Python and Data Science Step by Step guide Material

Welcome!!!

My name is Sojor, and I will be taking you through this tutorial.

In this study guide, I will be taking you through key concepts which are needed in order to master python language.

We will then use those python skills in order to learn data science, remember that python is just a tool needed to help in data science.

So first we will begin by learning python and then proceed to data science.

I will be providing an explanation in each concept and then some few examples to help with better understanding.

In case you have any further question, or you might need more examples, kindly let me know and I will be glad to assist.

That said let's begin with the topics to be covered!

- Variables
- Basic inbuilt operators
- If, else Conditions
- For and while loops
- Lists, tuples, dictionary and their functions
- Functions
- Object Oriented Programming
- List comprehensions
- Lambda functions
- Data structures
- Argument's operators (*args and **kwargs)
- PIP and Environments

The above are the most important concepts needed to become a Pythonista, however, there are still other advanced concepts you can master with time.

- Generators and Decorator functions
- Meta classes
- Parallelism
- Packages and Modules

Question for you..., Why are learning python and where do you want to apply it?

Answer…, I want to become a data scientist.

Perfect, now after you have understood python language, we now jump into data science.

I will list the concepts to be covered!!

- Microsoft Excel functionality
- Statistics
- Visualization and clean up libraries

- Machine Learning
- Deep learning
- Databases
- Power BI and Tableau tools

And yes, these are the topics needed to become a data scientist,

However, data science is an ever-changing subject with new concepts being introduced all-time, so it's important to keep up with any new changes.

Ready?

Let's get started!!!

Variables

There are different types of variables in python

- ✓ Numbers
- ✓ Booleans
- ✓ Strings

- ✓ Numbers

These can either be floats or integers

For example, we have

```
number_of_books=10, the #10 is an integer value

cost_of_fuel=112.45, the #112.45 is a float value
```

- ✓ Booleans

These are either True or False

For example

```
its_raining=False,it is raining can either be True or False

shes_studying=True
```

Note that python is a case sensitive language ,for instance #false and #False does not imply the same thing, so it is important to obey its syntax otherwise you get an error

For example if we set , `shes_studying=true , we get an error`

```
shes_studying=true
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-880b7e40012e> in <module>()
----> 1 shes_studying=true

NameError: name 'true' is not defined
```

So keep in mind to always start with a capital letter

- ✓ Strings

For example,

```
course="Python Course"  # "Python Course" is a string
```

A string is always enclosed by “” or ‘’, “Enter Text” or ‘Enter Text’

Lets look at variable names

So there is a specific way in which we name our variables

For instance ,recall this example `its_raining=False ,`

Note that we wrote the variable name as --its_raining– ,but not --It’s Raining--, and that is the point,when we are naming our variables we always start with a lowercase lettered name followed by __ and then the second part of the variable name,

For example

students_count ,reynolds_number, number_of_vehicles,course,rating

Note: We do not separate our variables names with a space in python but with an underscore _

➢ Basic inbuilt operators

Comments are used in python to explain the functionality of the code,

To write a comment we begin with the # symbol, followed by the statement,

Note that comments are not executed when you run the code

comments

```
[3] #comments--these are used in python to explain the workaround the code, to write a comment, we begin with # symbol
    #
```

print

```
[4] course_name='Python Programming'
    print(course_name)
```

```
Python Programming
```

```
[5] print('Python Programming')
```

```
Python Programming
```

input

```
[7] name=input('Enter your name: ')
```

```
Enter your name: sojor
```

```
[8] name=input('Enter your name: ')
    print('My name is :' ,name)
```

```
Enter your name: sojor
My name is : sojor
```

```
[9] name=input('Enter your name: ')# use of formatted string but the output is the same
    print(f'My name is : {name}' )
```

```
Enter your name: sojor
My name is : sojor
```

mathematics operators

```
[10] 3+4
```

```
7
```

```
[11] print(3+4) #addition
```

```
7
```

```
[12] print(4*6) #multiplication
```

```
24
```

```
[13] print(10/5) # division
```

```
2.0
```

```
[14] print(16-4) #subtraction
```

```
12
```

```
[15] #lets do some mathematics using these variables
     value_1=10
     value_2=2
     value_3=7
     value_4=54
```

```
[16] #addition
     print(value_4+value_1)
```

```
64
```

```
[17] #multiplication
     print(value_3*value_2)
```

```
14
```

power (**)

```
[18] print(value_2**2) #we are getting the square of value_2
```

```
4
```

modulus(//) -this divides values and leaves no remainder

```
[19] print(value_3/value_2) #with a single / -just division
```

```
3.5
```

```
[20] print(value_3//value_2) #with double // =modulus
```

```
3
```

int --returns a no decimal value

```
[27] value_5=int(input('Enter value 5: '))
     value_6=int(input('Enter value 6: '))
     result=value_5+value_6
     print(f'The result of the summation is equal to :{result}')
```

```
Enter value 5: 11
Enter value 6: 32
The result of the summation is equal to :43
```

```
[29] value_7=int(input('Enter value 7: '))
     value_8=int(input('Enter value 8: '))
     result=value_7 * value_8
     print(f'The result of the multipilcation is equal to :{result}')
```

```
Enter value 7: 5
Enter value 8: 7
The result of the multipilcation is equal to :35
```

float--returns a decimal value

```
[30] value_9=float(input('Enter value 9: '))
     value_10=float(input('Enter value 10: '))
     result=value_9 / value_10
     print(f'The result of the division is equal to :{result}')
```

```
Enter value 9: 10
Enter value 10: 2
The result of the division is equal to :5.0
```

str --converts numbers to strings

```
[36] first_name=str(input('Enter your first name: '))
     last_name=str(input('Enter your second name: '))
     print(f'My name is {first_name } {last_name}')
```

```
Enter your first name: joh
Enter your second name: ojor
My name is joh ojor
```

```
[37] value_9=str(input('Enter value 9: '))
     value_10=str(input('Enter value 10: '))
     print(f'The result of the is equal to :{value_9+value_10}')
```

```
Enter value 9: 2
Enter value 10: 3
The result of the is equal to :23
```

```
#note that the str function returns a different value compared to int f
unction so you have
#to be careful when choosing which function to use
```

➢ If, else .. ,and Conditions

If,else and conditions

```
[ ] <--less than
    >--greater than
    >=--greater than or equal
    <=--less than or equal
    !=--not equal to
    ==--equal to
```

```
[39] print(4<1)
```

```
False
```

```
[40] print(4>=2)
```

```
True
```

```
[41] print(6*4 != 5)
```

```
True
```

if ,else,elif conditions

if

```
[44] if 4==4:
       print('True')
```

```
True
```

```
[54] #another example
     score=float(input('Enter your total score: '))

     if score == 100:
       print('Your score is 100')
```

```
Enter your total score: 100
Your score is 100
```

```
[55] #now lets try this example again but we put diffrent score
     score=float(input('Enter your total score: '))

     if score == 100:
       print('Your score is 100')
```

```
Enter your total score: 50
```

you will notice that nothing happens on the printout section in regard to your score,and that is because we did not tell python what to do in the case when the score is not 100

so python automatically treats that as a none and thus nothing is printed

```
[66] #we can also add an elif ans else statement to the above code  as follows
     #you can use as many elif statements as possible to specify your required output
     #but you can only use one else statement
     score=float(input('Enter your total score: '))

     if score == 100:
       print('Your score is 100')
     elif score==50:
       print('Your score is 50')
     elif score >=150:
       print('Your score is greater or equal 150')
     elif score >10 and score <=40:
       print('Your score is less or equal to 40 and greater than 10')
     elif (score >0 and score <=10) or score ==6 :
       print('Your score is between 0 and 10')
     else:
       print('Your score is invalid')
```

```
Enter your total score: -4
Your score is invalid
```

```
father='Mike'
mother='Ileana'
if father =='Mike':
  print('His mother is ',mother)
```

```
His mother is  Ileana
```

else

```
[47] father='Tom'  #what if his father's name was not mike then we use the else statement as below
     mother='Ileana'
     if father =='Mike':
       print('His mother is ',mother)
     else:
       print('His mother is not actually ',mother)

     #the output is executed by the else statement because his father was not Mike
     #But if his father was Mike then output will be executed by the if statement
```

```
His mother is not actually  Ileana
```

```
[49] #lets  look at another example, where we combine a bit of what we have learned so far
     #first lets run the programe and put name as sojor and compare the output with when the name isn't sojor
     my_name='sojor'
     name=str(input('Enter name: '))
     if name ==my_name:
       print(f'My name is actually  {name}')
     else:
       print('My name is not',name)
```

```
Enter name: sojor
My name is actually  sojor
```

```
[50] #lets try this once more but this time we put a different name as see the output
     my_name='sojor'
     name=str(input('Enter name: '))
     if name ==my_name:
       print(f'My name is actually  {name}')
     else:
       print('My name is not',name)
```

```
Enter name: tom
My name is not tom
```

elif statement

```
[53] #lets use the exact same example but we add an elif statement
     my_first_name='sojor'
     my_last_name='joh'

     name=str(input('Enter name: '))
     if name ==my_first_name:
       print(f'My First name is {name}')
     elif name ==my_last_name:
       print('My Last name is ',name)
     else:
       print('My name is not',name)
```

```
Enter name: joh
My Last name is  joh
```

➢ For loops and while loops

for loops and while loops

```
[67] #What loops does is that they help to iterate through a range ,list of certain values
     #Say for example we have a list with 5 values,then we need to print each value within
     #that list,
     #Then we use loops to iterate through all the values and print them out
     #for example we have
     for item in range(5):
       print(item)
```

```
0
1
2
3
4
```

```
[68] for value in range(3,9):
       print(value)
```

```
3
4
5
6
7
8
```

```
[74] #how to break out of for loops,
     #for instance we want to stop immediately after we print the first value in the list i.e range(5)
     value=True
     for item in range(5):
       if value is not None:
         print(item)
         break
```

```
0
```

```
[76] #nested loops
     #this is putting more than one for-loop in one programw
     #for example
     for i in range(4):
       for j in range(2):
         print((i,j))
```

```
(0, 0)
(0, 1)
(1, 0)
(1, 1)
(2, 0)
(2, 1)
(3, 0)
(3, 1)
```

```
[77] #we can use for loops to iterate through a string as well
     for letter in 'STUDENT':
       print(letter)
```

```
S
T
U
D
E
N
T
```

```
[78] #iterating through a list,denoted by []
     for value in [5,6,7,8]:
       print(value)
```

```
5
6
7
8
```

```
[79] #iterating through a tuple,denoted by ()
     for value in (5,6,7,8):
       print(value)
```

```
5
6
7
8
```

```
[80] #iterating through a dictionary,denoted by {}
     for value in {5,6,7,8}:
       print(value)
```

```
8
5
6
7
```

```
[83] #while loops
     value=1 #this is the initializiation
     while value <=5:  #this is the condition
       print(value)
       value=value+1 #this line of code prevents the code from runing forever called the increment/decrement
```

```
1
2
3
4
5
```

```
[84] #for decrementing values we have
     value=4 #this is the initializiation
     while value >=1:  #this is the condition
       print(value)
       value=value-1  #decrement
```

```
4
3
2
1
```

```
[89] #another example
     i=0
     while i <=5:
       print(i)
       i+=1
```

```
0
1
2
3
4
5
```

```
[90] #using a break statement to break out of a loop
     i=0
     while i <=5:
       if i==3:
         break
       print(i)
       i+=1
```

```
0
1
2
```

```
[92] i=0
     while True:
       if i==3:
         break
       print(i)
       i=i+1
```

```
0
1
2
```

- **Lists, tuples, dictionary and their functions**

list,tuples ,dictionary

```
[ ]  #list-this allows to work with a range of values
     #it is formed with [] brackets
     #for example
     subjects=['math','biology','chem','history']
     #we can get the length of this list by running the below command
     print(len(subjects))
```

```
4
```

```
[ ]  #now to access the individual subject in the list we use the [] after the subject ,i.e
     print(subjects[1]) #index 1 represents biology ,indexes starts from 0,1,2....
```

```
biology
```

```
[ ]  #we can use negative index,which represents the last item
     print(subjects[-1])
```

```
history
```

```
[ ]  #to obtain certain number of values in list or slicing
     print(subjects[1:3])
```

```
['biology', 'chem']
```

```
[29] print(subjects[1:])
```

```
['biology', 'chem', 'history']
```

```
[30] print(subjects[:3])
```

```
['math', 'biology', 'chem']
```

```
[31] #adding values to the list
     subjects.append('physics')
     print(subjects)
```

```
['math', 'biology', 'chem', 'history', 'physics']
```

```
[32] #adding at start of list
     subjects.insert(0,'english')
     print(subjects)
```

```
['english', 'math', 'biology', 'chem', 'history', 'physics']
```

```
[33] #extend method to add list of values in list
     subjects.extend(['art','compsci'])
     print(subjects)
```

```
['english', 'math', 'biology', 'chem', 'history', 'physics', 'art', 'compsci']
```

```
[34] #removing items in list
     subjects.remove('physics')
     print(subjects)
```

```
['english', 'math', 'biology', 'chem', 'history', 'art', 'compsci']
```

```
[35] #pop method to remove last item in list
     subjects.pop()
     print(subjects)
```

```
['english', 'math', 'biology', 'chem', 'history', 'art']
```

```
[36] #sorting the list
     subjects.reverse()
     print(subjects)
```

```
['art', 'history', 'chem', 'biology', 'math', 'english']
```

```
[37] #sorting the list
     subjects.sort()
     print(subjects)
```

```
['art', 'biology', 'chem', 'english', 'history', 'math']
```

```
[38] numbers=[3,5,7,1,8]
```

```
[39] (numbers.sort()) #ascending order
     print(numbers)
```

```
[1, 3, 5, 7, 8]
```

```
[40] (numbers.sort(reverse=True)) #descending order
     print(numbers)
```

```
[8, 7, 5, 3, 1]
```

```
[41] #sorted function to sort the list ,if you dont want to alter the original list
     sorted_sub=sorted(subjects)
     print(sorted_sub)
```

```
['art', 'biology', 'chem', 'english', 'history', 'math']
```

```
[42] #to get minimum number in list
     print(min(numbers))
```

```
1
```

```
[43] #to get maximum number in list
     print(max(numbers))
```

```
8
```

```
[44] #to get total sum  in list
     print(sum(numbers))
```

```
24
```

```
[45] #to obtain index of any subject
     print(subjects.index('art'))
```

```
0
```

```
[46] #using -in- function to see if item is in the list
     print('physics' in subjects)
```

```
False
```

```
[47] #using for loop and enumerate function to print all the subjects in our list
     for index,subject in enumerate(subjects):
       print(index,subject) #this will print both the subject at its index
```

```
0 art
1 biology
2 chem
3 english
4 history
5 math
```

tuples and sets

```
[48] #tuples are similar to lists but we can not modify them
     #they are represented by ()
     subjects1=tuple(subjects)
     subjects1
```

```
('art', 'biology', 'chem', 'english', 'history', 'math')
```

```
[49] subjects1=('art', 'biology', 'chem', 'english', 'history', 'math')
```

```
[50] print(subjects1.insert('compsci')) #we get an error because tuples can not be editted or modified
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-50-9fdc83e71777> in <module>()
----> 1 print(subjects1.insert('compsci')) #we get an error because tuples can not be editted or modified

AttributeError: 'tuple' object has no attribute 'insert'

SEARCH STACK OVERFLOW
```

```
[65] #we mainly use tuples if we need to form range of values which we dont need them modified
```

```
[65]
```

sets

```
[66] #sets are similar also to list but they are represented by {} brackets
     subjects2={'art', 'biology', 'chem', 'english', 'history', 'math'}
```

```
[67] print(subjects2)
```

```
     {'biology', 'art', 'history', 'english', 'chem', 'math'}
```

```
[68] #with sets helps to remove duplicates
     subjects2={'art', 'biology', 'chem', 'english','chem', 'history', 'math'}
     print(subjects)
```

```
     ['art', 'biology', 'chem', 'english', 'history', 'math']
```

```
[69] #using -in- function to see if item is in the set
     print('chem' in subjects)
```

```
     True
```

```
[70] #we can use the below to create empty lists
     list1=[]
     #or
     list1=list()

     #empty sets
     set1={}
     set1=set()

     #empty tuples
     tuple1=()
     tuple1=tuple()
```

dictionaries

```
[71] #this allows to work with a key value pairs
```

```
[72] #how to generate a dictionary
     employee={'name': 'Tom','age':30,'salary':2500}
     print(employee)
```

```
     {'name': 'Tom', 'age': 30, 'salary': 2500}
```

```
[73] #to slice the dictionary to access only the age key
     print(employee['age'])
```

```
     30
```

```
[74] print(employee['name'])
```

```
Tom
```

```
[75] #accessing a key that does not exist we get an error
     print(employee['sub'])
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-75-ce1bef48da6f> in <module>()
      1 #accessing a key that does not exist we get an error
----> 2 print(employee['sub'])

KeyError: 'sub'
```

SEARCH STACK OVERFLOW

```
[76] #get method to get values
     print(employee.get('age'))
```

```
30
```

```
[77] #adding values to dictionary
     employee['number']='236-854-84'
     print(employee)
```

```
{'name': 'Tom', 'age': 30, 'salary': 2500, 'number': '236-854-84'}
```

```
[78] #updating values in the dictionary
     #get method to get values
     employee.update({'name':'Yori'})
```

```
[79] #get method to get values
     print(employee)
```

```
{'name': 'Yori', 'age': 30, 'salary': 2500, 'number': '236-854-84'}
```

```
[80] #deleting values in dictionary using the key
     del employee ['number']
     print(employee)
```

```
{'name': 'Yori', 'age': 30, 'salary': 2500}
```

```
[81] #pop method to remove values
     employee_age=employee.pop('age')
     print(employee_age)
     print(employee)
```

```
30
{'name': 'Yori', 'salary': 2500}
```

```
[82] #length of the dictionary
     print(len(employee))
```

```
2
```

```
[83] #to print all keys in the dict
     print(employee.keys())
```

```
dict_keys(['name', 'salary'])
```

```
[84] #to print all values in the dict
     print(employee.values())
```

```
dict_values(['Yori', 2500])
```

```
[85] #to print both keys and values in the dict
     print(employee.items())
```

```
dict_items([('name', 'Yori'), ('salary', 2500)])
```

```
[86] #using for loop to loop all items in the dictionary
     for key ,value in employee.items():
       print(key,value)
```

```
name Yori
salary 2500
```

➢ Functions

functions

```
[87] #to create a function we use the def keyword ,which  means definition
     def hello_function():
       print('Hello')
```

```
[88] #to execute this function we do the following
     hello_function()
```

```
Hello
```

```
[89] #function allows us to re-use code without having to repeat them
     hello_function()
     hello_function()
     hello_function()
```

```
Hello
Hello
Hello
```

```
#function allows us to re-use code without having to repeat them
hello_function()
hello_function()
hello_function()
```

```
Hello
Hello
Hello
```

```
#return word on functions
def hello_function():
  return 'Hello'
```

[91]
```
#to execute this we have
print(hello_function())
```

```
Hello
```

[92]
```
#obtaining upper case for our function
print(hello_function().upper())
```

```
HELLO
```

[93]
```
#how to pass arguments to a function
def hello_function(hey):
  return 'Hello'
```

[94]
```
#to execute this
print(hello_function('hey'))
```

```
Hello
```

[95]
```
def hello_function(hey,name='You'):
  return '{} ,{}'.format(hey,name)
```

[96]
```
print(hello_function('hey',name='Roy'))
```

```
hey ,Roy
```

[97]
```
#another example
def greet():
  print('Hey there')
  print('Welcome to class')
```

```
[98] #to execute this
     greet()
```

```
Hey there
Welcome to class
```

```
[99] #functions with parameter
     def greet(first_name,last_name):
       print(f'Hey {first_name} {last_name}')
       print('Welcome to class')
```

```
[100] greet('Roy','Keane')
```

```
Hey Roy Keane
Welcome to class
```

```
[101] #function which returns a value using a return keyword
      def greet(first_name):
        return (f'Hey {first_name} Welcome to class')
```

```
[102] print(greet('Roy'))
```

```
Hey Roy Welcome to class
```

```
[103] def increment(num,x):
        return num +x
```

```
[104] print(increment(5,7))
```

```
12
```

```
[105] def save_user(**user): #using kwargs or key-word-arguments
        print(user)

      save_user(name='Roy',age=25)
```

```
{'name': 'Roy', 'age': 25}
```

```
[106] def save_user(*user): #using args
        print(user)

      save_user('Roy','Tom','Keane')
```

```
('Roy', 'Tom', 'Keane')
```

```
[107] #another example

      def fizz_buzz(input):
        if (input % 3==0) and (input % 5==0):
          return 'FizzBuzz'
        elif input % 3 ==0: # % refers to modulus ,it means that the input is divisble by 3
          return 'Fizz'
        elif input % 5==0:
          return 'Buzz'
        else:
          return 'Invalid Value'


      print(fizz_buzz(15))
```

```
FizzBuzz
```

```
[108] def fizz_buzz(input):
        if (input % 3==0) and (input % 5==0):
          return 'FizzBuzz'
        elif input % 3 ==0: # % refers to modulus ,it means that the input is divisble by 3
          return 'Fizz'
        elif input % 5==0:
          return 'Buzz'
        else:
          return 'Invalid Value'


      print(fizz_buzz(4))
```

➢ Object Oriented Programming (OOP)

object oriented programming

```
[109] #This is composed of a class and several functions which we call methods
      #To execute a class then we will need to create an object from the class
      #In oop we can build an entire store by defining all the different types of
      #operations which shoud be carried out
      #These specific operations carried out ,we can them methods which are coded as
      #normal functions with the def key word
```

```
[110] #to create a class we use class keyword

      class Computer: #remember the definition computer must start with a capital letter

        #now we can define some methods
        def config(self):  #each method must take in the word self
          print('Hp,4gb ,1000Gb')
```

```
[111] #to execute this,we first create an object of the class as follows

      computer1=Computer()
      computer1.config()
```

```
Hp,4gb ,1000Gb
```

```
[112] #init method in oop
      #this is used for initializing the methods
      #defined as def __init__(self)

      class Computer:

        def __init__(self,cpu,ram):
          #creating the variables
          self.cpu=cpu
          self.ram=ram


        #now we can define some methods
        def config(self):
          print('Config is ', self.cpu ,self.ram )



      #to execute this we have
      computer1=Computer('ir',16)
      computer2=Computer('grw',8)


      computer1.config()
      computer2.config()
```

```
Config is  ir 16
Config is  grw 8
```

```
[113] #another example
      class Robot:
        def introduce_self(self):
          print('My name is ' + self.name)
```

```
[114] #to execute this
      r1=Robot()
      r1.name='Roy'

      r1.introduce_self()
```

```
My name is Roy
```

```
[115] #using init keyword to do exactly the above execution
      #another example
      class Robot:

        def __init__(self,name):
          self.name=name

        def introduce_self(self):
          print('My name is ' + self.name)
```

```
[116] r1=Robot('Roy')

      r1.introduce_self()
```

```
      My name is Roy
```

```
[117] #same examle with more functionality
      class Robot:

        def __init__(self,name,color,weight):
          self.color=color  #you can name the variables any name,we just choose these for simplicity
          self.weight=weight
          self.name=name

        def introduce_self(self):
          print(f"My name is  {self.name}  and i like color  {self.color} and i'm  {self.weight} Lbs of weight")



      #to execute this
      r1=Robot('Roy','blue',25)
      r2=Robot('KEane','red',30)
      r1.introduce_self()
```

```
      My name is  Roy  and i like color  blue and i'm  25 Lbs of weight
```

```
[118] #lets do another example

        class Person:
          def __init__(self,name,personality,is_sitting):
            self.name=name
            self.perosnality=personality
            self.is_sitting=is_sitting

          def sit_down(self):
            self.is_sitting=True

          def stand_up(self):
            self.is_sitting=False
```

```
[119] #to execute the above class

        p1=Person('Clare','talkative',False)
        p2=Person('Geof','hubbled',True)
```

```
[120] #now we can combine both classes i.e Robot and Person class as follows

        p1.robot_owned=r2
        p2.robot_owned=r1
```

```
[121] p1.robot_owned.introduce_self()
```

```
      My name is  KEane  and i like color  red and i'm  30 Lbs of weight
```

➢ List Comprehensions

List comprehensions helps in writing code in one single line

```
#List comprehensions
my_list=[i for i in range(1,11)]
my_list
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
my_list=[i*i for i in range(1,11)]
my_list
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

```
my_list=[i for i in range(1,11) if i%2==0]
my_list
```

```
[2, 4, 6, 8, 10]
```

```
my_list=[i**2 for i in range(1,11)]
my_list
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

```
my_list=[i**2 for i in range(1,11) if i%2==0]
my_list
```

```
[4, 16, 36, 64, 100]
```

```
my_list=[i for i in range(1,11) if i>3 if i!=7 if i!=9]
my_list
```

```
[4, 5, 6, 8, 10]
```

```
#if else statement in list comprehensins
my_list=[i if i>3 else i+1 for i in range(1,11)]
my_list # this clearly means that we will print values of i which are greater than 3 and if the values are not greater than 3
#we add i+1 i.e we add 1 to those values n then we print them too
```

```
[2, 3, 4, 4, 5, 6, 7, 8, 9, 10]
```

```
nums=[2,3,4,5]
fruit=['apple','banana','pears','peaches']
for num in nums:
    if num>2:
        for f in fruit:
            print(num,f)
```

```
3 apple
3 banana
3 pears
3 peaches
4 apple
4 banana
4 pears
4 peaches
5 apple
5 banana
5 pears
5 peaches
```

```
#in list comprehension we can have the same as above although in tuples
my_list=[(num,f) for num in nums if num>2 for f in fruit]
my_list
```

```
[(3, 'apple'),
 (3, 'banana'),
 (3, 'pears'),
 (3, 'peaches'),
 (4, 'apple'),
 (4, 'banana'),
 (4, 'pears'),
 (4, 'peaches'),
 (5, 'apple'),
 (5, 'banana'),
 (5, 'pears'),
 (5, 'peaches')]
```

```
my_list=[(f,num) for f in fruit if f!='banana' for num in nums ]
my_list
```

```
[('apple', 2),
 ('apple', 3),
 ('apple', 4),
 ('apple', 5),
 ('pears', 2),
 ('pears', 3),
 ('pears', 4),
 ('pears', 5),
 ('peaches', 2),
 ('peaches', 3),
 ('peaches', 4),
 ('peaches', 5)]
```

```
my_list=[(num,f) for num in nums if num!=5 if num!=3 if num!=1 for f in fruit if f!='peaches']
print(my_list)
```

```
[(2, 'apple'), (2, 'banana'), (2, 'pears'), (4, 'apple'), (4, 'banana'), (4, 'pears')]
```

```
#the above list comprehension can be writtin in nested for loops and if's as below
for num in nums:
    if num!=5:
        if num!=3:
            if num!=1:
                for f in fruit:
                    if f !='peaches':
                        print(num,f)
```

```
2 apple
2 banana
2 pears
4 apple
4 banana
4 pears
```

```
my_list=[(f,num) for f in fruit if f!='banana' for num in nums if num!=1]
print(my_list)
```

```
[('apple', 2), ('apple', 3), ('apple', 4), ('apple', 5), ('pears', 2), ('pears', 3), ('pears', 4), ('pears', 5), ('peaches',
2), ('peaches', 3), ('peaches', 4), ('peaches', 5)]
```

```
#dictionary comprehensions
my_dict={f:num for f ,num in zip(fruit,nums) }
print(my_dict)
```

```
{'apple': 2, 'banana': 3, 'pears': 4, 'peaches': 5}
```

```
my_dict
```

```
{'apple': 2, 'banana': 3, 'pears': 4, 'peaches': 5}
```

```
my_dict={f:num for f,num in zip(fruit,nums) if f!='banana'}
my_dict
```

```
{'apple': 2, 'pears': 4, 'peaches': 5}
```

```
my_dict={f:num for f,num in zip(fruit ,nums) if num!=2 if f!='peaches'}
my_dict
```

```
{'banana': 3, 'pears': 4}
```

```
#set comprehensions
numbers=[1,1,1,2,3,4,5,5,6,8,7,7,9,5]
numbers
```

```
[1, 1, 1, 2, 3, 4, 5, 5, 6, 8, 7, 7, 9, 5]
```

```
my_set={i for i in numbers}
my_set
```

```
{1, 2, 3, 4, 5, 6, 7, 8, 9}
```

```
my_set={i**2 for i in numbers if i%2==0}
my_set
```

```
{4, 16, 36, 64}
```

```
my_set={i if i>4 else i*3 for i in numbers }# the reason we aint getting lesser i value multiples is because set dont store
#duplicates
my_set
```

```
{3, 5, 6, 7, 8, 9, 12}
```

```
my_dict={f:num for f,num in zip(fruit,nums)}
print(my_dict)
```

```
{'apple': 2, 'banana': 3, 'pears': 4, 'peaches': 5}
```

```
my_set={i**2 if i%2==0 else i+2 for i in numbers}#this simply means that we will be squaring the even numbers in numbers list
print(my_set)            #if not so then we take the values of numbers not even and add 2 to them,finally we print them all
```

```
{64, 3, 4, 5, 36, 7, 9, 11, 16}
```

```
######### more examples
```

```
n_ll=[str(i) for i in [1,2,3,4]]
print(n_ll)
```

```
['1', '2', '3', '4']
```

```
n_ll=[int(i) for i in range(1,5)]
n_ll
```

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

```
n_ll=[float(i) for i in range(1,5)]
n_ll
```

```
[1.0, 2.0, 3.0, 4.0]
```

```
n_ll=[bool(i) for i in range(6)]
n_ll
```

```
[False, True, True, True, True, True]
```

```
n_ll=[str(i) for i in range(1,5) if i>2]
n_ll
```

```
['3', '4']
```

```
#when using normal for loops
squares=[]
for i in range(1,100):
    squares.append(i**2)
print(squares)
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256, 289, 324, 361, 400, 441, 484, 529, 576, 625, 676, 729, 78
4, 841, 900, 961, 1024, 1089, 1156, 1225, 1296, 1369, 1444, 1521, 1600, 1681, 1764, 1849, 1936, 2025, 2116, 2209, 2304, 2401, 2
500, 2601, 2704, 2809, 2916, 3025, 3136, 3249, 3364, 3481, 3600, 3721, 3844, 3969, 4096, 4225, 4356, 4489, 4624, 4761, 4900, 50
41, 5184, 5329, 5476, 5625, 5776, 5929, 6084, 6241, 6400, 6561, 6724, 6889, 7056, 7225, 7396, 7569, 7744, 7921, 8100, 8281, 846
4, 8649, 8836, 9025, 9216, 9409, 9604, 9801]
```

```
#the above using list comprehension
squares=[i**2 for i in range(1,100)]
print(squares)
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256, 289, 324, 361, 400, 441, 484, 529, 576, 625, 676, 729, 78
4, 841, 900, 961, 1024, 1089, 1156, 1225, 1296, 1369, 1444, 1521, 1600, 1681, 1764, 1849, 1936, 2025, 2116, 2209, 2304, 2401, 2
500, 2601, 2704, 2809, 2916, 3025, 3136, 3249, 3364, 3481, 3600, 3721, 3844, 3969, 4096, 4225, 4356, 4489, 4624, 4761, 4900, 50
41, 5184, 5329, 5476, 5625, 5776, 5929, 6084, 6241, 6400, 6561, 6724, 6889, 7056, 7225, 7396, 7569, 7744, 7921, 8100, 8281, 846
4, 8649, 8836, 9025, 9216, 9409, 9604, 9801]
```

```
names=['john','sarah','ernest','edmud','elly']
e_names=[name for name in names if name.startswith('e')]
print(e_names)
#the above means that we print the name amongest the names if the name itself startswith a letter 'e'
```

```
['ernest', 'edmud', 'elly']
```

```
#using list comprehensions to compute the cartesian product of 2 sets
#i.e A*B where A ={1,3} ,B ={x,y}
A={1,3,5,7}
B={2,4,6,8}
cartesian_product=[(a,b) for a in A for b in B]
print(cartesian_product)
```

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

- **Lambda functions**

Just like list comprehensions ,Lambda function helps write functions in a single line of code

lambda functions

```
#lambda functions
f=lambda a: a*a
f
```

```
<function __main__.<lambda>(a)>
```

```
#filter function
def is_even(n):
    return n%2==0

nums=[1,2,3,4,5,6,7,8,9]
evens=list(filter(is_even,nums))
print(evens)
```

```
[2, 4, 6, 8]
```

lambda within filter()

```
#or we can uselambda function
evens=list(filter(lambda n:n%2==0,nums))
print(evens)
```

```
[2, 4, 6, 8]
```

```
#map function
def update(n):
    return n+2

evens=list(filter(lambda n:n%2==0,nums))
doubles=list(map(update,evens))
print(doubles)
```

```
[4, 6, 8, 10]
```

lambda within map()

```
#or we can use lambda to perform the exact above operation
evens=list(filter(lambda n:n%2==0,nums))
doubles=list(map(lambda n: n+2,evens))
print(doubles)
```

```
[4, 6, 8, 10]
```

```
#reduce function

from functools import reduce
def add_all(a,b):
    return a+b

doubles=list(map(lambda n: n+2,evens))
sum=reduce(add_all,doubles)
print(sum)
```

```
28
```

lambda within reduce

```
#we can also use lambda function to perfom the above operation
#reduce function
#the reduce function returns a single value where it adds the values in a sequence  NB to be looked at in details
#reduce function adds each values in a sequence recursively and returns a single value NB proved already
from functools import reduce

doubles=list(map(lambda n: n+2,evens))
sum=reduce(lambda a,b:a+b,doubles)
print(doubles)
print(sum)
```

```
[4, 6, 8, 10]
28
```

```
add5=lambda x:x+5
print(add5(7))
```

```
12
```

```
square=lambda x:x*x
print(square(8))
```

```
64
```

```
numbers=[1,2,3,4,5]
get_evens=list(filter(lambda x:x%2==0,numbers))
print(get_evens)
```

```
[2, 4]
```

```
#sorting a list of Tuples using lambda
list1=[('eggs',3.44),('hooney',6.9),('carrots',1.34),('peaches',2.45)]
list1.sort(key=lambda x:x[0]) # can be sorted on the basis of specified index at x[?]
print(list1)
```

```
[('carrots', 1.34), ('eggs', 3.44), ('hooney', 6.9), ('peaches', 2.45)]
```

```
#sorting a list of dictionaries using lambda
import pprint as pp
list1=[{'make':'ford','model':'focus','year':2013},{'make':'feels','model':'x','year':1999}]
list2=sorted(list1,key=lambda x:x['make'])
pp.pprint(list2)
```

```
[{'make': 'feels', 'model': 'x', 'year': 1999},
 {'make': 'ford', 'model': 'focus', 'year': 2013}]
```

```
#lambda conditionals
starts_with=lambda x: True if x.startswith('J')else False
print(starts_with('Jerey'))
```

```
True
```

```
#lambdas on datetime objects
import datetime
now=datetime.datetime.now()
print(now)
year=lambda x:x.year
print(year(now))
```

```
2021-04-06 11:46:22.964734
2021
```

```
import datetime
now=datetime.datetime.now()
print(now)
year=lambda x:x.year
print(year(now))
```

```
2021-04-06 11:46:23.233385
2021
```

```
def do_something(f,val):
    return f(val)

func=lambda x:x**3
print(func(16))
print(do_something(func,5))
```

```
4096
125
```

lambda for Algebra

linear equations

linear equations are of a degree one

```
s=lambda a:a*a
s(4)
```

```
16
```

```
#3x+4y
d=lambda x,y:3*x+4*y
d(4,7)
```

```
40
```

quadratic equations (equations of a degree 2)

```
#(a+b)**2
x=lambda a,b:(a+b)**2
x(3,4)
```

```
49
```

```
def build_quandratic(a,b,c):
    return lambda x:a*x**2+b*x+c
f=build_quandratic(2,3,-5)
print(f(0))
print(f(6))
print(f(10))
```

```
-5
85
225
```

```
import statistics
data=[1.3,2.7,0.6,4.1,4.7,0.1]
avg=statistics.mean(data)
avg
```

```
2.25
```

```
filtered=list(filter(lambda x:x>avg,data))
filtered
```

```
[2.7, 4.1, 4.7]
```

```
data=[1,3,5,3,6,8,9,41,23,45,78,43]
multiplier=reduce(lambda x,y:x*y,data)# this returns the product of all numbers in the data sequence
multiplier
```

```
2766836685600
```

```
#map function exercise
a=[4,5,6,7]
b=[11,22,45,67]
mapped=list(map(lambda x:x+1,b))
mapped
```

```
[12, 23, 46, 68]
```

```
a=[1,2,3,4]
b=[1,1,1,1]
added=list(map(lambda x,y:x+y,a,b))
added
```

```
[2, 3, 4, 5]
```

```
#filter function exercise
a=[3,5,1,6,7,4,9]
def odd(n):
    return lambda x:x%2==1#this is wrong ,you cant use  lambda in the return function when getting only odd or even values
#this can only work if you are creating a quadratic function
filtered=list(filter(odd,a))
filtered
```

```
[3, 5, 1, 6, 7, 4, 9]
```

```
a=[3,5,1,6,7,4,9]
filtered=list(filter(lambda x: x>5,a))
filtered
```

```
[6, 7, 9]
```

```
#reduce function exercise
from functools import reduce
reduced=reduce(lambda x,y: x*y,a)
reduced
```

```
22680
```

```
a=[3,5,1,6,7,4,9]
def odd(n):
    return n%2==1
filtered=list(filter(odd,a))
filtered
```

```
[3, 5, 1, 7, 9]
```

➢ Data structures and Algorithms

Data structures are particular ways of organizing data in a computer so that it can be used effectively.
For example, we can store a list of items having the same data-type using the *array* data structure.

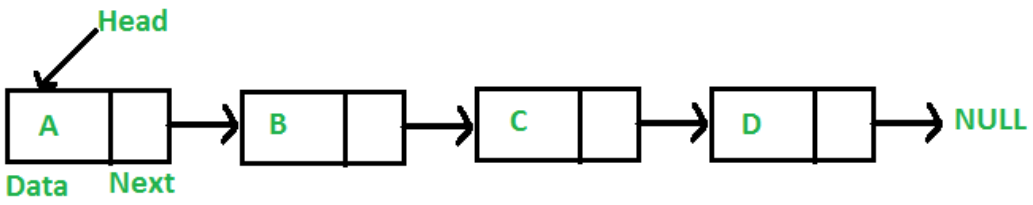
Examples of data structures

- Array
- Binary Search Tree
- Heap
- Queue
- Linked List
- Stack
- Binary Tree
- Hashing
- Graph

Lets have a look at some data structures before jumping into the code behind them

- Linked List

Like arrays, Linked List is a linear data structure. Unlike arrays, linked list elements are not stored at a contiguous location; the elements are linked using pointers.



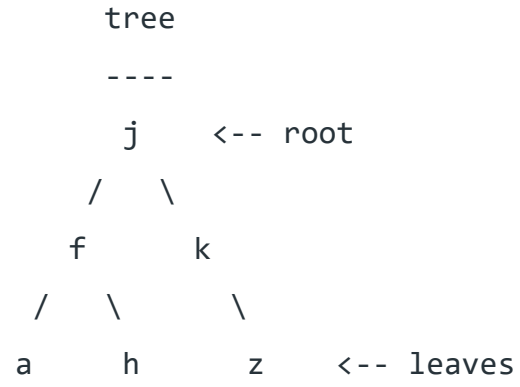
Reasons for using Linked Lists over simple python list or arrays

-Ease of insertion/deletion

-Dynamic size

❖ Binary Trees

Trees: Unlike Arrays, Linked Lists, Stack and queues, which are linear data structures, trees are hierarchical data structures.
Tree Vocabulary: The topmost node is called root of the tree. The elements that are directly under an element are called its children. The element directly above something is called its parent. For example, ‘a’ is a child of ‘f’, and ‘f’ is the parent of ‘a’. Finally, elements with no children are called leaves.



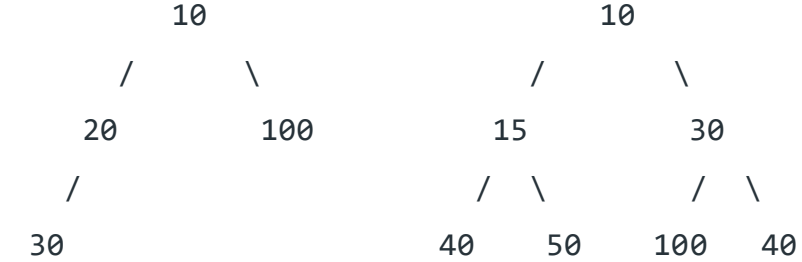
The main reason why we use Trees is because you might want to store information that naturally forms a hierarchy. For example, the file system on a computer

❖ Heap

A Binary Heap is a Binary Tree with following properties.
1) It’s a complete tree (All levels are completely filled except possibly the last level and the last level has all keys as left as possible). This property of Binary Heap makes them suitable to be stored in an array.

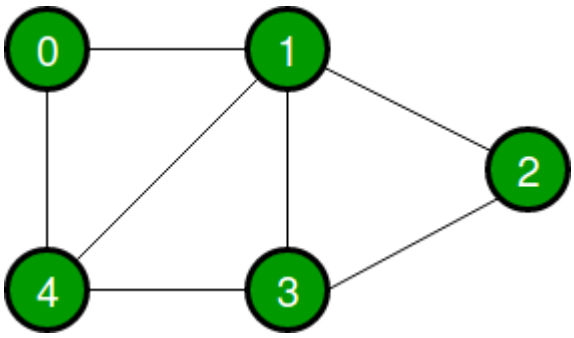
2) A Binary Heap is either Min Heap or Max Heap. In a Min Binary Heap, the key at root must be minimum among all keys present in Binary Heap. The same property must be recursively true for all nodes in Binary Tree. Max Binary Heap is similar to MinHeap.

Examples of Min Heap:



❖ Graphs

A graph is a data structure that consists of the following two components:
1. A finite set of vertices also called as nodes.
2. A finite set of ordered pair of the form (u, v) called as edge. The pair is ordered because (u, v) is not the same as (v, u) in case of a directed graph(di-graph). The pair of the form (u, v) indicates that there is an edge from vertex u to vertex v. The edges may contain weight/value/cost.
Graphs are used to represent many real-life applications: Graphs are used to represent networks. The networks may include paths in a city or telephone network or circuit network. Graphs are also used in social networks like linkedIn, Facebook. For example, in Facebook, each person is represented with a vertex(or node). Each node is a structure and contains information like person id, name, gender, and locale. See this for more applications of graph.
Following is an example of an undirected graph with 5 vertices.



The following two are the most commonly used representations of a graph.
1. Adjacency Matrix
2. Adjacency List
There are other representations also like, Incidence Matrix and Incidence List. The choice of graph representation is situation-specific. It totally depends on the type of operations to be performed and ease of use.
Adjacency Matrix:
Adjacency Matrix is a 2D array of size V x V where V is the number of vertices in a graph. Let the 2D array be adj[][], a slot adj[i][j] = 1 indicates that there is an edge from vertex i to vertex j. Adjacency matrix for undirected graph is always symmetric. Adjacency Matrix is also used to represent weighted graphs. If adj[i][j] = w, then there is an edge from vertex i to vertex j with weight w.

Note: Check out https://www.geeksforgeeks.org/data-structures/

for more in-depth explanations on data structures

Now lets look at the code behind some of these data structures

Stack

Stack is a linear data structure that follows a particular order in which the operations are performed. The order may be LIFO(Last In First Out) or FILO(First In Last Out).

Mainly the following three basic operations are performed in the stack:

- **Push:** Adds an item in the stack. If the stack is full, then it is said to be an Overflow condition.
- **Pop:** Removes an item from the stack. The items are popped in the reversed order in which they are pushed. If the stack is empty, then it is said to be an Underflow condition.
- **Peek or Top:** Returns the top element of the stack.
- **isEmpty:** Returns true if the stack is empty, else false

```
#stack
from collections import deque
class Stack:
    def __init__(self):
        self.stack=deque()
    def push(self,data):
        self.stack.append(data)
    def pop(self):
        if len(self.stack)>0:
            return self.stack.pop()
        else:
            return None
    def peek(self):
        if len(self.stack)>0:
            return self.stack[len(self.stack)-1]
    def __str__(self):
        return str(self.stack)

my_stack=Stack()
my_stack.push(3)
my_stack.push(4)
my_stack.push(5)
print(my_stack)
my_stack.pop()
print(my_stack.peek())
```

```
deque([3, 4, 5])
4
```

Queue

Like Stack, Queue is a linear structure which follows a particular order in which the operations are performed. The order is **F**irst **I**n **F**irst **O**ut (FIFO). A good example of queue is any queue of consumers for a resource where the consumer that came first is served first.
The difference between stacks and queues is in removing. In a stack we remove the item the most recently added; in a queue, we remove the item the least recently added.

Operations on Queue:

Mainly the following four basic operations are performed on queue:

Enqueue: Adds an item to the queue. If the queue is full, then it is said to be an Overflow condition.

Dequeue: Removes an item from the queue. The items are popped in the same order in which they are pushed. If the queue is empty, then it is said to be an Underflow condition.

Front: Get the front item from queue.

Rear: Get the last item from queue.

```
#Queue
from collections import deque
class Queue:
    def __init__(self):
        self.queue=deque()
    def push(self,data):
        self.queue.append(data)
    def pop(self):
        if len(self.queue)>0:
            return self.queue.popleft()
        else:
            return None
    def peek(self):
        if len(self.queue)>0:
            return self.queue[len(self.queue)-1]
    def __str__(self):
        return str(self.queue)

my_queue=Queue()
my_queue.push(4)
my_queue.push(5)
my_queue.push(6)
print(my_queue)
my_queue.pop()
print(my_queue)
print(my_queue.peek())
```

```
deque([4, 5, 6])
deque([5, 6])
6
```

Linked Lists

```
#Linked list(single linked list)
class Node:
    def __init__(self,data):
        self.item=data
        self.ref=None

class LinkedList:
    def __init__(self):
        self.head=None
    def insert_at_start(self,data):
        new_node=Node(data)
        new_node.ref=self.head
        self.head=new_node
    def insert_at_end(self,data):
        if self.head is None:
            self.head=Node(data)
        else:
            temp=self.head
            while temp.ref is not None:
                temp=temp.ref
            temp.ref=Node(data)
    def traverse(self):
        if self.head is None:
            print('LL is empty')
            return
        else:
            temp=self.head
            while temp is not None:
                print(temp.item)
                temp=temp.ref
    def find(self,data):
        temp=self.head
        while temp is not None:
            if temp.item==data:
                return True
            else:
                temp=temp.ref
        return False
    def get_size(self):
        temp=self.head
        count=0
        while temp is not None:
            count+=1
            temp=temp.ref
        return count
    def delete(self,data):
        temp=self.head
        temp_prev=None
        while temp is not None:
            if temp.item==data:
                if temp_prev is not None:
                    temp_prev.ref=temp.ref
                else:
                    self.head=temp.ref
                return True
            else:
                temp_prev=temp
                temp=temp.ref
        return False
    def print(self):
        temp=self.head
        while temp is not None:
            print(temp.item,end='->')
            temp=temp.ref
        print('None')

new_ll=LinkedList()
new_ll.insert_at_start(4)
new_ll.insert_at_start(5)
new_ll.insert_at_start(6)
```

```
new_ll.insert_at_end(7)
new_ll.insert_at_end(8)
new_ll.insert_at_end(9)
new_ll.print()
print(new_ll.find(7))
print(new_ll.find(799))
print(new_ll.get_size())
new_ll.delete(4)
new_ll.delete(6)
new_ll.delete(9)
new_ll.print()
```

```
6->5->4->7->8->9->None
True
False
6
5->7->8->None
```

Heaps

```
#max heap
class MaxHeap:
    def __init__(self,items=[]):
        super().__init__()
        self.heap=[0]
        for item in items:
            self.heap.append(item)
            self.__floatup(len(self.heap)-1)
    def push(self,data):
        self.heap.append(data)
        self.__floatup(len(self.heap)-1)
    def pop(self):
        if len(self.heap)>2:
            self.__swap(1,len(self.heap)-1)
            max=self.heap.pop()
            self.__bubbledown(1)
        elif len(self.heap)==2:
            max=self.heap.pop()
        else:
            max=False
        return max
    def peek(self):
        if self.heap[1] is not None:
            return self.heap[1]
    def __swap(self,i,j):
        self.heap[i],self.heap[j]=self.heap[j],self.heap[i]
    def __floatup(self,index):
        parent=index//2
        if index<=1:
            return
        else:
            if self.heap[parent]<self.heap[index]:
                self.__swap(index,parent)
```

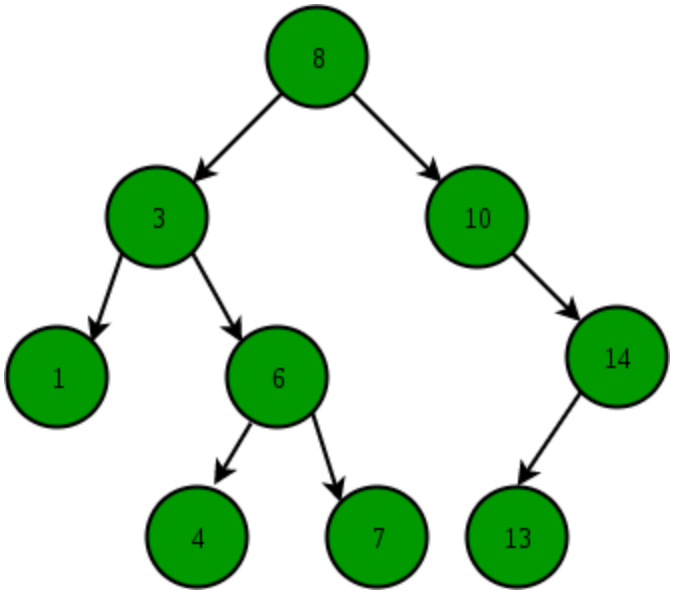
```
                self.__swap(index,parent)
                self.__floatup(parent)
    def __bubbledown(self,index):
        left=index*2
        right=index*2+1
        largest=index
        if len(self.heap)>left and self.heap[largest]<self.heap[left]:
            largest=left
        if len(self.heap)>right and self.heap[largest]<self.heap[right]:
            largest=right
        if largest!=index:
            self.__swap(index,largest)
            self.__bubbledown(largest)
    def __str__(self):
        return str(self.heap)

my_heap=MaxHeap()
my_heap.push(4)
my_heap.push(5)
my_heap.push(6)
print(my_heap)
print(my_heap.peek())
my_heap.pop()
print(my_heap)
```

```
[0, 6, 4, 5]
6
[0, 5, 4]
```

Binary Search Tree

The following is the definition of Binary Search Tree(BST) according to Wikipedia
Binary Search Tree is a node-based binary tree data structure which has the following properties:

- The left subtree of a node contains only nodes with keys lesser than the node's key.
- The right subtree of a node contains only nodes with keys greater than the node's key.
- The left and right subtree each must also be a binary search tree. There must be no duplicate nodes.



The above properties of Binary Search Tree provides an ordering among keys so that the operations like search, minimum and maximum can be done fast. If there is no ordering, then we may have to compare every key to search for a given key.

More on this check out : https://www.geeksforgeeks.org/binary-search-tree-set-1-search-and-insertion/?ref=lbp

```
#binary search tree
class Tree:
    def __init__(self,data,left=None,right=None):
        self.data=data
        self.left=left
        self.right=right
    def insert(self,data):
        if self.data==data:
            return False
        elif self.data>data:
            if self.left is not None:
                self.left.insert(data)
            else:
                self.left=Tree(data)
        elif self.data<data:
            if self.right is not None:
                self.right.insert(data)
            else:
                self.right=Tree(data)
    def find(self,data):
        if self.data==data:
            return True
        else:
            if self.data>data:
                if self.left is None:
                    return False
                else:
                    return self.left.find(data)
            elif self.data<data:
                if self.right is None:
                    return False
                else:
                    return self.right.find(data)
    def get_size(self):
```

```
    def get_size(self):
        if self.left is not None and self.right is not None:
            return 1 +self.left.get_size()+self.right.get_size()
        elif self.left is not None:
            return 1+ self.left.get_size()
        elif self.right is not None:
            return 1+ self.right.get_size()
        else:
            return 1
    def find_max(self):
        if self.right is None:
            return self.data
        return self.right.find_max()
    def find_min(self):
        if self.left is None:
            return self.data
        return self.left.find_min()
    def delete(self,data):
```

```
    def delete(self,data):
        if self.data>data:
            if self.left is not None:
                self.left=self.left.delete(data)
            else:
                return None
        elif self.data<data:
            if self.right is not None:
                self.right=self.right.delete(data)
            else:
                return None
        else:
            if self.left is None and self.right is None:
                return None
            elif self.left is None:
                return self.right
            elif self.right is None:
                return self.right
            min_val=self.right.find_min()
            self.data=min_val
            self.right=self.right.delete(min_val)
        return self
    def preorder(self):
        if self is not None:
            print(self.data,end=' ')
            if self.left is not None:
                self.left.preorder()
            if self.right is not None:
                self.right.preorder()
    def inorder(self):
```

```
    def inorder(self):
        if self is not None:
            if self.left is not None:
                self.left.inorder()
            print(self.data,end=' ')
            if self.right is not None:
                self.right.inorder()
t=Tree(99)
t.insert(4)
t.insert(5)
t.insert(7)
t.insert(8)
t.insert(23)
t.delete(23)
t.delete(5)
t.inorder()
```

```
4 7 8 99
```

Lets have a look at some sorting algorithms

These assorting algorithms works by sorting values stored within the above data structures

```
#Algorithms

#bubble sort

def bubble_sort(arr):
    size=len(arr)
    for i in range(size-1):
        swapped=False
        for j in range(size-1-i):
            if arr[j]>arr[j+1]:
                arr[j],arr[j+1]=arr[j+1],arr[j]
                swapped=True
        if not swapped:
            break
```

```
#selection sort
def selection_sort(arr):
    size=len(arr)
    for i in range(size-1):
        min_index=i
        for j in range(min_index+1,size):
            if arr[j]<arr[min_index]:
                min_index=j
        if i!=min_index:
            arr[i],arr[min_index]=arr[min_index],arr[i]
```

```
#insertion sort
def insertion_sort(arr):
    for i in range(1,len(arr)):
        anchor=arr[i]
        j=i-1
        while j>=0 and anchor<arr[j]:
            arr[j+1]=arr[j]
            j-=1
        arr[j+1]=anchor
```

```
#shell sort
def shell_sort(arr):
    size=len(arr)
    gap=size//2
    while gap>0:
        for i in range(gap,size):
            anchor=arr[i]
            j=i
            while j>=gap and anchor < arr[j-gap]:
                arr[j]=arr[j-gap]
                j-=gap
            arr[j]=anchor
        gap=gap//2
```

```python
#quick sort
def swap(a,b,arr):
    if a!=b:
        arr[a],arr[b]=arr[b],arr[a]
def partition(elements,start,end):
    pivot_index=start
    pivot_element=elements[pivot_index]
    while start<end:
        while start <len(elements) and elements[start]<=pivot_element:
            start+=1
        while elements[end]>pivot_element:
            end-=1
        if start<end:
            swap(start,end,elements)
    swap(pivot_index,end,elements)
    return end
def quick_sort(elements,start,end):
    if start<end:
        pi=partition(elements,start,end)
        quick_sort(elements,start,pi-1)
        quick_sort(elements,pi+1,end)
        partition(elements,start,end)
```

```python
#merge sort
def merge_sort(arr):
    if len(arr)<=1:
        return
    mid=len(arr)//2
    left=arr[:mid]
    right=arr[mid:]

    merge_sort(left)
    merge_sort(right)

    merge_two_sorted_lists(left,right,arr)

def merge_two_sorted_lists(a,b,arr):
    len_a=len(a)
    len_b=len(b)

    i=j=k=0
    while i<len_a and j<len_b:
        if a[i]<=b[j]:
            arr[k]=a[i]
            i+=1
        elif a[i]>b[j]:
            arr[k]=b[j]
            j+=1
        k+=1
    while i<len_a:
        arr[k]=a[i]
        i+=1
        k+=1
    while j<len_b:
        arr[k]=b[j]
        j+=1
        k+=1
```

```python
#search methods

#linear search
def linear_search(numbers_list,number_to_find):
    for index,element in enumerate(numbers_list):
        if element ==number_to_find:
            return index
    return None
```

```python
#binary search
def binary_search_recursive(numbers_list,number_to_find,left_index,right_index):
    if right_index<left_index:
        return -1
    mid_index=(left_index+right_index)//2
    if mid_index>=len(numbers_list) or mid_index<0:
        return -1
    mid_number=numbers_list[mid_index]
    if mid_number==number_to_find:
        return mid_index
    elif mid_number<number_to_find:
        left_index=mid_index+1
    elif mid_number>number_to_find:
        right_index=mid_index-1
    return binary_search_recursive(numbers_list,number_to_find,left_index,right_index)
```

```python
tests_scores=[[7,54,65,87,21,4,6,8,68],
              [2,3,4,5],
              [6],
              [7,6,5,4],
              []]
for elements in tests_scores[0]:
    print(elements)
    number_to_find=21
    index=binary_search_recursive(elements,number_to_find,0,len(elements)-1)
    print(index)
```

```
7
```

- **Argument's operators (*args and **kwargs)**

-*args stands for arguments and **kwargs represents key word arguments

-These are basically used within a function to help in keying in more than one parameter

unpacking

```
a,b,c='4 5 6'.split()
print(a,b,c)
```

```
4 5 6
```

```
print(a+b+c)
```

```
456
```

```
my_list=[5,6,7]
a,b,c=my_list
print(a)
print(b)
print(c)
```

```
5
6
7
```

*args for functions

```
def pack_it(*args):#args put the defined variables into a tuple
    print(args)
    print(type(args))

x='forest';y='hill';z='high'
pack_it(x,y,z)
```

```
('forest', 'hill', 'high')
<class 'tuple'>
```

```
def unpack_it(x,y):
    print(x)
    print(y)

args=['cotias','macdonalds']
unpack_it(*args)
```

```
cotias
macdonalds
```

** kwargs for functions

```
def func(**winners):# kwargs or (key word arguments) puts the defined variables into a dictionary
    print(winners)
    print(winners['a'])
    print(type(winners))
func(a='eddie',b='edsoon',c='edmaray')
```

```
{'a': 'eddie', 'b': 'edsoon', 'c': 'edmaray'}
eddie
<class 'dict'>
```

```
def unpackfunc(a,b,c):
    print(a)
winners={'a':'eddie','b':'edsoon','c':'edmaray'}
unpackfunc(**winners)
```

```
eddie
```

```
def func(**kwargs):
    print(kwargs)
    for key,value in kwargs.items():
        print(key)
        print(value)

func(a='arry',b='teries',c='yotki')
```

```
{'a': 'arry', 'b': 'teries', 'c': 'yotki'}
a
arry
b
teries
c
yotki
```

```
func(**{'a': 'arry', 'b': 'teries', 'c': 'yotki'})
```

```
{'a': 'arry', 'b': 'teries', 'c': 'yotki'}
a
arry
b
teries
c
yotki
```

```
#using both *args and **kwargs in the same function
def func(*args,**kwargs):
    print(args)
    for item in args:
        print(item)
    print(kwargs)
    for key,value in kwargs.items():
        print(f'key -> {key}')
        print(f'value -> {value}')
#func(10,20,30,[10,20])
func(10,20,30,[10,20],name='murray',age='35')
```

```
(10, 20, 30, [10, 20])
10
20
30
[10, 20]
{'name': 'murray', 'age': '35'}
key -> name
value -> murray
key -> age
value -> 35
```

```
func(*[10,20,30],**{'name':'murray','age':'35','height':20})
```

```
(10, 20, 30)
10
20
30
{'name': 'murray', 'age': '35', 'height': 20}
key -> name
value -> murray
key -> age
value -> 35
key -> height
value -> 20
```

```
#note that *args are kinda like pyhton list,returns list(tuples) while **kwargs returns dictionaries
```

```
#having normal argumenrs,args and kwargs in the same  function
def food(title,*args,**kwargs):
    print(title)
    for arg in args:
        print(arg)
    for key,value in kwargs.items():
        print(f'key-> {key }',f': value->{value}')

food('rice',[2,3,4,],'34',4,5, name='arrisc',age=43,likes='hang out')
```

```
rice
[2, 3, 4]
34
4
5
key-> name : value->arrisc
key-> age : value->43
key-> likes : value->hang out
```

```
#exercises
def sum_numbers(num1,num2,*list1):
    return num1+num2
print(sum_numbers(3,4))
```

```
7
```

```
from functools import reduce
def sum_numbers(num1,num2,*list1):
    print(num1+num2)
    for item in list1:# although this for loop is not executed i.e not necesary
        reduced=reduce(lambda x,y:x+y,*list1)
        print(reduced)


list1=[1,2,3,4]
#sum_numbers(2,3)
sum_numbers(2,3,list1)
```

```
5
10
```

```
from functools import reduce
def product_numbers(num1,num2,*list1):
    print(num1*num2)
    #for item in list:# although this for loop is not executed i.e not necesary
    reduced=reduce(lambda x,y:x*y,*list1)
    print(reduced)


list1=[1,2,3,4]
#sum_numbers(2,3)
product_numbers(2,3,list1)
```

```
6
24
```

```
from functools import reduce
def product_numbers(num1,num2,*list1):
    multiplied=(num1*num2)
    print(multiplied)
    #for item in list:# although this for loop is not executed i.e not necesary
    reduced=reduce(lambda x,y:x*y,*list1)
    print(reduced)


list1=[1,2,3,4]
#sum_numbers(2,3)
product_numbers(2,3,list1)
```

```
6
24
```

```
#some recursion exercises for fun for finding sum of certain range of numbers; haha
def find_sum(n):
    if n==1:
        return 1
    return n +find_sum(n-1)
find_sum(5)
```

```
15
```

➢ PIP and Environments

```
We have several Intergrated development environments
-These are used to run the code
-We have Anaconda environment where you can open the jupyter  notebooks to run the code
-Other environment includes:
    PyCharm
    Spynder
    Google colab

PIP
-This is a command in which we use to install packages in our environment
-For example:

    !pip install tensorflow
    !pip install keras
```

➢ Meta classes

These are useful to store a class within another class

metaclasses

```
#metaclass 1.1
class Meta(type):
    def __new__(self,name,age,likes):
        print(likes)
        return type(name,age,likes)
```

```
class Python(metaclass=Meta):
    x=10
    y=40

a=Python()
a.x
```

```
{'__module__': '__main__', '__qualname__': 'Python', 'x': 10, 'y': 40}

10
```

```
#metaclass 1.2
class Meta(type):
    def  __new__(self,class_name,bases,attrs):# this new method modifies the attributes of an object .tbla
        print(attrs)
        a={}
        for name,val in attrs.items():
            if name.startswith('__'):

                a[name]=val  # this is how we add items into a dictionary
            else:
                a[name.upper()]=val
        print(a)
        return type(class_name,bases,attrs)

class Dog(metaclass=Meta):
    x=5
    y=10
    def hello(self):
        print('hello')

d=Dog()
print(d.hello())
```

```
{'__module__': '__main__', '__qualname__': 'Dog', 'x': 5, 'y': 10, 'hello': <function Dog.hello at 0x0000029B82237D30>}
{'__module__': '__main__', '__qualname__': 'Dog', 'X': 5, 'Y': 10, 'HELLO': <function Dog.hello at 0x0000029B82237D30>}
hello
None
```

NOTE:

We have now gone through all the most pivotal concepts in python which will help us out in understanding data science

Now to data science,there are important key concepts needed to get started in this field

First,

- Microsoft Excel

This is very important tool to understand as a data scientist since most a times you will be storing datasets in excel format,

Some keys concepts to note about excel is:

-How to create and edit an excel file

-How to do basic calculations e.g sum , average .. using excel

-How to filter a dataset

-How to save an excel file

Second,

- Mathematics and Statistics

These two fields are very wide to study at a go, but there are the most pivotal concepts you need to understand;

-How to find average of any given data

-How to find standard deviation of any data

-How to find variance of a data

-Some basic calculus on how to integrate and differentiate equations-this is useful in deep learning

-How to do partial derivatives

Next,

We now look at the visualization tools

- **Visualization tools**

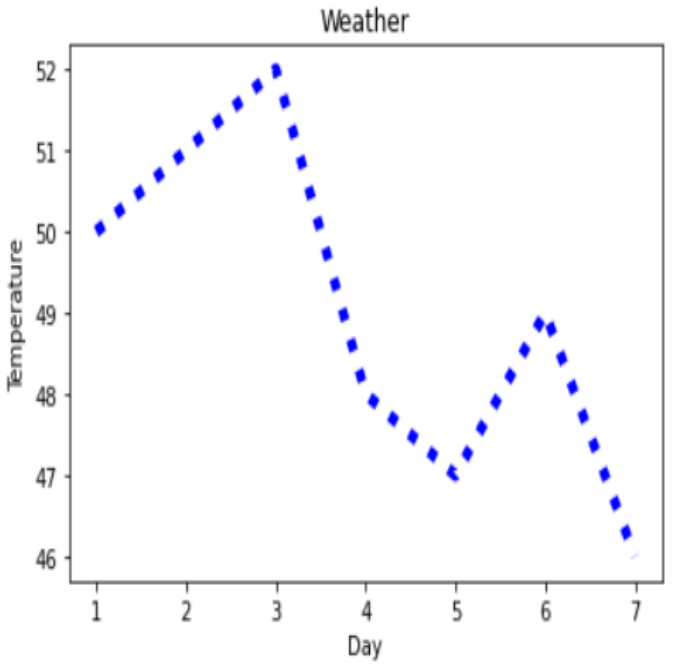
Some of the visualization tools we will be looking at includes matplotlib, seaborn

These tools are so important since they will help you visualize the data for any analysis, enables you to identify trends which can not be seen from statistical viewpoint

Let's get started:

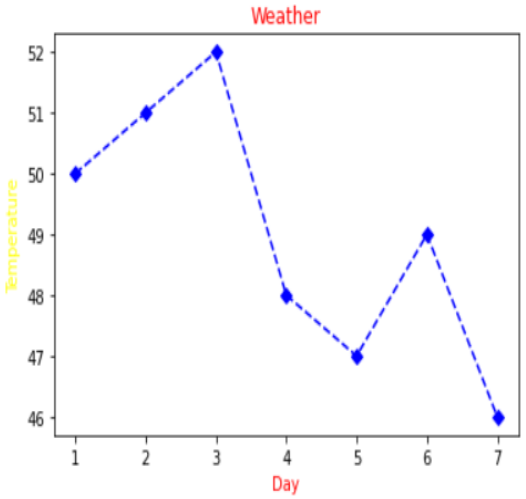
```
#weather
import matplotlib.pyplot as plt
%matplotlib inline
x=[1,2,3,4,5,6,7]
y=[50,51,52,48,47,49,46]
plt.xlabel('Day')
plt.ylabel('Temperature')
plt.title('Weather')
plt.plot(x,y,color='blue',linewidth=5,linestyle='dotted')
```

```
[<matplotlib.lines.Line2D at 0x1d76505ff10>]
```



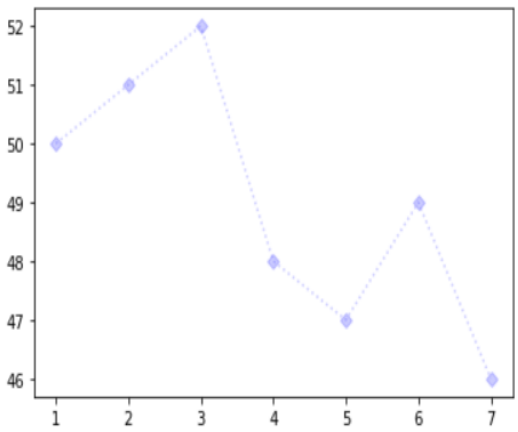
```
import matplotlib.pyplot as plt
%matplotlib inline
plt.xlabel('Day',color='red')
plt.ylabel('Temperature',color='yellow')
plt.title('Weather',color='red')
x=[1,2,3,4,5,6,7]
y=[50,51,52,48,47,49,46]
plt.plot(x,y,'bD',linestyle='dashed')#(bD refers to color blue and the line being Dashed=can also be denoted by color and linesty
```

```
[<matplotlib.lines.Line2D at 0x1d765154b20>]
```




```
plt.plot(x,y,color='blue',marker='D',linestyle='dotted',alpha=0.2)#alpha have a range from 0-1(shows brightness of the curve)
```

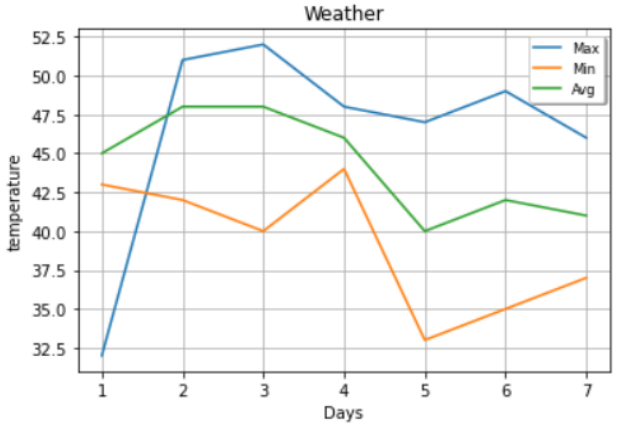
```
[<matplotlib.lines.Line2D at 0x1d7651b9880>]
```




```
#tutorial 3
import matplotlib.pyplot as plt
%matplotlib inline
```

```
days=[1,2,3,4,5,6,7]
max_t=[32,51,52,48,47,49,46]
min_t=[43,42,40,44,33,35,37]
avg_t=[45,48,48,46,40,42,41]
```

```
plt.xlabel('Days')
plt.ylabel('temperature')
plt.title('Weather')
plt.plot(days,max_t,label='Max')
plt.plot(days,min_t,label='Min')
plt.plot(days,avg_t,label='Avg')


plt.legend(loc='best',shadow=True,fontsize='small')#can be upper right ,lower left,lower right,upper left,best

plt.grid()#inserting a grid in chart
```




```
#tutorial 4 Bar chart
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline
```

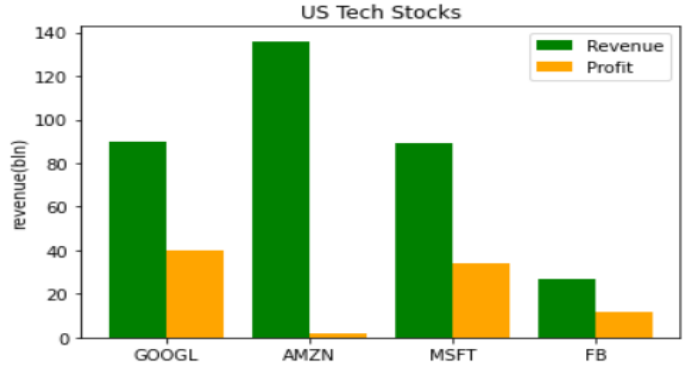
```
company=['GOOGL','AMZN','MSFT','FB']
revenue=[90,136,89,27]
profit=[40,2,34,12]
```

```
xpos=np.arange(len(company))
xpos
```

```
array([0, 1, 2, 3])
```

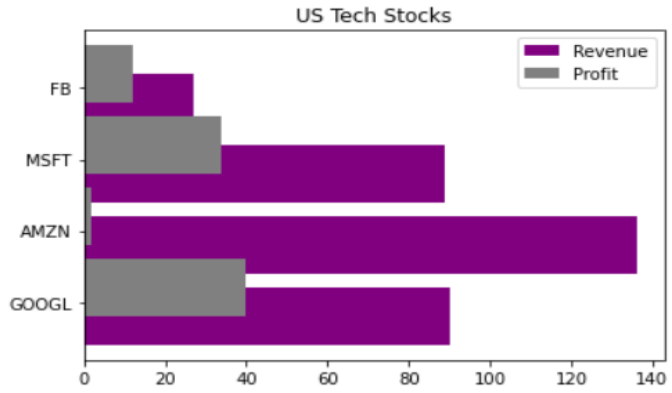
```
plt.xticks(xpos,company)
plt.ylabel('revenue(bln)')
plt.title('US Tech Stocks')
plt.bar(xpos-0.2,revenue,color ='green',label='Revenue',width=0.4)
plt.bar(xpos+0.2,profit,label='Profit',color='orange',width=0.4)

plt.legend(loc='best')
```

```
<matplotlib.legend.Legend at 0x1af86f1baf0>
```



```
#horizontal graph
plt.yticks(xpos,company)
plt.title('US Tech Stocks')

plt.barh(xpos-0.2,revenue,color ='purple',label='Revenue')
plt.barh(xpos+0.2,profit,label='Profit',color='grey')

plt.legend(loc='best')
```

```
<matplotlib.legend.Legend at 0x1af87015880>
```

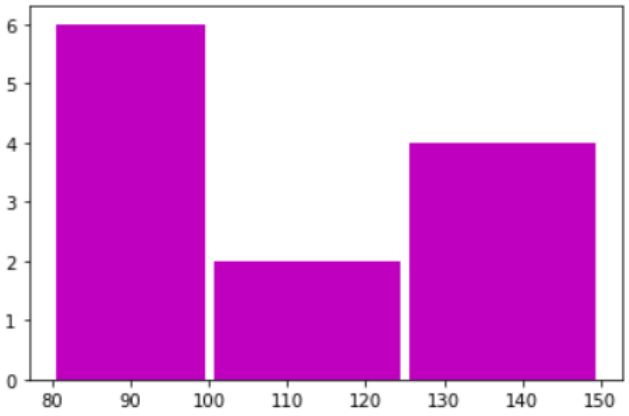



```
#tutorial 5(Histograms)
import matplotlib.pyplot as plt
%matplotlib inline
```

```
blood_sugar=[113,85,90,150,149,88,93,115,135,80,77,82,129]
```

```
plt.hist(blood_sugar,color='m',bins=[80,100,125,150],rwidth=0.95)
#by default bins value=10, rwidth is the relative width(0-1), also bins refers to range
```

```
(array([6., 2., 4.]),
 array([ 80, 100, 125, 150]),
 <BarContainer object of 3 artists>)
```

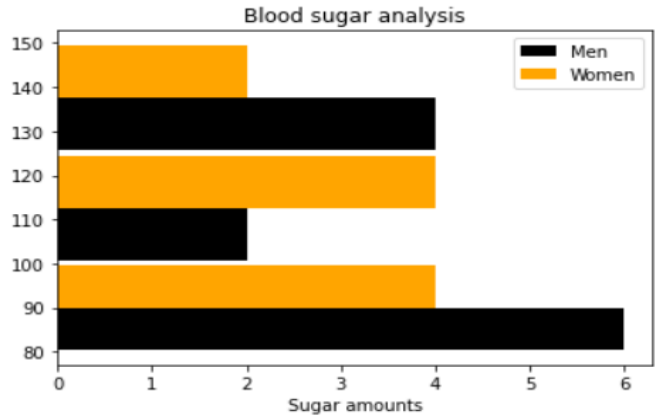


```
import matplotlib.pyplot as plt
%matplotlib inline
```

```
blood_sugar_men=[113,85,90,150,149,88,93,115,135,80,77,82,129]
blood_sugar_women=[67,98,89,120,133,150,84,69,89,79,120,112,100]
```

```
plt.hist([blood_sugar_men,blood_sugar_women],bins=[80,100,125,150],rwidth=0.95,color=['k','orange'],
        label=['Men','Women'],orientation='horizontal')
#plt.ylbael('Sugar range')
plt.xlabel('Sugar amounts')
plt.title('Blood sugar analysis')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x1d7653a7130>
```

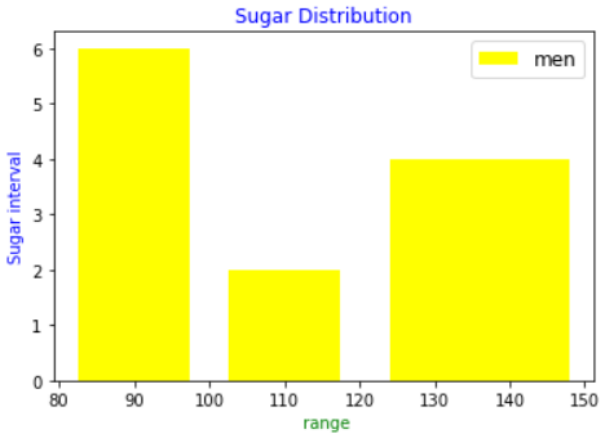



```
import matplotlib.pyplot as plt
%matplotlib inline
```

```
blood_sugar_men=[113,85,90,150,149,88,93,115,135,80,77,82,129]
```

```
plt.hist(blood_sugar_men,color='yellow',rwidth=0.75,bins=[80,100,120,152],orientation='vertical',label='men')
plt.ylabel('Sugar interval',color='blue')
plt.xlabel(' range',color='green')
plt.title('Sugar Distribution',color='blue')
plt.legend(loc='upper right',fontsize='large')
```

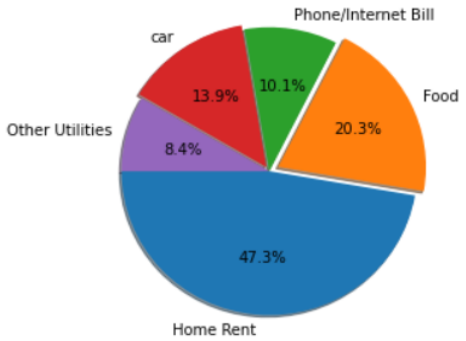
```
<matplotlib.legend.Legend at 0x1d765398bb0>
```



```
#Learning Pie Chart
import matplotlib.pyplot as plt
%matplotlib inline
```

```
exp_vals=[1400,600,300,410,250]
exp_labels=['Home Rent','Food','Phone/Internet Bill','car','Other Utilities']
```

```
plt.pie(exp_vals,labels=exp_labels,radius=1.5,autopct='%0.1f%%',shadow=True,explode=[0,0.1,0,0.05,0],startangle=180)
plt.axis('equal')

plt.show()
```

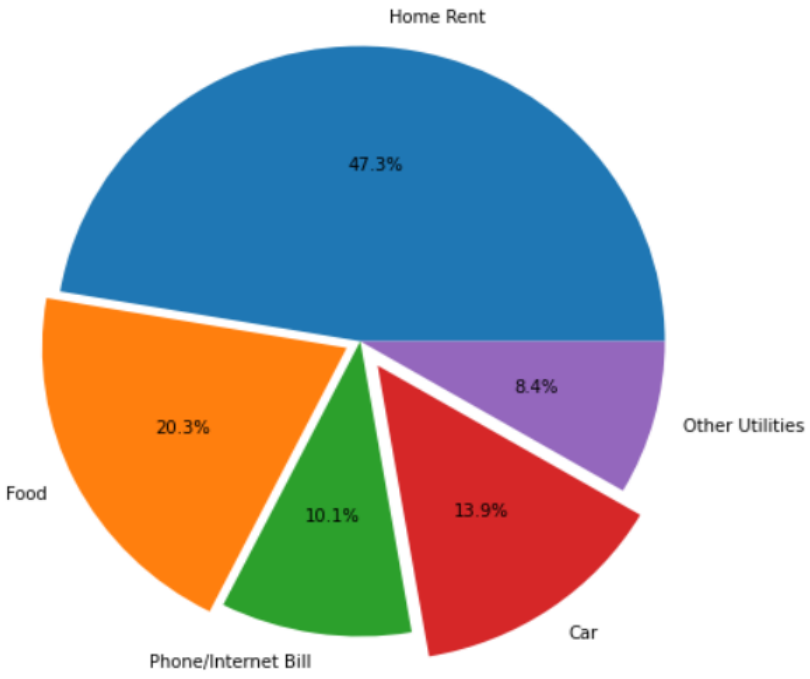


Type *Markdown* and LaTeX: α^2

```
import matplotlib.pyplot as plt
%matplotlib inline
```

```
exp_vals=[1400,600,300,410,250]
exp_lables=['Home Rent','Food','Phone/Internet Bill','Car','Other Utilities']
```

```
plt.axis('equal')
plt.pie(exp_vals,labels=exp_lables,radius=2,autopct='%0.1f%%',explode=[0,0.1,0,0.2,0])
plt.savefig('Piechart.png',bbox_inches='tight',pad_inches=2,transparent=True)
```



- Clean UP Libraries

As a data scientist, most of the time is spend in cleaning up the data, preparing data for model creation and deployment

There are tools which helps in all the above processes:

-Pandas

-NumPy

NumPy

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays

```
import numpy  as np
import time
import sys
```

```
size=1000000
l1=range(size)
l2=range(size)
```

```
a1=np.arange(size)
a2=np.arange(size)
```

```
start=time.time()
result=[(x+y) for x,y in zip(l1,l2)]
print('python list took ',(time.time()-start)*1000)
```

```
python list took  288.23137283325195
```

```
start=time.time()
result=a1+a2
print('numpy took ', (time.time()-start)*1000)
```

```
numpy took  51.01132392883301
```

```
a1=np.array([1,2,3])
a2=np.array([4,5,6])
a1+a2
```

```
array([5, 7, 9])
```

```
a2-a1
```

```
array([3, 3, 3])
```

```
#numpy basic operations
import numpy as np
a=np.array([5,6,9])# this is one dimensional array
#for two dimensional numpy array
a=np.array([[1,2],[3,4],[5,6]])
a.ndim
```

```
2
```

```
a=np.array([5,6,9])
a.ndim
```

```
1
```

```
a.itemsize
```

```
4
```

```
a.dtype
```

```
dtype('int32')
```

```
a=np.array([[1,2],[3,4],[5,6]] ,dtype=np.float64)
a.itemsize
```

```
8
```

```
a
```

```
array([[1., 2.],
       [3., 4.],
       [5., 6.]])
```

```
a.size
```

```
6
```

```
a.shape
```

```
(3, 2)
```

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

```
a
```

```
array([[1.+0.j, 2.+0.j],
       [3.+0.j, 4.+0.j],
       [5.+0.j, 6.+0.j]])
```

```
np.zeros((3,4))
```

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

```
np.ones((3,4))
```

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

```
l=range(5)
l
```

```
range(0, 5)
```

```
np.arange(1,5)
```

```
array([1, 2, 3, 4])
```

```
np.arange(1,5,2)
```

```
array([1, 3])
```

```
np.linspace(1,5,10)
```

```
array([1.        , 1.44444444, 1.88888889, 2.33333333, 2.77777778,
       3.22222222, 3.66666667, 4.11111111, 4.55555556, 5.        ])
```

```
np.linspace(1,5,20)
```

```
array([1.        , 1.21052632, 1.42105263, 1.63157895, 1.84210526,
       2.05263158, 2.26315789, 2.47368421, 2.68421053, 2.89473684,
       3.10526316, 3.31578947, 3.52631579, 3.73684211, 3.94736842,
       4.15789474, 4.36842105, 4.57894737, 4.78947368, 5.        ])
```

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

```
array([[1, 2],
       [3, 4],
       [5, 6]])
```

```
a.shape
```

```
(3, 2)
```

```
a.reshape(2,3)#2 rows ,3 colums
```

```
array([[1, 2, 3],
       [4, 5, 6]])
```

```
a.reshape(6,1)
```

```
array([[1],
       [2],
       [3],
       [4],
       [5],
       [6]])
```

```
a.ravel()#makes the array one dimensional
```

```
array([1, 2, 3, 4, 5, 6])
```

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

```
array([[1, 2],
       [3, 4],
       [5, 6]])
```

```
a.min()
```

```
1
```

```
a.max()
```

```
6
```

```
a.sum()
```

```
21
```

```
a.sum(axis=0)#axis =0  represents the columns whilest axis=1 represents rows
```

```
array([ 9, 12])
```

```
a.sum(axis=1)
```

```
array([ 3,  7, 11])
```

```
np.sqrt(a)
```

```
array([[1.        , 1.41421356],
       [1.73205081, 2.        ],
       [2.23606798, 2.44948974]])
```

```
np.std(a)#standard deviation
```

```
1.707825127659933
```

```
a=np.array([[1,1],
          [0,1]])
```

```
a=np.array([[1,2],[3,4]])
b=np.array([[5,6],[7,8]])
```

```
a
```

```
array([[1, 2],
       [3, 4]])
```

```
b
```

```
array([[5, 6],
       [7, 8]])
```

```
a+b
```

```
array([[ 6,  8],
       [10, 12]])
```

```
a/b
```

```
array([[0.2       , 0.33333333],
       [0.42857143, 0.5       ]])
```

```
a.dot(b)#matrix multiplication
```

```
array([[19, 22],
       [43, 50]])
```

```
#numpy slicing
a=np.array([6,7,8])
a[0:2]
```

```
array([6, 7])
```

```
a[-1]
```

```
8
```

```
a=np.array([[6,7,8],[1,2,3],[9,3,2]])
a
```

```
array([[6, 7, 8],
       [1, 2, 3],
       [9, 3, 2]])
```

```
a[1,2] #rows and columns
```

```
3
```

```
a[0:2,2] #means 2rd column 0 to 2 element
```

```
array([8, 3])
```

```
a[-1]
```

```
array([9, 3, 2])
```

```
a[-1,0:2]#means last row 0 to 2 element
```

```
array([9, 3])
```

```
a[:,1:3]
```

```
array([[7, 8],
       [2, 3],
       [3, 2]])
```

```
a
```

```
array([[6, 7, 8],
       [1, 2, 3],
       [9, 3, 2]])
```

```
for row in a:
    print(row)
```

```
[6 7 8]
[1 2 3]
[9 3 2]
```

```
for cell in a.flat:
    print(cell)
```

```
6
7
8
1
2
3
9
3
2
```

```
a=np.arange(6).reshape(3,2)
b=np.arange(6,12).reshape(3,2)
a
```

```
array([[0, 1],
       [2, 3],
       [4, 5]])
```

```
b
```

```
array([[ 6,  7],
       [ 8,  9],
       [10, 11]])
```

```
np.vstack((a,b))#vertical stacking
```

```
array([[ 0,  1],
       [ 2,  3],
       [ 4,  5],
       [ 6,  7],
       [ 8,  9],
       [10, 11]])
```

```
np.hstack((a,b))#horizontal stacking
```

```
array([[ 0,  1,  6,  7],
       [ 2,  3,  8,  9],
       [ 4,  5, 10, 11]])
```

```
a=np.arange(30).reshape(2,15)
a
```

```
array([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29]])
```

```
np.hsplit(a,3)
```

```
[array([[ 0,  1,  2,  3,  4],
        [15, 16, 17, 18, 19]]),
 array([[ 5,  6,  7,  8,  9],
        [20, 21, 22, 23, 24]]),
 array([[10, 11, 12, 13, 14],
        [25, 26, 27, 28, 29]])]
```

```
result=np.hsplit(a,3)
result[0]
```

```
array([[ 0,  1,  2,  3,  4],
       [15, 16, 17, 18, 19]])
```

```
result[1]
```

```
array([[ 5,  6,  7,  8,  9],
       [20, 21, 22, 23, 24]])
```

```
result[2]
```

```
array([[10, 11, 12, 13, 14],
       [25, 26, 27, 28, 29]])
```

```
a=np.arange(12).reshape(3,4)
a
```

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

```
b=a>4
b
```

```
array([[False, False, False, False],
       [False,  True,  True,  True],
       [ True,  True,  True,  True]])
```

```
a[b]
```

```
array([ 5,  6,  7,  8,  9, 10, 11])
```

```
a[b]=-1
a
```

```
array([[ 0,  1,  2,  3],
       [ 4, -1, -1, -1],
       [-1, -1, -1, -1]])
```

```
import numpy as np
a=np.arange(12).reshape(3,4)
a
```

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

```
for row in a:
    for cell in row:
        print(cell)
```

```
0
1
2
3
4
5
6
7
8
9
10
11
```

```
#or in other terms we can stil have
for cell in a.flatten():
    print(cell)
```

```
0
1
2
3
4
5
6
7
8
9
10
11
```

```
for x in np.nditer(a,order='f',flags=['external_loop']):#or order 'c'
    print(x)
```

```
[0 4 8]
[1 5 9]
[ 2  6 10]
[ 3  7 11]
```

```
for x in np.nditer(a,op_flags=['readwrite']):
    x[...]=x*x
a
```

```
array([[  0,   1,   4,   9],
       [ 16,  25,  36,  49],
       [ 64,  81, 100, 121]])
```

```
b=np.arange(3,15,4).reshape(3,1)
b
```

```
array([[ 3],
       [ 7],
       [11]])
```

```
for x,y in np.nditer([a,b]):
    print(x,y)
```

```
0 3
1 3
4 3
9 3
16 7
25 7
36 7
49 7
64 11
```

Pandas

pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series.

It is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language

```
How to open files in python
```

```
import pandas as pd
import os
```

```
pwd
```

```
'C:\\Users\\sojore\\Documents\\pyhthon s'
```

```
f=open('C:\\\\Users\\sojore\\Documents\\testtt','w')
f
```

```
<_io.TextIOWrapper name='C:\\\\Users\\sojore\\Documents\\testtt' mode='w' encoding='cp1252'>
```

```
print(f)
```

```
<_io.TextIOWrapper name='C:\\\\Users\\sojore\\Documents\\testtt' mode='w' encoding='cp1252'>
```

```
f=open('C:\\Users\\sojore\\Documents\\testtt','w')
f.write('i love python')
f.close()
```

```
f=open('C:\\Users\\sojore\\Documents\\testtt','a')##a is for append
f.write('i love javascript')
f.close()
```

```
f=open('C:\\Users\\sojore\\Documents\\testtt','r')##r is for append,,how to read a file
print(f.read())
f.close()
```

```
i love pythoni love javascript
```

```
f=open('C:\\Users\\sojore\\Documents\\testtt','w+')##this does both writting and reading same as r+
f.write('i love python')#this will also create a new file if its not already in existence
f.close()
```

```
import pandas as pd
#dataframe df
weather_data={
     'day':['1/1/2017','1/2/2017','1/3/2017','1/4/2017','1/5/2017','1/6/2017'],
     'temperature': [32,35,28,24,32,31],
     'windspeed':[6,7,2,7,4,2],
     'event':['Rain','Sunny','Snow','Snow','Rain','Sunny']
 }

df=pd.DataFrame(weather_data)
print(df)
```

```
        day  temperature  windspeed  event
0  1/1/2017           32          6   Rain
1  1/2/2017           35          7  Sunny
2  1/3/2017           28          2   Snow
3  1/4/2017           24          7   Snow
4  1/5/2017           32          4   Rain
5  1/6/2017           31          2  Sunny
```

```
df
```

	day	temperature	windspeed	event
0	1/1/2017	32	6	Rain
1	1/2/2017	35	7	Sunny
2	1/3/2017	28	2	Snow
3	1/4/2017	24	7	Snow
4	1/5/2017	32	4	Rain
5	1/6/2017	31	2	Sunny

```
rows,columns=df.shape
```

```
rows
```

```
6
```

```
df.head(2)#this will print only the first starting 2 rows
```

	day	temperature	windspeed	event
0	1/1/2017	32	6	Rain
1	1/2/2017	35	7	Sunny

```
df.tail()
```

	day	temperature	windspeed	event
1	1/2/2017	35	7	Sunny
2	1/3/2017	28	2	Snow
3	1/4/2017	24	7	Snow
4	1/5/2017	32	4	Rain
5	1/6/2017	31	2	Sunny

```
df.tail(3)# this will print only 3 ending rows
```

	day	temperature	windspeed	event
3	1/4/2017	24	7	Snow
4	1/5/2017	32	4	Rain
5	1/6/2017	31	2	Sunny

```
df[2:5]# this will slice rows from 2rd row to 4th row but not including the 5th row
```

	day	temperature	windspeed	event
2	1/3/2017	28	2	Snow
3	1/4/2017	24	7	Snow
4	1/5/2017	32	4	Rain

```
df[:]# printing everything
```

	day	temperature	windspeed	event
0	1/1/2017	32	6	Rain
1	1/2/2017	35	7	Sunny
2	1/3/2017	28	2	Snow
3	1/4/2017	24	7	Snow
4	1/5/2017	32	4	Rain
5	1/6/2017	31	2	Sunny

```
#now to printing columns
df.columns
```

```
Index(['day', 'temperature', 'windspeed', 'event'], dtype='object')
```

```
df.day
```

```
0    1/1/2017
1    1/2/2017
2    1/3/2017
3    1/4/2017
4    1/5/2017
5    1/6/2017
Name: day, dtype: object
```

```
df.event
```

```
0     Rain
1    Sunny
2     Snow
3     Snow
4     Rain
5    Sunny
Name: event, dtype: object
```

```
df['event']#both works the same as above
```

```
0     Rain
1    Sunny
2     Snow
3     Snow
4     Rain
5    Sunny
Name: event, dtype: object
```

```
type(df['event'])
```

```
pandas.core.series.Series
```

```
df[['event','day']] # this is when you only wanna print the event column and day column only
```

	event	day
0	Rain	1/1/2017
1	Sunny	1/2/2017
2	Snow	1/3/2017
3	Snow	1/4/2017
4	Rain	1/5/2017
5	Sunny	1/6/2017

```
df
```

	day	temperature	windspeed	event
0	1/1/2017	32	6	Rain
1	1/2/2017	35	7	Sunny
2	1/3/2017	28	2	Snow
3	1/4/2017	24	7	Snow
4	1/5/2017	32	4	Rain
5	1/6/2017	31	2	Sunny

```
df['temperature'].max()# for maximum value
```

```
35
```

```
df['temperature'].mean() # for mean value or average
```

```
30.333333333333332
```

```
df['temperature'].min()# for minimum value
```

```
24
```

```
df['temperature'].std()# for standard deviation
```

```
3.8297084310253524
```

```
df.describe()# this prints quickly the statistics of our data
```

	temperature	windspeed
count	6.000000	6.000000
mean	30.333333	4.666667
std	3.829708	2.338090
min	24.000000	2.000000
25%	28.750000	2.500000
50%	31.500000	5.000000
75%	32.000000	6.750000
max	35.000000	7.000000

```
type(df)
```

```
pandas.core.frame.DataFrame
```

```
type(df.event)
```

```
pandas.core.series.Series
```

```
rows
```

```
6
```

```
df.columns
```

```
Index(['day', 'temperature', 'windspeed', 'event'], dtype='object')
```

```
df[2:5]
```

	day	temperature	windspeed	event
2	1/3/2017	28	2	Snow
3	1/4/2017	24	7	Snow
4	1/5/2017	32	4	Rain

```
import pandas as pd
df=pd.read_excel('C:\\Users\sojore\Documents\weather data.xlsx')
df
```

	day	temperature	windspeed	event
0	2017-01-01	32.0	6.0	Rain
1	2017-04-01	NaN	9.0	Sunny
2	2017-05-01	28.0	NaN	Snow
3	2017-06-01	NaN	7.0	NaN
4	2017-07-01	32.0	NaN	Rain
5	2017-08-01	NaN	NaN	Sunny
6	2017-09-01	NaN	NaN	NaN
7	2017-10-01	34.0	8.0	Cloudy
8	2017-11-01	40.0	12.0	Sunny

```
#this is how you read an excel file
#for example we have that
import pandas as pd
df=pd.read_excel('C:\\Users\sojore\Documents\weather data.xlsx')
df
```

	day	temperature	windspeed	event
0	2017-01-01	32.0	6.0	Rain
1	2017-04-01	NaN	9.0	Sunny
2	2017-05-01	28.0	NaN	Snow
3	2017-06-01	NaN	7.0	NaN
4	2017-07-01	32.0	NaN	Rain

```
df=pd.read_excel('C:\\Users\sojore\Documents\weather data.xlsx',parse_dates=['day'])
df
```

	day	temperature	windspeed	event
0	2017-01-01	32.0	6.0	Rain
1	2017-04-01	NaN	9.0	Sunny
2	2017-05-01	28.0	NaN	Snow
3	2017-06-01	NaN	7.0	NaN
4	2017-07-01	32.0	NaN	Rain
5	2017-08-01	NaN	NaN	Sunny
6	2017-09-01	NaN	NaN	NaN
7	2017-10-01	34.0	8.0	Cloudy
8	2017-11-01	40.0	12.0	Sunny

```
df.set_index('day',inplace=True)#this is when we wanna modify the original dataframe to take day as the first column index
df
```

	temperature	windspeed	event
day			
2017-01-01	32.0	6.0	Rain
2017-04-01	NaN	9.0	Sunny
2017-05-01	28.0	NaN	Snow

```
#we trying to replace these Nan values with valuable meaningful data
#fill Nan
new_df=df.fillna(0)
new_df
```

	temperature	windspeed	event
day			
2017-01-01	32.0	6.0	Rain
2017-04-01	0.0	9.0	Sunny
2017-05-01	28.0	0.0	Snow
2017-06-01	0.0	7.0	0
2017-07-01	32.0	0.0	Rain
2017-08-01	0.0	0.0	Sunny
2017-09-01	0.0	0.0	0
2017-10-01	34.0	8.0	Cloudy
2017-11-01	40.0	12.0	Sunny

```
new_df.describe()
```

	temperature	windspeed
count	9.000000	9.000000
mean	18.444444	4.666667
std	17.770137	4.716991
min	0.000000	0.000000
25%	0.000000	0.000000

```
new_df=df.fillna({'temperature':0,
                  'windspeed':0,
                  'event':'no event'}) #we normally pass in a dict if we wanna new  values for each specific column data
new_df
```

day	temperature	windspeed	event
2017-01-01	32.0	6.0	Rain
2017-04-01	0.0	9.0	Sunny
2017-05-01	28.0	0.0	Snow
2017-06-01	0.0	7.0	no event
2017-07-01	32.0	0.0	Rain
2017-08-01	0.0	0.0	Sunny
2017-09-01	0.0	0.0	no event
2017-10-01	34.0	8.0	Cloudy
2017-11-01	40.0	12.0	Sunny

```
new_df=df.fillna(method='ffill')# ffill or forward fill means that if we have an Nan value the carry forward the previous data
#n fill it in that position
new_df
```

day	temperature	windspeed	event
2017-01-01	32.0	6.0	Rain
2017-04-01	32.0	9.0	Sunny
2017-05-01	28.0	9.0	Snow

```
new_df=df.fillna(method='bfill')#backward fill,filling in the next data to previous Nan slot
new_df
```

day	temperature	windspeed	event
2017-01-01	32.0	6.0	Rain
2017-04-01	28.0	9.0	Sunny
2017-05-01	28.0	7.0	Snow
2017-06-01	32.0	7.0	Rain
2017-07-01	32.0	8.0	Rain
2017-08-01	34.0	8.0	Sunny
2017-09-01	34.0	8.0	Cloudy
2017-10-01	34.0	8.0	Cloudy
2017-11-01	40.0	12.0	Sunny

```
new_df=df.fillna(method='bfill',axis='columns')
new_df
```

day	temperature	windspeed	event
2017-01-01	32	6	Rain
2017-04-01	9	9	Sunny
2017-05-01	28	Snow	Snow
2017-06-01	7	7	NaN
2017-07-01	32	Rain	Rain

```
new_df=df.fillna(method='ffill',limit=1)#this will carry forawrd the value in prev cell one(1) way forward but not 2
new_df
```

day	temperature	windspeed	event
2017-01-01	32.0	6.0	Rain
2017-04-01	32.0	9.0	Sunny
2017-05-01	28.0	9.0	Snow
2017-06-01	28.0	7.0	Snow
2017-07-01	32.0	7.0	Rain
2017-08-01	32.0	NaN	Sunny
2017-09-01	NaN	NaN	Sunny
2017-10-01	34.0	8.0	Cloudy
2017-11-01	40.0	12.0	Sunny

```
new_df=df.interpolate()#this uses linear inerpolation to come with the new values(better guesses)
new_df
```

day	temperature	windspeed	event
2017-01-01	32.000000	6.00	Rain
2017-04-01	30.000000	9.00	Sunny
2017-05-01	28.000000	8.00	Snow
2017-06-01	30.000000	7.00	NaN
2017-07-01	32.000000	7.25	Rain
2017-08-01	32.666667	7.50	Sunny
2017-09-01	33.333333	7.75	NaN

```
new_df=df.interpolate('quadratic')
new_df
```

day	temperature	windspeed	event
2017-01-01	32.000000	6.000000	Rain
2017-04-01	27.176305	9.000000	Sunny
2017-05-01	28.000000	8.188609	Snow
2017-06-01	30.128031	7.000000	NaN
2017-07-01	32.000000	6.111697	Rain
2017-08-01	32.316475	5.464641	Sunny
2017-09-01	31.697769	5.868905	NaN
2017-10-01	34.000000	8.000000	Cloudy
2017-11-01	40.000000	12.000000	Sunny

```
new_df=df.interpolate('time')# or have method='time'
new_df
```

day	temperature	windspeed	event
2017-01-01	32.000000	6.000000	Rain
2017-04-01	29.000000	9.000000	Sunny
2017-05-01	28.000000	8.016393	Snow
2017-06-01	30.032787	7.000000	NaN

```
new_df=df.dropna()#this drops all rows that had na value in them
new_df
```

day	temperature	windspeed	event
2017-01-01	32.0	6.0	Rain
2017-10-01	34.0	8.0	Cloudy
2017-11-01	40.0	12.0	Sunny

```
new_df=df.dropna(how='all')
new_df
```

day	temperature	windspeed	event
2017-01-01	32.0	6.0	Rain
2017-04-01	NaN	9.0	Sunny
2017-05-01	28.0	NaN	Snow
2017-06-01	NaN	7.0	NaN
2017-07-01	32.0	NaN	Rain
2017-08-01	NaN	NaN	Sunny
2017-10-01	34.0	8.0	Cloudy
2017-11-01	40.0	12.0	Sunny

```
new_df=df.dropna(thresh=2)# means if we have one row with atleast one Nan value keep that row otherwise dont show the rest
new_df
```

```
#creatig a daterange
dt=pd.date_range('01-01-2017','01-11-2017')
idx=pd.DatetimeIndex(dt)
df=df.reindex(idx)
df
```

	temperature	windspeed	event
2017-01-01	32.0	6.0	Rain
2017-01-02	NaN	NaN	NaN
2017-01-03	NaN	NaN	NaN
2017-01-04	NaN	NaN	NaN
2017-01-05	NaN	NaN	NaN
2017-01-06	NaN	NaN	NaN
2017-01-07	NaN	NaN	NaN
2017-01-08	NaN	NaN	NaN
2017-01-09	NaN	NaN	NaN
2017-01-10	NaN	NaN	NaN
2017-01-11	NaN	NaN	NaN

```
####using replace method to deal with missing data
```

```
import pandas as pd
import numpy as np
df=pd.read_excel('C:\\Users\sojore\Documents\weather_data1.xlsx')
df
```

	day	temperature	windspeed	event
0	2017-01-01	32	6	Rain
1	2017-02-01	-99999	7	Sunny
2	2017-03-01	28	-99999	Snow
3	2017-04-01	-99999	7	No event
4	2017-05-01	32	-88888	Rain
5	2017-06-01	31	2	Sunny
6	2017-07-01	34	5	No event

```
df.set_index('day',inplace=True)
df
```

	temperature	windspeed	event
day			
2017-01-01	32	6	Rain
2017-02-01	-99999	7	Sunny
2017-03-01	28	-99999	Snow
2017-04-01	-99999	7	No event
2017-05-01	32	-88888	Rain
2017-06-01	31	2	Sunny
2017-07-01	34	5	No event

```
df=pd.read_excel('C:\\Users\sojore\Documents\weather_data1.xlsx')
df
```

	day	temperature	windspeed	event
0	2017-01-01	32	6	Rain
1	2017-02-01	-99999	7	Sunny
2	2017-03-01	28	-99999	Snow
3	2017-04-01	-99999	7	No event
4	2017-05-01	32	-88888	Rain
5	2017-06-01	31	2	Sunny
6	2017-07-01	34	5	No event

```
new_df=df.replace(-99999,np.NaN)
new_df
```

	day	temperature	windspeed	event
0	2017-01-01	32.0	6.0	Rain
1	2017-02-01	NaN	7.0	Sunny
2	2017-03-01	28.0	NaN	Snow
3	2017-04-01	NaN	7.0	No event
4	2017-05-01	32.0	-88888.0	Rain
5	2017-06-01	31.0	2.0	Sunny
6	2017-07-01	34.0	5.0	No event

```
#how about if we have different special values,so we gonna ammend the data above
```

```
new_df=df.replace([-99999,-88888],np.NaN)#for us to change both special values n replace with NaN then we supply both in a list
new_df
```

	day	temperature	windspeed	event
0	2017-01-01	32.0	6.0	Rain
1	2017-02-01	NaN	7.0	Sunny
2	2017-03-01	28.0	NaN	Snow
3	2017-04-01	NaN	7.0	No event
4	2017-05-01	32.0	NaN	Rain
5	2017-06-01	31.0	2.0	Sunny
6	2017-07-01	34.0	5.0	No event

```
new1_df=df.replace({'temperature':-99999,
                    'windspeed':-88888,
                    'event':'No event'},np.NaN)
new1_df
```

	day	temperature	windspeed	event
0	2017-01-01	32.0	6.0	Rain
1	2017-02-01	NaN	7.0	Sunny
2	2017-03-01	28.0	-99999.0	Snow
3	2017-04-01	NaN	7.0	NaN

```
new_df=new1_df.replace({'windspeed':-99999},np.NaN)
new_df
```

	day	temperature	windspeed	event
0	2017-01-01	32.0	6.0	Rain
1	2017-02-01	NaN	7.0	Sunny
2	2017-03-01	28.0	NaN	Snow
3	2017-04-01	NaN	7.0	NaN
4	2017-05-01	32.0	NaN	Rain
5	2017-06-01	31.0	2.0	Sunny
6	2017-07-01	34.0	5.0	NaN

```
new_df
```

	day	temperature	windspeed	event
0	2017-01-01	32.0	6.0	Rain
1	2017-02-01	NaN	7.0	Sunny
2	2017-03-01	28.0	NaN	Snow
3	2017-04-01	NaN	7.0	NaN
4	2017-05-01	32.0	NaN	Rain
5	2017-06-01	31.0	2.0	Sunny
6	2017-07-01	34.0	5.0	NaN

```
#to remove F,C and mph in these data above,we need to use regular expressions or regex
#regex are used to detect patterns
new_df=df.replace('[A-Za-z]','',regex=True)#this '[A-Za-z]' means that any chrs between capital A to Z and also from small a to z
new_df
```

	day	temperature	windspeed	event
0	2017-01-01	32	6	
1	2017-02-01	-99999	7	
2	2017-03-01	28	-99999	
3	2017-04-01	-99999	7	
4	2017-05-01	32	-88888	
5	2017-06-01	31	2	
6	2017-07-01	34	5	

```
new_df=df.replace({'temperature':'[A-Za-z]',
                   'windspeed':'[A-Za-z]'},'',regex=True)#this is when we dont want to touch the event section we only specify the
new_df
```

	day	temperature	windspeed	event
0	2017-01-01	32	6	Rain
1	2017-02-01	-99999	7	Sunny
2	2017-03-01	28	-99999	Snow
3	2017-04-01	-99999	7	No event

```
#replacing list of values with another list of values
#now we construct a different dataframe
df=pd.DataFrame({'score':['exceptional','average','good','poor','average','exceptional'],
                 'student':['rob','maya','parthiv','tom','julian','erica']})
df
```

	score	student
0	exceptional	rob
1	average	maya
2	good	parthiv
3	poor	tom
4	average	julian
5	exceptional	erica

```
new_df=df.replace(['poor','average','good','exceptional'],[1,2,3,4])
new_df
```

	score	student
0	4	rob
1	2	maya
2	3	parthiv
3	1	tom
4	2	julian
5	4	erica

```
####Group by(split,apply,combine)
```

```
import pandas as pd
import numpy as np
df=pd.read_excel('C:\\Users\sojore\Documents\weather_data2.xlsx')
df
```

	day	city	temperature	windspeed	event
0	2017-01-01 00:00:00	new york	32	6	Rain
1	2017-02-01 00:00:00	new york	36	7	Sunny
2	1/3//2017	new york	28	12	Snow
3	2017-04-01 00:00:00	new york	33	7	Sunny
4	1/1//2017	mumbai	90	5	Sunny
5	2017-02-01 00:00:00	mumbai	85	12	Fog
6	2017-03-01 00:00:00	mumbai	87	15	Fog
7	2017-04-01 00:00:00	mumbai	92	5	Rain
8	2017-01-01 00:00:00	paris	45	20	Sunny
9	2017-02-01 00:00:00	paris	50	13	Cloudy
10	2017-03-01 00:00:00	paris	54	8	Cloudy
11	2017-04-01 00:00:00	paris	42	10	Cloudy

```
g=df.groupby('city')#we wanna group our data based on city
g
```

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x0000019DC315FD30>
```

```
for city,city_df in g:
    print(city)
    print(city_df)
```

```
mumbai
                  day    city  temperature  windspeed  event
4           1/1//2017  mumbai           90          5  Sunny
5  2017-02-01 00:00:00  mumbai           85         12    Fog
6  2017-03-01 00:00:00  mumbai           87         15    Fog
7  2017-04-01 00:00:00  mumbai           92          5   Rain
new york
                  day      city  temperature  windspeed  event
0  2017-01-01 00:00:00  new york           32          6   Rain
1  2017-02-01 00:00:00  new york           36          7  Sunny
2           1/3//2017  new york           28         12   Snow
3  2017-04-01 00:00:00  new york           33          7  Sunny
paris
                  day   city  temperature  windspeed   event
8   2017-01-01 00:00:00  paris           45         20   Sunny
9   2017-02-01 00:00:00  paris           50         13  Cloudy
10  2017-03-01 00:00:00  paris           54          8  Cloudy
11  2017-04-01 00:00:00  paris           42         10  Cloudy
```

```
g1=g.get_group('mumbai')
g1
```

	day	city	temperature	windspeed	event
4	1/1//2017	mumbai	90	5	Sunny
5	2017-02-01 00:00:00	mumbai	85	12	Fog
6	2017-03-01 00:00:00	mumbai	87	15	Fog

```
g=df.groupby('city')#we wanna group our data based on city
g
for city,city_df in g:
    print(city)
    print(city_df)
```

```
mumbai
                  day    city  temperature  windspeed  event
4           1/1//2017  mumbai           90          5  Sunny
5  2017-02-01 00:00:00  mumbai           85         12    Fog
6  2017-03-01 00:00:00  mumbai           87         15    Fog
7  2017-04-01 00:00:00  mumbai           92          5   Rain
new york
                  day      city  temperature  windspeed  event
0  2017-01-01 00:00:00  new york           32          6   Rain
1  2017-02-01 00:00:00  new york           36          7  Sunny
2           1/3//2017  new york           28         12   Snow
3  2017-04-01 00:00:00  new york           33          7  Sunny
paris
                  day   city  temperature  windspeed   event
8   2017-01-01 00:00:00  paris           45         20   Sunny
9   2017-02-01 00:00:00  paris           50         13  Cloudy
10  2017-03-01 00:00:00  paris           54          8  Cloudy
11  2017-04-01 00:00:00  paris           42         10  Cloudy
```

```
g.max('mumbai')
```

	temperature	windspeed
city		
mumbai	92	15
new york	36	12
paris	54	20

concat Dataframes

concatenation implies joining 2 or more dataframes

```
import pandas as pd
india_weather=pd.DataFrame({'city':['mumbai','delhi','banglore'],
                            'temperature':[32,45,30],
                            'humidity':[80,60,78]})
india_weather
```

	city	temperature	humidity
0	mumbai	32	80
1	delhi	45	60
2	banglore	30	78

```
us_weather=pd.DataFrame({'city':['new york','chicago','orlando'],
                            'temperature':[21,14,35],
                            'humidity':[68,65,75]})
us_weather
```

	city	temperature	humidity
0	new york	21	68
1	chicago	14	65
2	orlando	35	75

```
#now we wanna join these 2 dataframes
pd.concat([india_weather,us_weather])
```

	city	temperature	humidity
0	mumbai	32	80
1	delhi	45	60
2	banglore	30	78
0	new york	21	68
1	chicago	14	65
2	orlando	35	75

```
pd.concat([india_weather,us_weather],ignore_index=True)
```

	city	temperature	humidity
0	mumbai	32	80
1	delhi	45	60
2	banglore	30	78
3	new york	21	68
4	chicago	14	65
5	orlando	35	75

```
df=pd.concat([india_weather,us_weather],keys=['india','us'])
df
```

```
temperature_df=pd.DataFrame({'city':['mumbai','delhi','banglore'],
                             'temperature':[32,45,30],})
temperature_df
```

	city	temperature
0	mumbai	32
1	delhi	45
2	banglore	30

```
windspeed_df=pd.DataFrame({'city':['mumbai','delhi','banglore'],
                             'windspeed':[7,12,9]})
windspeed_df
```

	city	windspeed
0	mumbai	7
1	delhi	12
2	banglore	9

```
#we now append the 2 dataframes above
df=pd.concat([temperature_df,windspeed_df],axis=0)
df
```

	city	temperature	windspeed
0	mumbai	32.0	NaN
1	delhi	45.0	NaN
2	banglore	30.0	NaN
0	mumbai	NaN	7.0

```
df=pd.concat([temperature_df,windspeed_df],axis=1)
df
```

	city	temperature	city	windspeed
0	mumbai	32	mumbai	7
1	delhi	45	delhi	12
2	banglore	30	banglore	9

```
#now we can a case whereby the city names in both temp_df and wind_df are not in same order as below
temperature_df=pd.DataFrame({'city':['delhi','mumbai','banglore'],
                             'temperature':[32,45,30],})
temperature_df
```

	city	temperature
0	delhi	32
1	mumbai	45
2	banglore	30

```
windspeed_df=pd.DataFrame({'city':['mumbai','delhi'],
                             'windspeed':[7,12]})
windspeed_df
```

	city	windspeed
0	mumbai	7
1	delhi	12

```
df=pd.concat([temperature_df,windspeed_df],axis=1)
df
```

	city	temperature	city	windspeed
0	delhi	32	mumbai	7.0
1	mumbai	45	delhi	12.0
2	banglore	30	NaN	NaN

```
#now the above doesnt look good,so do some modifications when creating the dataframes
```

```
temperature_df=pd.DataFrame({'city':['delhi','mumbai','banglore'],
                             'temperature':[32,45,30]},index=[0,1,2])
temperature_df
```

	city	temperature
0	delhi	32
1	mumbai	45
2	banglore	30

```
windspeed_df=pd.DataFrame({'city':['mumbai','delhi'],
                             'windspeed':[7,12]},index=[1,0])
windspeed_df
```

	city	windspeed
1	mumbai	7

```
df=pd.concat([temperature_df,windspeed_df],axis=1)
df
```

	city	temperature	city	windspeed
0	delhi	32	delhi	12.0
1	mumbai	45	mumbai	7.0
2	banglore	30	NaN	NaN

```
temperature_df
```

	city	temperature
0	delhi	32
1	mumbai	45
2	banglore	30

```
s=pd.Series(['Humid','Dry','Rain'],name='event')
s
```

```
0    Humid
1      Dry
2     Rain
Name: event, dtype: object
```

```
df=pd.concat([temperature_df,s],axis=1)
df
```

	city	temperature	event
0	delhi	32	Humid
1	mumbai	45	Dry

Merge Dataframes ¶

```
import pandas as pd
df1=pd.DataFrame({'city':['new york','chicago','orlando','baltimore'],
                  'temperature':[21,14,35,32]})
df1
```

	city	temperature
0	new york	21
1	chicago	14
2	orlando	35
3	baltimore	32

```
df2=pd.DataFrame({'city':['chicago','new york','san francisco'],
                  'humidity':[65,68,75]})
df2
```

	city	humidity
0	chicago	65
1	new york	68
2	san francisco	75

```
df3=pd.merge(df1,df2,on='city',how='inner')
df3
```

	city	temperature	humidity
0	new york	21	68

pivot table

```
import pandas as pd
df =pd.read_excel('C:\\Users\sojore\Documents\weather_data3.xlsx')
df
```

	date	city	temperature	humidity
0	2017-01-05	new york	65	56
1	2017-02-05	new york	66	58
2	2017-03-05	new york	65	60
3	2017-01-05	mumbai	75	80
4	2017-02-05	mumbai	78	83
5	2017-03-05	mumbai	82	85
6	2017-01-05	beijing	80	26
7	2017-02-05	beijing	77	30
8	2017-03-05	beijing	79	35

```
df.pivot(index='date',columns='city')
```

	temperature			humidity		
city	beijing	mumbai	new york	beijing	mumbai	new york
date						
2017-01-05	80	75	65	26	80	56
2017-02-05	77	78	66	30	83	58
2017-03-05	79	82	65	35	85	60

```
df=pd.read_excel('C:\\Users\sojore\Documents\weather_data4.xlsx')
df
```

	date	city	temperature	humidity
0	2017-01-05 00:00:00	new york	65	56
1	2017-01-05 00:00:00	new york	61	54
2	2017-02-05 00:00:00	new york	70	60
3	2017-02-05 00:00:00	new york	72	62
4	2017-01-05 00:00:00	mumbai	75	80
5	5/1//2017	mumbai	78	83
6	2017-02-05 00:00:00	mumbai	82	85
7	2017-02-05 00:00:00	mumbai	80	26

```
df.pivot_table(index='city',columns='date',aggfunc='sum')#the default aggfunc='mean'
```

	humidity			temperature		
date	2017-01-05 00:00:00	2017-02-05 00:00:00	5/1//2017	2017-01-05 00:00:00	2017-02-05 00:00:00	5/1//2017
city						
mumbai	80.0	111.0	83.0	75.0	162.0	78.0
new york	110.0	122.0	NaN	126.0	142.0	NaN

```
df.pivot_table(index='city',columns='date',margins=True)
```

	humidity				temperature			
date	2017-01-05 00:00:00	2017-02-05 00:00:00	5/1//2017	All	2017-01-05 00:00:00	2017-02-05 00:00:00	5/1//2017	All
city								

#melt for reshaping dateframe

```
df=pd.read_excel('C:\\Users\sojore\Documents\weather_data5.xlsx')
df
```

	day	chicago	chennai	berlin
0	Monday	32	75	41
1	Tuesday	30	77	43
2	wednseday	28	75	45
3	Thursday	22	82	38
4	Friday	30	83	30
5	Saturday	20	81	45
6	Sunday	25	77	47

```
df1=pd.melt(df,id_vars=['day'])
df1
```

	day	variable	value
0	Monday	chicago	32
1	Tuesday	chicago	30
2	wednseday	chicago	28
3	Thursday	chicago	22
4	Friday	chicago	30

```
df1=pd.melt(df,id_vars=['day'],var_name='city')
df1
```

	day	city	value
0	Monday	chicago	32
1	Tuesday	chicago	30
2	wednseday	chicago	28
3	Thursday	chicago	22
4	Friday	chicago	30
5	Saturday	chicago	20
6	Sunday	chicago	25
7	Monday	chennai	75
8	Tuesday	chennai	77
9	wednseday	chennai	75
10	Thursday	chennai	82

stacking,unstacking

```
import pandas as pd
df=pd.read_excel('C:\\Users\sojore\Documents\stocks.xlsx',header=[0,1])
df
```

	price				price to earnings ratio(p/t)		
	company	facebook	google	microsoft	facebook	google	microsoft
0	2017-06-05	155	955	66	37.10	32.0	30.31
1	2017-06-06	150	987	69	36.98	31.3	30.56
2	2017-06-07	153	963	62	36.78	31.7	30.46
3	2017-06-08	155	1000	61	36.11	31.2	30.11
4	2017-06-09	156	1012	66	37.07	30.0	31.00

```
df.stack()
```

		price	price to earnings ratio(p/t)
0	company	2017-06-05 00:00:00	NaN
	facebook	155	37.10
	google	955	32.00
	microsoft	66	30.31
1	company	2017-06-06 00:00:00	NaN
	facebook	150	36.98
	google	987	31.30
	microsoft	69	30.56
2	company	2017-06-07 00:00:00	NaN

```
df_stacked.unstack()
```

	company	facebook		google		microsoft	
	price	price to earnings ratio(p/t)	price	price to earnings ratio(p/t)	price	price to earnings ratio(p/t)	price	price to earnings ratio(p/t)
0	2017-06-05	NaT	155.0	37.10	955.0	32.0	66.0	30.31
1	2017-06-06	NaT	150.0	36.98	987.0	31.3	69.0	30.56
2	2017-06-07	NaT	153.0	36.78	963.0	31.7	62.0	30.46
3	2017-06-08	NaT	155.0	36.11	1000.0	31.2	61.0	30.11
4	2017-06-09	NaT	156.0	37.07	1012.0	30.0	66.0	31.00

```
df
```

	price				price to earnings ratio(p/t)		
	company	facebook	google	microsoft	facebook	google	microsoft
0	2017-06-05	155	955	66	37.10	32.0	30.31
1	2017-06-06	150	987	69	36.98	31.3	30.56
2	2017-06-07	153	963	62	36.78	31.7	30.46
3	2017-06-08	155	1000	61	36.11	31.2	30.11
4	2017-06-09	156	1012	66	37.07	30.0	31.00

```
df.stack(level =1)
```

#contigency table(crosstab)

```
import pandas as pd
df=pd.read_excel('C:\\Users\sojore\Documents\survey.xlsx')
df
```

	name	nationality	sex	age	handedness
0	kathy	usa	female	23	right
1	linda	usa	female	18	right
2	peter	usa	male	19	right
3	john	usa	male	22	left
4	fatima	bangadesh	female	31	left
5	kadir	bangadesh	male	25	left
6	dhaval	india	male	35	left
7	sudhir	india	male	31	left
8	parvir	india	male	37	right
9	yan	china	female	52	right
10	juan	china	female	58	left
11	liang	china	male	43	left

```
pd.crosstab(df.nationality,df.handedness)
```

handedness	left	right
nationality		
bangadesh	2	0

```
pd.crosstab(df.nationality,df.handedness)
```

handedness	left	right
nationality		
bangadesh	2	0
china	2	1
india	2	1
usa	1	3

```
pd.crosstab(df.sex,df.handedness)
```

handedness	left	right
sex		
female	2	3
male	5	2

```
pd.crosstab(df.nationality,df.handedness,margins=True)#the margins generates totals
```

handedness	left	right	All
nationality			
bangadesh	2	0	2
china	2	1	3
india	2	1	3
usa	1	3	4
All	7	5	12

```
pd.crosstab(df.sex,[df.handedness,df.nationality],margins=True)
```

handedness	left				right			All
nationality	bangadesh	china	india	usa	china	india	usa	
sex								
female	1	1	0	0	1	0	2	5
male	1	1	2	1	0	1	1	7
All	2	2	2	1	1	1	3	12

```
pd.crosstab(df.sex,df.handedness,margins=True)#the margins generates totals
```

handedness	left	right	All
sex			
female	2	3	5
male	5	2	7
All	7	5	12

```
pd.crosstab([df.sex],[df.handedness],normalize='index')
```

handedness	left	right
sex		
female	0.400000	0.600000
male	0.714286	0.285714

#datetime index and resample

```
df=pd.read_excel('C:\\Users\sojore\Documents\stocks.xlsx')
df
```

	price	Unnamed: 1	Unnamed: 2	Unnamed: 3	price to earnings ratio(p/t)	Unnamed: 5	Unnamed: 6
0	company	facebook	google	microsoft	facebook	google	microsoft
1	2017-06-05 00:00:00	155	955	66	37.1	32	30.31
2	2017-06-06 00:00:00	150	987	69	36.98	31.3	30.56
3	2017-06-07 00:00:00	153	963	62	36.78	31.7	30.46
4	2017-06-08 00:00:00	155	1000	61	36.11	31.2	30.11
5	2017-06-09 00:00:00	156	1012	66	37.07	30	31

```
df.index
```

```
RangeIndex(start=0, stop=6, step=1)
```

```
import pandas as pd
df=pd.read_excel('C:\\Users\sojore\Documents\stocks.xlsx',header=[0,1])
df
```

	price				price to earnings ratio(p/t)		
	company	facebook	google	microsoft	facebook	google	microsoft
0	2017-06-05	155	955	66	37.10	32.0	30.31
1	2017-06-06	150	987	69	36.98	31.3	30.56
2	2017-06-07	153	963	62	36.78	31.7	30.46
3	2017-06-08	155	1000	61	36.11	31.2	30.11
4	2017-06-09	156	1012	66	37.07	30.0	31.00

```
pd.to_datetime('5#1#2017',format='%d#%m#%Y')
```

```
Timestamp('2017-01-05 00:00:00')
```

```
dates=['2017-01-05','Jan 5,2017','01/05/2017','2017.01.05','2017/01/05','20170105','abc']

pd.to_datetime(dates,errors='ignore')
```

```
Index(['2017-01-05', 'Jan 5,2017', '01/05/2017', '2017.01.05', '2017/01/05',
       '20170105', 'abc'],
      dtype='object')
```

```
dates=['2017-01-05','Jan 5,2017','01/05/2017','2017.01.05','2017/01/05','20170105','abc']

pd.to_datetime(dates,errors='coerce')
```

```
DatetimeIndex(['2017-01-05', '2017-01-05', '2017-01-05', '2017-01-05',
               '2017-01-05', '2017-01-05',        'NaT'],
              dtype='datetime64[ns]', freq=None)
```

```
import pandas as pd
y=pd.Period('2016')
y
```

```
Period('2016', 'A-DEC')
```

```
dir(y)
```

```
['__add__',
 '__class__',
 '__delattr__',
 '__dict__',
 '__dir__',
 '__doc__',
```

Now we have looked at all concepts and key procedures to do when using pandas and NumPy to do data cleaning, we will now look at our next topic which is machine learning and deep learning

These two topics entails creation of a model using the data that we have cleaned using pandas and NumPy

- **Machine Learning**

This is a broad topic, but to our interest as data scientist, we will be looking at how to use machine learning algorithms to develop models based on the cleaned data to help in prediction

In this, you will see how we clean up data, import machine learning algorithms and use them to build models, and lastly use those models to do predictions

```
import pandas as pd
import numpy as np
from sklearn import linear_model
```

```
df=pd.read_excel('C:\\Users\sojore\Documents\homeprice_2.xlsx')
df
```

	area	bedrooms	age	price
0	2600	3.0	20	550000
1	3000	4.0	15	565000
2	3200	NaN	18	610000
3	3600	3.0	30	595000
4	4000	5.0	8	760000

```
df.bedrooms
```

```
0    3.0
1    4.0
2    NaN
3    3.0
4    5.0
Name: bedrooms, dtype: float64
```

```
df['bedrooms']
```

```
0    3.0
1    4.0
2    NaN
3    3.0
4    5.0
```

```
df['bedrooms'].median()
```

```
3.5
```

```
import math
median_bedrooms=math.floor(df['bedrooms'].median())
median_bedrooms
```

```
3
```

```
df.bedrooms=df.bedrooms.fillna(median_bedrooms)
```

```
df
```

	area	bedrooms	age	price
0	2600	3.0	20	550000
1	3000	4.0	15	565000
2	3200	3.0	18	610000
3	3600	3.0	30	595000
4	4000	5.0	8	760000

```
reg=linear_model.LinearRegression()
reg.fit(df[['area','bedrooms','age']],df.price)#this method is used to train our model, in array are the independent variables
```

```
LinearRegression()
```

```
reg=linear_model.LinearRegression()
reg.fit(df[['area','bedrooms','age']],df['price'])
```

```
LinearRegression()
```

```
reg.coef_
```

```
array([   137.25, -26025.  ,  -6825.  ])
```

```
reg.intercept_
```

```
383724.9999999998
```

```
reg.predict([[3000,3,40]])
```

```
array([444400.])
```

```
137.25*3000+26025*3+6825*40+383724.9999999998
```

```
1146549.9999999998
```

```
reg.predict([[2500,4,5]])
```

```
array([588625.])
```

```
#above is linear regression
```

```
#we now look at gradient descent and cost function
```

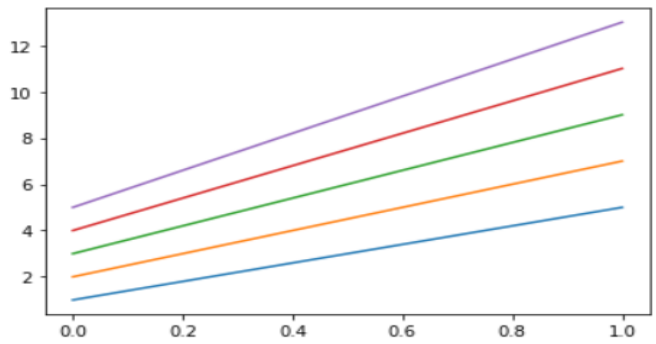
```python
# coding the gradient descent function algorithm to come up with the values of m and b in a linear(cost function)
import numpy as np

def gradient_descent(x,y):
    m_cur=b_cur=0 # this represnts the initial starting value of b and m as 0
    #we now using these small steps so as to get the global minimum
    iterations=10000 # these are no. of iterations are willing to perfrom to attain global minimu
    n=len(x)
    learning_rate=0.08
    for i in range(iterations):
        y_preiceted=m_cur*x+b_cur
        cost=(1/n)*sum([val **2 for val in (y-y_preiceted)])
        #reason we need to track the cost values is that its value must be decreasing
        #now next step we calculate the m derivertive and b derivertive
        md=-(2/n)*sum(x*(y-y_preiceted))
        bd = -(2 / n) * sum( (y - y_preiceted))
        #we now adjust our new m_cur value to the next m_cur using the learning rate and beloe equation
        m_cur=m_cur-learning_rate*md
        b_cur=b_cur-learning_rate*bd
        print('m {}, b {},,cost {},iteration {} '.format(m_cur,b_cur,cost,i))



x=np.array([1,2,3,4,5])
y=np.array([5,7,9,11,13])


gradient_descent(x,y)
```

```
m 4.96, b 1.44,,cost 89.0,iteration 0
m 0.4991999999999983, b 0.26879999999999993,,cost 71.10560000000002,iteration 1
m 4.451584000000002, b 1.426176000000001,,cost 56.8297702400001,iteration 2
m 0.892231679999997, b 0.5012275199999995,,cost 45.43965675929613,iteration 3
m 4.041214713600002, b 1.432759910400001,,cost 36.35088701894832,iteration 4
```

25Machine Learning11%25%25.ipyn

```python
%matplotlib inline
import matplotlib.pyplot as plt
plt.plot(((x,y)))
```

```
[<matplotlib.lines.Line2D at 0x23820767310>,
 <matplotlib.lines.Line2D at 0x238207673d0>,
 <matplotlib.lines.Line2D at 0x23820767490>,
 <matplotlib.lines.Line2D at 0x23820767550>,
 <matplotlib.lines.Line2D at 0x23820767610>]
```




```python
#saving a trained model using
#1.pickle
#2. sklearn joblib
```

```python
import pandas as pd
import numpy as np
from sklearn import linear_model
```

```
df=pd.read_excel('C:\\Users\sojore\Documents\homeprice.xlsx')
df.head()
```

	area	price
0	2600	550000
1	3000	565000
2	3200	610000
3	3600	680000
4	4000	725000

```
model=linear_model.LinearRegression()
model.fit(df[['area']],df['price'])
```

```
LinearRegression()
```

```
model.coef_
```

```
array([135.78767123])
```

```
model.intercept_
```

```
180616.43835616432
```

```
model.predict([[5000]])
```

```
array([859554.79452055])
```

```
import  pickle# this module allows us to serialise my model into a file
```

```
with open('model_pickle','wb') as f:
    pickle.dump(model,f)
```

```
with open('model_pickle','rb')  as f:
    mp=model=pickle.load(f)
```

```
mp.predict([[5000]])
```

```
array([859554.79452055])
```

```
#2rd approach of saving models as files
```

```
from sklearn.externals import joblib
```

```
joblib.dump(model,'model_joblib')
```

```
mj=joblib.load('model_joblib')
mj.predict([[5000]])
```

```
#### Dummy  variables
```

```
# one hot encoding
```

```
import pandas as pd
df=pd.read_excel('C:\\Users\sojore\Documents\homeprice_3.xlsx')
df
```

	town	area	price
0	monroe toenship	2600	550000
1	monroe toenship	3000	565000
2	monroe toenship	3200	610000
3	monroe toenship	3600	680000
4	monroe toenship	4000	725000
5	west windsor	2600	585000
6	west windsor	2800	615000
7	west windsor	3300	650000
8	west windsor	3600	710000
9	robbinsville	2600	575000
10	robbinsville	2900	600000

```
dummies=pd.get_dummies(df.town)
dummies
```

	monroe toenship	robbinsville	west windsor
0	1	0	0
1	1	0	0
2	1	0	0
3	1	0	0
4	1	0	0
5	0	0	1
6	0	0	1
7	0	0	1
8	0	0	1
9	0	1	0
10	0	1	0
11	0	1	0
12	0	1	0

```
merged=pd.concat([df,dummies],axis='columns')
merged
```

	town	area	price	monroe toenship	robbinsville	west windsor
0	monroe toenship	2600	550000	1	0	0
1	monroe toenship	3000	565000	1	0	0
2	monroe toenship	3200	610000	1	0	0

```
final=merged.drop(['town','west windsor'],axis='columns')
final
```

	area	price	monroe toenship	robbinsville
0	2600	550000	1	0
1	3000	565000	1	0
2	3200	610000	1	0
3	3600	680000	1	0
4	4000	725000	1	0
5	2600	585000	0	0
6	2800	615000	0	0
7	3300	650000	0	0
8	3600	710000	0	0
9	2600	575000	0	1
10	2900	600000	0	1
11	3100	620000	0	1
12	3600	695000	0	1

```
from sklearn.linear_model import LinearRegression
model=LinearRegression()
```

```
#X  this is values in area,monroe toenship and robbinvile except the price which is the Y value(dependent value)
X=final.drop(['price'],axis='columns')
X
```

	area	monroe toenship	robbinsville

```
y=final['price']
y
```

```
0     550000
1     565000
2     610000
3     680000
4     725000
5     585000
6     615000
7     650000
8     710000
9     575000
10    600000
11    620000
12    695000
Name: price, dtype: int64
```

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

```
LinearRegression()
```

```
model.predict([[2800,0,1]])
```

```
array([590775.63964739])
```

```
model.predict([[3400,0,0]])
```

```
array([681241.66845839])
```

```
model.score(X,y)
```

```
0.9573929037221873
```

```
#now we use the one hot encoding to perform the same operations as above
```

```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
```

```
dfle=df
dfle.town=le.fit_transform(dfle.town)
dfle
```

	town	area	price
0	0	2600	550000
1	0	3000	565000
2	0	3200	610000
3	0	3600	680000
4	0	4000	725000
5	2	2600	585000
6	2	2800	615000
7	2	3300	650000
8	2	3600	710000
9	1	2600	575000
10	1	2900	600000
11	1	3100	620000
12	1	3600	695000

```
X=dfle[['town','area']].values
X
```

```
y=dfle['price']
y
```

```
0     550000
1     565000
2     610000
3     680000
4     725000
5     585000
6     615000
7     650000
8     710000
9     575000
10    600000
11    620000
12    695000
Name: price, dtype: int64
```

```
from sklearn.preprocessing import OneHotEncoder
ohe=OneHotEncoder(categorical_features=[0]) # means that the zero column is my categorical_features
```

```
X=ohe.fit_transform(X).toarray()
```

```
X
```

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

```
X=X[:,1:]
X
```

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

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

```
LinearRegression()
```

```
model.predict([[1,0,2800]])
```

```
##Training and testing data and splitting
```

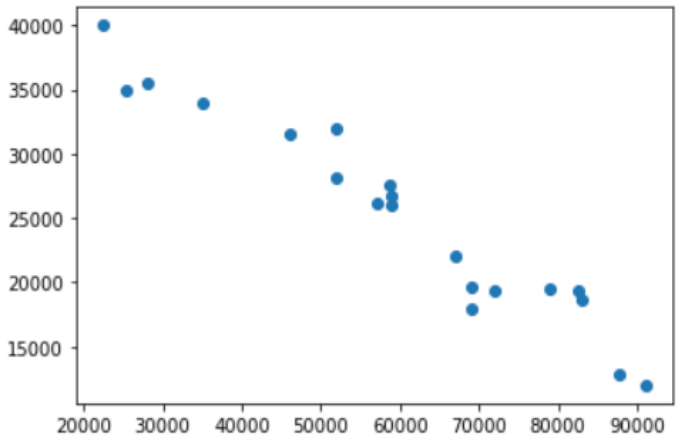
```
import pandas as pd
df=pd.read_excel('C:\\Users\sojore\Documents\carprices.xlsx')
df
```

	mileage	age	sell price
0	69000	6	18000
1	35000	3	34000
2	57000	5	26100
3	22500	2	40000
4	46000	4	31500
5	59000	5	26750
6	52000	5	32000
7	72000	6	19300
8	91000	8	12000
9	67000	6	22000
10	83000	7	18700
11	79000	7	19500

```
import matplotlib.pyplot as plt
%matplotlib inline
```

```
plt.scatter(df['mileage'],df['sell price'])
```

```
<matplotlib.collections.PathCollection at 0x238204f52b0>
```




```
X=df[['mileage','age']]
y=df['sell price']
```

```
X
```

	mileage	age
0	69000	6
1	35000	3
2	57000	5
3	22500	2
4	46000	4
5	59000	5
6	52000	5
7	72000	6
8	91000	8
9	67000	6
10	83000	7
11	79000	7
12	59000	5

```
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=10)
```

```
len(X_train)
```

```
16
```

```
len(X_test)
```

```
4
```

```
X_train
```

	mileage	age
3	22500	2
18	87600	8
13	58780	4
2	57000	5
14	82450	7
8	91000	8
17	69000	5
16	28000	2
19	52000	5
12	59000	5

```
from sklearn.linear_model import LinearRegression
clf= LinearRegression()
```

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

```
LinearRegression()
```

```
clf.predict(X_test)
```

```
array([20474.0745775 , 16352.07892168, 25174.14834912, 27197.42175439])
```

```
y_test
```

```
7     19300
10    18700
5     26750
6     32000
Name: sell price, dtype: int64
```

```
clf.score(X_test,y_test)
```

```
0.7332339593090138
```

logistic regression

```
import pandas as pd
from matplotlib import pyplot as plt
%matplotlib inline
```

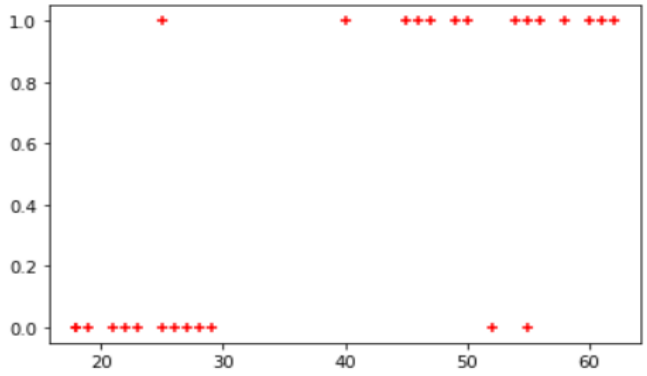
```
df=pd.read_excel('C:\\Users\sojore\Documents\insurance.xlsx')
df
```

```
df.head()
```

	age	bought_insurance
0	22	0
1	25	0
2	47	1
3	52	0
4	46	1

```
plt.scatter(df.age,df.bought_insurance,marker='+',color='red')
```

```
<matplotlib.collections.PathCollection at 0x23822c03520>
```




```
df.shape
```

```
(27, 2)
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test =train_test_split(df[['age']],df.bought_insurance,train_size=0.9)#you can use shift tab with cur
```

```
X_test
```

	age
14	49
5	56
0	22

```
X_train
```

	age
6	55
18	19
20	21
22	40
24	50
21	26
15	55
10	18

```
from sklearn.linear_model import LogisticRegression
model=LogisticRegression()
```

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

```
LogisticRegression()
```

```
model.predict(X_test)##array below([1,1,0]) means that from our X_test data 1 is going to buy insurance but 0 wont
```

```
array([1, 1, 0], dtype=int64)
```

```
model.score(X_test,y_test)
```

```
1.0
```

```
model.predict_proba(X_test)# this predicts the probablity of one class being in the other
```

```
array([[0.23008978, 0.76991022],
       [0.11054412, 0.88945588],
       [0.89811171, 0.10188829]])
```

```
#the above can be explained as folowing my X_test data above ,now using the first array in my predict probablity function
#we have that age was (49) and that means such person has 23% prob of not buying insurance but 76% prob of buying the insurance
# n this works for all array values against the age
```

```
model.predict([[74]])# means he will buy the insurance
```

```
array([1], dtype=int64)
```

```
model.predict([[14]])# means he wont buy the insurance
```

```
array([0], dtype=int64)
```

##logistic regression part 2

```
%matplotlib inline
import matplotlib.pyplot as plt
from sklearn.datasets import load_digits
```

```
<frozen importlib._bootstrap>:219: RuntimeWarning: numpy.ufunc size changed, may indicate
from C header, got 216 from PyObject
```

```
digits=load_digits()
```

```
dir(digits)
```

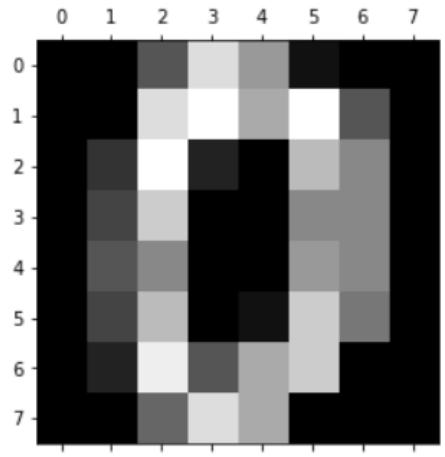
```
['DESCR', 'data', 'feature_names', 'frame', 'images', 'target', 'target_names']
```

```
digits.data[0]
```

```
array([ 0.,  0.,  5., 13.,  9.,  1.,  0.,  0.,  0.,  0., 13., 15., 10.,
       15.,  5.,  0.,  0.,  3., 15.,  2.,  0., 11.,  8.,  0.,  0.,  4.,
       12.,  0.,  0.,  8.,  8.,  0.,  0.,  5.,  8.,  0.,  0.,  9.,  8.,
        0.,  0.,  4., 11.,  0.,  1., 12.,  7.,  0.,  0.,  2., 14.,  5.,
       10., 12.,  0.,  0.,  0.,  0.,  6., 13., 10.,  0.,  0.,  0.])
```

```
plt.gray()
plt.matshow(digits.images[0])
```

```
<matplotlib.image.AxesImage at 0x2698a2e0610>

<Figure size 432x288 with 0 Axes>
```




```
plt.gray()
for i in range(5):
    plt.matshow(digits.images[i])
```

```
digits.target[[0,5]]
```

```
array([0, 5])
```

```
from sklearn.model_selection import train_test_split
```

```
 X_train, X_test, y_train, y_test=train_test_split(digits.data,digits.target,test_size=0.2)
```

```
len(X_train)
```

```
1437
```

```
len(X_test)
```

```
360
```

```
from sklearn.linear_model import LogisticRegression
model=LogisticRegression()
```

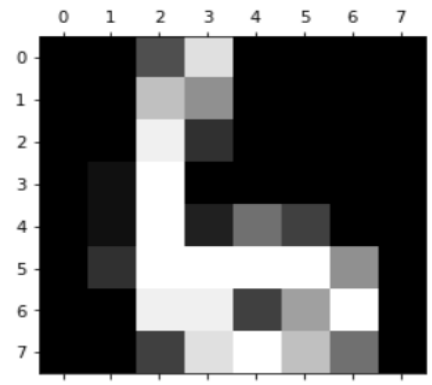
```
model.fit(X_train,y_train)
```

```
model.score(X_test,y_test)
```

```
0.9583333333333334
```

```
plt.matshow(digits.images[67])
```

```
<matplotlib.image.AxesImage at 0x2698ba29a90>
```



```
digits.target[67]
```

```
6
```

```
model.predict([digits.data[67]])
```

```
array([6])
```

```
model.predict(digits.data[0:5])
```

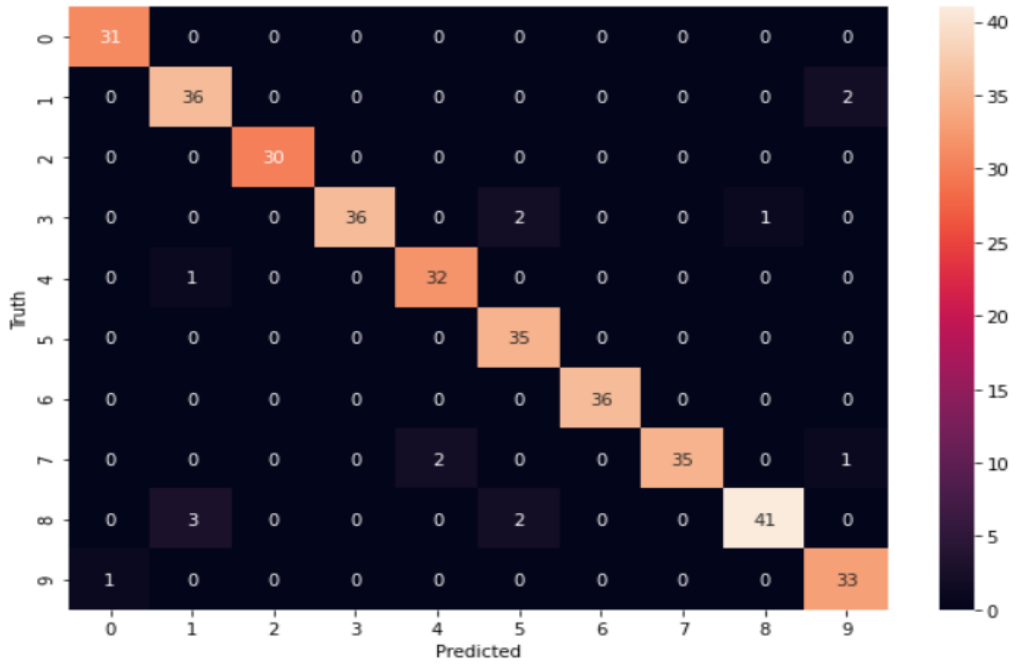
```
array([0, 1, 2, 3, 4])
```

```
y_predicted=model.predict(X_test)
from sklearn.metrics import confusion_matrix

cm=confusion_matrix(y_test,y_predicted)
cm
```

```
array([[31,  0,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 0, 36,  0,  0,  0,  0,  0,  0,  0,  2],
       [ 0,  0, 30,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0, 36,  0,  2,  0,  0,  1,  0],
       [ 0,  1,  0,  0, 32,  0,  0,  0,  0,  0],
       [ 0,  0,  0,  0,  0, 35,  0,  0,  0,  0],
       [ 0,  0,  0,  0,  0,  0, 36,  0,  0,  0],
       [ 0,  0,  0,  0,  2,  0,  0, 35,  0,  1],
       [ 0,  3,  0,  0,  0,  2,  0,  0, 41,  0],
       [ 1,  0,  0,  0,  0,  0,  0,  0,  0, 33]], dtype=int64)
```

```
import seaborn as sn
plt.figure(figsize=(10,7))
sn.heatmap(cm,annot=True)
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

```
Text(69.0, 0.5, 'Truth')
```




```
#the above heatmap shows that with for example 2 value ,when we fed it a truth value of 7 the model predicted
#4 and that shows the model isnt doing good at those instances
#i.e for example when we provided a truth value of 7 tor value 31 the model predicted 0 with intersect being 0 means is a good
#prediction,for 36 predicted 1 with intersect 0,thus it did pretty good,we go all the way to the last one,and for 32 the model
#predicted 4 with intersect at 2 thus it didnt do that good
```

```
#decision tree
import pandas as pd
df=pd.read_excel('C:\\Users\sojore\Documents\salaries.xlsx')
df
```

	company	job	degree	salary_more_than_100k
0	google	sales executive	bachelors	0
1	google	sales executive	masters	0
2	google	business manager	bachelors	1
3	google	business manager	masters	1
4	google	computer programmer	bachelors	0
5	google	computer programmer	masters	1
6	abc pharma	sales executive	masters	0
7	abc pharma	computer programmer	bachelors	0
8	abc pharma	business manager	bachelors	0
9	abc pharma	business manager	masters	1
10	facebook	sales executive	bachelors	1
11	facebook	sales executive	masters	1
12	facebook	business manager	bachelors	1
13	facebook	business manager	masters	1
14	facebook	computer programmer	bachelors	1

```
inputs=df.drop('salary_more_than_100k',axis='columns')
target=df['salary_more_than_100k']
```

```
inputs
```

	company	job	degree
0	google	sales executive	bachelors
1	google	sales executive	masters
2	google	business manager	bachelors
3	google	business manager	masters
4	google	computer programmer	bachelors
5	google	computer programmer	masters
6	abc pharma	sales executive	masters
7	abc pharma	computer programmer	bachelors
8	abc pharma	business manager	bachelors
9	abc pharma	business manager	masters
10	facebook	sales executive	bachelors
11	facebook	sales executive	masters
12	facebook	business manager	bachelors
13	facebook	business manager	masters
14	facebook	computer programmer	bachelors

```
from sklearn.preprocessing import LabelEncoder
```

```
le_company=LabelEncoder()# we have 3 columns so we need to create 3 objects
le_job=LabelEncoder()
le_degree=LabelEncoder()
```

```
inputs['company_n']=le_company.fit_transform(inputs['company'])#we now trynna add more columns in our dataframe n this is how
inputs['job_n']=le_company.fit_transform(inputs['job'])
inputs['degree_n']=le_company.fit_transform(inputs['degree'])
inputs.head()
```

	company	job	degree	company_n	job_n	degree_n
0	google	sales executive	bachelors	2	2	0
1	google	sales executive	masters	2	2	1
2	google	business manager	bachelors	2	0	0
3	google	business manager	masters	2	0	1
4	google	computer programmer	bachelors	2	1	0

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

	company_n	job_n	degree_n
0	2	2	0
1	2	2	1
2	2	0	0
3	2	0	1
4	2	1	0

```
from sklearn import tree
```

```
<frozen importlib._bootstrap>:219: RuntimeWarning: numpy.ufunc size changed, may indicate binary inc
from C header, got 216 from PyObject
<frozen importlib._bootstrap>:219: RuntimeWarning: numpy.ufunc size changed, may indicate binary inc
from C header, got 216 from PyObject
```

```
model=tree.DecisionTreeClassifier()
```

```
model.fit(inputs_n,target)
```

```
DecisionTreeClassifier()
```

```
model.score(inputs_n,target)
```

```
1.0
```

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

```
array([0], dtype=int64)
```

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

```
array([1], dtype=int64)
```

Support Vector Machine(SVM)

```
#svm works by trying to maximise the margins between differnt datapoints of diff. datasets
#we aim at drawing a better decision boundary
```

```
import pandas as pd
from sklearn.datasets import load_iris
iris=load_iris()
```

```
dir(iris)#this dir refers to basic features
```

```
['DESCR',
 'data',
 'feature_names',
 'filename',
 'frame',
 'target',
 'target_names']
```

```
iris.feature_names
```

```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

```
df=pd.DataFrame(iris.data,columns=iris.feature_names)
df.head()
```

```
df['target']=iris.target
df.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

```
iris.target_names
```

```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

```
df[df.target==1].head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
50	7.0	3.2	4.7	1.4	1
51	6.4	3.2	4.5	1.5	1
52	6.9	3.1	4.9	1.5	1
53	5.5	2.3	4.0	1.3	1
54	6.5	2.8	4.6	1.5	1

```
df[df.target==2].head()
```

```
df['flower_name']=df.target.apply(lambda x: iris.target_names[x])#these are methods of inserting more columns to our dataframe
df.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	flower_name
0	5.1	3.5	1.4	0.2	0	setosa
1	4.9	3.0	1.4	0.2	0	setosa
2	4.7	3.2	1.3	0.2	0	setosa
3	4.6	3.1	1.5	0.2	0	setosa
4	5.0	3.6	1.4	0.2	0	setosa

```
#above ways are used to add more columns to our dataset
```

```
from matplotlib import pyplot as plt
```

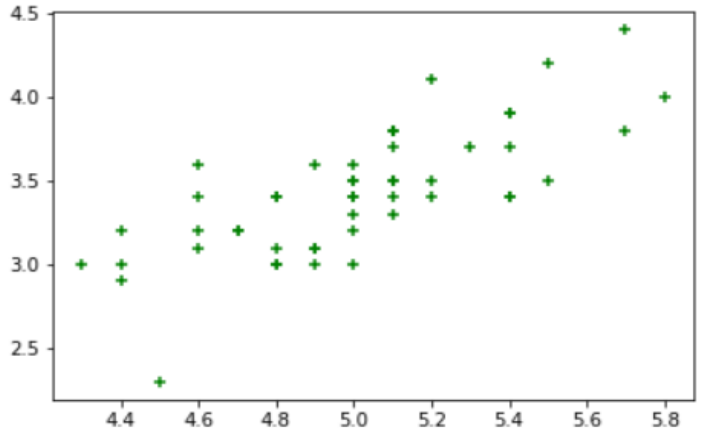
```
%matplotlib inline
```

```
df0=df[df.target==0]
df1=df[df.target==1]
df2=df[df.target==2]
```

```
df1.head()
```

```
plt.scatter(df0['sepal length (cm)'],df0['sepal width (cm)'],color='green',marker='+')
```

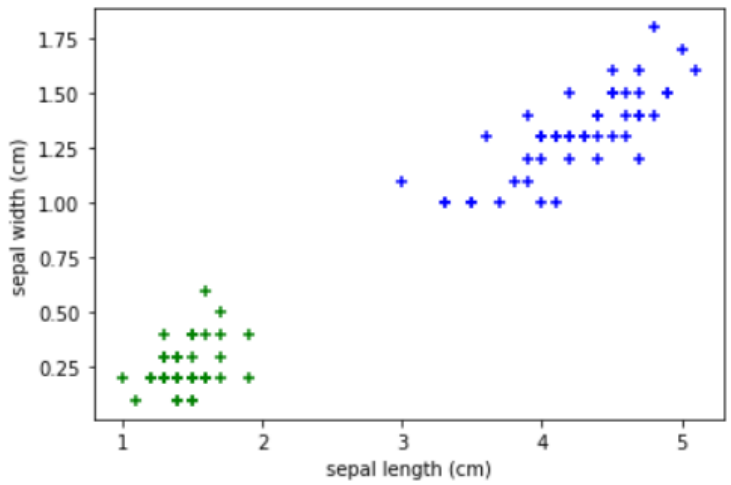
```
<matplotlib.collections.PathCollection at 0x2698f9b91f0>
```




```
plt.xlabel('sepal length (cm)')
plt.ylabel('sepal width (cm)')

plt.scatter(df0['sepal length (cm)'],df0['sepal width (cm)'],color='green',marker='+')
plt.scatter(df1['sepal length (cm)'],df1['sepal width (cm)'],color='blue',marker='+')
```

```
plt.xlabel('sepal length (cm)')
plt.ylabel('sepal width (cm)')

plt.scatter(df0['petal length (cm)'],df0['petal width (cm)'],color='green',marker='+')
plt.scatter(df1['petal length (cm)'],df1['petal width (cm)'],color='blue',marker='+')
```

```
<matplotlib.collections.PathCollection at 0x2698fd3c220>
```




```
from sklearn.model_selection import train_test_split
```

```
X=df.drop(['target','flower_name'],axis='columns')
X.head()
```

```
y=df.target
```

```
y
```

```
0      0
1      0
2      0
3      0
4      0
      ..
145    2
146    2
147    2
148    2
149    2
Name: target, Length: 150, dtype: int32
```

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

```
len(X_train)
```

```
120
```

```
len(X_test)
```

```
30
```

```
from sklearn.svm import SVC
model=SVC()#click shift tab for more SCV attributes
```

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

```
model.score(X_test,y_test)
```

```
0.9666666666666667
```

```
from sklearn.svm import SVC
model=SVC(C=10,kernel='linear',gamma='auto')
```

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

```
SVC(C=10, gamma='auto', kernel='linear')
```

```
model.score(X_test,y_test)
```

```
1.0
```

```
from sklearn.svm import SVC
model=SVC(C=10,kernel='linear',gamma='auto',
    degree=4,
    coef0=0.4,
    shrinking=True,
    probability=True,
    tol=0.002,
    cache_size=200,
    class_weight=None,
    verbose=True,
    max_iter=-1,
    decision_function_shape='ovr',
    break_ties=True,
    random_state=None,)
model.fit(X_train,y_train)
```

Random Forest Algorithm

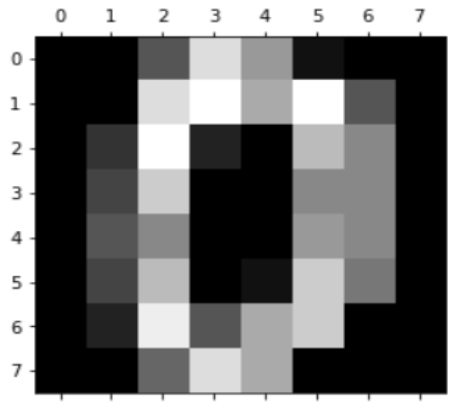
```
import pandas as pd
from sklearn.datasets import load_digits
digits=load_digits()
```

```
dir(digits)
```

```
['DESCR', 'data', 'feature_names', 'frame', 'images', 'target', 'target_names']
```

```
%matplotlib inline
import matplotlib.pyplot as plt
plt.gray()
for i in range(4):
    plt.matshow(digits.images[i])
```

```
<Figure size 432x288 with 0 Axes>
```




```
digits.data[:5]
```

```
array([[ 0.,  0.,  5., 13.,  9.,  1.,  0.,  0.,  0.,  0., 13., 15., 10.,
        15.,  5.,  0.,  0.,  3., 15.,  2.,  0., 11.,  8.,  0.,  0.,  4.,
        12.,  0.,  0.,  8.,  8.,  0.,  0.,  5.,  8.,  0.,  0.,  9.,  8.,
         0.,  0.,  4., 11.,  0.,  1., 12.,  7.,  0.,  0.,  2., 14.,  5.,
        10., 12.,  0.,  0.,  0.,  0.,  6., 13., 10.,  0.,  0.,  0.],
       [ 0.,  0.,  0., 12., 13.,  5.,  0.,  0.,  0.,  0.,  0., 11., 16.,
         9.,  0.,  0.,  0.,  0.,  3., 15., 16.,  6.,  0.,  0.,  0.,  7.,
        15., 16., 16.,  2.,  0.,  0.,  0.,  0.,  1., 16., 16.,  3.,  0.,
         0.,  0.,  0.,  1., 16., 16.,  6.,  0.,  0.,  0.,  0.,  1., 16.,
        16.,  6.,  0.,  0.,  0.,  0.,  0., 11., 16., 10.,  0.,  0.],
       [ 0.,  0.,  0.,  4., 15., 12.,  0.,  0.,  0.,  0.,  3., 16., 15.,
        14.,  0.,  0.,  0.,  0.,  8., 13.,  8., 16.,  0.,  0.,  0.,  0.,
         1.,  6., 15., 11.,  0.,  0.,  0.,  1.,  8., 13., 15.,  1.,  0.,
         0.,  0.,  9., 16., 16.,  5.,  0.,  0.,  0.,  0.,  3., 13., 16.,
        16., 11.,  5.,  0.,  0.,  0.,  0.,  3., 11., 16.,  9.,  0.],
       [ 0.,  0.,  7., 15., 13.,  1.,  0.,  0.,  0.,  8., 13.,  6., 15.,
         4.,  0.,  0.,  0.,  2.,  1., 13., 13.,  0.,  0.,  0.,  0.,  0.,
         2., 15., 11.,  1.,  0.,  0.,  0.,  0.,  0.,  1., 12., 12.,  1.,
         0.,  0.,  0.,  0.,  0.,  1., 10.,  8.,  0.,  0.,  0.,  8.,  4.,
         5., 14.,  9.,  0.,  0.,  0.,  7., 13., 13.,  9.,  0.,  0.],
       [ 0.,  0.,  0.,  1., 11.,  0.,  0.,  0.,  0.,  0.,  0.,  7.,  8.,
         0.,  0.,  0.,  0.,  0.,  1., 13.,  6.,  2.,  2.,  0.,  0.,  0.,
         7., 15.,  0.,  9.,  8.,  0.,  0.,  5., 16., 10.,  0., 16.,  6.,
         0.,  0.,  4., 15., 16., 13., 16.,  1.,  0.,  0.,  0.,  0.,  3.,
        15., 10.,  0.,  0.,  0.,  0.,  0.,  2., 16.,  4.,  0.,  0.]])
```

```
df=pd.DataFrame(digits.data)
df
```

```
df['target']=digits.target
df.head()
```

	0	1	2	3	4	5	6	7	8	9	...	55	56	57	58	59	60	61	62	63	target
0	0.0	0.0	5.0	13.0	9.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	6.0	13.0	10.0	0.0	0.0	0.0	0
1	0.0	0.0	0.0	12.0	13.0	5.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	11.0	16.0	10.0	0.0	0.0	1
2	0.0	0.0	0.0	4.0	15.0	12.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	3.0	11.0	16.0	9.0	0.0	2
3	0.0	0.0	7.0	15.0	13.0	1.0	0.0	0.0	0.0	8.0	...	0.0	0.0	0.0	7.0	13.0	13.0	9.0	0.0	0.0	3
4	0.0	0.0	0.0	1.0	11.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	2.0	16.0	4.0	0.0	0.0	4

5 rows × 65 columns

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(df.drop(['target'],axis='columns'),df.target,test_size=0.2)
```

```
len(X_train)
```

```
1437
```

```
len(X_test)
```

```
360
```

```
from sklearn.ensemble import RandomForestClassifier#ensemble module is used when we are tynna predict several outcomes
model=RandomForestClassifier(n_estimators=50)
```

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

```
RandomForestClassifier(n_estimators=50)
```

```
model.score(X_test,y_test)
```

```
0.9805555555555555
```

```
y_predicted=model.predict(X_test)
```

```
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,y_predicted)
cm
```

```
array([[39,  0,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 0, 30,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 1,  0, 37,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0, 41,  0,  1,  0,  0,  0,  0],
       [ 0,  0,  0,  0, 33,  0,  0,  0,  0,  0],
       [ 0,  0,  0,  0,  0, 32,  0,  0,  1,  1],
       [ 0,  0,  0,  0,  0,  0, 32,  0,  0,  0],
       [ 0,  0,  0,  0,  0,  0,  0, 46,  0,  1],
       [ 0,  1,  0,  0,  0,  0,  0,  0, 37,  0],
       [ 0,  0,  0,  0,  0,  0,  0,  0,  1, 26]], dtype=int64)
```

```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sn
plt.figure(figsize=(20,7))
sn.heatmap(cm,annot=True)
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

K Fold Cross Validation

```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
import numpy as np
from sklearn.datasets import load_digits
digits=load_digits()
```

```
<frozen importlib._bootstrap>:219: RuntimeWarning: numpy.ufunc size changed, may indicate binary in
from C header, got 216 from PyObject
<frozen importlib._bootstrap>:219: RuntimeWarning: numpy.ufunc size changed, may indicate binary in
from C header, got 216 from PyObject
```

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(digits.data,digits.target,test_size=0.3)
```

```
lr=LogisticRegression()
lr.fit(X_train,y_train)
lr.score(X_test,y_test)
```

```
C:\Users\sojore\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:762: ConvergenceWarni
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
0.9592592592592593
```

```
svm=SVC()
```

```
svm.fit(X_train,y_train)
svm.score(X_test,y_test)
```

```
0.9851851851851852
```

```
rf=RandomForestClassifier()
rf.fit(X_train,y_train)
rf.score(X_test,y_test)
```

```
0.9703703703703703
```

```
from sklearn.model_selection import KFold
kf=KFold(n_splits=3)#you can supply any value for n_splits eg 10 and it will work accordingly
kf
```

```
KFold(n_splits=3, random_state=None, shuffle=False)
```

```
for train_index,test_index in kf.split([1,2,3,4,5,6,7,8,9]):
    print(train_index,test_index)
```

```
[3 4 5 6 7 8] [0 1 2]
[0 1 2 6 7 8] [3 4 5]
[0 1 2 3 4 5] [6 7 8]
```

```
def get_score(model,X_train,X_test,y_train,y_test):# we can use this powerful method to still perform the above operations for s
    model.fit(X_train,y_train,)
    return model.score(X_test,y_test)
```

```
get_score(LogisticRegression(),X_train,X_test,y_train,y_test)
```

```
from sklearn.model_selection import StratifiedKFold#better tham kfold since it splits data uniformly
folds=StratifiedKFold(n_splits=3)
```

```
scores_lr=[]
scores_svm=[]
scores_rf=[]

for train_index,test_index in kf.split(digits.data):

    X_train,X_test,y_train,y_test=digits.data[train_index],digits.data[test_index],digits.target[train_index],digits.target[test_
       #this for loop is gonna run 3 times based on provided data
    print(get_score(LogisticRegression(),X_train,X_test,y_train,y_test) )
    print(get_score(SVC(),X_train,X_test,y_train,y_test) )
    print(get_score(RandomForestClassifier(),X_train,X_test,y_train,y_test) )
```

```
C:\Users\sojore\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:762: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
0.9232053422370617
0.9666110183639399
0.9365609348914858
```

```
scores_lr=[]
scores_svm=[]
scores_rf=[]

for train_index,test_index in kf.split(digits.data):

    X_train,X_test,y_train,y_test=digits.data[train_index],digits.data[test_index],digits.target[train_index],digits.target[test_
       #this for loop is gonna run 3 times based on provided data
    scores_lr.append(get_score(LogisticRegression(),X_train,X_test,y_train,y_test) )
    scores_svm.append(get_score(SVC(),X_train,X_test,y_train,y_test) )
    scores_rf.append(get_score(RandomForestClassifier(),X_train,X_test,y_train,y_test) )
```

K Means Clustering Algorithm

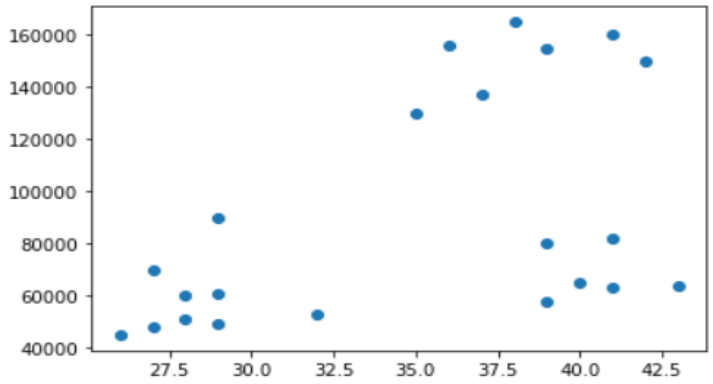
```
from sklearn.cluster import KMeans
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from matplotlib import pyplot as plt
%matplotlib inline
```

```
df=pd.read_excel('C:\\Users\sojore\Documents\income.xlsx')
df
```

	name	age	income
0	rob	27	70000
1	michael	29	90000
2	michan	29	61000
3	bmuli	28	60000
4	karoy	42	150000
5	guatum	39	155000
6	david	41	160000
7	andrea	38	165000
8	brad	36	156000
9	andelina	35	130000
10	donald	37	137000

```
plt.scatter(df.age,df.income)
```

```
<matplotlib.collections.PathCollection at 0x1cee29daa00>
```




```
km=KMeans(n_clusters=3)
km
```

```
KMeans(n_clusters=3)
```

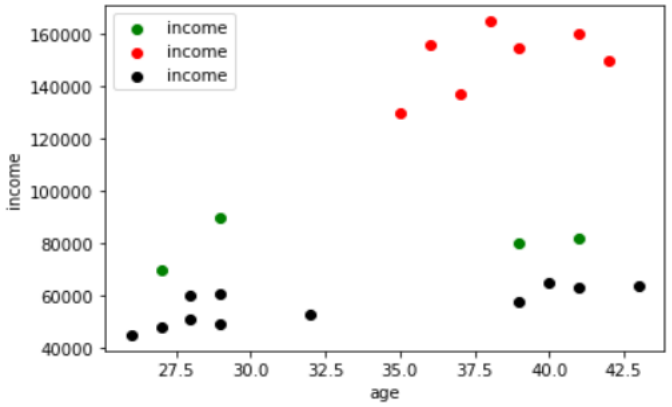
```
y_predicted=km.fit_predict(df[['age','income']])
y_predicted
```

```
array([0, 0, 2, 2, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0, 2])
```

```
df['cluster']=y_predicted
df.head()
```

```
df1=df[df.cluster==0]#this will return all dataframes where cluster is zero
df2=df[df.cluster==1]
df3=df[df.cluster==2]

plt.scatter(df1.age,df1.income,color='green',label='income')
plt.scatter(df2.age,df2.income,color='red',label='income')
plt.scatter(df3.age,df3.income,color='black',label='income')

plt.xlabel('age')
plt.ylabel('income')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x1cee28ee490>
```




```
#our scatter data plot above isnt well scaled so we use the minmaxscaler to come up with a better scalling
```

```
scaler=MinMaxScaler()
scaler.fit(df[['income']])#our sclaer is gonna make income to  have a range od 0-1
df['income']=scaler.transform(df[['income']])
df
```

```
#we now achieve the same range 0f 0-1 for age as well
```

```
scaler.fit(df[['age']])
df.age=scaler.transform(df[['age']])
df
```

	name	age	income	cluster
0	rob	0.058824	0.208333	0
1	michael	0.176471	0.375000	0
2	michan	0.176471	0.133333	2
3	bmuli	0.117647	0.125000	2
4	karoy	0.941176	0.875000	1
5	guatum	0.764706	0.916667	1
6	david	0.882353	0.958333	1
7	andrea	0.705882	1.000000	1

```
km=KMeans(n_clusters=3)
y_predicted=km.fit_predict(df[['age','income']])
y_predicted
```

```
array([0, 0, 0, 0, 2, 2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1])
```

```
df['cluster']=y_predicted

df
```

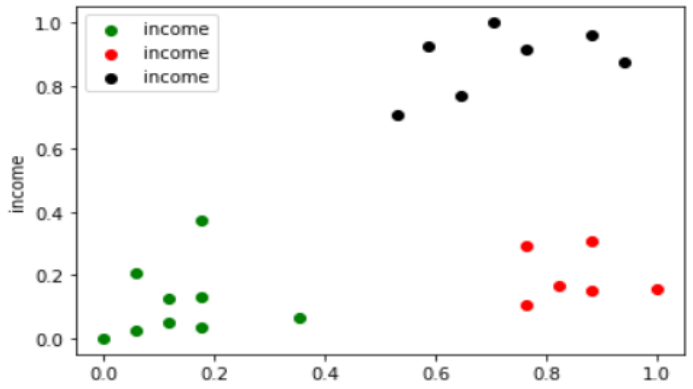
	name	age	income	cluster
0	rob	0.058824	0.208333	0
1	michael	0.176471	0.375000	0
2	michan	0.176471	0.133333	0
3	bmuli	0.117647	0.125000	0
4	karoy	0.941176	0.875000	2
5	guatum	0.764706	0.916667	2
6	david	0.882353	0.958333	2
7	andrea	0.705882	1.000000	2

```
km.cluster_centers_## the below are centroids to our clusters
```

```
array([[0.1372549 , 0.11342593],
       [0.85294118, 0.19722222],
       [0.72268908, 0.87857143]])
```

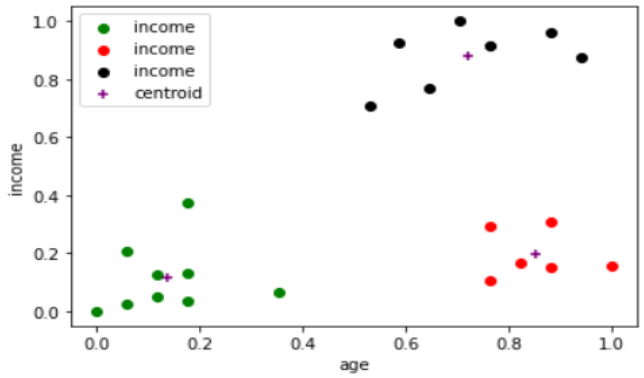
```
df1=df[df.cluster==0]#this will return all dataframes where cluster is zero
df2=df[df.cluster==1]
df3=df[df.cluster==2]

plt.scatter(df1.age,df1.income,color='green',label='income')
plt.scatter(df2.age,df2.income,color='red',label='income')
plt.scatter(df3.age,df3.income,color='black',label='income')

plt.xlabel('age')
plt.ylabel('income')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x1cee3edbf10>
```



```
df1=df[df.cluster==0]#this will return all dataframes where cluster is zero
df2=df[df.cluster==1]
df3=df[df.cluster==2]

plt.scatter(df1.age,df1.income,color='green',label='income')
plt.scatter(df2.age,df2.income,color='red',label='income')
plt.scatter(df3.age,df3.income,color='black',label='income')
plt.scatter(km.cluster_centers_[:,0],km.cluster_centers_[:,1],color='purple',marker='+',label='centroid')
plt.xlabel('age')
plt.ylabel('income')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x1cee3f69880>
```




```
#using elbow technique to get the sum of squared error using k
```

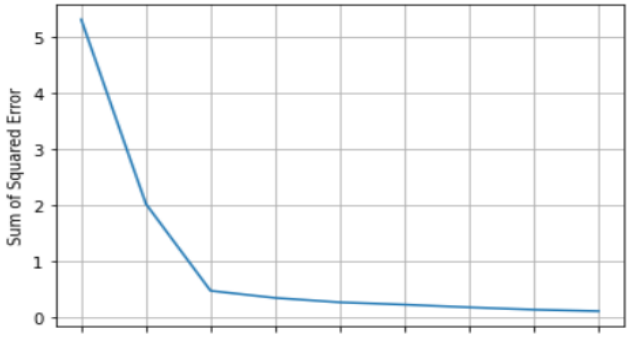
```
k_rng=range(1,10)
```

```
sse=[]
for k in k_rng:
    km=KMeans(n_clusters=k)
    km.fit(df[['age','income']])
    sse.append(km.inertia_)#this inertia method gives sse
```

```
sse
```

```
[5.321060160973927,
 2.024612197149683,
 0.46995844852102076,
 0.3426391825793499,
 0.26393545801694873,
 0.22277825085303737,
 0.17416046913046265,
 0.1326567963786822,
 0.10588652301198255]
```

```
plt.xlabel('k')
plt.ylabel('Sum of Squared Error')
plt.plot(k_rng,sse)
plt.grid()
```



Hyper Parameter Tuning

```
#this is a method used to determine which algorithm and speciallly parameters to use in  aa certain model
```

```
from sklearn import svm,datasets
iris=datasets.load_iris()
```

```
import pandas as pd
df=pd.DataFrame(iris.data,columns=iris.feature_names)
df['flower']=iris.target
df['flower']=df['flower'].apply(lambda x:iris.target_names[x])
df[47:52]
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	flower
47	4.6	3.2	1.4	0.2	setosa
48	5.3	3.7	1.5	0.2	setosa
49	5.0	3.3	1.4	0.2	setosa
50	7.0	3.2	4.7	1.4	versicolor
51	6.4	3.2	4.5	1.5	versicolor

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(iris.data,iris.target,test_size=0.3)
```

```
model=svm.SVC(kernel='rbf',C=30,gamma='auto')
model.fit(X_train,y_train)
model.score(X_test,y_test)
```

```
0.9777777777777777
```

```
#we now do the exact same thing as above using a for loop
#import numpy as np
#from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
#from sklearn.ensemble import RandomForestClassifier
import numpy as np
#from sklearn.datasets import load_digits
#digits=load_digits()
kernels=['rbf','linear']
C=[1,10,20]
avg_scores={}
for kval in kernels:
    for cval in C:
        cv_scores=cross_val_score(svm.SVC(kernel=kval,C=cval,gamma='auto'),iris.data,iris.target,cv=5)
        avg_scores[kval + '_' + str(cval)]=np.average(cv_scores)

avg_scores
```

```
{'rbf_1': 0.9800000000000001,
 'rbf_10': 0.9800000000000001,
 'rbf_20': 0.9666666666666668,
 'linear_1': 0.9800000000000001,
 'linear_10': 0.9733333333333334,
 'linear_20': 0.9666666666666666}
```

```
#also this method above isnt convinient so thes a method in sklearn we can use to solve the above taks without having to run for
```

```
from sklearn.model_selection import GridSearchCV#GridSearchCV takes 2 parameters
clf=GridSearchCV(svm.SVC(gamma='auto'),{'C':[1,101,20],'kernel':['rbf','linear']},cv=5,return_train_score=False)
```

```
clf.fit(iris.data,iris.target)
clf.cv_results_
```

choosing the best model code

```
from sklearn import svm
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
```

```
#now we define our parameter grid as a jason object or simple python dictionary where we supply each model
#with its parameters as shown below
```

```
model_params={
    'svm':{
        'model':svm.SVC(gamma='auto'),
        'params':{
            'C':[1,10,20],
            'kernel':['rbf','linear']
        }
    },
    'random_forest':{
        'model':RandomForestClassifier(),
        'params':{
            'n_estimators':[1,5,10]
        }
    },
    'logistic_regression':{
        'model':LogisticRegression(solver='liblinear',multi_class='auto'),
        'params':{
            'C':[1,5,10]
        }
    }
}
```

```
#now we write a for loop which goes through all the above mode_params (model and params) and then we append the values to a score
```

```
scores=[]

for model_name,mp in model_params.items():
    clf=GridSearchCV(mp['model'],mp['params'],cv=5,return_train_score=False)
    clf.fit(iris.data,iris.target)
    scores.append({
        'model':model_name,
        'best_score':clf.best_score_,
        'best_params':clf.best_params_
    })
```

```
df=pd.DataFrame(scores,columns=['model','best_score','best_params'])
df
```

	model	best_score	best_params
0	svm	0.980000	{'C': 1, 'kernel': 'rbf'}
1	random_forest	0.966667	{'n_estimators': 5}
2	logistic_regression	0.966667	{'C': 5}

```
#so from the above i can tell of which model and parameters is best for my analysis
```

L1 and L2 Regularization

```
#this is used to control overfitting of a model t.b.l.a
```

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
```

```
#supress warnings for clean notebook
import warnings
warnings.filterwarnings('ignore')
```

```
#read the dataset
dataset=pd.read_excel('C:\\Users\sojore\Documents\housing.xlsx')
dataset
```

	Address	Rooms	Type	Method	SellerG	Date	Distance	Postcode	Bedrooom	Bathroom	Car	Landsize	BulidingAr	Price
0	68 studly	2	h	SS	Jellis	2016-01-01	2.5	3067	2.0	1.0	1.0	126.0	NaN	140000
1	85 turner	2	h	S	Biggin	2016-02-01	2.5	3067	2.0	1.0	1.0	202.0	79.0	45000
2	25 bloomt	4	u	S	Biggin	2016-03-01	2.5	3067	3.0	2.0	1.0	156.0	NaN	76000
3	18 vic	3	h	VB	Rounds	2016-04-01	2.5	3067	3.0	2.0	1.0	0.0	150.0	47000
4	345 yhjk	1	u	SP	Nelson	2016-05-01	2.5	3067	4.0	1.0	2.0	134.0	142.0	87000
5	76 tre	2	u	PI	Biggin	2016-06-01	2.5	3067	3.0	1.0	2.0	94.0	220.0	98000
6	89 mngf	3	t	SN	Biggin	2016-07-01	2.5	3067	2.0	1.0	2.0	120.0	NaN	103500
7	45 dfgh	2	u	S	Collins	2016-08-01	2.5	3067	NaN	NaN	2.0	400.0	210.0	114100

```
col_to_fill_zero=['Distance','Bedrooom','Bathroom','Car']
dataset[col_to_fill_zero]=dataset[col_to_fill_zero].fillna(0)
dataset.isna().sum()
```

```
Address       0
Rooms         0
Type          0
Method        0
SellerG       0
Date          0
Distance      0
Postcode      0
Bedrooom      0
Bathroom      0
Car           0
Landsize      4
BulidingAr    8
Price         0
dtype: int64
```

```
dataset['Landsize']=dataset['Landsize'].fillna(dataset.Landsize.mean())
dataset['BulidingAr']=dataset['BulidingAr'].fillna(dataset.BulidingAr.mean())
```

```
dataset.isna().sum()
```

```
Address       0
Rooms         0
Type          0
Method        0
```

```
X=dataset.drop('Price',axis=1)
y=dataset['Price']
```

```
from sklearn.model_selection import train_test_split
train_X,X_test,train_y,test_y=train_test_split(X,y,test_size=0.2,random_state=2)
```

```
X=X[['Landsize','BulidingAr','Rooms','Bathroom','Bedrooom']]
X
```

	Landsize	BulidingAr	Rooms	Bathroom	Bedrooom
0	126.00	124.25	2	1.0	2.0
1	202.00	79.00	2	1.0	2.0
2	156.00	124.25	4	2.0	3.0
3	0.00	150.00	3	2.0	3.0
4	134.00	142.00	1	1.0	4.0
5	94.00	220.00	2	1.0	3.0

```
from sklearn.model_selection import train_test_split
train_X,test_X,train_y,test_y=train_test_split(X,y,test_size=0.3,random_state=2)
```

```
from sklearn.linear_model import LinearRegression
reg=LinearRegression().fit(train_X,train_y)
```

```
reg.score(test_X,test_y)
```

```
-19.115765479724793
```

```
reg.score(train_X,train_y)
```

```
0.3395504457805941
```

```
#the above LR model doesnt give a good score value so we use Lasso to get a better score
from sklearn import linear_model
lasso_reg=linear_model.Lasso(alpha=50,max_iter=100,tol=0.1)
lasso_reg.fit(train_X,train_y)
```

```
Lasso(alpha=50, max_iter=100, tol=0.1)
```

```
lasso_reg.score(test_X,test_y)
```

```
-18.876177972108298
```

```
lasso_reg.score(train_X,train_y)####these scores changes when using Lasso
```

```
0.33954180243397103
```

```
#####the above is L1 Regularization
```

```
#############now we can do L2 Regularization
```

```
from sklearn.linear_model import Ridge

ridge_reg=Ridge(alpha=50,max_iter=100,tol=0.1)
ridge_reg.fit(train_X,train_y)
```

```
Ridge(alpha=50, max_iter=100, tol=0.1)
```

```
ridge_reg.score(test_X,test_y)##now even using Ridge for L2 Regularization our score is impoved even further
```

```
-3.0983321768882783
```

```
ridge_reg.score(train_X,train_y)
```

```
0.07778375625923373
```

```
X=dataset.drop('Price',axis=1)
y=dataset['Price']
```

```
from sklearn.model_selection import train_test_split
train_X,X_test,train_y,test_y=train_test_split(X,y,test_size=0.2,random_state=2)
```

```
from sklearn.linear_model import LinearRegression
reg=LinearRegression().fit(train_X,train_y)
```

```
reg.score(test_X,test_y)
```

- **Deep Learning**

Just like machine learning, in deep learning we will using the same steps from data cleaning to model development, but in this case we will be developing models using deep learning algorithms such as:

-Artificial Neural Network (ANN)

-Convolutional Neural Network (CNN)

-Recurrent Neural Network (RNN)

Deep learning

```
import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
```

```
(X_train,y_train),(X_test,y_test)=keras.datasets.mnist.load_data()
```

```
len(X_train)
```

```
60000
```

```
len(X_test)
```

```
10000
```

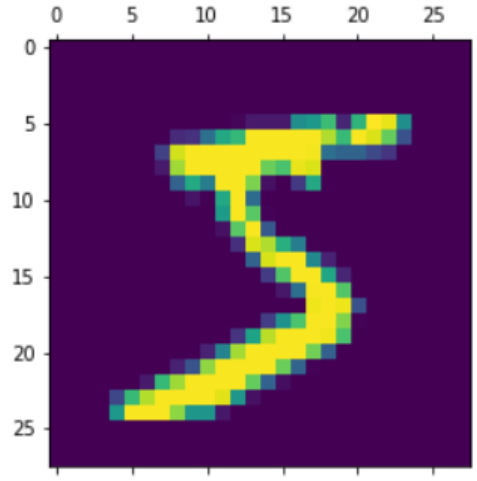
```
X_train[0].shape
```

```
(28, 28)
```

```
X_train[0]
```

```
plt.matshow(X_train[0])
```

```
<matplotlib.image.AxesImage at 0x1f55c386b50>
```




```
plt.matshow(X_train[2])
```

```
y_train[2]
```

```
4
```

```
y_train[:5]
```

```
array([5, 0, 4, 1, 9], dtype=uint8)
```

```
##now we are going to flatten our dataset
```

```
#if we are to check out the dimension of the X_train its a 2,2
X_train.shape
```

```
(60000, 28, 28)
```

```
X_train=X_train/255
X_test=X_test/255
```

```
X_train_flattened=X_train.reshape(len(X_train),28*28)
X_train_flattened
```

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

```
X_train_flattened.shape
```

```
X_train_flattened.shape
```

```
(60000, 784)
```

```
#we can as well flatten X_test as well
```

```
X_test_flattened=X_test.reshape(len(X_test),28*28)
X_test_flattened
```

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

```
X_test_flattened.shape
```

```
(10000, 784)
```

```
##when we take a look at the first individual element we have
X_train_flattened[0]
```

```
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
```

```
###comparing the 2 arrays above we see that we have converted out 2 dim array into a 1 dim array
```

```
##so now we will be creating a simple neural network,below is how you do it
```

here am doing my tensorboard visualization

```
model=keras.Sequential([
    keras.layers.Flatten(input_shape=(28,28)),
    keras.layers.Dense(100,activation='relu'),
    keras.layers.Dense(10,activation='sigmoid')
])

tb_callback=tf.keras.callbacks.TensorBoard(log_dir='logs/',histogram_freq=1)

model.compile(
    optimizer='SGD',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)
model.fit(X_train,y_train,epochs=5,callbacks=[tb_callback])
##using the same code previously buh doing some cahnges to visualize the tensorflow workflow
```

```
Epoch 1/5
1875/1875 [==============================] - ETA: 0s - loss: 0.6719 - accuracy: 0.8285 ETA: 4s - l - ET
- 88s 9ms/step - loss: 0.6707 - accuracy: 0.8288
Epoch 2/5
1875/1875 [==============================] - 13s 7ms/step - loss: 0.3392 - accuracy: 0.9050
Epoch 3/5
1875/1875 [==============================] - 10s 5ms/step - loss: 0.2896 - accuracy: 0.9186 0s - loss:
Epoch 4/5
1875/1875 [==============================] - 12s 7ms/step - loss: 0.2594 - accuracy: 0.9268
Epoch 5/5
```

```
model=keras.Sequential([
    keras.layers.Dense(10,input_shape=(784,),activation='sigmoid')
])##this sequantial means i have a stack of layers in myneural network

#output is set as 10,while input is set as 784
#we have also to specify the activiation function which is sigmoid
model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)
model.fit(X_train_flattened,y_train,epochs=5)#epochs is like the no. of iterations
```

```
Epoch 1/5
1875/1875 [==============================] - 23s 5ms/step - loss: 0.4715 - accuracy: 0.8761
Epoch 2/5
1875/1875 [==============================] - 8s 4ms/step - loss: 0.3036 - accuracy: 0.9154
Epoch 3/5
1875/1875 [==============================] - 10s 5ms/step - loss: 0.2835 - accuracy: 0.9206
Epoch 4/5
1875/1875 [==============================] - 9s 5ms/step - loss: 0.2733 - accuracy: 0.9232
Epoch 5/5
1875/1875 [==============================] - 12s 6ms/step - loss: 0.2669 - accuracy: 0.9257

<tensorflow.python.keras.callbacks.History at 0x1f55874ec10>
```

```
##if the accuracy above is not very high we can do scaling to improve the accuracy immediately after having your
#split X_train and y_train data
#using the code below
#X_train=X_train/255
#X_test=X_test/255(255 is the max number in the converted 1 dim array)
```

```
##now my model is trained in away that 88% of times it will make accurate prediction
```

```
##now lets test the model accuaracy on the test dataset
```

```
model.evaluate(X_test_flattened,y_test)
```

```
313/313 [==============================] - 11s 5ms/step - loss: 0.2662 - accuracy: 0.9261

[0.2662290632724762, 0.9261000156402588]
```

```
model.predict(X_test_flattened)
```

```
array([[1.86064839e-02, 3.48662780e-07, 5.03577888e-02, ...,
        9.99750137e-01, 1.01597577e-01, 5.97275257e-01],
       [4.51864928e-01, 3.82184982e-03, 9.99380827e-01, ...,
        8.22585110e-13, 1.86180651e-01, 2.07315942e-09],
       [3.65108252e-04, 9.92665827e-01, 6.52064919e-01, ...,
        1.17013276e-01, 3.66518468e-01, 4.53564823e-02],
       ...,
       [3.54939380e-06, 3.38986683e-06, 1.17710233e-03, ...,
        1.67916000e-01, 5.28737664e-01, 7.06049204e-01],
       [1.08627042e-04, 9.65756481e-05, 9.79038741e-05, ...,
        2.47069711e-05, 6.03402436e-01, 7.64286742e-05],
       [5.58257103e-03, 2.64248179e-10, 1.27466619e-01, ...,
        7.56885221e-09, 2.17109919e-04, 6.13280577e-07]], dtype=float32)
```

```
plt.matshow(X_test[0])
```

```
np.argmax(y_predicted[1])
```

```
2
```

```
##we can use confusion matrix to get a look at all y_predicted values
```

```
y_predicted_labels=[np.argmax(i) for i in y_predicted]##this is to convert the y_predicted lenghty values to match the y_test
#single integer value as below
y_predicted_labels[:5]
```

```
[7, 2, 1, 0, 4]
```

```
y_test[:5]
```

```
array([7, 2, 1, 0, 4], dtype=uint8)
```

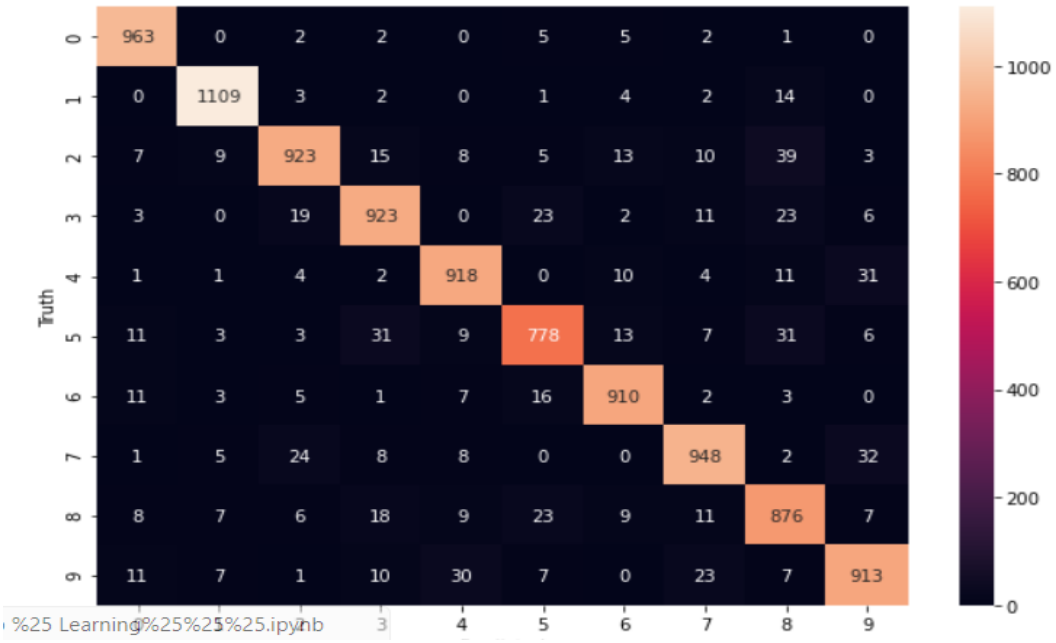
```
cm=tf.math.confusion_matrix(labels=y_test,predictions=y_predicted_labels)
cm
```

```
<tf.Tensor: shape=(10, 10), dtype=int32, numpy=
array([[ 963,    0,    2,    2,    0,    5,    5,    2,    1,    0],
       [   0, 1109,    3,    2,    0,    1,    4,    2,   14,    0],
       [   7,    9,  923,   15,    8,    5,   13,   10,   39,    3],
       [   3,    0,   19,  923,    0,   23,    2,   11,   23,    6],
       [   1,    1,    4,    2,  918,    0,   10,    4,   11,   31],
       [  11,    3,    3,   31,    9,  778,   13,    7,   31,    6],
       [  11,    3,    5,    1,    7,   16,  910,    2,    3,    0],
       [   1,    5,   24,    8,    8,    0,    0,  948,    2,   32],
       [   8,    7,    6,   18,    9,   23,    9,   11,  876,    7],
       [  11,    7,    1,   10,   30,    7,    0,   23,    7,  913]])>
```

```
#we will now use seaborn for a better visualization
```

```
import seaborn as sn
plt.figure(figsize=(10,7))
sn.heatmap(cm,annot=True,fmt='d')
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

```
Text(69.0, 0.5, 'Truth')
```




```
##this means that e.g for 962 times for value 0,our model predicted it to be actually 0
#now for with the 9 it means that for a 2 truth value it predicted wrongly to be 1
#anything not in the diagonal is an error
#
```

```
##now we write the same model buh this time we add an hidden layer to see the accuracy if it improves
```

```
model=keras.Sequential([
    keras.layers.Dense(100,input_shape=(784,),activation='relu'),##with 100 you kinda like specify the number of neurals youd
    #love the model to have(its sort of trial n error) but always less than input_shape
    #for the activation function we will use 'relu'
    keras.layers.Dense(10,activation='sigmoid')##my 2rd layer doesnt need an input
])##this sequantial means i have a stack of layers in myneural network

#output is set as 10,while input is set as 784
#we have also to specify the activiation function which is sigmoid
model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)
model.fit(X_train_flattened,y_train,epochs=5)#epochs is like the no. of iterations
```

```
Epoch 1/5
1875/1875 [==============================] - 25s 7ms/step - loss: 0.2778 - accuracy: 0.9206 0s - loss: 0.285
Epoch 2/5
1875/1875 [==============================] - 10s 5ms/step - loss: 0.1255 - accuracy: 0.9636
Epoch 3/5
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0887 - accuracy: 0.9737
Epoch 4/5
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0680 - accuracy: 0.9793
Epoch 5/5
1875/1875 [==============================] - 13s 7ms/step - loss: 0.0539 - accuracy: 0.9833
```

```
##after my model is trained i can evaluate its perfomance on the test dataset as below
```

```
model.evaluate(X_test_flattened,y_test)
```

```
313/313 [==============================] - 11s 5ms/step - loss: 0.0818 - accuracy: 0.9760

[0.0818413496017456, 0.9760000109672546]
```

```
###now clearly this model with an extra hiddden layer has an improved perfomance compared to the firt one
```

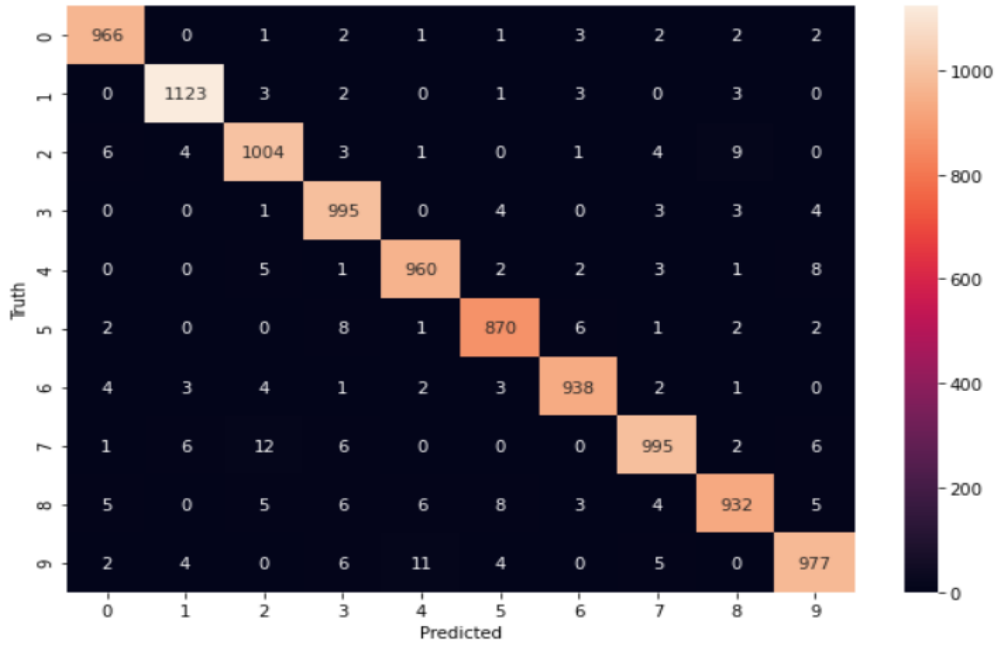
```
##we still check out the y_predicted in the confusion matrix similar to what we did above
```

```
y_predicted=model.predict(X_test_flattened)
y_predicted_labels=[np.argmax(i) for i in y_predicted]
cm=tf.math.confusion_matrix(labels=y_test,predictions=y_predicted_labels)
cm
```

```
<tf.Tensor: shape=(10, 10), dtype=int32, numpy=
array([[ 966,    0,    1,    2,    1,    1,    3,    2,    2,    2],
       [   0, 1123,    3,    2,    0,    1,    3,    0,    3,    0],
       [   6,    4, 1004,    3,    1,    0,    1,    4,    9,    0],
       [   0,    0,    1,  995,    0,    4,    0,    3,    3,    4],
       [   0,    0,    5,    1,  960,    2,    2,    3,    1,    8],
       [   2,    0,    0,    8,    1,  870,    6,    1,    2,    2],
       [   4,    3,    4,    1,    2,    3,  938,    2,    1,    0],
       [   1,    6,   12,    6,    0,    0,    0,  995,    2,    6],
       [   5,    0,    5,    6,    6,    8,    3,    4,  932,    5],
       [   2,    4,    0,    6,   11,    4,    0,    5,    0,  977]])>
```

```
import seaborn as sn
plt.figure(figsize=(10,7))
sn.heatmap(cm,annot=True,fmt='d')
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

```
Text(69.0, 0.5, 'Truth')
```



```
##the number of errors have in the blacks have decreased
#n for a better perfoming models the values in the diagonal have higher numeric values ,n zeros in the black boxes
```

```
######now also if you dont want to use the flatten function as above,,keras comes with its own flatten function
#so we will reuse the same code buh use keras flatten function
```

```
model=keras.Sequential([
    keras.layers.Flatten(input_shape=(28,28)),
    keras.layers.Dense(100,input_shape=(784,),activation='relu'),
    keras.layers.Dense(10,activation='sigmoid')
])
model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)
model.fit(X_train,y_train,epochs=5)
```

```
Epoch 1/5
1875/1875 [==============================] - 24s 8ms/step - loss: 0.2740 - accuracy: 0.9229
Epoch 2/5
1875/1875 [==============================] - 14s 8ms/step - loss: 0.1222 - accuracy: 0.9642
Epoch 3/5
1875/1875 [==============================] - 14s 8ms/step - loss: 0.0852 - accuracy: 0.9745
Epoch 4/5
1875/1875 [==============================] - 19s 10ms/step - loss: 0.0649 - accuracy: 0.9803
Epoch 5/5
1875/1875 [==============================] - 15s 8ms/step - loss: 0.0510 - accuracy: 0.9844

<tensorflow.python.keras.callbacks.History at 0x1f55876d7f0>
```

Activation Functions

```
##we have these several activation funfions
#step function
#sigmoid
#tanh
#relu
#leaky relu
```

```
##now i will write a function for the sigmoid equation as below
```

```
import math

def sigmoid(x):
    return 1/(1+math.exp(-x))
```

```
sigmoid(100)
```

```
1.0
```

```
sigmoid(1)
```

```
0.7310585786300049
```

```
sigmoid(-56)
```

```
4.780892883885469e-25
```

```
#so the sigmoid function always converts any number into a range from 0-1
```

```
##lets also code the tanh equation function as below
```

```
def tanh(x):
    return (math.exp(x)-math.exp(-x))/(math.exp(x)+math.exp(-x))
```

```
#so this function always converts any number into a range from -1 --1
```

```
tanh(-56)
```

```
-1.0
```

```
tanh(50)
```

```
1.0
```

```
tanh(1)
```

```
0.7615941559557649
```

```
##relu function
```

```
def relu(x):
    return max(0,x)
```

```
relu(-7)
```

```
0
```

```
relu(1)
```

```
1
```

```
relu(6)
```

```
6
```

```
##relu converts any number less than 0 to 0 buh any value greater than 0 ,returns the same number
```

```
##leaky relu
```

```
def leaky_relu(x):
    return max(0.1*x,x)
```

```
leaky_relu(-10)
```

```
-1.0
```

```
leaky_relu(8)
```

```
8
```

```
#leaky relu converts all values below 0 by a factor of 0.1 but for values greater than 0,it returns the same values
```

Derivatives ,Matrix Basics

```
import numpy as np
```

```
revenue=np.array([[180,200,220],[24,36,40],[12,18,20]])
expenses=np.array([[80,90,100],[10,16,20],[8,10,10]])
```

```
profit=revenue-expenses
profit
```

```
array([[100, 110, 120],
       [ 14,  20,  20],
       [  4,   8,  10]])
```

```
price_per_unit=np.array([1000,400,1200])
units=np.array([[30,40,50],[5,10,15],[2,5,7]])
```

```
price_per_unit*units##so this operation is not what we need for a toatl price for each so we use the dot product instead
```

```
array([[30000, 16000, 60000],
       [ 5000,  4000, 18000],
       [ 2000,  2000,  8400]])
```

```
np.dot(price_per_unit,units)##sales amount using the dot product
```

```
array([34400, 50000, 64400])
```

```
new_a=2*profit
new_a
```

```
array([[200, 220, 240],
       [ 28,  40,  40],
       [  8,  16,  20]])
```

Loss or Cost Function ¶

```
##in my above model training i used loss as sparse_categorical_crossentropy,buh you can as well use other
#tensorflow loss values as
#binary_crossentropy or Log loss
#categorical_crossentropy
#mean_absolute_error
#mean_squared_error
#this loss function is mainly used in neural network training

#for logistic regression we use Log loss
```

```
import numpy as np

y_predicted=np.array([1,1,0,0,1])
y_true=np.array([0.30,0.7,1,0,0.5])
```

```
def mae(y_true,y_predicted):
    total_error=0
    for yt,yp in zip(y_true,y_predicted):
        total_error+=abs(yt-yp)
    print('Total error: ',total_error)
    mae=total_error/len(y_true)
    print('MAE: ',mae)
    return mae
```

```
mae(y_true,y_predicted)
```

```
Total error:  2.5
MAE:  0.5
```

```
##writing the log loss formula
```

```
-np.mean(y_true*np.log(y_predicted_new)+(1-y_true)*np.log(1-y_predicted_new))##implementing log loss function using numpy
#without having to write the below function
```

```
17.2696280766844
```

```
def log_loss(y_true,y_predicted):
    y_predicted_new=[max(i,epsilon) for i in y_predicted]
    y_predicted_new=[min(i,1-epsilon) for i in y_predicted_new]
    y_predicted_new=np.array(y_predicted_new)
    return -np.mean(y_true*np.log(y_predicted_new)+(1-y_true)*np.log(1-y_predicted_new))
```

```
log_loss(y_true,y_predicted)
```

```
17.2696280766844
```

Gradient Descent for Neural Network

```
import pandas as pd
import tensorflow as tf
from tensorflow import keras
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

```
df=pd.read_excel('C:\\Users\sojore\Documents\insurance.xlsx')
df.head()
```

	age	affordability	bought_insurance
0	22	1	0
1	25	0	0
2	47	1	1
3	52	0	0
4	46	1	1

```
##first i will split my data for training and testing
```

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(df[['age','affordability']],df.bought_insurance,test_size=0.2,random_state=25)
```

```
len(X_train)
```

```
22
```

```
df.shape
```

```
(28, 3)
```

```
##now we need to scale our dataset columns to a better range
```

```
X_train_scalled=X_train.copy()
X_train_scalled['age']=X_train_scalled['age']/100

X_test_scalled=X_test.copy()
X_test_scalled['age']=X_test_scalled['age']/100
```

```
X_train_scalled
```

	age	affordability
0	0.22	1
13	0.29	0
6	0.55	0
17	0.58	1
24	0.50	1
19	0.18	1
25	0.54	1
16	0.25	0
20	0.21	1
3	0.52	0
7	0.60	0
1	0.25	0
5	0.56	1

```
##the reason we are scalling the age is inorder to bring both age and affrodaility to the same scale
```

```
##we will use tensorflow to create a neural network for training my model
```

```
model=keras.Sequential([
    keras.layers.Dense(1,input_shape=(2,),activation='sigmoid',kernel_initializer='ones',bias_initializer='zeros')##2 refers to i
])
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

model.fit(X_train_scalled,y_train,epochs=5000)
```

```
1/1 [==============================] - 0s 15ms/step - loss: 0.7030 - accuracy: 0.5000
Epoch 3/5000
1/1 [==============================] - 0s 9ms/step - loss: 0.7027 - accuracy: 0.5000
Epoch 4/5000
1/1 [==============================] - 0s 9ms/step - loss: 0.7023 - accuracy: 0.5000
Epoch 5/5000
1/1 [==============================] - 0s 32ms/step - loss: 0.7019 - accuracy: 0.5000
Epoch 6/5000
1/1 [==============================] - 0s 9ms/step - loss: 0.7016 - accuracy: 0.5000
Epoch 7/5000
1/1 [==============================] - 0s 12ms/step - loss: 0.7012 - accuracy: 0.5000
Epoch 8/5000
```

```
###now i wanna know my weights1,2.. and bias as well
#in this case we use the get_weights() function
```

```
coef,intercept=model.get_weights()
coef,intercept##so in the aaray the first one is coef(weigt1 and weight 2 ) and then in the 2rd array is the bias
```

```
(array([[5.422585 ],
        [1.4887357]], dtype=float32),
 array([-3.0377512], dtype=float32))
```

```
import math

def sigmoid(x):
    return 1/(1+math.exp(-x))
sigmoid(18)
```

```
0.9999999847700205
```

```
##so now i will write a prediction function which takes both age and afforadability
```

```
def prediction_function(age,affordability):
    weighted_sum=coef[0]*age+coef[1]*affordability+intercept
    return sigmoid(weighted_sum)
```

```
prediction_function(0.47,1)##now i will call my function on the first values 0.47 and 1
#so we got the exact same values as above using the function
```

```
0.7309798197665749
```

```
prediction_function(.18,1)
```

```
0.7605563789104385
```

```
#######now i am going to implement the gradient descent function in python from scratch
```

```
def log_loss(y_true,y_predicted):
    y_predicted_new=[max(i,epsilon) for i in y_predicted]
    y_predicted_new=[min(i,1-epsilon) for i in y_predicted_new]
    y_predicted_new=np.array(y_predicted_new)
    return -np.mean(y_true*np.log(y_predicted_new)+(1-y_true)*np.log(1-y_predicted_new))
```

```
def sigmoid_numpy(x):
    return 1/(1+np.exp(-x))

sigmoid_numpy(np.array([12,0,1]))
```

```
array([0.99999386, 0.5       , 0.73105858])
```

```
##now we write the gradient descent function to get the weights and bias
```

```
##now we write the gradient descent function to get the weights and bias
```

```
def gradient_descent(age,affordability,y_true,epochs):
    w1=w2=1
    bias=0
    rate=0.5##learning rate (you can start with .5 and keep adjusting this value)
    n=len(age)

    for i in range(epochs):
        weighted_sum=w1*age+w2*affordability+bias
        y_predicted=sigmoid_numpy(weighted_sum)

        loss=log_loss(y_true,y_predicted)
        #now we implement the derivertive function for loss
        w1d=(1/n)*np.dot(np.transpose(age),(y_predicted-y_true))##this is how in numpy we implement the 1/n sum of age(y_pred-y_t
        #the above is w1 derivertive
        w2d=(1/n)*np.dot(np.transpose(affordability),(y_predicted-y_true))
        #bias derivertive
        bias_d=np.mean(y_predicted-y_true)

        w1=w1-rate*w1d
        w2=w2-rate*w2d
        bias=bias-rate*bias_d

        print(f'Epoch: {i}, w1: {w1}, w2: {w2}, bias: {bias}, loss: {loss}')

    return w1,w2,bias
```

```
gradient_descent(X_train_scalled['age'],X_train_scalled['affordability'],y_train,1000)
```

```
###so we can stop when the loss value attains levels out or attains some consistency as above
```

```
##now i will do some modifications to my above above function such that i will stop ata a ceratin thresold as my previous functi
```

```
def gradient_descent(age,affordability,y_true,epochs,loss_thresold):
    w1=w2=1
    bias=0
    rate=0.5##learning rate (you can start with .5 and keep adjusting this value)
    n=len(age)

    for i in range(epochs):
        weighted_sum=w1*age+w2*affordability+bias
        y_predicted=sigmoid_numpy(weighted_sum)

        loss=log_loss(y_true,y_predicted)
        #now we implement the derivertive function for loss
        w1d=(1/n)*np.dot(np.transpose(age),(y_predicted-y_true))##this is how in numpy we implement the 1/n sum of age(y_pred-y_t
        #the above is w1 derivertive
        w2d=(1/n)*np.dot(np.transpose(affordability),(y_predicted-y_true))
        #bias derivertive
        bias_d=np.mean(y_predicted-y_true)

        w1=w1-rate*w1d
        w2=w2-rate*w2d
        bias=bias-rate*bias_d

        print(f'Epoch: {i}, w1: {w1}, w2: {w2}, bias: {bias}, loss: {loss}')

        if loss<=loss_thresold:
            break

    return w1,w2,bias
```

```
##these values almost matches the tensorflow function values internally
```

```
##implementing neural network class from scratch in python( instead of using keras or tensorflow class)
```

```
##lets write our own class
```

```
class myNN:
    def __init__(self):
        self.w1=1
        self.w2=1
        self.bias=0

    def fit(self,X,y,epochs,loss_thresold):
        self.w1,self.w2,self.bias=self.gradient_descent(X['age'],X['affordability'],y,epochs,loss_thresold)


    def predict(self,X_test):
        ##note that in any predict function thes only weighted sum and sigmoid
        weighted_sum=self.w1*X_test['age']+self.w2*X_test['affordability']+self.bias
        return sigmoid_numpy(weighted_sum)



    #now we simply put our gradient function in our myNN class
    def gradient_descent(self,age,affordability,y_true,epochs,loss_thresold):
        w1=w2=1
        bias=0
        rate=0.5##learning rate (you can start with .5 and keep adjusting this value)
        n=len(age)

        for i in range(epochs):
            weighted_sum=w1*age+w2*affordability+bias
            y_predicted=sigmoid_numpy(weighted_sum)

            loss=log_loss(y_true,y_predicted)
            #now we implement the derivertive function for loss
            w1d=(1/n)*np.dot(np.transpose(age),(y_predicted-y_true))##this is how in numpy we implement the 1/n sum of age(y_pre
            #the above is w1 derivertive
            w2d=(1/n)*np.dot(np.transpose(affordability),(y_predicted-y_true))
            #bias derivertive
            bias_d=np.mean(y_predicted-y_true)

            w1=w1-rate*w1d
            w2=w2-rate*w2d
            bias=bias-rate*bias_d

            if i%50==0:##this prints at every 50 interval to minimise the lots of printing
                print(f'Epoch: {i}, w1: {w1}, w2: {w2}, bias: {bias}, loss: {loss}')

            if loss<=loss_thresold:
                print(f'Epoch: {i}, w1: {w1}, w2: {w2}, bias: {bias}, loss: {loss}')
                #this means that i wanna see what are my values when am breaking my loop
                break

        return w1,w2,bias
```

```
customModel=myNN()
customModel.fit(X_train_scalled,y_train,epochs=500,loss_thresold=0.4256)
```

```
Epoch: 0, w1: 0.9791614895378375, w2: 0.9491655949517359, bias: -0.11260120780647886, loss: 0.7034009654546431
Epoch: 50, w1: 1.635691989355835, w2: 1.1093071191868196, bias: -1.2621186307254795, loss: 0.5535927791214487
```

Dropout Regularization

```
##ways of trying to control overfitting in deep learning
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sn
```

```
import warnings
warnings.filterwarnings('ignore')

df=pd.read_excel('C:\\Users\sojore\Documents\dropout_data.xlsx',header=None)
df.sample(5)
```

	0	1	2	3	4	5	6	7	8	9	...	51	52	53	54	55	5
30	0.0240	0.0218	0.0324	0.0569	0.0330	0.0513	0.0897	0.0713	0.0569	0.0389	...	0.0162	0.0146	0.0093	0.0112	0.0094	0.005
120	0.0346	0.0509	0.0079	0.0243	0.0432	0.0735	0.0938	0.1134	0.1228	0.1508	...	0.0040	0.0122	0.0107	0.0112	0.0102	0.005
172	0.0180	0.0444	0.0476	0.0698	0.1615	0.0887	0.0596	0.1071	0.3175	0.2918	...	0.0122	0.0114	0.0098	0.0027	0.0025	0.002
118	0.0363	0.0478	0.0298	0.0210	0.1409	0.1916	0.1349	0.1613	0.1703	0.1444	...	0.0115	0.0190	0.0055	0.0096	0.0050	0.006
98	0.1313	0.2339	0.3059	0.4264	0.4010	0.1791	0.1853	0.0055	0.1929	0.2231	...	0.0362	0.0210	0.0154	0.0180	0.0013	0.010

5 rows × 61 columns

```
df.shape
```

```
(208, 61)
```

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.25,random_state=1)
```

```
X_train.shape,X_test.shape
```

```
((156, 60), (52, 60))
```

```
import tensorflow as tf
from tensorflow import keras
```

```
model=keras.Sequential([
    keras.layers.Dense(60,input_dim=60,activation='relu'),
    keras.layers.Dense(30,activation='relu'),
    keras.layers.Dense(15,activation='relu'),
    keras.layers.Dense(1,activation='sigmoid')
])
model.compile(
    loss='binary_crossentropy',
    optimizer='adam',
    metrics=['accuracy']
)
model.fit(X_train,y_train,epochs=100,batch_size=8)
```

```
Epoch 1/100
20/20 [==============================] - 22s 17ms/step - loss: 0.6922 - accuracy: 0.4808
Epoch 2/100
20/20 [==============================] - 0s 6ms/step - loss: 0.6758 - accuracy: 0.7436
Epoch 3/100
20/20 [==============================] - 0s 8ms/step - loss: 0.6601 - accuracy: 0.7564
Epoch 4/100
20/20 [==============================] - 0s 9ms/step - loss: 0.6433 - accuracy: 0.7885
Epoch 5/100
20/20 [==============================] - 0s 5ms/step - loss: 0.6203 - accuracy: 0.7564
Epoch 6/100
20/20 [==============================] - 0s 4ms/step - loss: 0.5926 - accuracy: 0.7692
```

```
###always try dropout layer n see if thes any improvements
```

Imbalanced Dataset

```
##this will aim at improving the F1 scores
```

Convolutional Neural Network (CNN)

```
##the convolution main idea is the feature extraction before applying the  neural network to it
#
```

```
##image classification using CNN
```

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

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

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170500096/170498071 [==============================] - 248s 1us/step

(50000, 32, 32, 3)
```

```
##now i will write a function which takes X,y and return or prints that specified particular image as below
```

```
y_train.shape
```

```
(50000, 1)
```

```
y_train[:5]##from this we can see we have a 2dim array bu we dont need this we need a direct category,so we gonna reshape
#this y_train
```

```
array([[6],
       [9],
       [9],
       [4],
       [1]], dtype=uint8)
```

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

```
array([6, 9, 9, 4, 1], dtype=uint8)
```

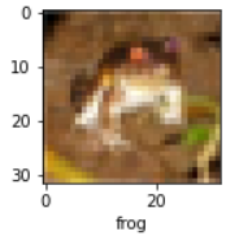
```
classes=['airplane','automobile','bird','cat','deer','dog','frog','horse','ship','truck']
```

```
classes[9]
```

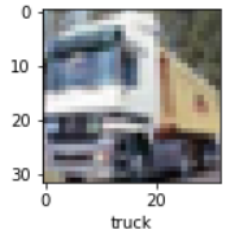
```
'truck'
```

```
def plot_sample(X,y,index):
    plt.figure(figsize=(15,2))
    plt.imshow(X[index])
    plt.xlabel(classes[y[index]])
```

```
plot_sample(X_train,y_train,0)
```




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




```
X_train=X_train/255
X_test=X_test/255#so with this the values are normalized
```

```
##so first we will build an Artificial Neural Network to see how it perfoms and then we will then
#build our Convolutional Neural NEtwork
```

```
#for the ANN we have

ann=models.Sequential([
    layers.Flatten(input_shape=(32,32,3)),
    layers.Dense(3000,activation='relu'),
    layers.Dense(1000,activation='relu'),
    layers.Dense(10,activation='sigmoid')
])
ann.compile(
    optimizer='SGD',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)
ann.fit(X_train,y_train,epochs=2)#i had to do 2 epochs since it will take forever to perfom 5 epochs since i am runnig
#my ANN on my CPU which might be slow compared to using a GPU
```

```
Epoch 1/2
1563/1563 [==============================] - 653s 399ms/step - loss: 1.8063 - accuracy: 0.3606
Epoch 2/2
1563/1563 [==============================] - 461s 295ms/step - loss: 1.6194 - accuracy: 0.4283

<tensorflow.python.keras.callbacks.History at 0x1bbe25fcbb0>
```

```
#sparse_categorical_crossentropy ,we use this when the data is like exact value of 1,2  i.e not one hot encoded
#buh when we have one hot encoded values we use the categorical_crossentropy
```

```
ann.evaluate(X_test,y_test)
```

```
##so from above we can see that ANN is perfoming very bad for our classification
```

```
#so now i can also print a classification report
```

```
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]

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

```
Classification Report :
               precision    recall  f1-score   support

           0       0.53      0.39      0.45      1000
           1       0.32      0.78      0.45      1000
           2       0.32      0.30      0.31      1000
           3       0.35      0.23      0.28      1000
           4       0.45      0.24      0.32      1000
           5       0.38      0.34      0.36      1000
           6       0.43      0.54      0.48      1000
           7       0.70      0.23      0.35      1000
           8       0.42      0.72      0.53      1000
           9       0.50      0.23      0.31      1000

    accuracy                           0.40     10000
   macro avg       0.44      0.40      0.38     10000
weighted avg       0.44      0.40      0.38     10000
```

```
###now we gonna use CNN to improve the perfomance of our model prediction
#as follows
```

```
################
```

```
cnn=models.Sequential([
    #cnn layers
    #so you can have several layers for convolution as pooling layers as well like in our case below
    layers.Conv2D(filters=32,activation='relu',kernel_size=(3,3),input_shape=(32,32,3)),
    layers.MaxPooling2D((2,2)),

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


    #Dense
    #so this down here is my dense network which in above it am gonna put my cnn layers for pooling,feature extraction as on..
    layers.Flatten(),###we dont need to specifily the input_shape=(32,32,3) since the system is gonna detect it automatically
    #since we in a midlle layers
    layers.Dense(64,activation='relu'),#i will only have 64 neural layers since the cnn will have done  most of the task
    layers.Dense(10,activation='softmax')##importance of softmax is that it will normalize my probability
])

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

cnn.fit(X_train,y_train,epochs=2)##i will do 2 epochs because im using a cpu to run my tensorflow neural networks ,
#gpu does it faster
```

```
Epoch 1/2
1563/1563 [==============================] - 374s 208ms/step - loss: 1.4387 - accuracy: 0.4818
Epoch 2/2
1563/1563 [==============================] - 243s 155ms/step - loss: 1.1004 - accuracy: 0.6135
```

```
##Data Augmentation to address overfitting

#this works by trying to generate new samples(this can include taking a small sample and dupicating it) or in the case of
#image classification you can appy some transformation to a small sample like
#horizintal flip,contrast,rotation,zoom ....
import matplotlib.pyplot as plt
import numpy as np
import cv2
import os
import PIL
import tensorflow as tf

from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
```

```
dataset_url='https://storage.googleapis.com/download.tensorflow.org/example_images/flower_photos.tgz'
```

```
##Transfer learning
#here we use a pre-trained model and re-train it again for a new problem
#########the above i will need to revisit the tutorials to write the codes now i have the modules already installed succeesfully
```

```
#image classification vs object detection vs image segmentation
```

```
#####YOLO algorithm
#
```

##Object detection using YOLO

Recurrent Neural Network (RNN)

```
##this is mainly used for (NLP) natural language proccessing tasks
#CNN is used for image processing
```

```
#vanishing and exploding gradients
#when you  have a very low gradient ,your learning process will be slow(vanishing gradients)
#we special types of RNN which adrresses and solves the issue of short memory problem
#i.e GRU(Gated Recurrent Units) and LSTM(Long Short Term Memory)
#Bidirectional RNN
```

```
import numpy as np
from tensorflow.keras.preprocessing.text import one_hot
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Embedding
```

```
reviews=[
    'nice food',
    'amazing restaurant',
    'too good',
    'just loved it',
    'will go again',
    'horrible food',g
    'never go there',
    'poor service',
    'poor quality',
    'needs improvement'
]
sentiment=np.array([1,1,1,1,1,0,0,0,0,0])
```

```
##we will first convert these reviews into one_hot vector
```

```
one_hot('amazing_restaurant',50)
```

```
[35, 6]
```

```
vocab_size=30
encoded_reviews=[one_hot(d,vocab_size) for d in reviews]
print(encoded_reviews)
```

```
[[8, 4], [9, 25], [8, 23], [7, 20, 13], [3, 15, 27], [6, 4], [15, 15, 20], [20, 6], [20, 11], [17, 25]]
```

```
encoded_reviews
```

```
[[8, 4],
 [9, 25],
 [8, 23],
 [7, 20, 13],
 [3, 15, 27],
 [6, 4],
 [15, 15, 20],
 [20, 6],
 [20, 11],
 [17, 25]]
```

```
##now we need to do padding ,i.e some terms have 2 values n others with 3,so we need to append zero to those
#text with 2 values to attain a standard 3 value format
```

```
max_length=3#can do 3 or 4
padded_reviews=pad_sequences(encoded_reviews,maxlen=max_length,padding='post')
print(padded_reviews)
```

```
[[ 8  4  0]
 [ 9 25  0]
```

```
embeded_vector_size=4

model=Sequential()##the first layer is an embedding layer
model.add(Embedding(vocab_size,embeded_vector_size,input_length=max_length,name='embedding'))
#second layer is flatten layer
model.add(Flatten())
#next layer is one layer sigmoid activation funtion
model.add(Dense(1,activation='sigmoid'))
```

```
X=padded_reviews
y=sentiment
```

```
model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 3, 4)              120
_________________________________________________________________
flatten_3 (Flatten)          (None, 12)                0
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 13
=================================================================
Total params: 133
Trainable params: 133
Non-trainable params: 0
_________________________________________________________________
```

now we gonna train our model

```
model.fit(X,y,epochs=50,verbose=0)
```

```
model.fit(X,y,epochs=50,verbose=0)
```

```
<tensorflow.python.keras.callbacks.History at 0x203b68a0c40>
```

```
loss,accuracy=model.evaluate(X,y)
accuracy
```

```
1/1 [==============================] - 0s 308ms/step - loss: 0.6399 - accuracy: 1.0000

1.0
```

```
weights=model.get_layer('embedding').get_weights()[0]
len(weights)
```

```
30
```

```
weights[8]
```

```
array([-0.03325002, -0.06556621, -0.02027671,  0.08029708], dtype=float32)
```

```
weights[9]
```

```
array([-0.08811332, -0.03716795,  0.06202875, -0.06848854], dtype=float32)
```

Word2Vec

```
#this is a method usefull in converting words to a vector
```

```
##first i will import the neccesary important libraries to use in this topic
```

```
import matplotlib.pyplot as plt
import numpy as np
import cv2
import os
import PIL

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

```
dataset_url='https://storage.googleapis.com/download.tensorflow.org/example_images/flower_photos.tgz'
##we are downloading the flowers dataset from google
data_dir=tf.keras.utils.get_file('flower_photos',origin=dataset_url, cache_dir='C:\\Users\sojore\Documents',untar=True)
#so the above code means we saving the data into a folder called flower_phots ,n the cache_dir is the directory i want ot saved
#untar=True ,means that since we are getting a zip file from the download,then we unzip it using this function(untar)
```

```
data_dir
```

```
'C:\\Users\\sojore\\Documents\\datasets\\flower_photos'
```

```
##next now im gonna convert my data_dir into a pathlib which makes it easier to access
```

```
import pathlib
data_dir=pathlib.Path(data_dir)
data_dir
```

```
WindowsPath('C:/Users/sojore/Documents/datasets/flower_photos')
```

```
##so we can do the following,meaning that,give me all the images which has an extension jpg
list(data_dir.glob('*/*.jpg'))
```

```
##so also i can get the length(count) of all the images as follows
len(list(data_dir.glob('*/*.jpg')))
```

```
3669
```

```
##so if i wanna get only the rose images i do the following
roses=list(data_dir.glob('roses/*'))
roses[:5]
```

```
[WindowsPath('C:/Users/sojore/Documents/datasets/flower_photos/roses/10090824183_d02c613f10_m.jpg'),
 WindowsPath('C:/Users/sojore/Documents/datasets/flower_photos/roses/102501987_3cdb8e5394_n.jpg'),
 WindowsPath('C:/Users/sojore/Documents/datasets/flower_photos/roses/10503217854_e66a804309.jpg'),
 WindowsPath('C:/Users/sojore/Documents/datasets/flower_photos/roses/10894627425_ec76bbc757_n.jpg'),
 WindowsPath('C:/Users/sojore/Documents/datasets/flower_photos/roses/110472418_87b6a3aa98_m.jpg')]
```

```
##so we wanna show all the above images using the pillow module
#we use the command below to show images in PIL
PIL.Image.open(str(roses[0]))
```

```
PIL.Image.open(str(roses[1]))
```

```
#now i will create a flower dictionary containing all paths to different types of flowers
#this will give all the paths of the flower images
```

```
flower_images_dict['roses']
```

```
[WindowsPath('C:/Users/sojore/Documents/datasets/flower_photos/roses/10090824183_d02c613f10_m.jpg'),
 WindowsPath('C:/Users/sojore/Documents/datasets/flower_photos/roses/102501987_3cdb8e5394_n.jpg'),
 WindowsPath('C:/Users/sojore/Documents/datasets/flower_photos/roses/10503217854_e66a804309.jpg'),
 WindowsPath('C:/Users/sojore/Documents/datasets/flower_photos/roses/10894627425_ec76bbc757_n.jpg'),
 WindowsPath('C:/Users/sojore/Documents/datasets/flower_photos/roses/110472418_87b6a3aa98_m.jpg'),
 WindowsPath('C:/Users/sojore/Documents/datasets/flower_photos/roses/11102341464_508d558dfc_n.jpg'),
 WindowsPath('C:/Users/sojore/Documents/datasets/flower_photos/roses/11233672494_d8bf0a3dbf_n.jpg'),
 WindowsPath('C:/Users/sojore/Documents/datasets/flower_photos/roses/11694025703_9a906fedc1_n.jpg'),
 WindowsPath('C:/Users/sojore/Documents/datasets/flower_photos/roses/118974357_0faa23cce9_n.jpg'),
 WindowsPath('C:/Users/sojore/Documents/datasets/flower_photos/roses/11944957684_2cc806276e.jpg'),
 WindowsPath('C:/Users/sojore/Documents/datasets/flower_photos/roses/12045735155_42547ce4e9_n.jpg'),
 WindowsPath('C:/Users/sojore/Documents/datasets/flower_photos/roses/12165480946_c4a3fe182d_n.jpg'),
 WindowsPath('C:/Users/sojore/Documents/datasets/flower_photos/roses/12202373204_34fb07205b.jpg'),
 WindowsPath('C:/Users/sojore/Documents/datasets/flower_photos/roses/12238827553_cf427bfd51_n.jpg'),
 WindowsPath('C:/Users/sojore/Documents/datasets/flower_photos/roses/12240165555_98625b1e88_n.jpg'),
 WindowsPath('C:/Users/sojore/Documents/datasets/flower_photos/roses/12240303_80d87f77a3_n.jpg'),
 WindowsPath('C:/Users/sojore/Documents/datasets/flower_photos/roses/12240577184_b0de0e53ea_n.jpg'),
 WindowsPath('C:/Users/sojore/Documents/datasets/flower_photos/roses/12243068283_ee4c2683e2_n.jpg'),
 WindowsPath('C:/Users/sojore/Documents/datasets/flower_photos/roses/12243069253_e512464095_n.jpg'),
```

```
##i wanna also have a labels dictionary as below
```

```
flowers_labels_dict={
    'roses':0,
```

```
flowers_labels_dict={
    'roses':0,
    'daisy':1,
    'dandelion':2,
    'sunflowers':3,
    'tulips':4
}
```

```
#we can use the open cv module to read these files into an open cv object(sort of like a numpy array)
```

```
str(flower_images_dict['roses'][0])
```

```
'C:\\Users\\sojore\\Documents\\datasets\\flower_photos\\roses\\10090824183_d02c613f10_m.jpg'
```

```
img=cv2.imread(str(flower_images_dict['roses'][0]))
img
```

```
array([[[ 0, 15,  6],
        [ 0, 15,  6],
        [ 0, 17,  8],
        ...,
        [24, 51, 31],
        [25, 52, 32],
        [26, 53, 33]],

       [[ 7, 22, 14],
        [ 7, 22, 14],
        [ 6, 21, 13],
```

```
img.shape
```

```
(240, 179, 3)
```

```
#now we will use open cv inorder to resize our images to some standard size
#this is because some of the images have diffent dimensions
cv2.resize(img,(180,180)).shape
```

```
(180, 180, 3)
```

```
X,y=[],[]

for flower_name,images in flower_images_dict.items():
    print(flower_name)
    print(len(images))
```

```
roses
640
daisy
1
dandelion
898
sunflowers
0
tulips
799
```

```
X,y=[],[]

for flower_name,images in flower_images_dict.items():
    for image in images:
        img=cv2.imread(str(image))
        resized_img=cv2.resize(img,(180,180))##we resize because ML models works with images with one dimension
        X.append(resized_img)
        y.append(flowers_labels_dict[flower_name])#this code i will get the number for each flower name
```

```
##now i will convert my X,y into a simple numpy array
```

```
X=np.array(X)
y=np.array(y)
```

```
X.shape
```

```
(640, 180, 180, 3)
```

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,random_state=0)
```

```
len(X_train)
```

```
480
```

```
len(X_test)
```

```
160
```

```
##now we need to do scaling
```

```
X_train_scaled=X_train/255
X_test_scaled=X_test/255
```

```

```

```
X_train_scaled[0]
```

```
array([[[0.56470588, 0.68235294, 0.65490196],
        [0.51764706, 0.67058824, 0.66666667],
        [0.47843137, 0.65098039, 0.6627451 ],
```

```
##now we are ready to build our model using CNN
```

```
num_classes=5
model=Sequential([
    layers.Conv2D(16,3,padding='same',activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(32,3,padding='same',activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(64,3,padding='same',activation='relu'),
    layers.MaxPooling2D(),

    #add Dense network
    layers.Flatten(),
    layers.Dense(128,activation='relu'),
    layers.Dense(num_classes)
])
model.compile(
    optimizer='adam',
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=['accuracy']
)
model.fit(X_train_scaled,y_train,epochs=5)
```

```
Epoch 1/5
15/15 [==============================] - 80s 3s/step - loss: 0.1023 - accuracy: 0.9542
Epoch 2/5
15/15 [==============================] - 44s 3s/step - loss: 0.0000e+00 - accuracy: 1.0000
Epoch 3/5
15/15 [==============================] - 37s 2s/step - loss: 0.0000e+00 - accuracy: 1.0000
Epoch 4/5
```

- **Databases**

Databases are very useful in storing data, we will be studying and looking at some of the most important steps in SQL database, there are other databases such as MongoDB,

Its important to note that as a data scientist you will be required to connect to these databases, in order to extract data from them, clean the data and build models from them

In this section, I will be sharing the most important techniques and procedures we use in MySQL

Remember to have your MySQL workbench installed:

creating a database using a query

```
Create Database Sample2
```

rename a database

```
Alter Database  Sample2 Modify Name = Sample3
```

dropping or deleting a database

```
Drop database Sample3
```

```
--creating tables
create table tblGender
(
ID int NOT NULL Primary key,
Gender nvarchar(50) not null

)
```

```
--adding foreign keys using a query

alter table tblPerson add constraint  tblPerson_GenderId_FK

foreign key (GenderId) references tblGender(ID)


--insering rows into a table using a query

insert into tblPerson (ID,Name,Email) values (7,'rich','r@r.com')


--we can use a default constraint to set the null values always to take
unknown  number 3

alter table tblPerson

add constraint DF_tblPerson_GenderId

default 3 for GenderId


select * from tblGender

select * from tblPerson


--insering rows into a table using a query

insert into tblPerson (ID,Name,Email,GenderId) values
(10,'johnny','jo@r.com',null)

--we can use a default constraint to set the null values always to take
unknown  number 3

alter table tblPerson

add constraint DF_tblPerson_GenderId

default 3 for GenderId
```

```
--how to drop a conrtaint

alter table tblPerson

drop constraint DF_tblPerson_GenderId

--how to delete rows in a table

delete from tblPerson where ID=7

delete from tblPerson where ID=10

--check constraint -its used to limit the range of values which can be entered into a specific column

alter table tblPerson

add constraint CK_tblPerson_Age check (Age>0 and Age<150)

--check constraint -its used to limit the range of values which can be entered into a specific column

alter table tblPerson

add constraint CK_tblPerson_Age check (Age>0 and Age<150)

insert into tblPerson values (4,'sara','s@s.com',2,3000)

---how to drop check constraint

alter table tblPerson

drop constraint CK_tblPerson_Age
```

```
select * from tblPerson1



--to perform the below insert operation ,i will need to turn the identity insert  on

set IDENTITY_INSERT tblPerson1 on

insert into tblPerson1(personid,name) values (1,'jane')


delete from tblPerson1 where personid=1


--to perform the below insert operation ,i will need to turn the identity insert  on

set IDENTITY_INSERT tblPerson1 off

insert into tblPerson1 values ('martin')


delete from tblPerson1 where personid=1




delete from tblPerson1


select * from tblPerson1



--now if we need to reset the identity column to start from 1 we do the following
```

```
DBCC checkident (tblPerson1,reseed,0)

set IDENTITY_INSERT tblPerson1 off
insert into tblPerson1 values ('martin')

delete from tblPerson1 where personid=1



create table test1
(
id int identity(1,1),
value nvarchar(20)
)

create table test2
(
id int identity(1,1),
value nvarchar(20)
)
--now i want to retrive the last inserted row i do the following
select SCOPE_IDENTITY()
select @@IDENTITY


insert into test1 values ('x')
```

```
select * from test1

select * from test2


--we gonna create a triggger such that when we insert a row into test1
,another one is automatically inserted by the trigger to test2

create trigger trForinsert on test1 for insert

as

begin

    insert into test2 values ('yyy')

end




--how to create a unique key constraint

--we can create one using the desing window by naviagating to tables
design,select indexes /keys n...

select* from tblPerson

--using a query to create a unique key constraint

alter table tblPerson

add constraint UQ_tblPerson_Email Unique(Email)


insert into tblPerson values(1,'abc','a@a.com',1,20)

insert into tblPerson values(2,'xyz','a@a.com',1,20)

--running the 2 above queries will give a unique key constraint error


--how to drop constraint

alter table tblPerson
```

drop constraint

--using select keyword

select * from tblPerson

select distinct Name,City from tblPerson

select * from tblPerson where City ='london'

select * from tblPerson where City <>'london'--not equal to operator

select * from tblPerson where Age =20 or Age=23 or Age=29

--or we can use the in operator as below

select * from tblPerson where Age in (20,23,29)

select * from tblPerson where Age between 20 and 25

--like operator

select * from tblPerson where City like 'l%'--this mean we need all city name starting with a l letter

select * from tblPerson where Email not like '%@%'--this filters ant email without the @ symbol

--now if i wanna select all eamils with only character before the @ symbol we do the following

select * from tblPerson where Email like '_@_.com'

```
select * from tblPerson where Name like '[mst]%'--we need only names starting with characters mst(i.e m, s,& t)

--the opposite of the above is given by

select * from tblPerson where Name like '[^mst]%'


--combining multiple conditions

select * from tblPerson where (City='london' or City='mumbai') and Age>25


--ordering the names in ascending order, sorting

select * from tblPerson order by Name desc

select * from tblPerson order by Name desc,Age asc


--to only select top rows

select top 2 * from tblPerson

select top 2 Name,Age from tblPerson

select top 50 percent * from tblPerson


--to determine the oldest person in the list we can use the top keyword

select top 1* from tblPerson order by Age desc



--group by

select * from tblEmployee


insert into tblEmployee values('todd','male',2800,'sydney')

insert into tblEmployee values('russel','male',7800,'london ')
```

```
--group by

select * from tblEmployee

select sum(Salary) from tblEmployee
select max(Salary) from tblEmployee
select min(Salary) from tblEmployee

select City , sum(Salary)  as TotalSalary
from tblEmployee
group by City

select City ,Gender, sum(Salary)  as TotalSalary
from tblEmployee
group by City,Gender
order by City

--to get the total count of employees
select count(*) from tblEmployee
select count(ID) from tblEmployee

select City ,Gender, sum(Salary)  as TotalSalary,count(ID) as [Total
Employees]
from tblEmployee
```

```
group by City,Gender

order by City


--filtering the gender

--here the aggregation is done only after the grouping by male is done unlike in having clause where the aggregation is done before grouping by choose target

select City ,Gender, sum(Salary)  as TotalSalary ,count(ID) as [Total Employees]

from tblEmployee

where Gender ='male'

group by City,Gender

order by City


--we can also use having clause to filter

select City ,Gender, sum(Salary)  as TotalSalary ,count(ID) as [Total Employees]

from tblEmployee

group by City,Gender

having Gender ='male'


select * from tblEmployee where sum(Salary) >5000--this is not allowed buh with the having clause you can achieve this


select City ,Gender, sum(Salary)  as TotalSalary ,count(ID) as [Total Employees]

from tblEmployee

group by City,Gender
```

```
having sum(Salary) >5000


--joins
--inner join--only matching rows in both tables are retrieved
set IDENTITY_INSERT tblEmployee on
select * from tblEmployee
delete from tblEmployee where ID=1005
insert into tblEmployee(ID,Name,Gender,Salary,City) values
(10,'russel','male',8800,null)


--joins
--inner join--only matching rows in both tables are retrieved
create table tblDepartment
(
Id int primary key,
DepartmentName nvarchar(20),
Location nvarchar(20),
DepartmentHead nvarchar(20)
)
select * from tblDepartment
insert into tblDepartment values (1,'IT','london','rick')
insert into tblDepartment values (2,'Payroll','delhi','ron')
insert into tblDepartment values (3,'HR','new york','christie')
insert into tblDepartment values (4,'other department','sydney','cindrela')
```

```
create table tblEmpoyee1
(
ID int primary key,
Name nvarchar(20),
Gender nvarchar(20),
Salary int,
DepartmentId int,
)

insert into tblEmpoyee1 values(1,'tom','male',4000,1)
insert into tblEmpoyee1 values(2,'pam','female',3000,3)
insert into tblEmpoyee1 values(3,'john','male',3500,1)
insert into tblEmpoyee1 values(4,'sam','male',4500,2)
insert into tblEmpoyee1 values(5,'todd','male',2800,2)
insert into tblEmpoyee1 values(6,'ben','male',7000,1)
insert into tblEmpoyee1 values(7,'sara','female',4800,3)
insert into tblEmpoyee1 values(8,'valarie','female',5500,1)
insert into tblEmpoyee1 values(9,'james','male',6500,null)
insert into tblEmpoyee1 values(10,'russel','male',8800,null)


--joins
--inner join--only matching rows in both tables are retrieved
select * from tblEmpoyee1
select * from tblDepartment
```

```
select Name,Gender ,Salary ,DepartmentName

from tblEmpoyee1

inner join tblDepartment

on tblEmpoyee1.DepartmentId=tblDepartment.Id


--left join or left outer join--returns all matching rows  + non-matching rows on left table

select Name,Gender ,Salary ,DepartmentName

from tblEmpoyee1

left join tblDepartment

on tblEmpoyee1.DepartmentId=tblDepartment.Id


--right join or right outer join--returns all matching rows + non matching rows on the right table

select Name,Gender ,Salary ,DepartmentName

from tblEmpoyee1

right outer join tblDepartment

on tblEmpoyee1.DepartmentId=tblDepartment.Id


--full join or full outer join--returns all rows from both left and right tables including the non-matching rows

select Name,Gender ,Salary ,DepartmentName

from tblEmpoyee1

full outer join tblDepartment

on tblEmpoyee1.DepartmentId=tblDepartment.Id


--cross join
```

```
--doesnt have an on clause
select Name,Gender ,Salary ,DepartmentName
from tblEmpoyee1
cross join tblDepartment


--advanced or intelligent joins--for returning only non matching rows from both left and right tables
select * from tblEmpoyee1
select * from tblDepartment

select Name,Gender,Salary,DepartmentName
from tblEmpoyee1
left join tblDepartment
on tblEmpoyee1.DepartmentId=tblDepartment.Id
where tblEmpoyee1.DepartmentId is null

--or other way
select Name,Gender,Salary,DepartmentName
from tblEmpoyee1
left join tblDepartment
on tblEmpoyee1.DepartmentId=tblDepartment.Id
where tblDepartment.Id is null

--for the right join
select Name,Gender,Salary,DepartmentName
from tblEmpoyee1
```

```
right join tblDepartment
on tblEmpoyee1.DepartmentId=tblDepartment.Id
where tblEmpoyee1.DepartmentId is null

--for non matching values in both tables
select Name,Gender,Salary,DepartmentName
from tblEmpoyee1
full join tblDepartment
on tblEmpoyee1.DepartmentId=tblDepartment.Id
where tblEmpoyee1.DepartmentId is null
or tblDepartment.Id is null


--self join--joining a table by itself
create table tblEmployee2
(
EmployeeID int primary key,
Name nvarchar(20),
ManagerID int
)

insert into tblEmployee2 values(1,'mike',3)
insert into tblEmployee2 values(2,'rob',1)
insert into tblEmployee2 values(3,'todd',null)
insert into tblEmployee2 values(4,'ben',1)
insert into tblEmployee2 values(5,'sam',1)
```

```
--self join--joining a table by itself
select * from tblEmployee2

select E.Name as Employee ,M.Name as Manager
from tblEmployee2 E
left join tblEmployee2 M
on E.ManagerID=M.EmployeeID

--for inner join
select E.Name as Employee ,M.Name as Manager
from tblEmployee2 E
inner join tblEmployee2 M
on E.ManagerID=M.EmployeeID

--for a cross join
select E.Name as Employee ,M.Name as Manager
from tblEmployee2 E
cross join tblEmployee2 M


--replacing null values in SQL server
select * from tblEmployee2
```

```
select E.Name as Employee,M.Name as Manager
from  tblEmployee2 E
left join tblEmployee2 M
on  E.ManagerID=M.EmployeeID

--isnull function
select isnull('sojore','No manager') as Manager
select isnull('null','No manager') as Manager

select E.Name as Employee,isnull(M.Name,'No manager') as Manager
from  tblEmployee2 E
left join tblEmployee2 M
on  E.ManagerID=M.EmployeeID


--using coalesce function to replace nulls
select coalesce('sojore','No manager') as Manager
select coalesce('null','No manager') as Manager

select E.Name as Employee,coalesce(M.Name,'No manager') as Manager
from  tblEmployee2 E
left join tblEmployee2 M
on  E.ManagerID=M.EmployeeID

--using case statement function to replace nulls
--case when expression then ' ' else ' ' end
```

```
select E.Name as Employee,case when M.Name is null then 'No manager'
else M.Name end as Manager

from  tblEmployee2 E

left join tblEmployee2 M

on  E.ManagerID=M.EmployeeID

--coalesce function--this returns the first non null value



create table tblIndiacustomers

(

Id int primary key,

Name nvarchar(20),

Email nvarchar(50)

)


create table tblUKcustomers

(

Id int primary key,

Name nvarchar(20),

Email nvarchar(50)

)


insert into tblIndiacustomers values(1,'raj','r@r.com')

insert into tblIndiacustomers values(2,'sam','s@s.com')


insert into tblUKcustomers values(1,'ben','b@b.com')
```

```
insert into tblUKcustomers values(1,'sam','s@s.com')


--union and union all
--union all
select * from tblIndiacustomers
union all
select * from tblUKcustomers
order by Name

--union
select * from tblIndiacustomers
union
select * from tblUKcustomers

--union kinda combines rows from different tables wherelse join combine columns in different tables
--stored procedure
--this is basically a group of transact SQL statements
select * from tblEmployee3
create table tblEmployee3
(
Id int primary key,
Name nvarchar(20),
Gender nvarchar(50),
DepartmentId int
)
```

```
insert into tblEmployee3 values(1,'sam','male',1)
insert into tblEmployee3 values(2,'ram','male',1)
insert into tblEmployee3 values(3,'sara','female',3)
insert into tblEmployee3 values(4,'todd','male',2)
insert into tblEmployee3 values(5,'john','male',3)
insert into tblEmployee3 values(6,'sara','female',2)
insert into tblEmployee3 values(7,'james','male',1)
insert into tblEmployee3 values(8,'rob','male',2)
insert into tblEmployee3 values(9,'steve','male',1)
insert into tblEmployee3 values(10,'pam','female',2)


--stored procedure
--this is basically a group of transact SQL statements
select * from tblEmployee3


--i will create a stored procedure to call the below query if i need to use it
create procedure spGetemployees--or create proc
as
begin
    select Name,Gender from tblEmployee3
end

--executing the sp
spGetemployees
```

```
--creating a sp with input parameters
create proc spGetemployeesbygenderanddepartment
@Gender nvarchar(20),
@DepartmentId int
as
begin
    select Name,Gender,DepartmentId from tblEmployee3 where
Gender=@Gender and DepartmentId=@DepartmentId
end

spGetemployeesbygenderanddepartment 'Male',1
spGetemployeesbygenderanddepartment 'Male',2

--changing the defination of a sp
alter procedure spGetemployees--or create proc
as
begin
    select Name,Gender from tblEmployee3 order by Name
end

spGetemployees
--to drop a sp
drop procedure spGetemployees

--encrypting a sp
```

```
alter proc spGetemployeesbygenderanddepartment
@Gender nvarchar(20),
@DepartmentId int
with encryption
as
begin
    select Name,Gender,DepartmentId from tblEmployee3 where
Gender=@Gender and DepartmentId=@DepartmentId
end

--reading the content of the sp
sp_helptext spGetemployeesbygenderanddepartment



--stored procedures with output parameters
select * from tblEmployee3
create proc spGetemployeecountbygender
@Gender nvarchar(20),
@Employeecount int output
as
begin
    select @Employeecount =count(Id) from tblEmployee3 where
Gender=@Gender
end

--how to execute sp with output paramters
```

```
declare @Totalcount int
execute spGetemployeecountbygender 'male',@Totalcount output
if(@Totalcount is null)
    print '@Totalcount is null'
else
    print @Totalcount

sp_help spGetemployeecountbygender
sp_helptext spGetemployeecountbygender--to view the definition



--stored procedure with return values (or output parameters)
select * from tblEmployee3
create proc spGettotalcount1
@Totalcount int out
as
begin
    select @Totalcount=count(Id) from tblEmployee3
end

declare @Total int
execute spGettotalcount1 @Total out
print @Total
```

```
--lets do the same as above buh using a return values
create proc spGettotalcount2
as
begin
    return(select count(Id) from tblEmployee3)
end

declare @Total int
execute @Total=spGettotalcount2
print @Total

--we look at different scenarios where output parameters can be used rather than return values
create proc spGetnamebyid
@Id int,
@Name nvarchar(20) output
as
begin
    select @Name=Name from tblEmployee3 where Id =@Id
end

declare @Name nvarchar(20)
execute spGetnamebyid 1,@Name out
print 'Name = ' + @Name
```

```
--we gonna do the same buh using return values

create proc spGetnamebyid1
@Id int
as
begin
    return (select Name from tblEmployee3 where Id =@Id)
end

declare @Name nvarchar(20)
execute @Name=spGetnamebyid1 1
print 'Name = ' + @Name
----executing this will result to an error


--commonly used string functions
select ASCII('A') --for small letters i use 'a' inplace of 'A'
--printing out the alphabet using a while loop and ascii function
select CHAR(65)
declare @start int
set @start =65
while (@start <=90)
begin
    print CHAR(@start)
    set @start=@start +1
end
```

```
--for small letters I can first check the ascii value for 'a' and then proceed to do the following

select ASCII('a')

select CHAR(97)

declare @start int

set @start =97

while (@start <=122)

begin

    print CHAR(@start)

    set @start=@start +1

end

--for numerical values

select ASCII('0')

select CHAR(48)

declare @start int

set @start =48

while (@start <=57)

begin

    print CHAR(@start)

    set @start=@start +1

end


--ltrim--removes blanks on the left side of a value,rtrim,lower(),upper()-case,len() ...

select  ltrim('      hello')

select firstname, len(ltrim(firstname)) as [total characters] from tblemployee
```

```
create table tblEmployee4
(
Id int primary key,
Firstname nvarchar(20),
Lastname nvarchar(20),
Email nvarchar(50)
)
insert into tblEmployee4 values (1,'sam','sony','sam@aaa.com')
insert into tblEmployee4 values (2,'ram','barber','ram@aaa.com')
insert into tblEmployee4 values (3,'sara','sanosky','sara@ccc.com')
insert into tblEmployee4 values (4,'todd','gartner','todd@bbb.com')
insert into tblEmployee4 values (5,'john','grover','john@aaa.com')
insert into tblEmployee4 values (6,'sana','lenin','sana@ccc.com')
insert into tblEmployee4 values (7,'james','bond','james@bbb.com')
insert into tblEmployee4 values (8,'rob','hunter','rob@ccc.com')
insert into tblEmployee4 values (9,'steve','wilson','steve@aaa.com')
insert into tblEmployee4 values (10,'pam','broker','pam@bbb.com')


select * from tblEmployee4

select LEFT('ABCDEF',3)
select RIGHT('ABCDEF',3)
```

```
--charindex()---i wanna find the position of the @ index in the below expression

select CHARINDEX('@', 'sara@aaa.com',1)--this function checks the starting position of the specified value


--substring

select SUBSTRING('sara@aaa.com',6,7)

--we can do the same as below

select SUBSTRING('pam@bbb.com',CHARINDEX('@','pam@bbb.com')+1,

LEN('pam@bbb.com')-CHARINDEX('@','pam@bbb.com'))


--example

select SUBSTRING(Email,CHARINDEX('@',Email)+1,

LEN(Email)-CHARINDEX('@',Email))  as Emaildomain,

count(Email) as Total

from tblEmployee4

group by  SUBSTRING(Email,CHARINDEX('@',Email)+1,

LEN(Email)-CHARINDEX('@',Email))



--replicate function

select Firstname,Lastname,

SUBSTRING(Email,1,2) + REPLICATE('*',5) +

SUBSTRING(Email,CHARINDEX('@',Email),LEN(Email)-CHARINDEX('@',Email) + 1 )as Email

from tblEmployee4
```

```
--replace function
select Email,REPLACE(Email,'.com','.net') as Convertedemail
from tblEmployee4

--Datetime datatypes
select GETDATE()
--ISDATE function---checks if a value is a datetime object and returns 1 for success,otherwise 0
select ISDATE('sojore')
select ISDATE(GETDATE())

select DAY(GETDATE())
select Month(GETDATE())
select Year(GETDATE())

--datename function
select DATENAME(Day,Getdate())
select DATENAME(WEEKDAY,Getdate())
select DATENAME(MONTH,Getdate())
select DATENAME(YEAR,Getdate())
select DATENAME(HOUR,Getdate())
--datepart,dateadd,datediff
select DATEPART(weekday,getdate())
select DATENAME(weekday,getdate())

select DATEADD(day,20,getdate())
select DATEDIFF(month,'11/12/2020',getdate())
```

```
--cast & convert functions

select Id,Name,Dateofbirth,CAST(Dateofbirth as nvarchar) as convertedDob
from tblEmployees

select Id,Name,Dateofbirth, CONVERT(nvarchar ,Dateofbirth) as
convertedDob from tblEmployees


select CAST(Getdate() as Date)--this gives you only the datepart

select CONVERT(Date,getdate())


--example---total registration of persons by date

select cast (registrationdate as date)  as Registrationdate,count(Id) as Total

from tblregistration

group by cast (registrationdate as date)



--functions

select * from tblEmployee4

select getdate() as [Date time]

create function fn_getnamebyid(@id int)

returns nvarchar(30)

as

begin

    return (select Firstname from tblEmployee4 where Id=@id)

end
```

```
select dbo.fn_getnamebyid(1)

--encryption of the function
alter function fn_getnamebyid(@id int)
returns nvarchar(30)
with encryption
as
begin
    return (select Firstname from tblEmployee4 where Id=@id)
end
--to view to function info
sp_helptext fn_getnamebyid

--schemabiding---this helps prevent any changes which can be done on the table tblEmployee4 since its solely dependend by the function
alter function fn_getnamebyid(@id int)
returns nvarchar(30)
with schemabinding
as
begin
    return (select Firstname from dbo.tblEmployee4 where Id=@id)
end

drop table tblEmployee4


--temporary tables
```

```
--local temporary table
create table #Persondetails(Id int,Name nvarchar(20))

insert into #Persondetails values(1,'mike')
insert into #Persondetails values(2,'john')
insert into #Persondetails values(3,'todd')

select * from #Persondetails

select name from tempdb..sysobjects
where name like '#Persondetails%'

--global temporary table
create table ##Persondetails(Id int,Name nvarchar(20))
select * from ##Persondetails


--indexes
create table tblEmployee5
(Id int primary key,
Name nvarchar(20),
Salary int,
Gender nvarchar(20))
insert into tblEmployee5 values(1,'sam',2500,'male')
insert into tblEmployee5 values(2,'pam',6500,'female')
insert into tblEmployee5 values(3,'john',4500,'male')
```

```
insert into tblEmployee5 values(4,'sara',5500,'female')

insert into tblEmployee5 values(5,'todd',3100,'male')

select * from tblEmployee5

create index IX_tblemployee5_salary

on tblEmployee5 (Salary ASC)

--dropping the index

drop index tblEmployee5.IX_tblemployee5_salary

--views---saved query

create table tblDepartment2(DepId int primary key,DeptName
nvarchar(20))

insert into tblDepartment2 values(1,'IT')

insert into tblDepartment2 values(2,'Payroll')

insert into tblDepartment2 values(3,'HR')

insert into tblDepartment2 values(4,'Admin')

create table tblEmployee6(Id int primary key,Name nvarchar(20),Salary
int,Gender nvarchar(20),DepartmentId int)

insert into tblEmployee6 values(1,'john',5000,'male',3)

insert into tblEmployee6 values(2,'mike',3400,'male',2)

insert into tblEmployee6 values(3,'pam',6000,'female',1)

insert into tblEmployee6 values(4,'todd',4800,'male',4)
```

```
insert into tblEmployee6 values(5,'sara',3200,'female',1)
insert into tblEmployee6 values(6,'ben',4800,'male',3)

select * from tblEmployee6
select * from tblDepartment2


--creating a view
alter view vwEmployeesbydepartment
as
select Id,Name,Salary,Gender,DeptName
from tblEmployee6
join tblDepartment2
on tblEmployee6.DepartmentId=tblDepartment2.DepId
--or creatthe above view including this statement where
tblDepartment2.DeptName='IT'

select  * from vwEmployeesbydepartment
select  * from vwEmployeesbydepartment  where DeptName='IT'


--indexed views
--when we index views then they are capable of storing data
create table tblProduct(ProductId int primary key,Name
nvarchar(20),Unitprice nvarchar(20))
insert into tblProduct values(1,'books',20)
insert into tblProduct values(2,'pens',14)
```

```
insert into tblProduct values(3,'pencils',11)
insert into tblProduct values(4,'clips',10)

create table tblProductsales(ProductId int ,Quantitysold int)

insert into tblProductsales values (1,10)
insert into tblProductsales values (3,23)
insert into tblProductsales values (4,21)
insert into tblProductsales values (2,12)
insert into tblProductsales values (1,13)
insert into tblProductsales values (3,12)
insert into tblProductsales values (4,13)
insert into tblProductsales values (1,11)
insert into tblProductsales values (2,12)
insert into tblProductsales values (1,14)

select * from tblProduct
select * from tblProductsales

alter view vwTotalsalesbyproduct
with schemabinding
as
select Name,
SUM(ISNULL(( Quantitysold * Unitprice),0) )as TotalSales,
COUNT_BIG(*) as TotalTransactions
from dbo.tblProductsales
```

```
join dbo.tblProduct
on dbo.tblProduct.ProductId=dbo.tblProductsales.ProductId
group by Name


select * from vwTotalsalesbyproduct


--creating an index on the view--this will convert the view to store the data instead of referencing the base tables everytime it gets executed
create unique clustered index UIX_vWTotalsalesbyproduct_name
on vwTotalsalesbyproduct(Name)

--DML triggers
----after triggers
create table tblEmployeeAudit(Id int primary key,AuditData nvarchar(50))

select *from tblEmployee6
select * from tblEmployeeAudit

 create trigger tr_tblEmployee_forinsert
 on tblEmployee6
 for insert
 as
 begin
     declare @Id int
     select @Id =Id from inserted
```

```
    insert into tblEmployeeAudit values ('New employee with Id = ' +
cast(@Id as nvarchar(5)) + 'is added at'
    + cast(getdate() as nvarchar(50)))
end

insert into tblEmployee6 values(8,'jane',1800,'female',3)



--delete trigger
 create trigger tr_tblEmployee_fordelete
 on tblEmployee6
 for delete
 as
 begin
    declare @Id int
    select @Id =Id from deleted

    insert into tblEmployeeAudit values ('An existing employee with Id = ' +
cast(@Id as nvarchar(5)) + 'is added at'
    + cast(getdate() as nvarchar(50)))
end
delete from  tblEmployee6 where Id =2
select * from tblEmployeeAudit
```

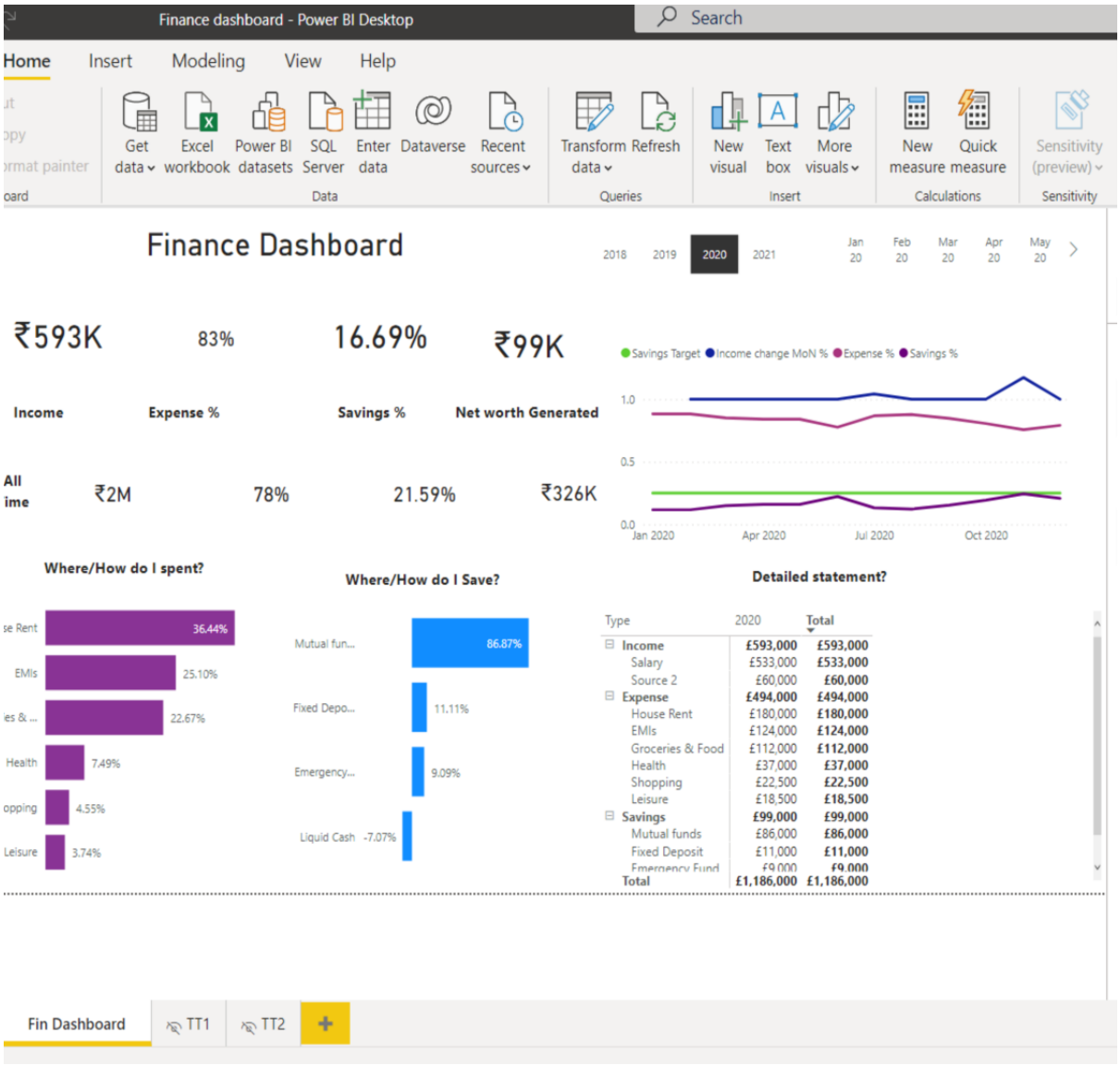
```
--after update triggers
select * from tblEmployee6
select * from tblEmployeeAudit
create trigger tr_tblEmployee_forupdate
on tblEmployee6
for update
as
begin
    select * from deleted
    select * from inserted
end

update tblEmployee6 set Name ='james',Salary=2000,Gender ='male'
where Id=8
drop  table tblEmployee6

create table tblEmployee6(Id int primary key,Name nvarchar(20),Salary
int,Gender nvarchar(20),DepartmentId int)
insert into tblEmployee6 values(1,'john',5000,'male',3)
insert into tblEmployee6 values(2,'mike',3400,'male',2)
insert into tblEmployee6 values(3,'pam',6000,'female',1)
insert into tblEmployee6 values(4,'todd',4800,'male',4)
insert into tblEmployee6 values(5,'sara',3200,'female',1)
insert into tblEmployee6 values(6,'ben',4800,'male',3)


---creating the trigger for after update
```

```
alter trigger tr_tblEmployee_forupdate
on tblEmployee6
for update
as
begin
    declare @Id int
    declare @Oldname nvarchar(20),@Newname nvarchar (20)
    declare @Oldsalary int,@Newsalary int
    declare @Oldgender nvarchar(20),@Newgender nvarchar(20)
    declare @Olddeptid int, @Newdeptid int
    declare @Auditstring nvarchar (1000)

    select *
    into #TempTable
    from inserted

    while (exists(select Id from #TempTable))
    begin
            set @Auditstring=''

            select Top 1 @Id=Id,@Newname=Name,
            @Newgender=Gender,@Newsalary=Salary,
            @Newdeptid=DepartmentId
            from #TempTable

            select @Oldname=Name,@Oldgender=Gender,
```

```
        @Oldsalary=Salary,@Olddeptid=DepartmentId

        from deleted where Id=@Id


        set @Auditstring = 'Employeewith Id = ' +cast (@Id as nvarchar(4))
+ ' changed'

        if (@Oldname <> @Newname)

                set @Auditstring =@Auditstring + ' Name from ' +
@Oldname+ ' to ' + @Newname


        if (@Oldgender<> @Newgender)

                set @Auditstring=@Auditstring +'Gender from' +
@Oldgender + 'to' + @Newgender


        if (@Oldsalary<>Newsalary)

                set @Auditstring=@Auditstring+'Salary from' + cast
(@Oldsalary as nvarchar(10)) +'to' + cast(@Newsalary as nvarchar (10))


        if (@Olddeptid<>@Newdeptid)

                set @Auditstring=@Auditstring + 'Department from ' +
cast(@Olddeptid as nvarchar(10)) + 'to' + cast (@Newdeptid as
nvarchar(10))


        insert into tblEmployeeAudit values (@Auditstring)


        delete from #TempTable where Id=@Id

    end

end
```

The last topic of data science is the BI tools, includes the powerBI and tableau tools

These are very powerful tools in that you can build a visualization board and use this for making analysis and predictions



And with that, you have learned all the concepts in Python as well as in Data science.

Thank you!!!