Python Assignment #1,2

Python Tutorials For Data Science and ML

Python	Pandas	ML
	pandas $y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$ 	 MACHINE LEARNING

Made By:

Name: Asad Haroon
Reg#: SP17-BCS-012
Section: C
Submitted to: Dr. Rao M.Adeel Nawab

Contents:

1. Introduction

    1.1. Features
    1.2. Applications

2. Python Basics

2.1. Data Types and Diff between Mutable and Immutable Objects
2.1.1. Numbers
2.1.2. Strings
2.1.3. Dictionary
2.1.4. Lists
2.1.5. Sets
2.1.6. Tuples
2.2. if/else statements
2.3. Loops
2.3.1. For Loop
2.3.2. While Loop
2.4. Type Casting
2.5. Functions
2.5.1. User Defined Functions
2.5.2. Lambda Functions
2.6. File Handling
2.7. Object Oriented Programming
2.7.1. Classes
2.7.2. Constructors
2.7.3. Functions

3. Python Libraries

3.1. Numpy
3.2. Pandas
3.3. Matplotlib

1.Introduction

Python has become most popular language now-a-days. And ranked 2nd best and widely used programming languages for Data Analysis and Statistical Operations. It was created by Guido van Rossum.

1.2. Features

- Its open source and free to use.
- Syntax is very easy as compared to other programming languages.
- Very easy to learn it.
- You can code in it in just few lines as compared to other languages which requires a large amou
nt of code.
- Many built in functions and libraries for complex calculations.

1.3. Applications

• Python is used for **Data Analysis, Data Mining, Artificial Intelligence and in Machine Learning**.

• As Data is increasing day by day, python is used for **Big Data Analytics**, so predict some patterns.

• Libraries and their uses:

1. **Pandas:** Used for Data Manipulation and Analysis.
   Documentation: https://pandas.pydata.org/pandas-docs/stable/pandas.pdf (https://pandas.pydata.org/pandas-docs/stable/pandas.pdf)
2. **Matplotlib:** Used for making Graphs from Data.
   Documentation: https://matplotlib.org/users/index.html (https://matplotlib.org/users/index.html)
3. **Numpy:** Contains Linear Algebra functions and used to perform mathematical operations on Data.
   Documentation: https://docs.scipy.org/doc/numpy/numpy-ref-1.17.0.pdf (https://docs.scipy.org/doc/numpy/numpy-ref-1.17.0.pdf)
4. **Scikit-Learn:** Contains tools for Machine Learning and Statistical Modeling.
   Documentation: https://scikit-learn.org/stable/_downloads/scikit-learn-docs.pdf (https://scikit-learn.org/stable/_downloads/scikit-learn-docs.pdf)

2. Python Basics

Print Statement Methods:

- By Using **.format(items)**
- By using **format specifiers.**
- By simply using print statement.

Note: Print Statement works only on string objects, if we want to print integer,float,and bool type values then we have to convert it first to string type.

In [1]:
```
#By using .format(items):
a=12
b=[1,2,3]
c="Asad"
print("a is {} and b is {} and c is {}".format(a,b,c))

#By using format specifiers.
print("a is %d and b is %s and c is %s"%(a,b,c))

#By using print statement.
print(a,b,c)
```

```
a is 12 and b is [1, 2, 3] and c is Asad
a is 12 and b is [1, 2, 3] and c is Asad
12 [1, 2, 3] Asad
```

2.1. Difference b/w Mutable and Immutable Objects

Mutable Objects can change their states or contents whereas immutable objects cannot change their states or contents. e.g, List,Sets and Dictionaries are mutable Objects. and integer,float,string are immutable Objects.

Data Types

1.Variables

In Python, we dont to write data type with the variable name, because it automatically detects the type of the variable from their value.

In [2]:
```
number=12 # integer value.
numfloat=4.0
string="Asad Haroon"
ch='A' # There is no specific character type in Python as compared to C, Java. So it considers it as string of l

# To find the data type of the variable we use type() function.
print(type(number))
print(type(numfloat))
print(type(string))
print(type(ch))
```

```
<class 'int'>
<class 'float'>
<class 'str'>
<class 'str'>
```

2. Strings

- Strings are used when we have to deal with characters or words which we cant store in numbers format.e.g, used to store name of student and their registeration number.
- Strings are used instead of integers,float or any other data structure because, data containing alphabets cannot store in any other data structure.

In [3]:
```
#Example 1.
st1="Hello I am Asad Haroon"
print(st1)
#Example 2.
st2='I love Python Language'
print(len(st2)) # String length is calculated with len() function.
```

```
Hello I am Asad Haroon
22
```

Slicing of String

For Slicing of string, we use variable**[initial_limit : final_limit : iterator]**, where initial_limit is starting index and final_index is ending index(exclusive). if final_index is not mentioned then, it automatically operates till full length of string. And iterator is number of iterator i.e, steps.

In [4]:
```
st2="Asad Haroon"
#Example 1.
print(st2[0:5]) # it will pick from 0 to 4 index
#Example 2.
print(st2[:6]) # it will pick from 0 index automatically and pick till 5th index.
#Example 3.
print(st2[-6:]) # it will pick 6th from (right to left) of string. and print till last of string.
```

```
Asad
Asad H
Haroon
```

Updating String

We can update string by concatenating with '+' operator.

In [5]:
```
#Example 1.
stt1="Hello My "
stt1=stt1+"name is Asad Haroon"
print("Updated string1 is "+stt1)
# Example 2.
stt2="Ali is my friend"
stt2='Now Zain'+stt2[3:] # it will pick Ali and update new string.
print("Updated string2 is "+stt2)
#Example 3.
str3="He is girl"
print("Updated string3 is "+str3[:-4]+"Boy") # now in that case our final limit is till -4th index
#(means 4th index from right which means from 0th index to 6th index) and update with new string.
```

```
Updated string1 is Hello My name is Asad Haroon
Updated string2 is Now Zain is my friend
Updated string3 is He is Boy
```

String Functions

String Data Structure have many built in functions. Some of functions are given below.

In [6]:
```
#Function.1: Capitialize func
st1="asad"
print("Capitialize String is "+st1.capitalize()) # it will capitalize the first letter of the string.
# Function 2: Index func
st2="asad"
print("Position of s is "+str(st2.index('s'))) # it will return the position/index of character.
print("Position of a is "+str(st2.index('a'))) # in case of duplication of character
#it will return first character position only.

# Function 3: Ends With
st1="Hello Python"
print("Ends with function returns "+str(st1.endswith('Python'))) # it will tell if string endswith specific stri
#it will return true else false.

#Function 4: Replace func
st4="These is my books"
print("Replaced string is "+st4.replace('is','are')) # first we write replacing item(want to replace)
#and then we write replaced item(new item)
```

```
Capitialize String is Asad
Position of s is 1
Position of a is 0
Ends with function returns True
Replaced string is These are my books
```

Delete String

To delete string value we use del function. It also deletes reference of object.

In [7]:
```
str1="Asad"
print(str1)
del str1
# after deleting string, str1 is not available now.
```

```
Asad
```

String Special Operators

In [8]:
```
# '*' operator is used for repitition of string.
#Example 1.
print("Example 1\n")
st12="Hello Asad"
print((st12+"\n")*3)

print("Example 2")
#Example 2. Using Loops to print pattern
str11="*"
for i in range(5):
    print(str11*i)
```

```
Example 1

Hello Asad
Hello Asad
Hello Asad

Example 2

*
**
***
****
```

In [9]:
```
# 'in' operator if given character/string is in original string.
# in operator is used when we have to find some pattern in the string.
#Example 1.
a="abc def"
print("Space in a returns "+str(' ' in a))
```

```
Space in a returns True
```

In [10]:
```
# 'not in operator' if given character/string not in original string.
# not in operator is used when we have to find some pattern in the string.
# Example
b="abcdef"
print("z in b returns "+str('z' not in b))
```

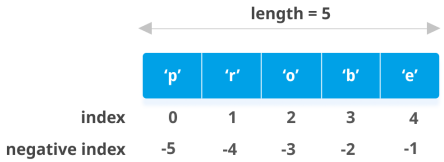
```
z in b returns True
```

3. Lists

Lists Data Structure is used when we have to make list of items/products/numbers/strings.
Lists contains data on which we perform some algorithm to get some type of data/pattern.
Rules:

- In **positive slicing** i.e,from left,Index starts from 0.
- In **negative slicing**,i.e, from right, index starts from -1.
- List can have values of multiple data types.i.e, One list can contains numbers, strings as well.
- Items are comma-seperated values in list.

Structure of List:



In [11]:
```
#Initalization
list1=['asad',6,3.51] # Name,Semester,CGPA.
print("Contents of list1: {}".format(list1))

#Access List Items

#Example 1
print("type(list1[0]): {}".format(type(list1[0])))
#Example 2
print("list1[-1]: {}".format(list1[-1]))
#Example 3.
print("list1[::2]: {}".format(list1[::2]))
```

```
Contents of list1: ['asad', 6, 3.51]
type(list1[0]): <class 'str'>
list1[-1]: 3.51
list1[::2]: ['asad', 3.51]
```

Updating Lists

- We can update lists by simply assigning some index of list to updated value.
- We can update lists by append() function, which will update list and write new content at end.

In [12]:
```
list1=['asad',6,3.51]
#Example 1.
list1[2]=3.7
print('After Updating list1: {}'.format(list1))

#Example 2.
subj=['ML','DS']
list1.append(subj) # appending subjects list at end.
print("list1: {}".format(list1))
```

```
After Updating list1: ['asad', 6, 3.7]
list1: ['asad', 6, 3.7, ['ML', 'DS']]
```

Delete List Elements

We can delete list elements by del statement and remove() function.

- **del** statement is used to remove the specific index from the list when index is known.
- **remove()** function is used to remove the specific object/value from list when index is unknown.

In [13]:
```
#Delete List Elements
list1=['asad',6,3.51,['ML','DS']]
print("Contents of list1: {}".format(list1))

#Example 1. By del
del list1[2]
print("After del list1[2]: {}".format(list1))

#Example 2. By remove

list1[2].remove('ML')
print("list1[2].remove('ML'): {}".format(list1))

#Example 3. By del
del list1 # in that case it will delete list1 reference also.
print(list1)
```

```
Contents of list1: ['asad', 6, 3.51, ['ML', 'DS']]
After del list1[2]: ['asad', 6, ['ML', 'DS']]
list1[2].remove('ML'): ['asad', 6, ['DS']]

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-13-40f0de88bcbe> in <module>
     14 #Example 3. By del
     15 del list1 # in that case it will delete list1 reference also.
---> 16 print(list1)

NameError: name 'list1' is not defined
```

List Functions

There are many built-in functions of lists.

- Sort() function: It will sort list in ascending order.
- Reverse() function: It will reverse array elements.
- Pop() function: It will remove and return last inserted element.
- Clear() function: It will clear all elements of array. i.e, delete all elements in array.

In [100]:
```
# Sort fun():
li=[10,9,8,7,4.3,3.9,3.2,3.8,4.5,5,1]
print("Original List: {}".format(li))
li.sort()
print("Sorted list: {}".format(li))
# Reverse fun():
li.reverse()
print("Reversed Sorted List: {}".format(li))
# Pop() fun:
element=li.pop()
print("Pop() from list returns: {}".format(element))
print("Now list contents are: {}".format(li))
```

```
Original List: [10, 9, 8, 7, 4.3, 3.9, 3.2, 3.8, 4.5, 5, 1]
Sorted list: [1, 3.2, 3.8, 3.9, 4.3, 4.5, 5, 7, 8, 9, 10]
Reversed Sorted List: [10, 9, 8, 7, 5, 4.5, 4.3, 3.9, 3.8, 3.2, 1]
Pop() from list returns: 1
Now list contents are: [10, 9, 8, 7, 5, 4.5, 4.3, 3.9, 3.8, 3.2]
```

In [101]:
```
#List clear fun:
li=[10,9,8,7,4.3,3.9,3.2,3.8,4.5,5,1]
li.clear()
print("Clear li: {}".format(li))

#List Copy fun:
li1=[10,9,8,7,4.3,3.9,3.2,3.8,4.5,5,1]
li2=li1.copy() # it will copy all array in another list li2.
del li1 # it will delete li1 list only.
print("After deleting List Li1, and li2 elements are: {}".format(li2))
```

```
Clear li: []
After deleting List Li1, and li2 elements are: [10, 9, 8, 7, 4.3, 3.9, 3.2, 3.8, 4.5, 5, 1]
```

4. Dictionary

- Dictionary belongs to **mutable** data type.
- Each key has a value seperated by a colon(:).
- **Keys are unique** in dictionary, but value may be same or different.
- Keys must be of immutable data type(strings,numbers,tuples) which we can change values in future.
- A dictionary is used to map or associate things you want to store the keys you need to get them like real world Dictionary.

Accessing Dictionary

In [102]:
```
mydict={'Name':'Asad Haroon','Age':21,'Semester':6,'Subjects':['DS','ML','CCN']}
#Example 1.
print("My Dictionary: {}".format(mydict))
print("My name is {} and my age is {}".format(mydict['Name'],mydict['Age']))

#Example 2.
print("My first subject is {}".format(mydict['Subjects'][0]))
```

```
My Dictionary: {'Name': 'Asad Haroon', 'Age': 21, 'Semester': 6, 'Subjects': ['DS', 'ML', 'CCN']}
My name is Asad Haroon and my age is 21
My first subject is DS
```

Dictionary Functions

- **dict.keys()**: It will return all keys of dictionary.
- **dict.items()**: It will return all dictionary items, i.e, key and value.
- **dict.get()**:It will return that key value.

In [103]:
```
mydict={'Name':'Asad Haroon','Age':21,'Semester':6,'Subjects':['DS','ML','CCN']}
#Example 3.
# To print Keys only.
print("\nKeys of mydict are listed below:")
i=0
for item in mydict.keys():
    print("{}.Key: {}".format(i,item))
    i=i+1
#Example 4
for key,val in mydict.items():
    print("Key is {} and Value is {}".format(key,val))
# Example 5.
print("My Semester is {}".format(mydict.get('Semester')))
```

```
Keys of mydict are listed below:
0.Key: Name
1.Key: Age
2.Key: Semester
3.Key: Subjects
Key is Name and Value is Asad Haroon
Key is Age and Value is 21
Key is Semester and Value is 6
Key is Subjects and Value is ['DS', 'ML', 'CCN']
My Semester is 6
```

Updating Dictionary

1st Method:
We can update dictionary by simply assigning that key to new value.

2nd Method:
We can update dictionary by dict.update() function. Syntax: dict.update({'key':'value'})

In [104]:
```
mydict={'Name':'Asad Haroon','Age':21,'Semester':6,'Subjects':['DS','ML','CCN']}
#1st Method:
mydict['Name']='Asad1234'
print("After updating in 1st Method: {}".format(mydict))
#2nd Method:
mydict.update({'Semester':7})
print("After updating with update() function: {}".format(mydict))

#If that key does not exists in dictionary then it will add that new key with value.
mydict['CGPA']=3.51
print("Adding CGPA in mydict: {}".format(mydict))
```

```
After updating in 1st Method: {'Name': 'Asad1234', 'Age': 21, 'Semester': 6, 'Subjects': ['DS', 'ML', 'CCN']}
After updating with update() function: {'Name': 'Asad1234', 'Age': 21, 'Semester': 7, 'Subjects': ['DS', 'ML',
'CCN']}
Adding CGPA in mydict: {'Name': 'Asad1234', 'Age': 21, 'Semester': 7, 'Subjects': ['DS', 'ML', 'CCN'], 'CGPA':
3.51}
```

Delete Dictionary Elements

We can delete dictionary elements by del statement and clear() function.

- del: Used to delete specific key from dictionary.
- clear(): Used to clear all keys,values from dictionary.

In [105]:
```
mydict={'Name':'Asad Haroon','Age':21,'Semester':6,'Subjects':['DS','ML','CCN'],'CGPA':3.7}
#By del:
del mydict['CGPA']
print("Original Dictionary: {}".format(mydict))
print("del mydict['CGPA']: {}".format(mydict))

# By clear():
mydict.clear()
print("By clear() function: {}".format(mydict))
```

```
Original Dictionary: {'Name': 'Asad Haroon', 'Age': 21, 'Semester': 6, 'Subjects': ['DS', 'ML', 'CCN']}
del mydict['CGPA']: {'Name': 'Asad Haroon', 'Age': 21, 'Semester': 6, 'Subjects': ['DS', 'ML', 'CCN']}
By clear() function: {}
```

5.Tuples

- Tuples is sequence of **immutable** objects, which we cant change.
- Tuple is created by placing comma-separated values between parenthesis().
- It can also have different data types.
- Example of Tuples,we use tuples in managing states NFA,DFA.
- Tuples are **faster than Lists**. And Tuples have structure and lists have order. Thats why, we use tuples.

In [106]:
```
tuples=('Asad',6,3.51,('ML','DS','CCN','Stats')) #Name,Sem,CGPA,(subjects)
# THis is tuple for all students its compulsory to have subjects studied before 6th semester and its constant.
print(tuples[-1][::2])
```

```
('ML', 'CCN')
```

Updating Tuples

In [107]:
```
tuples=('Asad',6,3.51,('ML','DS','CCN','Stats')) #Name,Sem,CGPA,(subjects)
# If we want to change some subjects, we cant update it.
tuples[-1][0]='SEC'
print(tuples[-1])
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-107-4501f0823c70> in <module>
      1 tuples=('Asad',6,3.51,('ML','DS','CCN','Stats')) #Name,Sem,CGPA,(subjects)
      2 # If we want to change some subjects, we cant update it.
----> 3 tuples[-1][0]='SEC'
      4 print(tuples[-1])

TypeError: 'tuple' object does not support item assignment
```

Concat of Tuples

We cant update tuple, but we can concat two tuples.

In [173]:
```
# Example
tuple1=('Asad',6,3.7)
tuple2=('ML','DS')
tup3=tuple1+tuple2
print(tup3)
```

```
('Asad', 6, 3.7, 'ML', 'DS')
```

Delete Tuple Elements

We cant delete elements from tuple, but we can delete whole tuple.

In [174]:
```
tuple1=('Asad',6,3.7)
del tuple1[0]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-174-bf4e949c1294> in <module>
      1 tuple1=('Asad',6,3.7)
----> 2 del tuple1[0]

TypeError: 'tuple' object doesn't support item deletion
```

In [264]:
```
tuple2=('ML','DS')
del tuple2
print(tuple2)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-264-8989a566f8a9> in <module>
      1 tuple2=('ML','DS')
      2 del tuple2
----> 3 print(tuple2)

NameError: name 'tuple2' is not defined
```

6. Sets

- Sets is **unordered collection** of items. Sets is **mutable** data structure
- Set elements are **unique**.
- We use sets in order to avoid duplication.
- Sets are helpful in working with huge datasets.
- Sets are used to include membership testing and eliminating duplicate entries
- It works on Hashing Data structure which takes O(1) time.

In [265]:
```
#Creating Set
sets={1,2,3,1,1,2,2} # it will keep only unique items.
print("Set is {}".format(sets))
print("Type of sets : "+str(type(sets)))

# Adding elements in set.
#Single element:
sets.add('Asad')
print("After adding asad to sets: {}".format(sets))
#Multiple elements
sets.update({2,5,5,4,6,9.1})
print("Adding elements now set={}".format(sets))
```

```
Set is {1, 2, 3}
Type of sets : <class 'set'>
After adding asad to sets: {1, 2, 3, 'Asad'}
Adding elements now set={1, 2, 3, 4, 5, 6, 9.1, 'Asad'}
```

Accessing Sets

Sets are not accessed using index,slicing etc.

In [266]:
```
sets={1,2,3,1,1,2,2}
print(sets[0])
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-266-78dc3d58ff6d> in <module>
      1 sets={1,2,3,1,1,2,2}
----> 2 print(sets[0])

TypeError: 'set' object is not subscriptable
```

In [175]:
```
#Printing Sets elements
sets={1,3,1,1,2,2,0,10,9}
print("Elements of sets are listed below")
for i in sets:
    print(i)
```

```
Elements of sets are listed below
0
1
2
3
9
10
```

Delete Set Elements

We can delete set elements by using remove() function.

In [176]:
```
sets={1,3,1,1,2,2,0,10,9}
sets.remove(2)
print("After removing 2 from set now set={}".format(sets))
```

```
After removing 2 from set now set={0, 1, 3, 9, 10}
```

2.2. if/else Statements

If/else statements are used for comparisions. These are very helpful in programming language.

In [177]:
```
#Example 1.
a=input("Enter the number for example1. ")
a=int(a)
if a%2==0:
    print("Yes it is even number.")
else:
    print("No, its odd number")
# Example 2.
a=input("Enter the number for example2. ")
a=int(a)
if a>10 and a<20:
    print("a is greater than 10 and less than 20")
elif a>20 and a<30:
    print("a is greater than 20 and less than 30")
elif a>30:
    print("a is greater than 30.")
```

```
Enter the number for example1. 12
Yes it is even number.
Enter the number for example2. 155
a is greater than 30.
```

2.3. for/while Loop.

for and while loop used to **iterate some pattern** which we cant do by hard code.e.g, if we have million of data then we use loops to perform calculations/ apply some pattern to produce result.

In [178]:
```
# For Loop:

li=[1,2,3,4]
#1st Method:
print("By 1st Method:")
for item in li:
    print(item)
#2nd Method:
print("By 2nd Method:")
for index in range(len(li)):
    print(li[index])
```

```
By 1st Method:
1
2
3
4
By 2nd Method:
1
2
3
4
```

In [179]:
```
# While Loop:
#1st Method:
i=10
print("While Loop Output:")
while(i<=20):
    print(i)
    i=i+1
#2nd Method:
i=25
print("While True Output:")
while(True):
    if i<=30:
        print(i)
        i=i+1
        continue
    else:
        break
```

```
While Loop Output:
10
11
12
13
14
15
16
17
18
19
20
While True Output:
25
26
27
28
29
30
```

2.4. Type Casting

Convert List into Sets.

In [180]:
```
#Convert List into Sets.
li=[1,2,3,4,5,6]
sets=set(li)
print("List into set: {}".format(sets))
print(type(sets))
```

```
List into set: {1, 2, 3, 4, 5, 6}
<class 'set'>
```

Other Conversions

In [181]:
```
#Convert float into int.
a=3.51
integer=int(a)
print("Float into int: %d"%(integer))

#Convert string into float
a="900.728"
fl=float(a)
print("String into float: %f"%(fl))

#Convert string into tuple
st="asad"
tup=tuple(st)
print("Str into tuple: {} ".format(tup))
```

```
Float into int: 3
String into float: 900.728000
Str into tuple: ('a', 's', 'a', 'd')
```

In [182]:
```
#Convert integer into complex form
comp=complex(4,9)
print("Complex form of real: 4 and Imaginary: 9 is {}".format(comp))

#Convert tuple into dictionary.
tup=(('Name','Asad Haroon'),('Semester',6))
dic=dict(tup)
print("Dictionary is {}".format(dic))

#Convert binary into decimal with base 2.
s = "110" # binary.
dec=int(s,2)
print("Decimal of {} is {}".format(s,dec))
```

```
Complex form of real: 4 and Imaginary: 9 is (4+9j)
Dictionary is {'Name': 'Asad Haroon', 'Semester': 6}
Decimal of 110 is 6
```

2.5. Functions

Functions are group of statements that perform specific tasks as per user requirements.
Basic Syntax is as follows:
def function_name(args):

```
    Statements

    return expression
```

User Defined Functions.

In [183]:
```
# Even number teller function.
def even(a):
    if a%2==0:
        return True
    else:
        return False
print("Example. 1")
num=input("Enter the number ")
num=int(num)
status=even(num)
if status==True:
    print("Yes its even number")
else:
    print("Its odd number")
#Example 2.
print("\nExample. 2")
def iterator(a):
    if a==1 or a==0:
        return 1
    else:
        return a*iterator(a-1)
a=iterator(5)
print("Function returns:"+str(a))
```

```
Example. 1
Enter the number 13
Its odd number

Example. 2
Function returns:120
```

Lambda Functions

They refers to Anonymous Functions. They dont need a return statement because they always return a expression.

In [184]:
```
#Lambda Function #1.
var2=lambda abc: len(abc)
print("Function var2 returns: {}".format(var2('Asad'))) # it will return Length of string.
#Lambda Function #2.
var3=lambda v: int(v)
print("Function var3 returns: {}".format(var3(3.5)))
```

```
Function var2 returns: 4
Function var3 returns: 3
```

Map() Function

- Map() function returns the list of results after applying the given function to each item iteratively.
- Syntax of Map(): map(function,iterable) where iterable should be list,tuple etc.

In [185]:
```
#Example 1.
lis1=[0,2,4,6,8,10] #list of even numbers
li=map(lambda a:a+1,lis1)
print("Lambda function of even to odd returns: {}".format(list(li))) # returns list of odd numbers.

#Example 2.
dict_a = {'name': 'Asad', 'marks': 10}, {'name': 'Z', 'marks': 8}
num=map(lambda a: a['marks']*100,dict_a)
print("Lambda function 2 returns: {}".format(list(num)))

#Example 3.
quiz_marks=[8,10,10,10]
assign_marks=[10,9,9,10]
marks=map(lambda x,y: x/40*15+(y)/40*10,quiz_marks,assign_marks)
print(list(marks))
```

```
Lambda function of even to odd returns: [1, 3, 5, 7, 9, 11]
Lambda function 2 returns: [1000, 800]
[5.5, 6.0, 6.0, 6.25]
```

Filter() Function:

- Filter() function is used to filter out elements from list.

- Syntax: filter(function,iter) where function returns boolean value

In [186]:
```
#Example 1.
lis=['aaa','a','baab','aa','aab']
func=filter(lambda x: x if (x.count('a')==2) else False,lis)
print(list(func))
```

```
['baab', 'aa', 'aab']
```

2.6. File Handling

Reading Input from User

In [187]:
```
string=input("Enter your Semester:")
print("My Semester is "+string)
```

```
Enter your Semester:6
My Semester is 6
```

From File

File Read and Write Modes are as follows:

- r: It opens file in read mode only.
- w: It opens file in write mode only.
- r+ :It opens file in read and write.
- a:It opens file in append mode.
- a+ :It opens file in read only and apeend mode.

In [188]:
```
#Read only Mode
data=open('datasets//File.txt','r')
for d in data.readlines():
    print(d)
```

```
Name: Asad

Semester:7

Department:BSCS

Subjects: DS,ML,CCN
```

In [189]:
```
# Write Only Mode
# It will delete all previous content of file. And write new content.
data=open('datasets//File.txt','w')
data.writelines('Name: Asad\nSemester:7\nDepartment:BSCS')
data.close

f=open('datasets//File.txt','r')
for d in f.readlines():
    print(d)
```

In [190]:
```
#Append Mode
data=open('datasets//File.txt','a')
data.write('\nSubjects: DS,ML,CCN')
data.close

f=open('datasets//File.txt','r')
f=f.read()
print(f)
```

```
Name: Asad
Semester:7
Department:BSCS
```

File Position

- tell():It tells the current position of the file.
- seek():It will change the current position of the file pointer.

In [191]:
```
#Tell():
data=open('datasets//File.txt','r')
print("Content is:\n"+data.read(21))
print("File current location is {}".format(data.tell()))
```

```
Content is:
Name: Asad
Semester:7
File current location is 22
```

In [192]:
```
#Seek():
data=open('datasets//File.txt','r')
print("Content is:\n"+data.read(21))

data.seek(39,0)
print("After changing the location")
print(data.read())
```

```
Content is:
Name: Asad
Semester:7
After changing the location

Subjects: DS,ML,CCN
```

2.7. Object Oriented Programming

Classes And Constructors

Classes are object constructors which have some properties and definitions. Syntax of defining class is **class ClassName:**
Constructors are always executed when class is initiated in java where as in Python we define constructors as **def init(self):**

In [193]:
```
class Employee:
    def __init__(self,name,sno):
        self.name=name
        self.sno=sno
e1=Employee('Asad',4) # Created Object of Employee
print(e1.name)
```

```
Asad
```

Functions in Class

In [194]:
```
class Employee:
    def __init__(self,name,sno):
        self.name=name
        self.sno=sno
    def prints(self):
        print("Employee Name is {} and Sno is {}".format(self.name,self.sno))
e1=Employee('Asad',4) # Created Object of Employee
e1.prints()
```

```
Employee Name is Asad and Sno is 4
```

3. Python Scientific Libraries

3.1. Numpy

- It is fast and require less space as compared to Python Lists.
- It is written in C language.
- It is used to do mathematical operations like Linear Algebra.
- It is used to do many operations on Array.
- It provides a high-performance multidimensional array object

Functions:

- arange(): It works like range function. And it will create an array from 0 value to inputted value.
- ones():It will create an matrix of all ones of specific dimension.
- repeat(array,x):It will repeat the elements of array each x times.
- tile(array,x):It will repeat the whole array x times.
- zeros(): It will create an array of all zeros of specific dimension.
- shape():It will return the order of the matrix.
- full():It will make an matrix of all input value in specific order.
- sum(axis):It will sum up col-wise elements of matrix if axis=0. else if axis=1, then it will sum up row-wise elements.
- where(condition):It will tell the location of elements of given condition.

In [195]:
```
import numpy as np
#arange function():

#Example1 arange()
arr=np.arange(10)
print("Array of range till 10: {}".format(arr))
#Example2 arange()
arr3=np.arange(10,50,5) # it will create an array of value starting from 10 and ending at 25 with step-iteration
#Syntax of arange is arange(start=10,stop=50,step=5)
print("Array of arrange function [10:25,5] is {}".format(arr3))

#ones function():
arr2=np.ones((2,3),dtype=int)
print("\nArray of 2x3 of ones: {}".format(arr2))

#repeat function():
arr=np.array([1,2,3])
arr=np.repeat(arr,3)
print("The contents of repeating arr,3 are {}".format(arr))

#tile function():
a=np.array([1,2,3])
arr=np.tile(a,4)
print("The contents of tile a,4 are {}".format(arr))
```

```
Array of range till 10: [0 1 2 3 4 5 6 7 8 9]
Array of arrange function [10:25,5] is [10 15 20 25 30 35 40 45]

Array of 2x3 of ones: [[1 1 1]
 [1 1 1]]
The contents of repeating arr,3 are [1 1 1 2 2 2 3 3 3]
The contents of tile a,4 are [1 2 3 1 2 3 1 2 3 1 2 3]
```

In [196]:

```
#zeros function():
zero=np.zeros((2,2),dtype=int)
print("Array of 2x2 of all zeros is {}\n".format(zero))

#shape function():
arr=np.zeros((3,4))
row,col=np.shape(arr) # shape function return tuple of order of matrix which is (rows,cols)
print("rows are {} and cols are {}\n".format(row,col))

#full function():
arrfull=np.full((3,4),10) # it will create an matrix of all values 10 with dimension of 3x4.
print("Matrix full with 10 is {}".format(arrfull))
#axis function():
arr1=np.array(([1,2,3],[4,5,6]))
print("Contents of arr1 is {}\n".format(arr1))
a=arr1.sum(axis=0)
print("Sum of arr1 on axis=0 is {}".format(a))
a=arr1.sum(axis=1)
print("Sum of arr1 on axis=1 is {}".format(a))

#where function():
arr=np.array([1,2,10,22,21])
a=np.where(arr>10)
print("The locations of arr>10 are {}".format(a))
```

```
Array of 2x2 of all zeros is [[0 0]
 [0 0]]

rows are 3 and cols are 4

Matrix full with 10 is [[10 10 10 10]
 [10 10 10 10]
 [10 10 10 10]]
Contents of arr1 is [[1 2 3]
 [4 5 6]]

Sum of arr1 on axis=0 is [5 7 9]
Sum of arr1 on axis=1 is [ 6 15]
The locations of arr>10 are (array([3, 4], dtype=int64),)
```

3.2.Pandas

- It is an open source library, provides high performance, easy to use functions and data structures and best for data analysis.
- It allows for fast analysis and data cleaning.

Components of Pandas

There are two components of pandas.

1. Series
2. Data Frame

1. Series

Series is essientially a column and it is very similar to Numpy Array. It is 1d Array of data. It has axis label and it can be indexed by the label.
To create a Series in Pandas, the following function is used a=pd.Series(data,index=index).
We can create Series from three different ways.

- Python Dictionary
- From ndarray(numpy random array)
- Scalar Value

1.1. From Random array

In [197]:
```
import pandas as pd
import numpy as np
a=pd.Series(np.random.rand(3),index=['A','B','C'])
print("Contents of a is\n{}".format(a))
print("Type of np.random.rand is {} and type of a is {}".format(type(np.random.rand(3)),type(a)))
```

```
Contents of a is
A    0.384594
B    0.294880
C    0.144340
dtype: float64
Type of np.random.rand is <class 'numpy.ndarray'> and type of a is <class 'pandas.core.series.Series'>
```

In [198]:
```
# To print the index of array.
print(a.index)
```

```
Index(['A', 'B', 'C'], dtype='object')
```

In [199]:
```
# IF we dont assign index in Series then it will automatically generate index from 0 to len(array-1)
#Example
series=pd.Series(np.random.rand(10))
print(series)
```

```
0    0.014739
1    0.501020
2    0.884920
3    0.405336
4    0.738612
5    0.876131
6    0.042329
7    0.878318
8    0.109824
9    0.161092
dtype: float64
```

1.2.From Dictionary.

In [200]:
```
# Example 1: Where dictionary has key, value pair and
#Series will automatically pick key as index and value as their corresponding index value
di={'Semester':'6','Name':'A'}
a=pd.Series(di)
print("Contents of Example 1 is \n{}".format(a))
print("\n")
#Example 2: Where index are given but not in order but Series will automatically pick their indexes values from
i={'Semester':'6','Name':'A','Subjects':{'ML','DS','OOP'}}
b=pd.Series(i,index=['Name','Subjects','Semester'])
print("Contents of Example 2 is \n{}".format(b))
```

```
Contents of Example 1 is
Semester    6
Name        A
dtype: object


Contents of Example 2 is
Name                    A
Subjects    {DS, OOP, ML}
Semester                6
dtype: object
```

1.3. From Scalar Value

If data is in scalar value then index must be provided.

In [201]:
```
#Example 1:
sc=pd.Series([10,2,3],index=['a','b','c'])
print("Output is \n{}".format(sc))

#Example 2: It will now use 4 as value of index and assign 4 to all index.
scc=pd.Series(4,index=[10,20,30,40,50,60])
print("\nOutput of 2nd Example is")
print(scc)
```

```
Output is
a    10
b     2
c     3
dtype: int64

Output of 2nd Example is
10    4
20    4
30    4
40    4
50    4
60    4
dtype: int64
```

1.4. Series Slicing

In [202]:
```
# Declaration of Series
i=pd.Series([1,2,3,4,5,6,7,9.23,92309,983.923],index=['a','b','c','d','e','f','g','h','i','j'])
print("Contents of Series is \n{}".format(i))
print("\n")
#Series Slicings:
#Example1:
print("Series[0] is",i[0])
#Example2:
print("\nResult of Series[0:4:2] is\n{}".format(i[0:4:2])) # it will pick from 0 index and skip index by 2.
#i.e, 0 and 2 index will be included.
print("\n")
#Example3:
print("Mean of series is {} and instruction i[i>i.mean()] result is \n{}".format(i.mean(),i[i>i.mean()]))
```

```
Contents of Series is
a        1.000
b        2.000
c        3.000
d        4.000
e        5.000
f        6.000
g        7.000
h        9.230
i    92309.000
j      983.923
dtype: float64


Series[0] is 1.0

Result of Series[0:4:2] is
a    1.0
c    3.0
dtype: float64


Mean of series is 9333.0153 and instruction i[i>i.mean()] result is
i    92309.0
dtype: float64
```

In [203]:
```
# To access value of index
print(i['f'])
```

```
6.0
```

In [204]:
```
# To update value of index simply assign new value.
i['j']=998398
print("Now updated Series is\n{}".format(i))
```

```
Now updated Series is
a         1.00
b         2.00
c         3.00
d         4.00
e         5.00
f         6.00
g         7.00
h         9.23
i     92309.00
j    998398.00
dtype: float64
```

In [205]:
```
# to check some index is present in Series or not.
st='a' in i
print(" 'a' in i returns",st)
#Example 2
st='a ' in i
print(" 'a ' in i returns",st) # because space after a index is not present in Series.
```

```
 'a' in i returns True
 'a ' in i returns False
```

1.5. Vectorized Operations on Series

In [206]:
```
# To add a Series contents (index-wise)
s1=pd.Series(np.random.rand(5),index=['a','b','c','d','e'])
print("Series is\n",s1)
print("\n")
print("After s1+s1 returns\n{}".format(s1+s1))

# If index of two Series does not match then it will dont add up.
s2=pd.Series(np.random.rand(5),index=['a1','b1','c1','d1','e1'])
print("\nAddition of s1 and s2 is as follows:\n")
print(s1+s2)
```

```
Series is
 a    0.295640
b    0.846414
c    0.879989
d    0.125169
e    0.964775
dtype: float64


After s1+s1 returns
a    0.591281
b    1.692828
c    1.759978
d    0.250339
e    1.929549
dtype: float64

Addition of s1 and s2 is as follows:

a    NaN
a1   NaN
b    NaN
b1   NaN
c    NaN
c1   NaN
d    NaN
d1   NaN
e    NaN
e1   NaN
dtype: float64
```

In [207]:
```
# TO name the series.
s1=pd.Series([1,2,3,4,5],index=['a','b','c','d','e'],name='Series of first five Natural Numbers')
print("Name of s1 is: ",s1.name)
# To rename the series, we cant rename the original series so copy it in other attribute first and then rename i
s2=s1.rename('2nd Series')
print("After renaming the series its name is now: ",s2.name)
```

```
Name of s1 is:  Series of first five Natural Numbers
After renaming the series its name is now:  2nd Series
```

2. Data Frames

A two-dimensional labeled data structure with columns of potentially different types. It accepts many kind of inputs.

- Dictionary of 1D ndarrays,dicts,lists or sets.
- 2D Numpy Arrays.
- A Series
- Data Frame.

The data in the dataframe is actually stored on memory as collection of Series. And it provides various functionalities to analyze, change and extract some information from the dataset.

2.1.From Dictionary

In [208]:
```
#Declaration
import pandas as pd
d={'height':pd.Series([5,6,7],index=['Ali','Basit','Asad']),
   'weight':pd.Series([40,50,60,70],index=['Ali','Basit','Asad','Ammar'])
  }
df=pd.DataFrame(d)
print("Data Frame is\n{}".format(df))
print("type of df is ",type(df))
```

```
Data Frame is
       height  weight
Ali       5.0      40
Ammar     NaN      70
Asad      7.0      60
Basit     6.0      50
type of df is  <class 'pandas.core.frame.DataFrame'>
```

In [209]:
```
# It will show only limited data i.e, only columns height and B data and given index data.
import pandas as pd
d={'height':pd.Series([5,6,7],index=['Ali','Basit','Asad']),
   'weight':pd.Series([40,50,60,70],index=['Ali','Basit','Asad','Ammar'])
  }
df=pd.DataFrame(d,index=['Ali','Basit','Asad','Ammar'],columns=['height','B'])
print(df)
```

```
       height    B
Ali       5.0  NaN
Basit     6.0  NaN
Asad      7.0  NaN
Ammar     NaN  NaN
```

2.2. From Dict of Lists and ndArrays(Numpy)

In [210]:
```
#Declaration
d={
    'height':[10,20,30,40],
    'weight':[100,200,300,400]
}
df=pd.DataFrame(d)
print("With Auto Generated Index\n {}".format(df))
print("\n")
#With Given Index.
df=pd.DataFrame(d,index=['a','b','c','d'])
print("With given index values\n{}".format(df))
```

```
With Auto Generated Index
    height  weight
0       10     100
1       20     200
2       30     300
3       40     400


With given index values
   height  weight
a      10     100
b      20     200
c      30     300
d      40     400
```

In [211]:
```
#nd arrays(Numpy Array)

df=pd.DataFrame(np.random.rand(5),index=['aa','bb','cc','dd','ee'],columns=['A'])
print(df)
```

```
          A
aa  0.584805
bb  0.133878
cc  0.031587
dd  0.739434
ee  0.859939
```

2.3.From List of Dicts.

In [212]:
```
#Declaration
li=[{'Semester':6,'Name':'Asad','GPA':4},{'Semester':6,'Name':'Zain','GPA':3}]
df=pd.DataFrame(li,index=['1st','2nd'],columns=['Name','Semester','GPA'])
print(df)
```

```
     Name  Semester  GPA
1st  Asad         6    4
2nd  Zain         6    3
```

2.4. Read Data From CSV

In [213]:
```
df=pd.read_csv('datasets//dataset.csv')
# Selection of columns from dataset
print(df.head())
```

```
   user_id  recipe_id        date  rating   u       i
0     8937      44551  12/23/2005       4   2  173538
1    56680     126118   10/7/2006       4  16  177847
2   349752     219596   4/12/2008       0  26   89896
3   628951      82783  11/13/2007       2  45  172637
4    92816     435013   7/31/2013       3  52  177935
```

2.5. DataFrame Functions for Viewing/Inspecting Data and Statistics:

Dataset Functions are listed below:

- **head(n)**: It will pick first n rows from datasets.
- **tail(n)**: It will pick last n rows from datasets.
- **shape()**: It will return the shape of the dataframe.
- **df.columns**: It will tell the columns of the dataframe.
- **df.index**: It will tell the starting and ending index of the datasets.
- **df.info()**:It will tell the information of the dataframes.
- **df.count()**:It will count the non NA values of each column i.e, not null.
- **df.sum()**:It will sum up column wise all the values of each column.
- **df.min()**:It will tell the minimum value of each column
- **df.max()**:It will tell the maximum value of each column.
- **df.mean()**:It will tell the mean of values of each column.
- **df.median()**:It will tell the median of values of each column.
- **df.describe()**:It will generate the statistical summary of the dataframe.

- df.describe():it will generate the statistical summary of the dataframe.
- df.dtypes:it will print the types of all column attributes.

In [214]:
```
print("Head of dataset:")
print(df.head()) # by default head function returns first five rows of datasets.
```

```
Head of dataset:
   user_id  recipe_id        date  rating   u       i
0     8937      44551  12/23/2005       4   2  173538
1    56680     126118   10/7/2006       4  16  177847
2   349752     219596   4/12/2008       0  26   89896
3   628951      82783  11/13/2007       2  45  172637
4    92816     435013   7/31/2013       3  52  177935
```

In [215]:
```
print("Tail of dataset:")
print(df.tail(10))
```

```
Tail of dataset:
       user_id  recipe_id        date  rating      u       i
12445   541949     175116    9/8/2007       2  25033  125465
12446  1248602     299886  12/14/2013       4  25041  133351
12447  1724643     375807   11/8/2011       5  25047  154459
12448   937528     375371    1/6/2010       5  25048  157275
12449   472815     187829  10/31/2011       0  25051  147807
12450   101053     179011    1/3/2009       5  25054  130258
12451   252205      81398  12/26/2005       2  25055  152255
12452   624305     142984   1/15/2011       1  25057  139864
12453   173575     104842  12/18/2004       3  25059  140646
12454  1249650     287280   4/28/2009       4  25070  166028
```

In [216]:
```
print("Shape of dataset:")
row,col=df.shape
print("Rows are ",row)
print("Cols are ",col)
```

```
Shape of dataset:
Rows are  12455
Cols are  6
```

In [217]:
```
print("Columns of datsets are:\n")
print(df.columns)
```

```
Columns of datsets are:

Index(['user_id', 'recipe_id', 'date', 'rating', 'u', 'i'], dtype='object')
```

In [218]:
```
print("It will tell the starting and ending index of the datasets.\n")
print(df.index)
```

```
It will tell the starting and ending index of the datasets.

RangeIndex(start=0, stop=12455, step=1)
```

In [219]:
```
print("It will tell the information of the dataframes.\n")
print(df.info())
```

```
It will tell the information of the dataframes.

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12455 entries, 0 to 12454
Data columns (total 6 columns):
user_id      12455 non-null int64
recipe_id    12455 non-null int64
date         12455 non-null object
rating       12455 non-null int64
u            12455 non-null int64
i            12455 non-null int64
dtypes: int64(5), object(1)
memory usage: 583.9+ KB
None
```

In [220]:
```
print("It will count the non NA values of each column i.e, not null\n")
print(df.count())
```

```
It will count the non NA values of each column i.e, not null

user_id      12455
recipe_id    12455
date         12455
rating       12455
u            12455
i            12455
dtype: int64
```

In [221]:
```
print("It will sum up column wise all the values of each column\n")
print(df.sum())
```

```
It will sum up column wise all the values of each column

user_id                                              362775382662
recipe_id                                              2607119516
date         12/23/200510/7/20064/12/200811/13/20077/31/201...
rating                                                     52474
u                                                      153053259
i                                                     1438404578
dtype: object
```

In [222]:
```
print("Max(): It will tell the maximum value of each column\n")
print(df.max())
print("\n")
print("Min(): It will tell the minimum value of each column\n")
print(df.min())
```

```
Max(): It will tell the maximum value of each column

user_id      2002254807
recipe_id        537716
date            9/9/2018
rating                 5
u                  25074
i                 178264
dtype: object


Min(): It will tell the minimum value of each column

user_id          1533
recipe_id         120
date          1/1/2005
rating               0
u                    2
i                  102
dtype: object
```

In [223]:
```
print("It will tell the mean of values of each column\n")
print(df.mean())
```

```
It will tell the mean of values of each column

user_id      2.912689e+07
recipe_id    2.093231e+05
rating       4.213087e+00
u            1.228850e+04
i            1.154881e+05
dtype: float64
```

In [224]:
```
print("It will tell the median of values of each column\n")
print(df.median())
```

```
It will tell the median of values of each column

user_id      382954.0
recipe_id    195040.0
rating            5.0
u             12023.0
i            127793.0
dtype: float64
```

In [225]:
```
print("Statistical Summary of the dataset is as follows:\n")
print(df.describe())
```

```
Statistical Summary of the dataset is as follows:

            user_id      recipe_id        rating             u              i
count  1.245500e+04   12455.000000  12455.000000  12455.000000   12455.000000
mean   2.912689e+07  209323.124528      4.213087  12288.499318  115488.123485
std    2.334357e+08  135001.832923      1.338503   6897.751394   50448.663212
min    1.533000e+03     120.000000      0.000000      2.000000     102.000000
25%    1.698420e+05   94616.000000      4.000000   6428.500000   76904.000000
50%    3.829540e+05  195040.000000      5.000000  12023.000000  127793.000000
75%    8.016370e+05  314928.500000      5.000000  17985.500000  160024.000000
max    2.002255e+09  537716.000000      5.000000  25074.000000  178264.000000
```

In [226]:
```
# it will print the types of all column attributes.
df.dtypes
```

Out[226]:
```
user_id       int64
recipe_id     int64
date         object
rating        int64
u             int64
i             int64
dtype: object
```

2.6. Data Cleaning

- df.dropna(): It will drop all rows that contains missing data(NaN).

- df.dropna(axis=1): It will drop all columns that contains missing data.
- df.dropna(axis=0): It will drop all rows that contains missing data.
- df.fillna(x): It will fill all nan(missing data) with x value.
- df.replace(value,newvalue): It will replace all values with newvalue in the dataset.

In [227]:
```
dic = {'First Score':[100, 90, np.nan, 95],
       'Second Score': [30, 45, 56, np.nan],
       'Third Score':[np.nan, 40, 80, 98]}
df=pd.DataFrame(dic)
print("Original Df is \n{}".format(df))
print(df.dropna()) # it will drop all rows which contains nan.
```

```
Original Df is
   First Score  Second Score  Third Score
0        100.0          30.0          NaN
1         90.0          45.0         40.0
2          NaN          56.0         80.0
3         95.0           NaN         98.0
   First Score  Second Score  Third Score
1         90.0          45.0         40.0
```

In [228]:
```
print(df.dropna(axis=1))
```

```
Empty DataFrame
Columns: []
Index: [0, 1, 2, 3]
```

In [229]:
```
print(df.dropna(axis=0))
```

```
   First Score  Second Score  Third Score
1         90.0          45.0         40.0
```

In [230]:
```
print(df.fillna('Missing'))
```

```
  First Score Second Score Third Score
0         100           30     Missing
1          90           45          40
2     Missing           56          80
3          95      Missing          98
```

In [231]:
```
df.replace('Missing',np.nan)
```

Out[231]:

	First Score	Second Score	Third Score
0	100.0	30.0	NaN
1	90.0	45.0	40.0
2	NaN	56.0	80.0
3	95.0	NaN	98.0

In [232]:
```
df.fillna(df.mean()) # it will fill missing data with their mean value column-wise.
```

Out[232]:

	First Score	Second Score	Third Score
0	100.0	30.000000	72.666667
1	90.0	45.000000	40.000000
2	95.0	56.000000	80.000000
3	95.0	43.666667	98.000000

2.7. Filer, Sort and Group by Functions

In [233]:
```
df=pd.read_csv('datasets//dataset.csv')
df.head(15)
```

Out[233]:

	user_id	recipe_id	date	rating	u	i
0	8937	44551	12/23/2005	4	2	173538
1	56680	126118	10/7/2006	4	16	177847
2	349752	219596	4/12/2008	0	26	89896
3	628951	82783	11/13/2007	2	45	172637
4	92816	435013	7/31/2013	3	52	177935
5	280271	228179	7/29/2007	5	55	178179
6	345569	186470	10/5/2008	0	57	177482
7	724516	298748	5/7/2011	5	71	177749
8	176615	118119	10/27/2006	0	82	178250
9	56112	166712	8/2/2007	5	89	177821
10	537179	78641	11/20/2009	4	91	177805
11	222478	437144	11/9/2010	5	97	178178
12	22898	65976	12/14/2003	5	101	173681
13	857489	311630	9/20/2009	5	120	178097
14	1056692	312579	8/31/2013	4	122	177486

In [234]:
```
# It will return those rows whose ratings is 3. and limit records to first 10 only.
df[df['rating']==3].head(10)
```

Out[234]:

	user_id	recipe_id	date	rating	u	i
4	92816	435013	7/31/2013	3	52	177935
57	310518	298141	4/29/2008	3	314	159728
106	464080	171138	1/19/2012	3	498	177711
127	863904	387364	9/2/2009	3	559	70544
134	371105	318432	9/3/2008	3	589	88159
140	1967997	431022	10/24/2012	3	601	164245
194	54716	149739	1/7/2007	3	724	171048
200	477439	393783	3/14/2010	3	737	154483
221	634323	320079	9/9/2008	3	815	176996
237	68884	110741	4/8/2006	3	852	167172

In [235]:
```
# it will sort values of date and return last 10 rows in Descending Order.
df.sort_values('date',ascending=False).tail(10)
```

Out[235]:

	user_id	recipe_id	date	rating	u	i
7505	215969	118696	1/1/2006	4	13305	155702
4234	62866	125882	1/1/2006	5	11777	82738
6509	8527	85331	1/1/2006	5	11445	43786
8283	95858	135483	1/1/2006	5	14800	164686
2092	37502	64424	1/1/2006	5	4208	98005
911	98467	133270	1/1/2006	5	2193	109942
1002	42720	91894	1/1/2005	5	2334	166245
7351	62474	99712	1/1/2005	5	12982	109510
2800	166746	102309	1/1/2005	5	5291	150945
10182	37909	95522	1/1/2005	4	18797	131052

In [236]:
```
# Sort values on basis of more than 1 attribute
df.sort_values(['date','rating'],ascending=[True,True]).head(10)
```

Out[236]:

	user_id	recipe_id	date	rating	u	i
10182	37909	95522	1/1/2005	4	18797	131052
1002	42720	91894	1/1/2005	5	2334	166245
2800	166746	102309	1/1/2005	5	5291	150945
7351	62474	99712	1/1/2005	5	12982	109510
7505	215969	118696	1/1/2006	4	13305	155702
911	98467	133270	1/1/2006	5	2193	109942
2092	37502	64424	1/1/2006	5	4208	98005
4234	62866	125882	1/1/2006	5	11777	82738
6509	8527	85331	1/1/2006	5	11445	43786
7846	124481	147661	1/1/2006	5	13919	175506

In [237]:
```
# Group by: It will group by date and count all records entries on that date.
df.groupby('date').count().tail(10)
```

Out[237]:

date	user_id	recipe_id	rating	u	i
9/9/2004	4	4	4	4	4
9/9/2005	1	1	1	1	1
9/9/2006	3	3	3	3	3
9/9/2007	3	3	3	3	3
9/9/2008	8	8	8	8	8
9/9/2009	3	3	3	3	3
9/9/2011	3	3	3	3	3
9/9/2012	2	2	2	2	2
9/9/2015	2	2	2	2	2
9/9/2018	1	1	1	1	1

2.8. Column Operations

In [238]:
```
# it will show the first 10 records of the ratings column
df['rating'].head(10)
```

Out[238]:
```
0    4
1    4
2    0
3    2
4    3
5    5
6    0
7    5
8    0
9    5
Name: rating, dtype: int64
```

In [239]:
```
# We can Create a new column and can multiply,add columns and save result in it.
df['MultRating']=df['rating']*df['rating']
df['MultRating'].head(10)
```

Out[239]:
```
0    16
1    16
2     0
3     4
4     9
5    25
6     0
7    25
8     0
9    25
Name: MultRating, dtype: int64
```

In [240]:
```
(df['MultRating']>16).head(10)
```

Out[240]:
```
0    False
1    False
2    False
3    False
4    False
5     True
6    False
7     True
8    False
9     True
Name: MultRating, dtype: bool
```

In [241]:
```
# it will delete multRating column now.
del df['MultRating']
```

In [242]:
```
# it will create a new column named NewCol and place it at 1 index means 2nd column and save the results df[rati
df=pd.read_csv('datasets//dataset.csv')
df.insert(1,'NewCol',df['rating']*df['rating'])
df.head(10)
```

Out[242]:

	user_id	NewCol	recipe_id	date	rating	u	i
0	8937	16	44551	12/23/2005	4	2	173538
1	56680	16	126118	10/7/2006	4	16	177847
2	349752	0	219596	4/12/2008	0	26	89896
3	628951	4	82783	11/13/2007	2	45	172637
4	92816	9	435013	7/31/2013	3	52	177935
5	280271	25	228179	7/29/2007	5	55	178179
6	345569	0	186470	10/5/2008	0	57	177482
7	724516	25	298748	5/7/2011	5	71	177749
8	176615	0	118119	10/27/2006	0	82	178250
9	56112	25	166712	8/2/2007	5	89	177821

2.9. Date Operations

In [243]:
```
# it will create indexes of date from 29092019 to next 6 days i.e, till 04102019.
dates=pd.date_range('20190929',periods=6)
dates
```

Out[243]:
```
DatetimeIndex(['2019-09-29', '2019-09-30', '2019-10-01', '2019-10-02',
               '2019-10-03', '2019-10-04'],
              dtype='datetime64[ns]', freq='D')
```

In [244]:
```
df=pd.DataFrame(np.random.rand(6,1),index=dates,columns=list('A'))
df
```

Out[244]:

	A
2019-09-29	0.480140
2019-09-30	0.678219
2019-10-01	0.191883
2019-10-02	0.314270
2019-10-03	0.086386
2019-10-04	0.490324

In [245]:
```
import pandas as pd
import numpy as np
dff=pd.DataFrame({'A':pd.Categorical(['test','train']),'B':np.array([1,2]),'C':pd.Timestamp('20130102'),'D':'Hel
                  index=['1','2'])
dff
```

Out[245]:

	A	B	C	D
1	test	1	2013-01-02	Hello
2	train	2	2013-01-02	Hello

2.10. Logical AND and OR

In [246]:
```
#Logical OR
df=pd.read_csv('datasets//dataset.csv')
(df['user_id'] | df['rating']).head(10)
```

Out[246]:
```
0      8941
1     56684
2    349752
3    628951
4     92819
5    280271
6    345569
7    724517
8    176615
9     56117
dtype: int64
```

In [247]:
```
#Logical AND
(df['user_id'] & df['rating']).head(10)
```

Out[247]:
```
0    0
1    0
2    0
3    2
4    0
5    5
6    0
7    4
8    0
9    0
dtype: int64
```

In [248]:
```
print("Actual Contents are:")
print(dff)
print("\nTransposed contents are:")
print(dff.T)
```

```
Actual Contents are:
       A  B          C      D
1   test  1 2013-01-02  Hello
2  train  2 2013-01-02  Hello

Transposed contents are:
                     1                   2
A                 test               train
B                    1                   2
C  2013-01-02 00:00:00  2013-01-02 00:00:00
D                Hello               Hello
```

2.11. loc and iloc function:

In [249]:
```
#iloc function:
# Row and column selection i.e, row,col
dff.iloc[0:2,0:1]
```

Out[249]:

	A
1	test
2	train

In [250]:
```
# loc function
# It will return the corresponding row.
dff.loc['2']
```

```
A                 train
B                     2
C    2013-01-02 00:00:00
D                 Hello
Name: 2, dtype: object
```

2.12. isin function:

In [251]:
```
# it will filter all the results based on specific column and their value.
dff[dff['D'].isin(['Hello'])]
```

Out[251]:

	A	B	C	D
1	test	1	2013-01-02	Hello
2	train	2	2013-01-02	Hello

In [252]:
```
# in this case in D column no value exists 'test'. So it will return nothing.
dff[dff['D'].isin(['test'])]
```

Out[252]:

	A	B	C	D

2.13. Save CSV File

In [253]:
```
# it will save your dataset in the file.
dff.to_csv('datasets//file.csv')
```

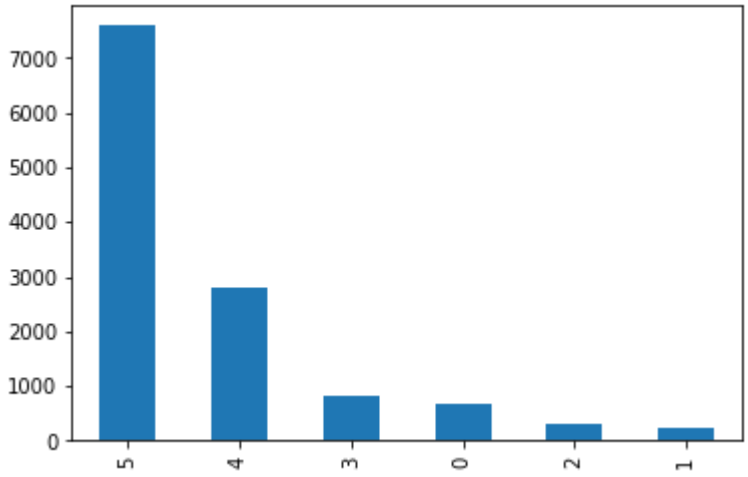
2.14. Plot Graphs of DataSet

Bar Chart

In [254]:
```
dff=pd.read_csv('datasets//dataset.csv')

dff['rating'].value_counts().plot.bar() # it will plot graph of rating entries.
```

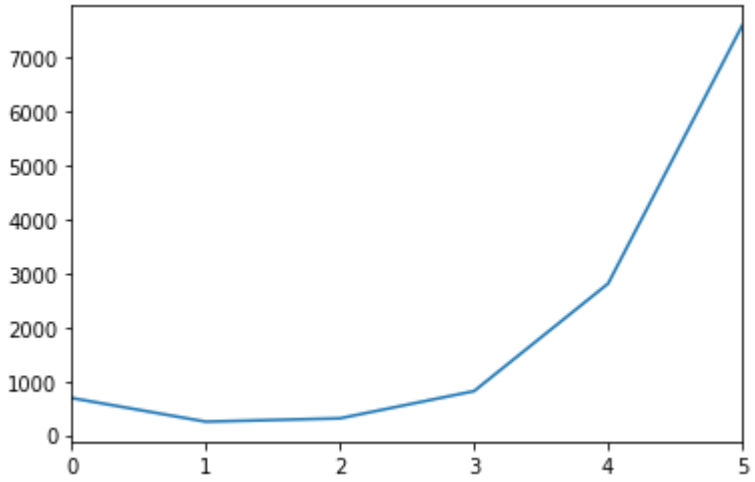
Out[254]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2b955569c18>
```



Line Charts

In [255]:
```
dff['rating'].value_counts().sort_index().plot.line()
```

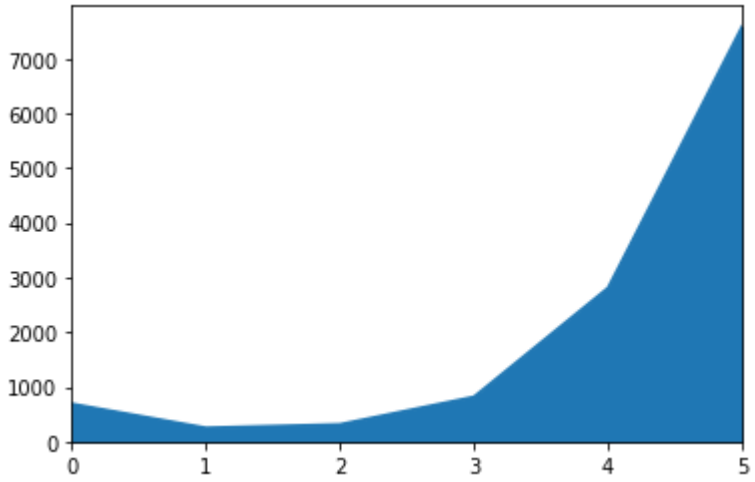
Out[255]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2b956683b00>
```



Area Charts

In [256]:
```
dff['rating'].value_counts().sort_index().plot.area()
```

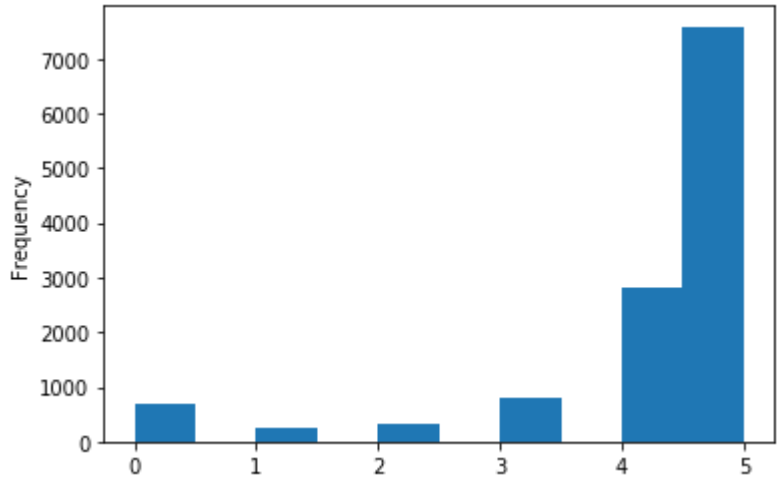
Out[256]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2b9566e1d30>
```



Histogram Charts

In [257]:
```
dff['rating'].plot.hist()
```

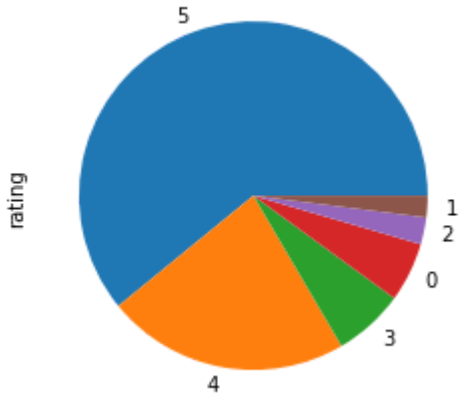
Out[257]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2b956739470>
```



Pie Charts

In [258]:
```
import pandas as pd
df=pd.read_csv('datasets//dataset.csv')
df['rating'].value_counts().plot.pie()
```

Out[258]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2b956739ba8>
```



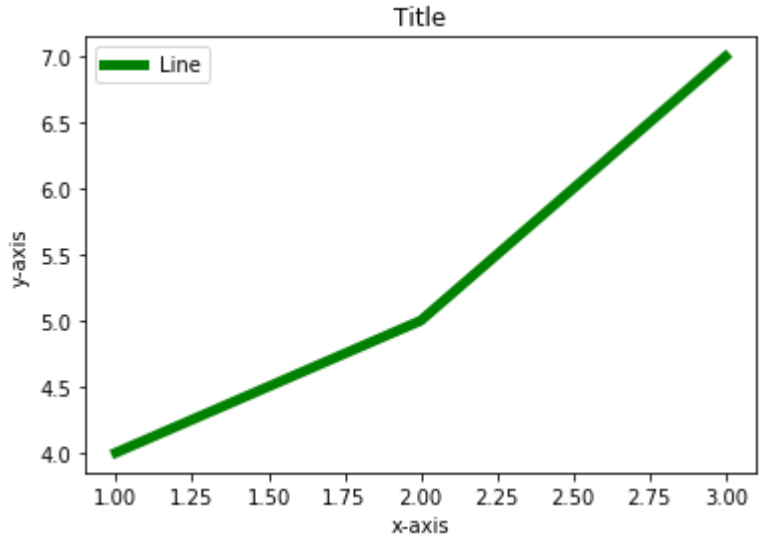
3.3. Matplotlib

It is useful for visualizations and after training model we have to test our model and see graph behavior and predictions.

In [259]:
```
import matplotlib.pyplot as plt
```

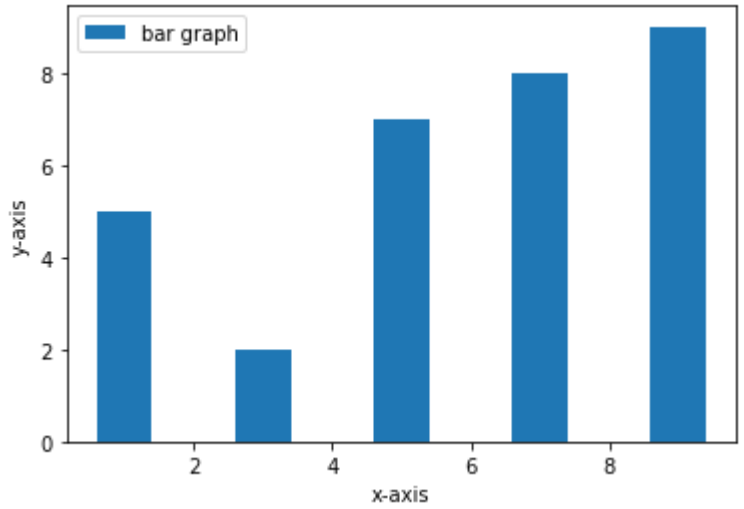
3.3.1. Simple Plot

In [260]:
```
from matplotlib import pyplot as plt
plt.plot([1,2,3],[4,5,7],'g',label="Line",linewidth=5) # g means green color and label is for Legend
plt.title('Title')
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.legend(loc="upper left")
plt.show()
```



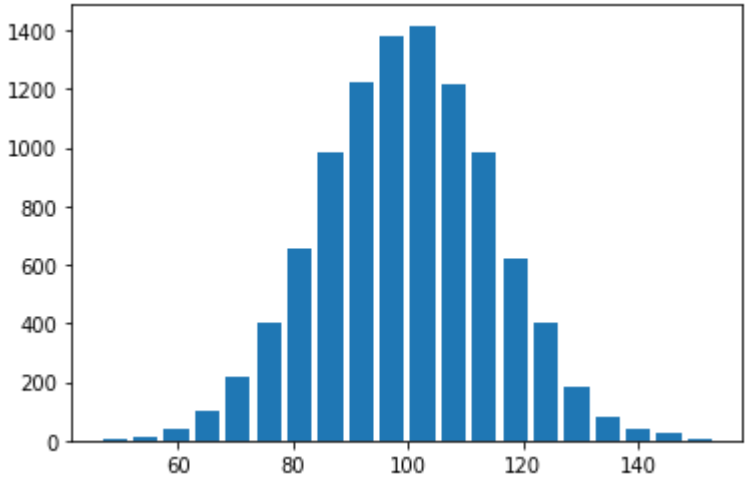
3.3.2. Bar Graph:

In [261]:
```
x=[1,3,5,7,9]
y=[5,2,7,8,9]
plt.bar(x,y,label="bar graph",linewidth=2)
plt.xlabel("x-axis")
plt.ylabel("y-axis")
plt.legend()
plt.show()
```



3.3.3. Histogram

In [262]:
```
x = [21,22,23,4,5,6,77,8,9,10,31,32,33,34,35,36,37,18,49,50,100]
mu = 100 # mean of distribution
sigma = 15 # standard deviation of distribution
x = mu + sigma * np.random.randn(10000)
bint = 20
plt.hist(x,bint,histtype='bar',rwidth=0.8)
plt.show()
```



3.3.4. Scatter Plot:

In [263]:
```
x=[1,2,3,4,5]
y=[2,2,4,4,5]
plt.scatter(x,y,label="Scatter",color='g')
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.legend()
plt.show()
```