Be Agile..Be in Demand

DLithe Consultancy Services Pvt Ltd

Internship on DATA SCIENCE

Trupthi M

Master of Computer Application[MCA]

BMS Institute of Technology and Management,

Avalahalli, Doddaballapura Main Road, Yelahanka, Bengaluru – 560 064.

Internship Dates :

01/03/2022 - 19/03/2022

Date of Submission :

04/03/2022

COMPANY BACKGROUND

The “DLithe Consultancy Services Pvt Ltd” group of industry veterans that have found the optimal balance of technology and human resources. Their aim is to make every student should be able to produce what the business requires right away. They provide students with the opportunity to gain excellent technology and coding abilities using the approaches employed by corporations.

BE AGILE. BE IN DEMAND!

The DLithe Internship@2022 Enables student’s community to bridge gaps between academics and industry. Developing an Agile workforce, Placement training & mentoring hackathons, Internship platforms are making resources employable.

Day 1

Introduction:

Data science is an interdisciplinary field that uses scientific methods, processes, algorithms and systems to extract knowledge from noisy, structured and unstructured data, and apply knowledge and actionable insights from data across a broad range of applications domains. Data science is related to data mining, machine learning and big data. Data science is growing at a staggering rate to fulfil the needs of Machine Learning and Deep Learning. Due to these rapid changes, individuals and businesses may find it baffling to keep track of the numerous changes in the industry. As the industry gets flooded with newer trends and techniques, businesses may feel the need to understand and adopt these to optimise their data handling and analysis process.

Customers Experience:

Get The Results You Want

Data allows organisations to measure the effectiveness of a given strategy: When strategies are put into place to overcome a challenge, collecting data will allow you to determine how well your solution is performing, and whether or not your approach needs to be tweaked or changed over the long-term.

Industrial Transformation

Data science has an incredible perspective for all types of businesses to design models that further define trends and use them as the foundation for transformative software, i.e. from locating IoT devices to predictive analytics.Digital transformation is the all-embracing transformation of multiple activities that an organisation control to leverage opportunities produced by digital technologies and data. It touches the ubiquitous era of digitalization regardless of the size and worthiness of the industry. Through a use of various technologies such as statistics, programming and mathematics, this new scientific field has grown into a highly lucrative career path.

SMAC

SMAC is the basis for an ecosystem that enables a business to transition from e-business to digital business. The four technologies improve business operations and help companies get closer to the customer with minimal overhead and maximum reach.

Social: Social media platforms such as Twitter, Facebook, Instagram and Snapchat have provided businesses with new ways to reach, interact with, target and acquire customers.

Mobile: Mobile technologies and platforms such as the iPhone and the iPad, have changed the way people communicate, shop and work.

Analytics: Data analytics allows businesses to understand how, when and where people consume certain goods and services. It is also used as a predictive indicator for future customer behaviour as well as when physical assets, such as parts of a jet engines, will experience degradation.

Cloud: Cloud computing provides a new way to access technology and the data a business needs to quickly respond to changing markets and solve business problems. It ushered in a new way to build infrastructure, platforms and services. Amazon Web Services was one of the big disruptors in this space.

Introduction to Python Programming

Python is a widely used high-level, general-purpose, interpreted, dynamic programming language. Its design philosophy emphasises code readability, and its syntax allows programmers to express concepts in fewer lines of code than would be possible in languages such as C++ or Java. The language provides constructs intended to enable clear programs on both a small and large scale. Python supports multiple programming paradigms, including object-oriented, imperative and functional programming or procedural styles. It features a dynamic type system and automatic memory management and has a large and comprehensive standard library. Python interpreters are available for installation on many operating systems, allowing Python code execution on a wide variety of systems.

Python advantages

1. Extensive support libraries(NumPy for numerical calculations, Pandas for data analytics etc)
2. Open source and community development
3. Versatile, Easy to read, learn and write
4. User-friendly data structures
5. High-level language
6. Dynamically typed language(No need to mention data type based on the value assigned, it takes data type)
7. Object-oriented language
8. Portable and Interactive
9. Ideal for prototypes – provide more functionality with less coding
10. Internet of things(IoT) opportunities
11. Interpreted Language
12. Portable across Operating system

Disadvantages of Python

1. Speed limitations
2. Weak in Mobile Computing
3. Design Restrictions
4. Memory Consumption

Day 2

Data types

Python provides various data types that define the storage method of each of them. Data types defined in python are:

1. Numbers
2. Sequence Types
3. Set
4. Dictionary
5. Boolean

❖ **Numbers**

Number stores numeric values. The integer, float, and complex values belong to a Python Numbers data-type. Python provides the type() function to know the data-type of the variable.

1. **Int -** Integer value can be any length such as integers 10, 2, 29, -20, -150 etc. Python has no restriction on the length of an integer. Its value belongs to **int**

   Eg: `type(1)`

   ```
   int
   ```

2. **Float -** Float is used to store floating-point numbers like 1.9, 9.902, 15.2, etc. It is accurate upto 15 decimal points.

Eg: type(12.4)

float

3. **complex -** A complex number contains an ordered pair, i.e., x + iy where x and y denote the real and imaginary parts, respectively. The complex numbers like 2.14j, 2.0 + 2.3j, etc.

Eg: type(1+3j)

complex

❖ Sequence Types

1. String

In Python, Strings are arrays of bytes representing Unicode characters. A string is a collection of one or more characters put in a single quote, double-quote or triple quote. In python there is no character data type, a character is a string of length one. It is represented by str class.

Eg: type('String')

Str

2. List

Lists are just like the arrays, declared in other languages which is an ordered collection of data. It is very flexible as the items in a list are of the same type. Lists in Python can be created by just placing the sequence inside the square brackets[].

Eg: type([1,2,3,4,5])

list

3. Tuple

Just like list, tuple is also an ordered collection of Python objects. The only difference between tuple and list is that tuples are immutable i.e. tuples cannot be modified after it is created. It is represented by tuple class.

Creating Tuple

In Python, tuples are created by placing a sequence of values separated by 'comma' with or without the use of parentheses for

grouping of the data sequence. Tuples can contain any number of elements and of any datatype (like strings, integers, list, etc.).

Note: Tuples can also be created with a single element, but it is a bit tricky. Having one element in the parentheses is not sufficient, there must be a trailing 'comma' to make it a tuple.

Eg: type((1,2,3,4,5,6))
tuple

❖ **Dictionary**

Dictionary in Python is an unordered collection of data values, used to store data values like a map, which unlike other Data Types that hold only single value as an element, Dictionary holds key:value pair. Key-value is provided in the dictionary to make it more optimized. Each key-value pair in a Dictionary is separated by a colon :, whereas each key is separated by a 'comma'.

Creating Dictionary

In Python, a Dictionary can be created by placing a sequence of elements within curly {} braces, separated by 'comma'. Values in a dictionary can be of any datatype and can be duplicated, whereas keys can't be repeated and must be immutable. Dictionary can also be created by the built-in function dict(). An empty dictionary can be created by just placing it to curly braces{}.

Note – Dictionary keys are case sensitive, same name but different cases of Key will be treated distinctly.

Eg: {'Name':'Trupthi','Roll num':57,'Age':22}
{'Age': 22, 'Name': 'Trupthi', 'Roll num': 57}

❖ **Boolean**

Boolean type provides two built-in values, True and False. These values are used to determine the given statement true or false. It denotes by the class bool. True can be represented by any non-zero value or 'T' whereas false can be represented by the 0 or 'F'. Consider the following example.

Eg: print(type(True))
<class 'bool'>

❖ **Set**

Python Set is the unordered collection of the data type. It is iterable, mutable(can modify after creation), and has unique elements. In set, the order of the elements is undefined; it may return the changed sequence of the element. The set is created by using a built-in function set(), or a sequence of elements is passed in the curly braces and separated by the comma. It can contain various types of values.

Eg:
```
set1 = {'Python', 1, 10, "Program"}
print((type(set1)))
```
<class 'set'>

Operators

1. **Arithmetic operators** : are used to performing mathematical operations like addition, subtraction, multiplication, and division.

Eg:
```
a=35
b=65

mul=a*b
div=a/b
add=a+b
sub=a-b
expo=a**b
floor_div=a//b

print(mul)
print(div)
print(add)
print(sub)
print(expo)
print(floor_div)
```

Output:
2275
0.5384615384615384
100
-30
23143171985618148385244286678913597734970588950324139906433544566777 2

```
717115440173074603080749511718750
```

2. **Relational Operator** : compares the values. It either returns **True** or **False** according to the condition.

   Eg:
   ```
   43>45
   False
   43!=45
   True
   45==45
   True
   ```

3. **Assignment Operator** : are used to assign values to the variables.

   Eg:
   ```
   x=15
   x+=5
   print(x)
   20
   ```

4. **Logical Operator** : perform **Logical AND**, **Logical OR**, and **Logical NOT** operations. It is used to combine conditional statements.

   Eg:
   ```
   12>25 and 25>20
   False
   12>25 or 25>20
   True
   not 'trupthi' == 'Trupthi'
   True
   ```

Day 3

 # What are Sequence in Python

Sequences are containers with items that are accessible by the indexing or string.

1. Concatenation
2. Repetition
3. Membership Testing
4. Slicing
5. Indexing

Creating a list of name lis

```
lis=[10,20,30,40,50]
print(lis)
```

#Append

```
lis.append(60)
print(lis)
```

```
      [10, 20, 30, 40, 50, 60]
```

#Insert

```
lis.insert(1,15)
print(lis)
```

```
      [10, 15, 20, 30, 40, 50, 60]
```

```
# to -do
#You have to insert 37 in 8th index and 2 in 12th index and 5 in 1000
index

lis=[10,15,20,30,40,50,60,70,80,90,25,35,45,55,65]
print(lis)
lis.insert(8,37)
lis.insert(12,2)
lis.insert(1000,5)
print(lis)
```

```
      [10, 15, 20, 30, 40, 50, 60, 70, 80, 90, 25, 35, 45, 55, 65]
[10, 15, 20, 30, 40, 50, 60, 70, 37, 80, 90, 25, 2, 35, 45, 55, 65, 5]
```

#Extend Function

```
lis2=[1,2,3,4,5]
lis.extend(lis2)
lis2.extend(lis)
print(lis)
print(lis2)
```

```
      [10, 15, 20, 30, 40, 50, 60, 70, 37, 80, 90, 25, 2, 35, 45, 55,
65, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5]
[1, 2, 3, 4, 5, 10, 15, 20, 30, 40, 50, 60, 70, 37, 80, 90, 25, 2, 35,
45, 55, 65, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5]
```

#Sum

```
print(sum(lis))
```

```
764
```

#count

```
print(lis.count(1))
```

```
2
```

#Indexing

```
lis.index(37)
```

```
8
```

```
lis[5]
```

```
50
```

```
lis[6:10]
```

```
[60, 70, 37, 80]
```

#min

```
min(lis)
```

```
1
```

#max

```
max(lis)
```

```
90
```

#popup function

```
a=lis.pop(8)
print(lis)
print(a)
```

```
[10, 15, 20, 30, 40, 50, 60, 70, 80, 90, 25, 2, 35, 45, 55, 65, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5]
37
```

```
lis.pop()
```

```
5
```

#delete

```
del lis[7]
print(lis)
```

```
[10, 15, 20, 30, 40, 50, 60, 80, 90, 25, 2, 35, 45, 55, 65, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4]
```

#remove

```
b=lis.remove(15)
```

```
print(lis)
print(b)
```

```
      [10, 20, 30, 40, 50, 60, 80, 90, 25, 2, 35, 45, 55, 65, 5, 1, 2,
3, 4, 5, 1, 2, 3, 4]
None
```

```
#To Do
#a = 12
#b = 3
# c = 45
#Write a swapping function where a == c, b ==a, c = b
```

```
a=12
b=3
c=45
a,b,c = c,a,b
print(a)
print(b)
print(c)
```

```
      45
      12
      3
```

String Methods

1. capitalize():Converts the first character to upper case
2. count():Returns the number of times a specified value occurs in a string
3. find():Searches the string for a specified value and returns the position of where it was found
4. islower():Returns True if all characters in the string are lower case
5. isupper():Returns True if all characters in the string are upper case
6. strip():Returns a trimmed version of the string
7. split():Splits the string at the specified separator, and returns a list
8. replace():Returns a string where a specified value is replaced with a specified value

```
name = 'dlitHe Institute'
course = 'python certification'
```

```
course.capitalize()
      Python Certification
course.count("t")
      3
course.find('c')
      7
course.isalpha()
      False
course.islower()
      True
course.isupper()
      False
course.lower()
      python certification
course.upper()
      PYTHON CERTIFICATION
course.strip()
      python certification
course.split(" ")
      ['python', 'certification']
a='python'
b='language'
a+" "+b
      python language
a*3
      Pythonpythonpython
str1='Hello, This is python programming class'
len(str1)
      39
str1[8]
      h
str1[5:10]
      , Thi
str1.replace('Hello','Hi')
      Hi, This is python programming class
```

Tuples

```
this1=('Banana','mango','orange','melon','kiwi','mango')
this1[2]
```

```
      Orange
this1[1:3]
      ('mango', 'orange')
tuple1=(1,2,3,4,5)
tuple2=('abhbd',2,3,6,'bhj')
tuple3 = tuple1+tuple2
print(tuple3)
      (1, 2, 3, 4, 5, 'abhbd', 2, 3, 6, 'bhj')
tup=tuple("hello world")
print(tup)
      ('h', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd')
tu=tuple("hello",) * 3
print(tu)
      ('h', 'e', 'l', 'l', 'o', 'h', 'e', 'l', 'l', 'o', 'h', 'e', 'l',
'l', 'o')
#tuple unpacking

tuple4=("hello","good","people")
a,b,c=tuple4
print(a)
print(b)
print(c)
      hello
      good
      people
```

#Dictionary

```
dic={"Name":"Trupthi","Age":22}
#deleting the key
del dic["Name"]
print(dic)
      {'Age': 22}
#adding key to the existing dictionary
dic["Number"] = 25648
dic["Name"] = "Trupthi"
print(dic)
      {'Age': 22, 'Number': 25648, 'Name': 'Trupthi'}
#pop
dic.pop("Age")
      22
print(dic)
      {'Number': 25648, 'Name': 'Trupthi'}
#creation of dictionary
```

```
abc = dict(Brand="Ford",Model="s-series",Year=2021)
print(abc)
      {'Brand': 'Ford', 'Model': 's-series', 'Year': 2021}
#copy dictionary
x = dict(abc)
print(x)
      {'Brand': 'Ford', 'Model': 's-series', 'Year': 2021}
#accessing a value from dictionary
x['Year']
      2021
#nested dictionary
di={'Dict1':{1:'get'},'Dict2':{'Name':'For'}}
di['Dict1'][1]
      get
#empty dictionary
di.clear()
print(di)
      {}
```

Conditional Statements in Python

Decision-making in a programming language is automated using conditional statements, in which Python evaluates the code to see if it meets the specified conditions.

The conditions are evaluated and processed as true or false. If this is found to be true, the program is run as needed. If the condition is found to be false, the statement following the If condition is executed.

The following are conditional statements in python :-

1. If the statement
2. If else statement
3. If..elif else statement
4. Nested if statement
5. Short Hand if statement

<u>If Statement</u>

The If statement is the most fundamental decision-making statement, in which the code is executed based on whether it meets the specified condition. It has a

code body that only executes if the condition in the if statement is true. The statement can be a single line or a block of code.

Syntax:

if expression:

Statement

Eg:

```
i = 5
if(i<25):
  print("i is less")

print("I am not in if block")
```

```
i is less
I am not in if block
```

If Else Statement

This statement is used when both the true and false parts of a given condition are specified to be executed. When the condition is true, the statement inside the if block is executed; if the condition is false, the statement outside the if block is executed.

Syntax:

if condition :

#Will executes this block if the condition is true

else :

#Will executes this block if the condition is false

Eg:

```
i=20

if(i<15):

  print("i is smaller than 15")

  print("I am in if block")
```

```
else:
  print("i is greater than or equal to 15")
  print("I am is else block")
```

```
i is greater than or equal to 15
I am is else block
```

Day 5

If…Elif..else Statement

In this case, the If condition is evaluated first. If it is false, the Elif statement will be executed; if it also comes false, the Else statement will be executed.

Syntax:

if condition :
Body of if
elif condition :
Body of elif
else:
Body of else

Eg:

```
num = 7
if num > 0:
    print("Positive number")
elif num == 0:
    print("Zero")
else:
    print("Negative number")
```

```
Positive number
```

Nested IF Statement

A Nested IF statement is one in which an If statement is nestled inside another If statement. This is used when a variable must be processed more than once. If, If-else, and If…elif…else statements can be used in the program. In Nested If statements, the indentation (whitespace at the beginning) to determine the scope of each statement should take precedence.

Syntax:

if (condition1):
#Executes if condition 1 is true
if (condition 2):
#Executes if condition 2 is true
#Condition 2 ends here
#Condition 1 ends here

Eg:

```
i=4

if(i==40):  #if i=40 then it enters into this block else the else block will be executed

  if(i<=15): #if it enters to the if block, if check if i<=15 since it is false it don't enter into if block

    print("i is smaller than 15")

  if(i<12): #next condition is checked and this condition also fails hence else block will be executed if the i value is equal to 40

    print("i is smaller than 12 too")

  else:

    print("i is greater than 15")

else:

  print("i is not 40")
```

```
i is not 40
```

Short Hand if statement

Short Hand if statement is used when only one statement needs to be executed inside the if block. This statement can be mentioned in the same line which holds the If statement.

Syntax:

if condition: statement

Eg:

```
i=10
if i<=10:print('i is less than 10')
```

```
i is less than 10
```

Looping statement

A loop statement allows us to execute a statement or group of statements multiple times.

Following are the types of loops:

1. For Loop
2. While Loop

For Loop : A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).This is less like the for keyword in other programming languages, and works more like an iterator method as found in other object-oriented programming languages. With the for loop we can execute a set of statements, once for each item in a list, tuple, set etc.

Syntax:

for val in sequence:
loop body

Here, val is the variable that takes the value of the item inside the sequence on each iteration.

Eg:

```
l=[1,2,3,4,5,6,7]
for v in l:
    print(v)
```

Output:

1
2
3
4
5
6
7

While Loop : With the while loop we can execute a set of statements as long as a condition is true.

Syntax:

```
while test_expression:
  Body of while
```

Eg:

```
count = 0
while (count < 3):
        count+=1
        print("hello",count)
```

Output: hello 1
hello 2
hello 3

Loop Control Statements:

1. break statement: Terminates the loop statement and transfers execution to the statement immediately following the loop.
2. continue statement: Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.
3. pass statement: The pass statement in Python is used when a statement is required syntactically but you do not want any command or code to execute.

Examples of break, continue and pass statements are given below respectively:

Break Statement :

```
i=0
a='helloworld'
while i<len(a):
  if a[i] == 'e':
```

```
    i+=1
    break
  print('Current letter',a[i])
  i+=1
```

Output:
```
Current letter h
```

Continue Statement:

```
i=0
a='helloworld'
while i<len(a):
  if a[i] == 'e':
    i+=1
    continue
  print('Current letter',a[i])
  i+=1
```

```
Output: Current letter h
        Current letter l
        Current letter l
        Current letter o
        Current letter w
        Current letter o
        Current letter r
        Current letter l
        Current letter d
```

Pass Statement:

```
i=0
a='helloworld'
while i<len(a):
  if a[i] == 'e':
    i+=1
    pass
  print('Current letter',a[i])
  i+=1
```

Output:
```
Current letter h
Current letter l
Current letter l
Current letter o
Current letter w
Current letter o
Current letter r
Current letter l
```

```
Current letter d
```

Day 6

Zip Function: The zip() function returns a zip object, which is an iterator of tuples where the first item in each passed iterator is paired together, and then the second item in each passed iterator are paired together etc.

Eg:

```
questions=['name','color','shape']
answers=['apple','red','a circle']
for question, answer in zip(questions,answers):
  print('What is your {0}? I am {1}.'.format(question,answer))
```

Output:

```
What is your name? I am apple.
What is your color? I am red.
What is your shape? I am a circle.
```

Map Function: The map() function executes a specified function for each item in an iterable. The item is sent to the function as a parameter.

Eg1:

```
b=list(map(int, input().strip()))
```

Output:

```
12345
```

Lambda Function: Python Lambda Functions are anonymous functions meaning that the function is without a name. As we already know that the *def* keyword is used to define a normal function in Python. Similarly, the *lambda* keyword is used to define an anonymous function in Python.

Eg:

```
def db(n):
  return lambda x:x*n
num_double = db(3)
print(num_double(13))
```

Output:

```
39
```

Separators: The separator between the arguments to print() function in Python is space by default (softspace feature) , which can be modified and can be made to any character, integer or string as per our choice. The 'sep' parameter is used to achieve the same, it is found only in python 3.x or later. It is also used for formatting the output strings.

Eg:

```
print('08','03','2022', sep = '-')
```

Output:

```
08-03-2022
```

Day 7

Pandas: Pandas is an open source library that is designed to make work with relational or labelled data both easy and intuitively. It provides various data structures and operations for manipulating numerical data and time series. This library is built on top of the NumPy library. Pandas is fast and it has high performance & productivity for users. It is also a high level building block for real time data analysis.

To install pandas all you need to do is:

!pip3 install pandas

To import pandas you need to do is:

import pandas as pd #here pd refers to alias name

Pandas Series: A Pandas Series is like a column in a table. It is a one-dimensional array holding data of any type.

Eg:

```
data=[1,2,3,4,5,6]
d=pd.Series(data)
print(d)
```

Output:

```
0    1
1    2
2    3
```

```
3    4
4    5
5    6
dtype: int64
```

DataFrame: A Pandas DataFrame is a 2 dimensional data structure, like a 2 dimensional array, or a table with rows and columns.

Eg:

```
s1=pd.Series([1,2,3,4,5,6,7])
s2=pd.Series([1.1,3.9,2.7,4.4,5.5,9.7,4.6])
s3=pd.Series(['a','b','c','d','e'])
data={'first':s1,'second':s2,'third':s3}
dfseries=pd.DataFrame(data)
print(dfseries)
```

Output:

```
      first  second third
0       1     1.1     a
1       2     3.9     b
2       3     2.7     c
3       4     4.4     d
4       5     5.5     e
5       6     9.7   NaN
6       7     4.6   NaN
```

Read CSV Files: To read a csv file we use the following syntax:df.to_csv('basic.csv')

Converting a data frame into csv file

```
df = pd.DataFrame({'A':[1,2,3,4],
                   'B':pd.Timestamp('20220310'),
                   'C':pd.Series(1,index=list(range(4))),
                   'D':np.array([5]*4, dtype = 'int32'),
                  'E': pd.Categorical(['test','train','train','test'])})
```

Output:

head(): The head() method returns the headers and a specified number of rows, starting from the top.

tail(): The tail() method returns the headers and a specified number of rows, starting from the bottom.

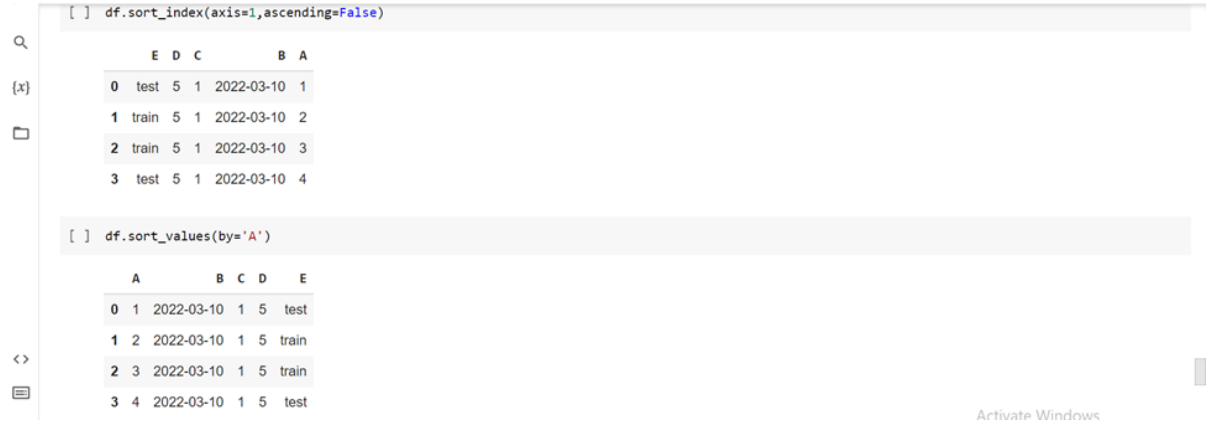
info(): The DataFrames object has a method called info(), that gives you more information about the data set.

isnull(): The isnull() method returns a DataFrame object where all the values are replaced with a Boolean value True for NULL values, and otherwise False.

dropna(): The dropna() method removes the rows that contains NULL values

fillna(): The fillna() method replaces the NULL values with a specified value.

df.sort_values(): The sort_values() method sorts the DataFrame by the specified label.



```
[ ] df.sort_index(axis=1,ascending=False)
```

	E	D	C	B	A
0	test	5	1	2022-03-10	1
1	train	5	1	2022-03-10	2
2	train	5	1	2022-03-10	3
3	test	5	1	2022-03-10	4

```
[ ] df.sort_values(by='A')
```

	A	B	C	D	E
0	1	2022-03-10	1	5	test
1	2	2022-03-10	1	5	train
2	3	2022-03-10	1	5	train
3	4	2022-03-10	1	5	test

df.iloc[]: The iloc property gets, or sets, the value(s) of the specified indexes.Specify both row and column with an index.To access more than one row, use double brackets and specify the indexes, separated by commas:df.iloc[[0,2]]

```
[ ] #From position
    # 3 is ur index
    df.iloc[3]
```

```
A                    4
B    2022-03-10 00:00:00
C                    1
D                    5
E                 test
Name: 3, dtype: object
```

```
[ ] #0:2 is row
    #3:5 is col
    df.iloc[0:2,3:5]
```

	D	E
0	5	test
1	5	train

For more explanation on Pandas please check the below link:
https://colab.research.google.com/drive/1CviYz78OAoOIFgOsh1mCWlp4utCgs97n?usp=sharing

Day 9

Numpy: NumPy, which stands for Numerical Python, is a library consisting of multidimensional array objects and a collection of routines for processing those arrays. Using NumPy, mathematical and logical operations on arrays can be performed. NumPy is a Python package. It stands for 'Numerical Python'.

To install numpy all you need to do is:
!pip3 install numpy
To import pandas you need to do is:
import numpy as np #here np refers to alias name

arange(): The arange([start,] stop[, step,][, dtype]) : Returns an array with evenly spaced elements as per the interval. The interval mentioned is half-opened i.e. [Start, Stop)

ndarray: An array class in Numpy is called ndarray. Elements in Numpy arrays are accessed by using square brackets and can be initialised by using nested Python Lists.

linspace(): The numpy.linspace() function returns number spaces evenly w.r.t interval. Similar to numpy.arange() function but instead of step it uses sample number.

random rand:The numpy.random.rand() function creates an array of specified shape and fills it with random values.

randint: The randint() method returns an integer number selected element from the specified range.

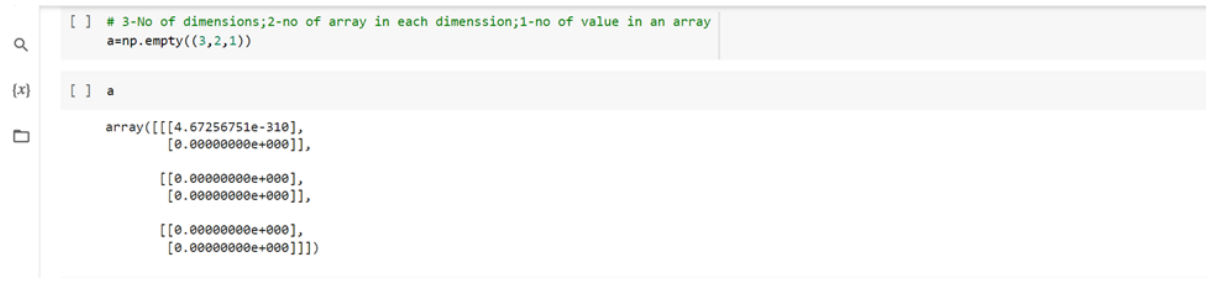
vstack():np.vastack()function is used to stack the sequence of input arrays vertically to make a single array.

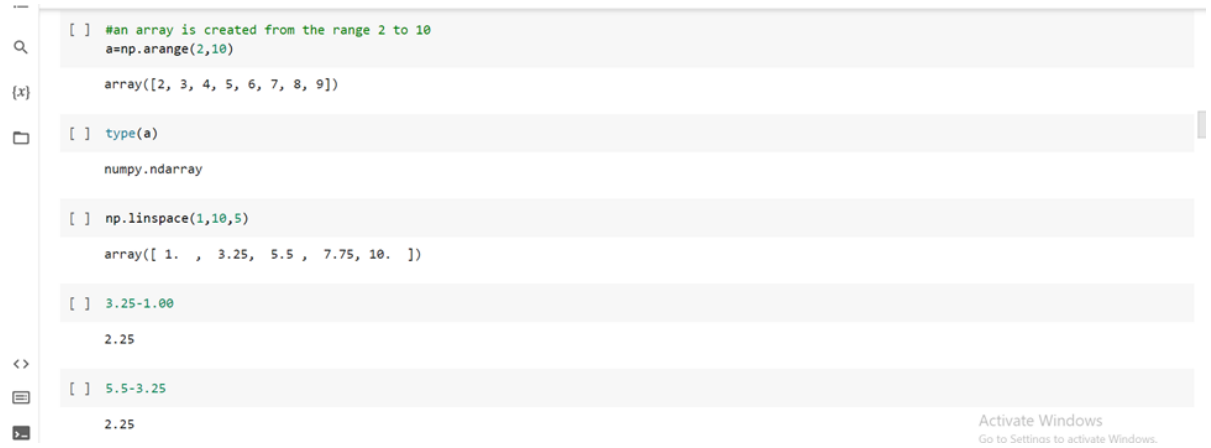
hstack():np.hstack() function is used to stack the sequence of input arrays horizontally (i.e. column wise) to make a single array.

vsplit():vsplit() function splits an array into multiple sub-arrays vertically (row-wise). vsplit is equivalent to split with axis=0 (default), the array is always split along the first axis regardless of the array dimension.

ndim():ndim() function **returns the number of dimensions of an array**. Parameters : arr : [array_like] Input array. If it is not already an ndarray, a conversion is attempted.

reshape(): Reshaping means changing the shape of an array.The shape of an array is the number of elements in each dimension.By reshaping we can add or remove dimensions or change the number of elements in each dimension.





```
[ ] np.random.rand(2,2,2)

    array([[[0.49878564, 0.15727884],
            [0.42390269, 0.11739902]],

           [[0.96864526, 0.26016479],
            [0.95684784, 0.14055984]]])

[ ] d=np.random.randint(0,10,size=(5,5))

[ ] d

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

```
[ ] #concatenation
    np.vstack([a,b])

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

[ ] np.hstack((a,b))

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

Day 10

Matplotlib: Matplotlib is an amazing visualisation library in Python for 2D plots of arrays. Matplotlib is a multi-platform data visualisation library built on NumPy arrays and designed to work with the broader SciPy stack.

To install matplotlib all you need to do is:

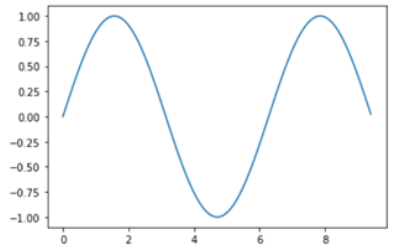
!pip3 install matplotlib

To import pandas you need to do is:

import matplotlib.pyplot as plt #here plt refers to alias name

A basic line plot example

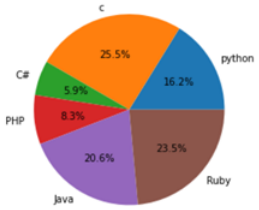
```
[ ] import numpy as np
    import matplotlib.pyplot as plt

[ ] x = np.arange(0,3*np.pi,0.1)
    y = np.sin(x)

[ ] plt.plot(x,y)

    [<matplotlib.lines.Line2D at 0x7f4468e7c190>]
```



Types of plot in matplotlib:

- **Pie Chart:**Pie charts represent data broken down into categories/labels. They're an intuitive and simple way to visualise proportional data — such as percentages.

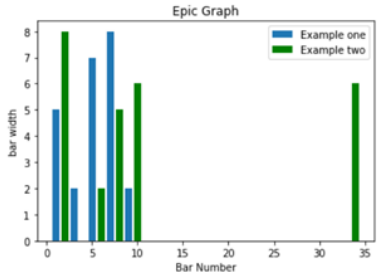
```
[ ] labels = ['python','c','C#','PHP','Java','Ruby']
    sizes = [33,52,12,17,42,48]
    plt.pie(sizes, labels= labels, autopct='%1.1f%%')
    plt.axis('equal')

    (-1.1037775963297032,
     1.1001799283090257,
     -1.117024172809282,
     1.1099314904660127)
```



- Bar Graph:A bar plot or bar chart is a graph that represents the category of data with rectangular bars with lengths and heights that is proportional to the values which they represent. The bar plots can be plotted horizontally or vertically.

```
[ ]  plt.bar([1,3,5,7,9],[5,2,7,8,2], label = 'Example one')
     plt.bar([2,34,6,8,10],[8,6,2,5,6],label = 'Example two',color = 'green')
     plt.legend()
     plt.ylabel('bar width')
     plt.xlabel('Bar Number')
     plt.title('Epic Graph')
```

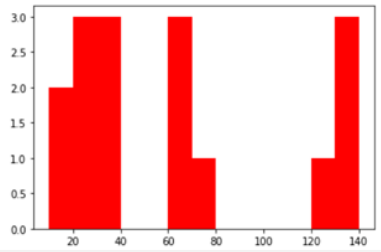
```
Text(0.5, 1.0, 'Epic Graph')
```



- **Histogram:**A histogram is a graphical representation that organizes a group of data points into user-specified ranges. Similar in appearance to a bar graph, the histogram condenses a data series into an easily interpreted visual by taking many data points and grouping them into logical ranges or bins.

```
[ ]  population_ages = [12,34,31,23,122,15,145,543,135,135,134,351,63,24,72,151,63,24,62,31]
     bins = [10,20,30,40,50,60,70,80,90,100,110,120,130,140]
     plt.hist(population_ages,bins,color = 'red')
```

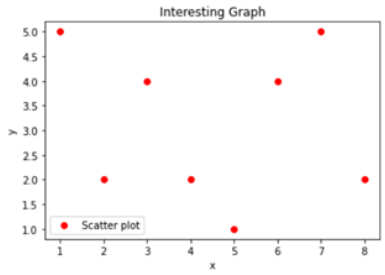
```
(array([2., 3., 3., 0., 0., 3., 1., 0., 0., 0., 0., 1., 3.]),
 array([ 10,  20,  30,  40,  50,  60,  70,  80,  90, 100, 110, 120, 130,
        140]),
 <a list of 13 Patch objects>)
```



- **Scatter Plot: The** scatter() function plots one dot for each observation. It needs two arrays of the same length, one for the values of the x-axis, and one for values on the y-axis:

```
[ ]  x = [1,2,3,4,5,6,7,8]
     y = [5,2,4,2,1,4,5,2]
     plt.scatter(x,y,label = 'Scatter plot',color = 'r')
     plt.ylabel('y')
     plt.xlabel('x')
     plt.title('Interesting Graph')
     plt.legend()
```

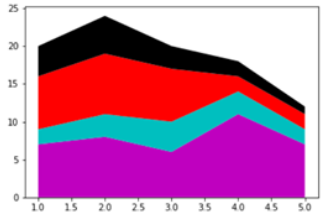
```
<matplotlib.legend.Legend at 0x7f2f8e9da090>
```



- **Stackplot:**Stackplot is used to draw a stacked area plot. It displays the complete data for visualization. It shows each part stacked onto one another and how each part makes the complete figure. It displays various constituents of data and it behaves like a pie chart. It has x-label, y-label, and title in which various parts can be represented by different colors.

```
[ ]  # Stackplot

     days = [1,2,3,4,5]
     sleeping = [7,8,6,11,7]
     eating = [2,3,4,3,2]
     working = [7,8,7,2,2]
     playing = [4,5,3,2,1]
     plt.stackplot(days,sleeping,eating,working,playing,colors = ['m','c','r','k'])
```

```
[<matplotlib.collections.PolyCollection at 0x7f2f8da10190>,
 <matplotlib.collections.PolyCollection at 0x7f2f8da10810>,
 <matplotlib.collections.PolyCollection at 0x7f2f8d9f9fd0>,
 <matplotlib.collections.PolyCollection at 0x7f2f8dec8a50>]
```



To know more about numpy and matplotlib check the below link
https://colab.research.google.com/drive/1CviYz78OAoOIFgOsh1mCWlp4utCgs97n?usp=sharing

Day 11

Working on churn dataset

```
[ ] import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt
```

```
[ ] data = pd.read_csv('telecom_churn - telecom_churn.csv')
```

```
[ ] data.head()
```

	State	Account length	Area code	International plan	Voice mail plan	Number vmail messages	Total day minutes	Total day calls	Total day charge	Total eve minutes	Total eve calls	Total eve charge	Total night minutes	Total night calls	Total night charge	Total intl minutes	Total intl calls	Total intl charge	Customer service calls	Churn
0	KS	128	415	No	Yes	25	265.1	110	45.07	197.4	99	16.78	244.7	91	11.01	10.0	3	2.70	1	False
1	OH	107	415	No	Yes	26	161.6	123	27.47	195.5	103	16.62	254.4	103	11.45	13.7	3	3.70	1	False
2	NJ	137	415	No	No	0	243.4	114	41.38	121.2	110	10.30	162.6	104	7.32	12.2	5	3.29	0	False
3	OH	84	408	Yes	No	0	299.4	71	50.90	61.9	88	5.26	196.9	89	8.86	6.6	7	1.78	2	False
4	OK	75	415	Yes	No	0	166.7	113	28.34	148.3	122	12.61	186.9	121	8.41	10.1	3	2.73	3	False

For more exploring on churn dataset follow the below link
https://colab.research.google.com/drive/1NymDwsYmmRY8nQRGO7oMQfGIvWXddFfn?usp=sharing

Conclusion

I had a good time doing this internship, which enabled me to know more about Data science. Thank you.