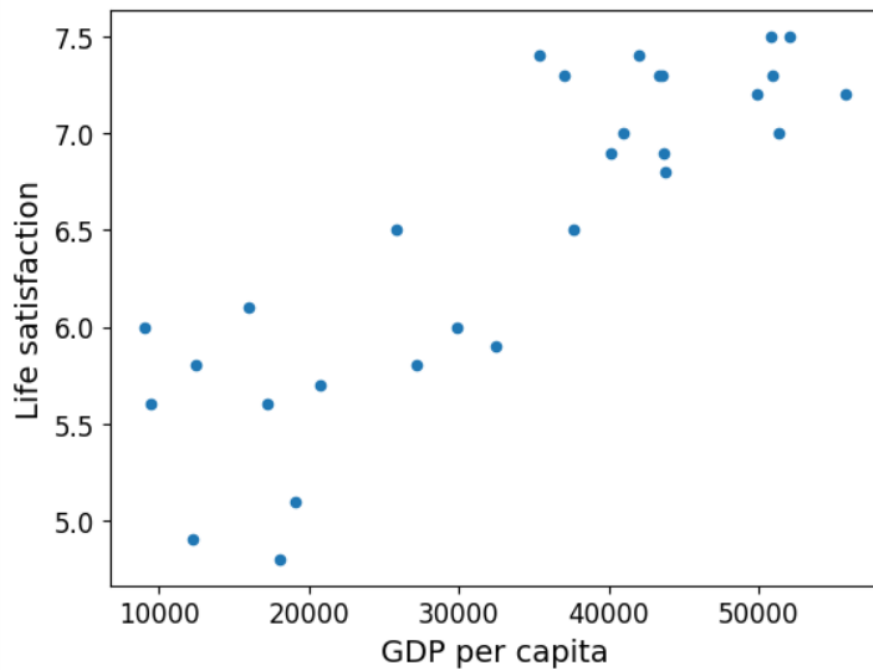
# Visualize the data
country_stats.plot(kind='scatter', x="GDP per capita", y="Life satisfaction")

# Select a linear model
model = sklearn.linear_model.LinearRegression()

# Train the model
model.fit(X, y)

# Make a prediction for Cyprus
X_new = [[22587]] # Cyprus' GDP per capita
print(model.predict(X_new)) # outputs [[5.96242338]]
```

	GDP per capita	Life satisfaction
Country		
Russia	9054.914	6.0
Turkey	9437.372	5.6
Hungary	12239.894	4.9
Poland	12495.334	5.8
Slovak Republic	15991.736	6.1
[[5.96242338]]		



### 3. Binary Classification using a Logistic Regression Model

#### Logistic Regression

Logistic Regression (also called Logit Regression) is commonly used to estimate the probability that an instance belongs to a particular class (e.g., what is the probability that this email is spam?).

If the estimated probability is greater than 50%, then the model predicts that the instance belongs to that class (called the positive class, labeled “1”), or else it predicts that it does not (i.e., it belongs to the negative class, labeled “0”).

Thus it acts as a binary classifier.

**Problem Statement 1:** Build and train a Logistic Regression Model to do binary classification of iris flowers using the iris dataset. In particular, the model should predict whether a particular iris flower instance belongs to the class Iris Virginica or not using only \*petal width\* as the input feature.

**Code:-**

```
import numpy as np
from sklearn import datasets

iris = datasets.load_iris()
print(type(iris))
print(list(iris.keys()))
X = iris["data"][:,3:] # petal width
y = (iris["target"] == 2).astype(np.int64) # 1 if Iris-Virginica, else 0
```

output:

```
<class 'sklearn.utils._bunch.Bunch'>
['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names', 'filename', 'data_module']
```

**Training the Logistic Regression Model**

```
from sklearn.linear_model import LogisticRegression

log_reg = LogisticRegression(solver="lbfgs", random_state=42)
log_reg.fit(X,y)
```

```
▼ LogisticRegression
LogisticRegression(random_state=42)
```

A look at the model's estimated probabilities for flowers with petal widths varying from 0 to 3 cm

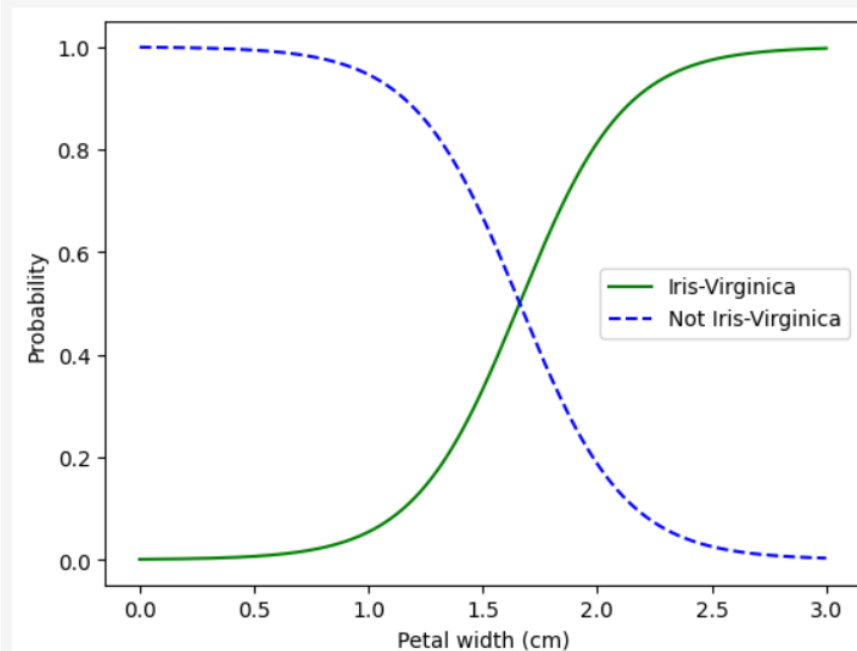
Code:-

```
import matplotlib.pyplot as plt

X_new = np.linspace(0,3,1000).reshape(-1,1)
y_proba = log_reg.predict_proba(X_new)
plt.plot(X_new, y_proba[:,1], "g-")
plt.plot(X_new, y_proba[:,0], "b--")
plt.xlabel('Petal width (cm)')
plt.ylabel('Probability')
plt.legend(['Iris-Virginica', 'Not Iris-Virginica'])
```

Output:-

<matplotlib.legend.Legend at 0x7f03fcb2bd30>



The petal width of Iris-Virginica flowers (represented by triangles) ranges from 1.4cm to 2.5 cm, while the other iris flowers (represented by squares) generally have a smaller petal width, ranging from 0.1 cm to 1.8 cm.

There is a bit of overlap. Above about 2 cm the classifier is highly confident that the flower is an Iris-Virginica (it outputs a high probability to that class), while below 1

cm it is highly confident that it is not an Iris-Virginica (high probability for the “Not Iris-Virginica” class). In between these extremes, the classifier is unsure.

There is a decision boundary at around 1.6 cm where both probabilities are equal to 50%: if the petal width is higher than 1.6 cm, the classifier will predict that the flower is an Iris-Virginica, or else it will predict that it is not (even if it is not very confident):

```
log_reg.predict([[1.7],[1.5]])
```

```
array([1, 0])
```

**Problem Statement 2:** Logistic Regression for predicting class using two features: Petal length and width.

Below we use the same dataset but use two features: petal width and petal length to train the Logistic Regression model to estimate the probability that a new flower is an Iris-Virginica based on these two features.

The dashed line represents the points where the model estimates a 50% probability. This is the model's decision boundary. Note that it is a linear boundary.

Each parallel line represents the

points where the model outputs a specific probability, from 15% (bottom left) to 90% (top right). All the flowers beyond the top-right line have an over 90% chance of being Iris-Virginica according to the model.

```
from sklearn.linear_model import LogisticRegression

X = iris["data"][:(2, 3)] # petal length, petal width
y = (iris["target"] == 2).astype(np.int64)

log_reg2 = LogisticRegression(solver="lbfgs", C=10**10, random_state=42)
log_reg2.fit(X, y)

x0, x1 = np.meshgrid(
    np.linspace(2.9, 7, 500).reshape(-1, 1),
    np.linspace(0.8, 2.7, 200).reshape(-1, 1),
)
X_new = np.c_[x0.ravel(), x1.ravel()]
print(X_new.shape)
```

```

y_proba = log_reg2.predict_proba(X_new)

plt.figure(figsize=(10, 4))
plt.plot(X[y==0, 0], X[y==0, 1], "bs")
plt.plot(X[y==1, 0], X[y==1, 1], "g^")

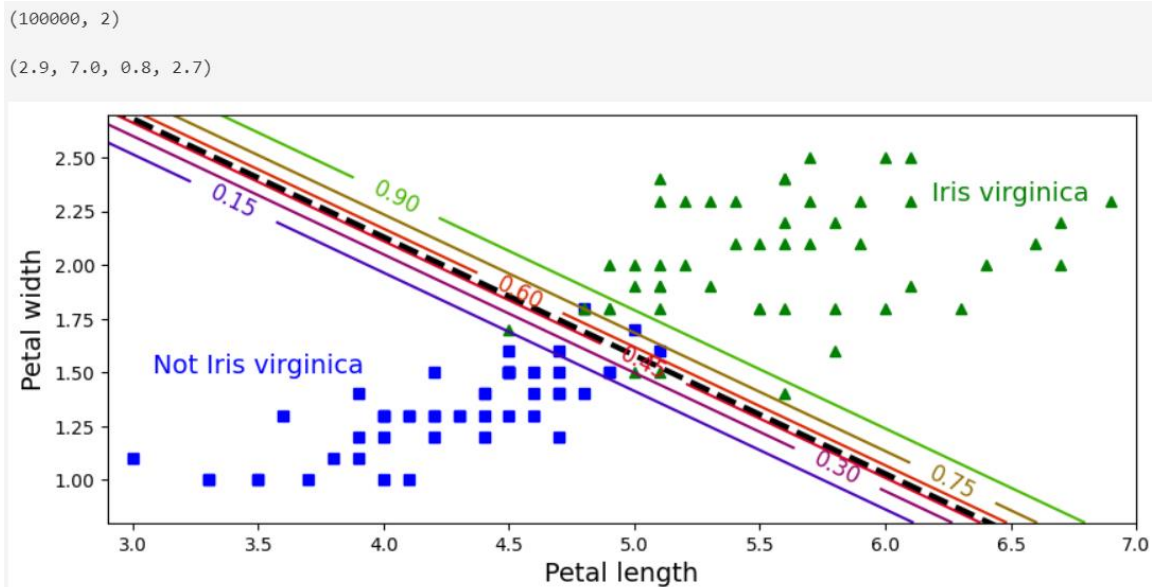
zz = y_proba[:, 1].reshape(x0.shape)
contour = plt.contour(x0, x1, zz, cmap=plt.cm.brg)

left_right = np.array([2.9, 7])
boundary = -(log_reg2.coef_[0][0] * left_right + log_reg2.intercept_[0]) / log_reg2.coef_[0][1]

plt.clabel(contour, inline=1, fontsize=12)
plt.plot(left_right, boundary, "k--", linewidth=3)
plt.text(3.5, 1.5, "Not Iris virginica", fontsize=14, color="b", ha="center")
plt.text(6.5, 2.3, "Iris virginica", fontsize=14, color="g", ha="center")
plt.xlabel("Petal length", fontsize=14)
plt.ylabel("Petal width", fontsize=14)
plt.axis([2.9, 7, 0.8, 2.7])

```

## Output:-



## 4. Multi-Class Classification using k-Nearest Neighbours

Implement & Train a k-Nearest Neighbour Model for Detecting Iris Flower Species

**Problem Statement:** Load the iris dataset. Build a k-NN model with k=3. Explore the Iris Data using visualizations to understand the relationships among different input features with respect to each species.

Then use all four features: sepal length (cm), sepal width (cm), petal length (cm), and petal width (cm) to train the k-NN model to classify a new iris flower into one of three species classes: Iris Virginica, Iris Setosa or Iris Versicolor.

Finally evaluate the model accuracy on the held-out Test Data.

### # Loading/Fetching Your Data

The data we will use for this example is the Iris dataset, a classical dataset in machine learning and statistics. It is included in scikit-learn in the datasets module. We can load it by calling the load\_iris function:

```
from sklearn.datasets import load_iris

iris_dataset = load_iris()

# Code that shows the short description of the Iris Dataset
print(iris_dataset['DESCR'][:193] + "\n...")
print("Keys of iris_dataset: \n{}".format(iris_dataset.keys()))

# The value of the key target_names is an array of strings, containing the species of
# flower that we want to predict:
print("\n Target names: {}".format(iris_dataset['target_names']))

# The value of feature_names is a list of strings, giving the description of each feature:
print("\n Feature names: \n{}".format(iris_dataset['feature_names']))

# The data itself is contained in the target and data fields. data contains the numeric
# measurements of sepal length, sepal width, petal length, and petal width in a NumPy array:
print("\n Type of data: {}".format(type(iris_dataset['data'])))

# The rows in the data array correspond to flowers, while the columns represent the four
# measurements that were taken for each flower:
print("\n Shape of data: {}".format(iris_dataset['data'].shape))

# We see that the array contains measurements for 150 different flowers.
# Here are the feature values for the first five samples:
```

```
print("\\n First five columns of data:\\n{ }".format(iris_dataset['data'][:5]))

# From this data, we can see that all of the first five flowers have a petal width of 0.2 cm
# and that the first flower has the longest sepal, at 5.1 cm
# The target array that contains the species of each of the flowers that were measured is a
# numpy array:

print("\\n Type of target: { }".format(type(iris_dataset['target'])))
print("\\n Shape of target: { }".format(iris_dataset['target'].shape))

# The species are encoded as integers from 0 to 2:

print("\\n Target:\\n{ }".format(iris_dataset['target']))
```

output:-

```

> .._iris_dataset:
> Iris plants dataset
> -----
>
> **Data Set Characteristics:**
>
> :Number of Instances: 150 (50 in each of three classes)
> :Number of Attributes: 4 numeric, pre
> ...
> Keys of iris_dataset:
> dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names', 'filename', 'data_module'])
>
> Target names: ['setosa' 'versicolor' 'virginica']
>
> Feature names:
> ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
>
> Type of data: <class 'numpy.ndarray'>
>
> Shape of data: (150, 4)
>
> First five columns of data:
> [[5.1 3.5 1.4 0.2]
>  [4.9 3. 1.4 0.2]
>  [4.7 3.2 1.3 0.2]
>  [4.6 3.1 1.5 0.2]
>  [5. 3.6 1.4 0.2]]
>
> Type of target: <class 'numpy.ndarray'>
>
> Shape of target: (150,)

```

```
Target:  
[000000000000000000000000000000000000000000  
000000000000000111111111111111111111111111  
1111111111111111111111111111111122222222222  
222222222222222222222222222222222222222222  
22]
```

## Splitting the Data into Training and Testing Data



To assess the model's performance we split the labelled data we have (our 150 flower measurements) into two parts: training data and testing data.

Scikit-learn contains a function that shuffles the dataset and splits it for you: the `train_test_split` function. This function extracts 75% of the rows in the data as the training set, together with the corresponding labels for this data. The remaining 25% of the data, together with the remaining labels, is declared as the test set.

Let's call `train_test_split` on our data and assign the outputs using this nomenclature:

```
from sklearn.model_selection import train_test_split

# The output of the train_test_split function is X_train, X_test, y_train, and y_test,
# which are all NumPy arrays.
X_train, X_test, y_train, y_test = train_test_split(iris_dataset['data'],
                                                    iris_dataset['target'], random_state=0)

# X_train contains 75% of the rows of the dataset, and X_test contains the remaining 25%:
print("X_train shape: {}".format(X_train.shape))
print("y_train shape: {}".format(y_train.shape))
print("X_test shape: {}".format(X_test.shape))
print("y_test shape: {}".format(y_test.shape))
```

```
X_train shape: (112, 4)
y_train shape: (112,)
X_test shape: (38, 4)
y_test shape: (38,)
```

## # Knowing Your Data

One of the best ways to inspect data is to visualize it. One way to do this is by using a scatter plot for each pair of features. This is also known as a scatter matrix or a pair plot. A pair plot looks at all possible pairs of features. If you have a small number of features, such as the four we have here, this is quite reasonable.

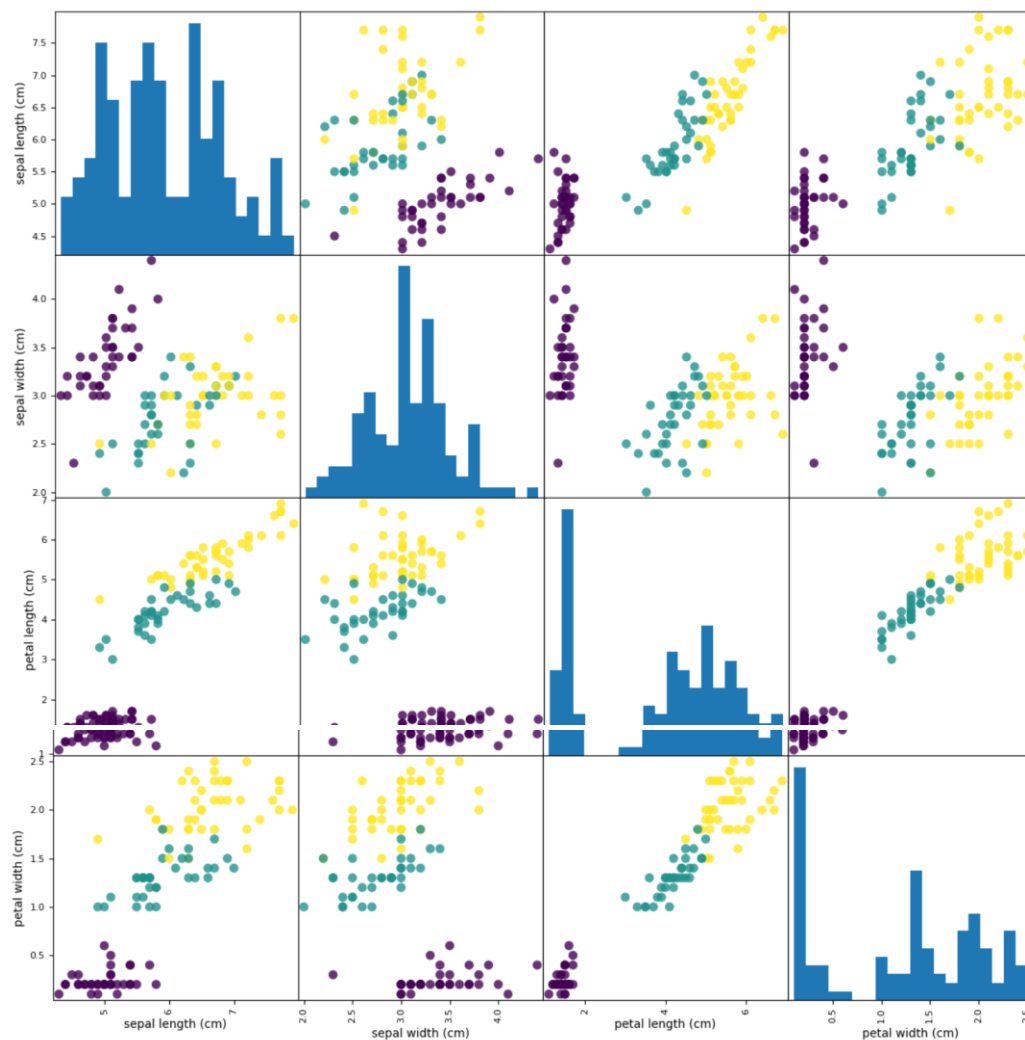
Below we draw a pair plot of the features in the training set. The data points are colored according to the species the iris belongs to.

To create the plot, we first convert the

NumPy array into a pandas DataFrame. pandas has a function to create pair plots called `scatter_matrix`. The diagonal of this matrix is filled with histograms of each feature:

```
import pandas as pd
# create dataframe from data in X_train
# label the columns using the strings in iris_dataset.feature_names
iris_dataframe = pd.DataFrame(X_train, columns=iris_dataset.feature_names)

# create a scatter matrix from the dataframe, color by y_train
grr = pd.plotting.scatter_matrix(iris_dataframe, c=y_train, figsize=(15, 15), marker='o',
hist_kws={'bins': 20}, s=60, alpha=.8)
```



From the plots, we can see that the three classes seem to be relatively well separated using the sepal and petal measurements.

This means that a machine learning model will likely be able to learn to separate them.

## # Building our K-Nearest Neighbours (K-NN) Model

We train a k-nearest neighbours classifier with our training data. Once trained our model should be able to predict the species of any new iris flower given its four features: petal length, petal width, sepal length and sepal width in centimeters.

To make a prediction for a new data point, the k-NN algorithm finds the k points in the training set that are closest to the new point. Then, a prediction is made using the majority class among these neighbors.

The k-nearest neighbors classification algorithm is implemented in the `KNeighborsClassifier` class in the `neighbors` module:

```
from sklearn.neighbors import KNeighborsClassifier

# The most important parameter of KNeighborsClassifier is the number of neighbors,
# which we will set to 3:
knn = KNeighborsClassifier(n_neighbors=3)
```

## # Training the k-NN Model

To train the model on the training set, we call the `fit` method of the `knn` object, which takes as arguments the NumPy array `X_train` containing the training data and the NumPy array `y_train` of the corresponding training labels.

The `fit` method returns the `knn` object itself (and modifies it in place), so we get a string representation of our classifier.

```
knn.fit(X_train, y_train)
```

```
▼ KNeighborsClassifier
KNeighborsClassifier(n_neighbors=3)
```

## # Making Predictions

We can now make predictions using this model on new data for which we might not know the correct labels.

Imagine we found an iris in the wild with a sepal length of 5 cm, a sepal width of 2.9 cm, a petal length of 1 cm, and a petal width of 0.2 cm. What species of iris would this be?

We can put this data into a NumPy array with shape as number of samples (1) multiplied by number of features (4):

```
import numpy as np

# Note that we made the measurements of this single flower into a row in a twodimensional
# NumPy array, as scikit-learn always expects two-dimensional arrays for the data.
X_new = np.array([[5, 2.9, 1, 0.2]])
print("X_new.shape: {}".format(X_new.shape))

# To make a prediction, we call the predict method of the knn object:
prediction = knn.predict(X_new)
print("Prediction: {}".format(prediction))
print("Predicted target name: {}".format(iris_dataset['target_names'][prediction]))
```

```
X_new.shape: (1, 4)
Prediction: [0]
Predicted target name: ['setosa']
```

## # Evaluating our k-NN model

we can make a prediction for each iris in the test data and compare it against its label (the known species).

We can measure how well the model works by computing the accuracy, which is the fraction of flowers for which the right species was predicted:

```
y_pred = knn.predict(X_test)
print("Test set predictions:\n {}".format(y_pred))
print("Test set score: {:.2f}".format(np.mean(y_pred == y_test)))

# We can also use the score method of the knn object, which will compute the test set
# accuracy for us:
print("Test set score: {:.2f}".format(knn.score(X_test, y_test)))
```

```
Test set predictions:
```

```
[2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0 0 1 1 0 2 1 0 2 2 1 0  
2]
```

```
Test set score: 0.97
```

```
Test set score: 0.97
```

## Conclusion

For this model, the test set accuracy is about 0.97, which means we made the right prediction for 97% of the irises in the test set. Under some mathematical assumptions, this means that we can expect our model to be correct 97% of the time for new irises.

## 5. Linear and Non-Linear SVM Classification

### Train & Implement a SVM Model for Linear and Non-Linear Classification.

Problem Statement 1 - Linear SVM Classification: Load the iris dataset, scale the petal-width and petal-length features and then train a linear SVM model to detect Iris-Virginica flowers. Use the petal length and petal width features to train the model. Scale the features before training the model. Use the resulting model to do the detection on some sample data point.

```
import numpy as np
from sklearn import datasets
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import LinearSVC

iris = datasets.load_iris()
X = iris["data"][:, (2, 3)] # petal length, petal width
y = (iris["target"] == 2).astype(np.float64) # Iris-Virginica has code 2

svm_clf = Pipeline([
    ("scaler", StandardScaler()),
    ("linear_svc", LinearSVC(C=1, loss="hinge"))
])

svm_clf.fit(X,y)

# predict for a sample iris flower with petal length 5.5 and petal width 1.7
svm_clf.predict([[5.5, 1.7]]) # detected as Iris Virginica
# [1 -- Virginica, 0 -- Not Virginica]
```

```
array([1.])
```

Problem Statement 2 - Nonlinear SVM Classification: Train and implement a SVM model to do binary classification on moons dataset. Since the dataset is linearly inseparable do a transformation and add polynomial features to the dataset.

```
# Plotting Moons data to illustrate its linear inseparability
import matplotlib.pyplot as plt
```

```
from sklearn.datasets import make_moons
```

```
X, y = make_moons(n_samples = 100, noise=0.4)
```

```
def plot_dataset(X,y,axes):
```

```
    plt.plot(X[:,0][y==0], X[:,1][y==0], "bs") # bs stands for blue square
```

```
    plt.plot(X[:,0][y==1], X[:,1][y==1], "g^") # g^ stands for green triangle
```

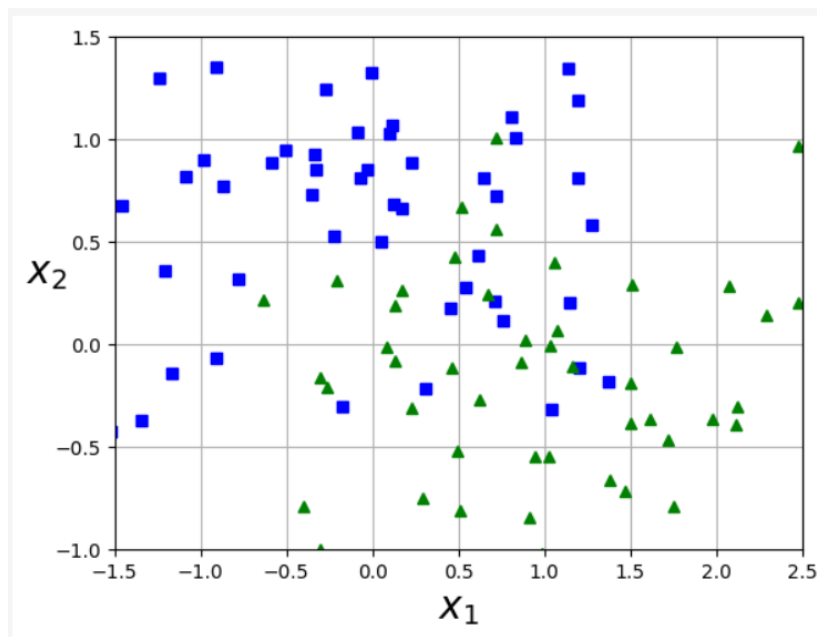
```
    plt.axis(axes)
```

```
    plt.grid(True, which='both')
```

```
    plt.xlabel(r"$x_1$", fontsize=20)
```

```
    plt.ylabel(r"$x_2$", fontsize=20, rotation=0)
```

```
plot_dataset(X,y,[-1.5,2.5,-1,1.5])
```



```
from sklearn.pipeline import Pipeline
```

```
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
```

```
from sklearn.svm import LinearSVC
```

```
polynomial_svm_clf = Pipeline([
```

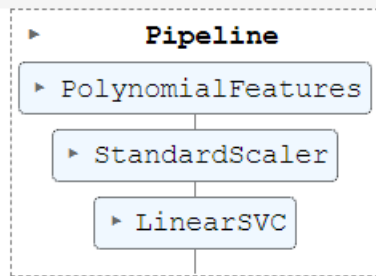
```
    ("poly_features", PolynomialFeatures(degree=3)),
```

```
    ("scaler", StandardScaler()),
```

```
    ("svm_clf", LinearSVC(C=10, loss='hinge', random_state=42))
```

```
])
```

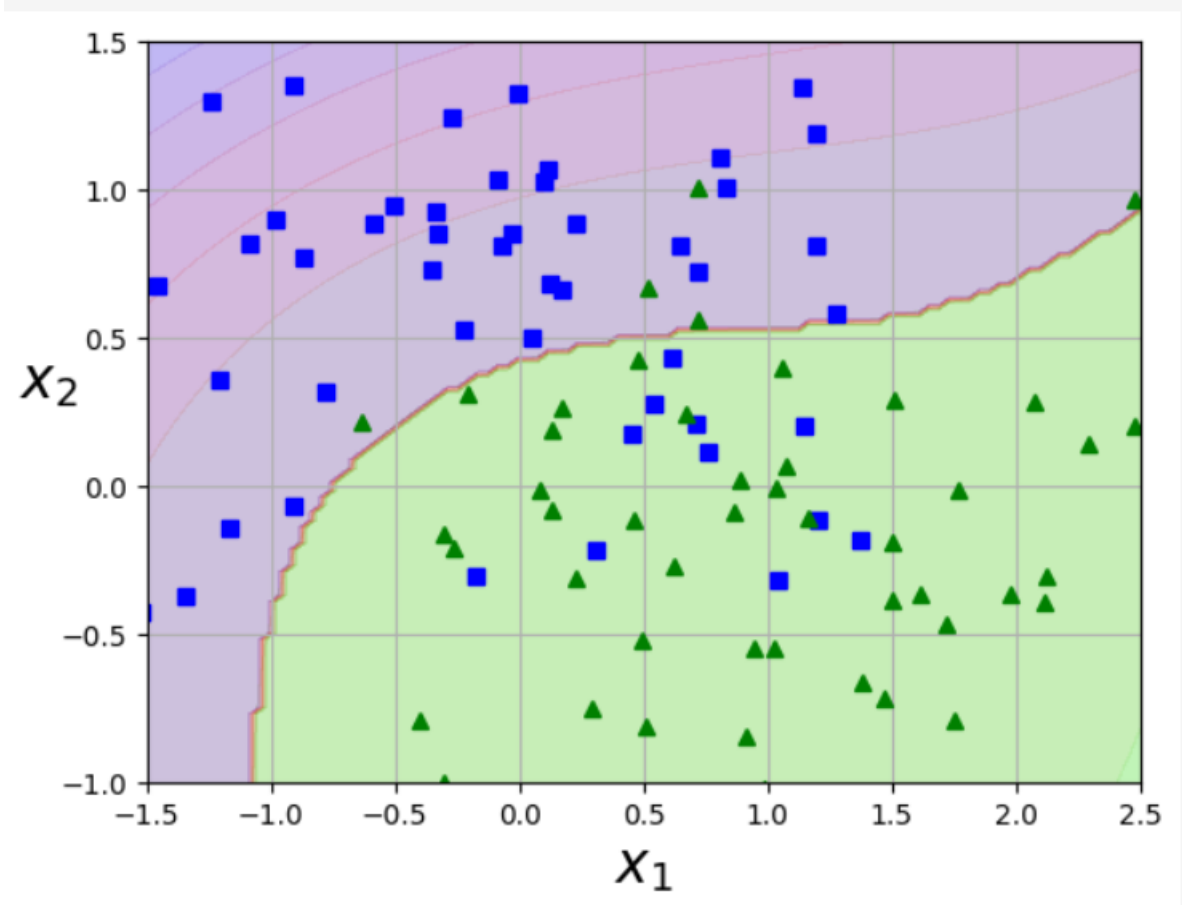
```
polynomial_svm_clf.fit(X,y)
```



## Visualizing the Non Linear SVM Classification Decision Boundary

```
def plot_predictions(clf, axes):  
    x0s = np.linspace(axes[0], axes[1], 100)  
    x1s = np.linspace(axes[2], axes[3], 100)  
    x0, x1 = np.meshgrid(x0s, x1s)  
    X = np.c_[x0.ravel(), x1.ravel()]  
    y_pred = clf.predict(X).reshape(x0.shape)  
    y_decision = clf.decision_function(X).reshape(x0.shape)  
    plt.contourf(x0, x1, y_pred, cmap=plt.cm.brg, alpha=0.2)  
    plt.contourf(x0, x1, y_decision, cmap=plt.cm.brg, alpha=0.1)  
  
# plot predictions and decision boundary  
# of the previously trained SVM classifier for given x and y axes values  
plot_predictions(polynomial_svm_clf, [-1.5, 2.5, -1, 1.5])  
  
# plot the linearly inseparable Moons Dataset as was done previously  
plot_dataset(X,y,[-1.5,2.5,-1,1.5])
```





## 6. Decision Tree Learning for Classification

### Learning Decision Trees for Classification from Iris Dataset

**Problem Statement:** Use the Iris dataset to train and visualize a decision tree for classifying an Iris flower based on its petal length (in cms) and petal width (in cms) features.

Use the trained model to predict the class for a flower having petal-length and petal-width of 5 cms and 1.5 cms respectively. Also try predicting class probabilities instead of a class for the same flower.

Finally, plot the decision boundaries for the induced decision tree classifier.

```
# creating images folder for decision tree lab
import os

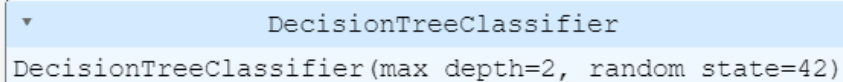
PROJECT_ROOT_DIR = "."
SUB_DIR = "decision_trees"
IMAGES_PATH = os.path.join(PROJECT_ROOT_DIR, "images", SUB_DIR)
os.makedirs(IMAGES_PATH, exist_ok=True)
```

#### # Step 1 - Training and Visualizing a Decision Tree

```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier

iris = load_iris()
X = iris.data[:, 2:] # petal length and width
y = iris.target

tree_clf = DecisionTreeClassifier(max_depth=2, random_state=42)
tree_clf.fit(X, y)
```

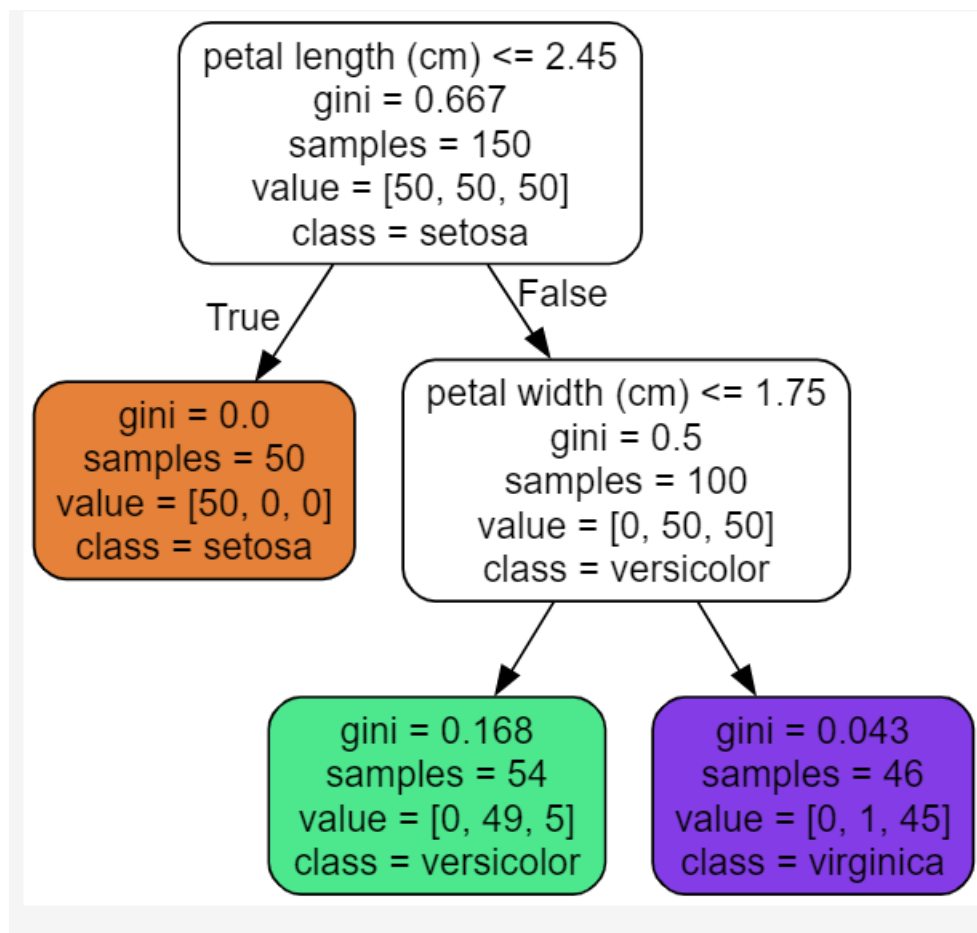


```
# Visualizing the Iris Decision Tree
from graphviz import Source
```

```
from sklearn.tree import export_graphviz
```

```
export_graphviz(
    tree_clf,
    out_file = os.path.join(IMAGES_PATH, "iris_tree.dot"),
    feature_names = iris.feature_names[2:],
    class_names = iris.target_names,
    rounded = True,
    filled = True
)
```

```
Source.from_file(os.path.join(IMAGES_PATH, "iris_tree.dot"))
```



## # Step 2- Making Predictions using the induced DT Classifier

Suppose you have found a flower whose petals are 5 cm long and 1.5 cms wide. The corresponding leaf node is the depth-2 left node, so the decision tree should predict the class as Iris-Versicolor (class 1).

```
# predict the class for flower with petal length 5 cm and petal-width 1.5 cm
```

```
tree_clf.predict([[5,1.5]]) # class 1 is predicted - which is for Iris-Versicolor
```

```
array([1])
```

If you ask it to predict the class probabilities instead, for the same flower, then it should output the following probabilities: 0 % for Iris-Setosa (0/54), 90.7 % for Iris-Versicolor (49/54), and 9.3 % for Iris-Virginica (class 1).

```
# predict class probabilities instead for the same flower
```

```
tree_clf.predict_proba([[5,1.5]])
```

```
array([[0.          , 0.90740741, 0.09259259]])
```

### # Step 3 - Visualizing the Decision Boundary for our Decision Tree

```
import numpy as np
from matplotlib import pyplot as plt
from matplotlib.colors import ListedColormap

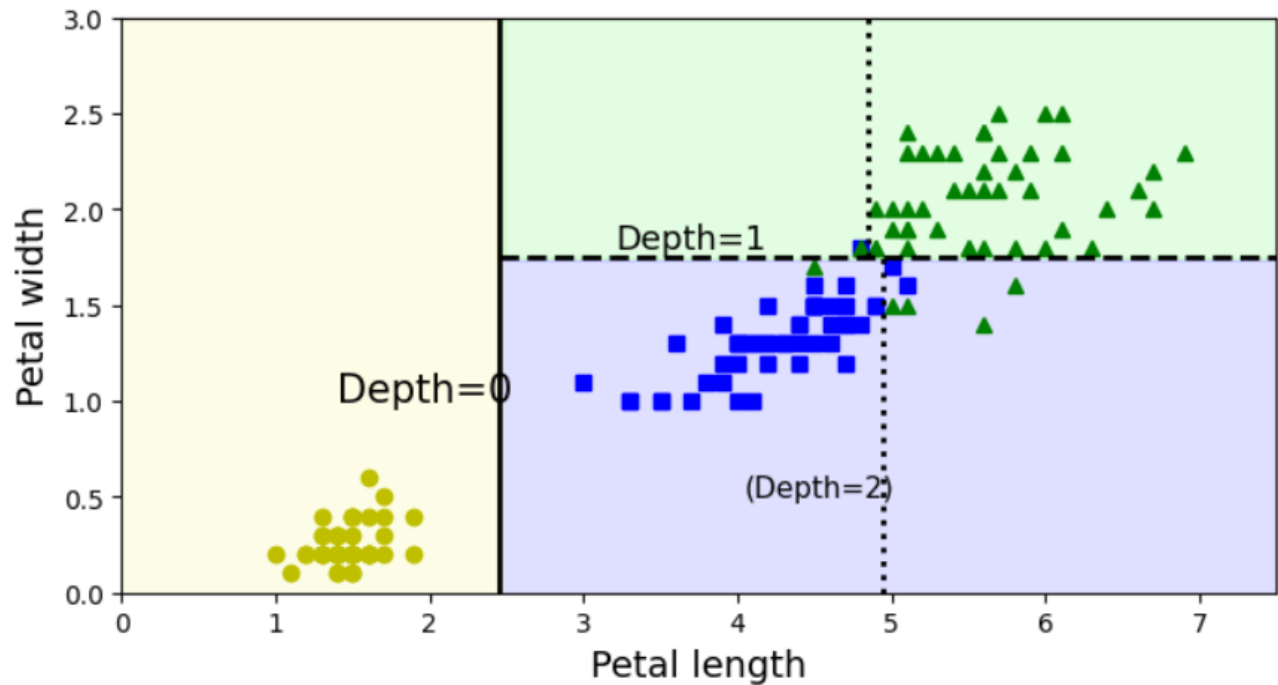
def plot_decision_boundary(clf, X, y, axes=[0, 7.5, 0, 3], iris=True, legend=False, plot_training=True):
    x1s = np.linspace(axes[0], axes[1], 100)
    x2s = np.linspace(axes[2], axes[3], 100)
    x1, x2 = np.meshgrid(x1s, x2s)
    X_new = np.c_[x1.ravel(), x2.ravel()]
    y_pred = clf.predict(X_new).reshape(x1.shape)
    custom_cmap = ListedColormap(['#fafab0', '#9898ff', '#a0faa0'])
    plt.contourf(x1, x2, y_pred, alpha=0.3, cmap=custom_cmap)

    if not iris:
        custom_cmap2 = ListedColormap(['#7d7d58', '#4c4c7f', '#507d50'])
        plt.contour(x1, x2, y_pred, cmap=custom_cmap2, alpha=0.8)
    if plot_training:
        plt.plot(X[:, 0][y==0], X[:, 1][y==0], "yo", label="Iris setosa")
        plt.plot(X[:, 0][y==1], X[:, 1][y==1], "bs", label="Iris versicolor")
        plt.plot(X[:, 0][y==2], X[:, 1][y==2], "g^", label="Iris virginica")
        plt.axis(axes)
    if iris:
        plt.xlabel("Petal length", fontsize=14)
        plt.ylabel("Petal width", fontsize=14)
    else:
        plt.xlabel(r"$x_1$", fontsize=18)
```

```
plt.ylabel(r"$x_2$", fontsize=18, rotation=0)
if legend:
    plt.legend(loc="lower right", fontsize=14)
```

```
plt.figure(figsize=(8, 4))
plot_decision_boundary(tree_clf, X, y)
plt.plot([2.45, 2.45], [0, 3], "k-", linewidth=2)
plt.plot([2.45, 7.5], [1.75, 1.75], "k--", linewidth=2)
plt.plot([4.95, 4.95], [0, 1.75], "k:", linewidth=2)
plt.plot([4.85, 4.85], [1.75, 3], "k:", linewidth=2)
plt.text(1.40, 1.0, "Depth=0", fontsize=15)
plt.text(3.2, 1.80, "Depth=1", fontsize=13)
plt.text(4.05, 0.5, "(Depth=2)", fontsize=11)
```

```
Text(4.05, 0.5, '(Depth=2)')
```



## 7. Ensemble Learning with AdaBoost Classifier

## Implement Ensemble Learning with AdaBoost Classifier

**Problem Statement:** Build and train an AdaBoost classifier where Decision Tree acts as the first base classifier. In particular, train an AdaBoost classifier based on 200 \*Decision Stumps\* on Moons Dataset using Scikit-Learn's AdaBoostClassifier class.

Compare the accuracy of the AdaBoostClassifier with the individual DecisionStump on the Test Set.

Finally draw the decision boundary of the AdaBoostClassifier.

# Step 1: Loading the Moons Data and Splitting into Training and Testing Sets

```
import sklearn
import numpy as np

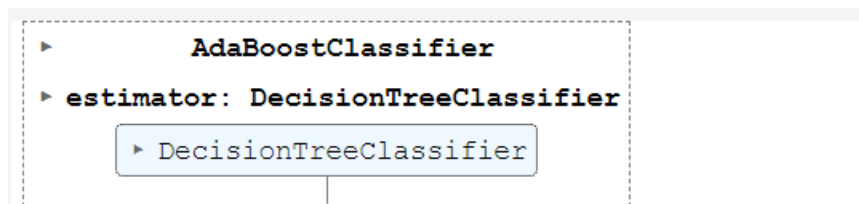
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_moons

X, y = make_moons(n_samples=500, noise=0.30, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
```

# Step 2: Building and Training the AdaBoost Classifier

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier

ada_clf = AdaBoostClassifier(
    DecisionTreeClassifier(max_depth=1, n_estimators=200,
        algorithm="SAMME.R", learning_rate=0.5, random_state=42)
ada_clf.fit(X_train, y_train)
```



# Step 3 : Comparing Performance of the AdaBoost Classifier with Individual Decision Stump on Test Set

```

from sklearn.metrics import accuracy_score

# first, build and train an individual Decision Stump
ds_clf = DecisionTreeClassifier(max_depth=1, random_state=42)
ds_clf.fit(X_train, y_train)

# second, compare the accuracy of the previously trained AdaBoost with the just trained DecisionStump
# on the test data
y_pred_ada_clf = ada_clf.predict(X_test)
y_pred_ds_clf = ds_clf.predict(X_test)

print(ds_clf.__class__.__name__, accuracy_score(y_test, y_pred_ds_clf))
print(ada_clf.__class__.__name__, accuracy_score(y_test, y_pred_ada_clf))

```

```

DecisionTreeClassifier 0.824
AdaBoostClassifier 0.896

```

#### #Step 4 : Visualizing the Decision Boundary of our AdaBoost Model

We define a function named **plot\_decision\_boundary** for the purpose and then invoke it passing our AdaBoost model and training data as arguments.

```

from matplotlib import pyplot as plt
from matplotlib.colors import ListedColormap

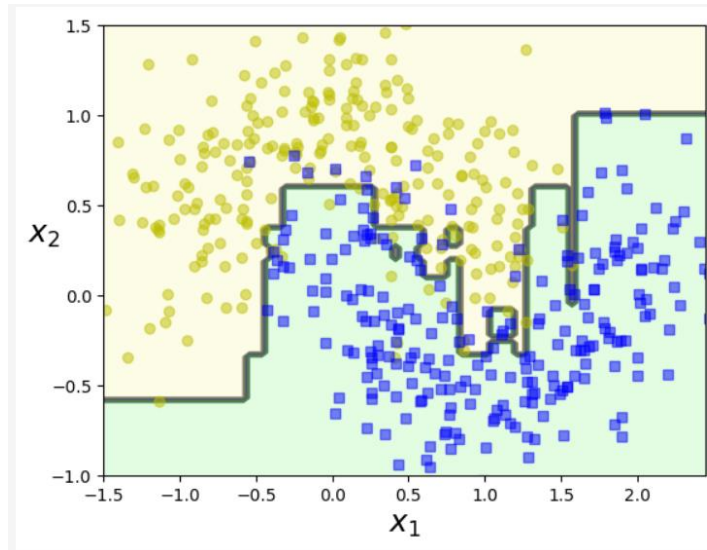
def plot_decision_boundary(clf, X, y, axes=[-1.5, 2.45, -1, 1.5], alpha=0.5, contour=True):
    x1s = np.linspace(axes[0], axes[1], 100)
    x2s = np.linspace(axes[2], axes[3], 100)
    x1, x2 = np.meshgrid(x1s, x2s)
    X_new = np.c_[x1.ravel(), x2.ravel()]
    y_pred = clf.predict(X_new).reshape(x1.shape)
    custom_cmap = ListedColormap(['#fafab0', '#9898ff', '#a0faa0'])
    plt.contourf(x1, x2, y_pred, alpha=0.3, cmap=custom_cmap)
    if contour:
        custom_cmap2 = ListedColormap(['#7d7d58', '#4c4c7f', '#507d50'])
        plt.contour(x1, x2, y_pred, cmap=custom_cmap2, alpha=0.8)
    plt.plot(X[:, 0][y==0], X[:, 1][y==0], "yo", alpha=alpha)
    plt.plot(X[:, 0][y==1], X[:, 1][y==1], "bs", alpha=alpha)
    plt.axis(axes)
    plt.xlabel(r"$x_1$", fontsize=18)
    plt.ylabel(r"$x_2$", fontsize=18, rotation=0)

```

```

plot_decision_boundary(ada_clf, X, y)

```



## 8. Clustering with k-Means Algorithm

# Implement k-Means Clustering on Iris Dataset

Problem Statement:\*\* Build the k-Means cluster model, with K=3 and using the two features: \*Sepal Length\* and \*Sepal Width\*, for the purpose of training the k-Means



clustering model. Finally, visualize the formed clusters along with their respective centroids.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import load_iris

# Load the Iris dataset
iris = load_iris()
X = iris.data[:, :2] # Take only the first two features for simplicity

# Perform K-means clustering
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X)

# Obtain cluster labels and centroids
labels = kmeans.labels_
centroids = kmeans.cluster_centers_

# Visualize the clusters
plt.scatter(X[:, 0], X[:, 1], c=labels)
plt.scatter(centroids[:, 0], centroids[:, 1], marker='x', color='red', label='Centroids')
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.title('K-means Clustering on Iris Dataset')
plt.legend()
```

