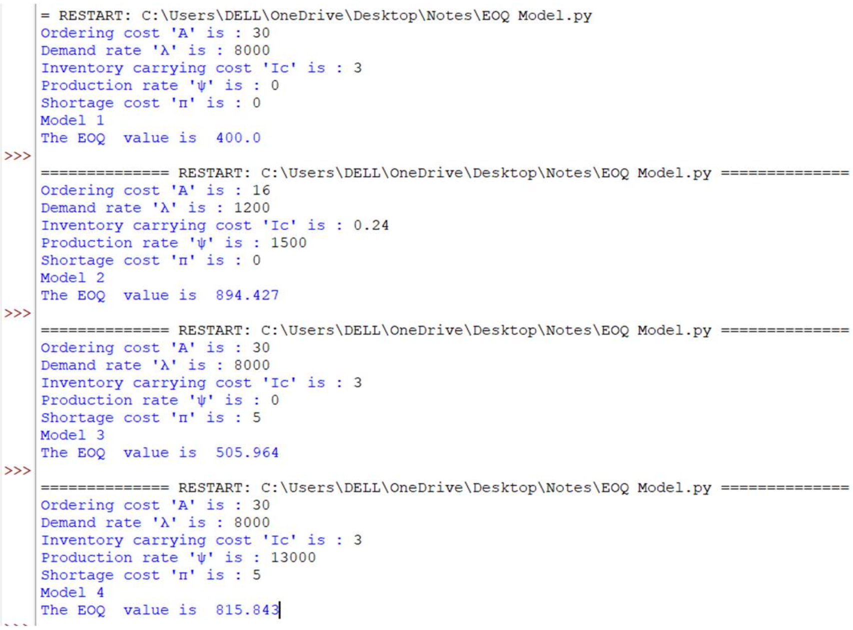
PRACTICAL 1

Objective : Find the Optimal order quantity(Q*) for each EOQ model.

CODE :

```
### ECONOMIC ORDER QUANTITY ###
# Import required libraries
import math

A=float(input("Ordering cost 'A' is : ") )
D=float(input("Demand rate 'λ' is : "))
Ic=float(input("Inventory carrying cost 'Ic' is : ") )
P=float(input("Production rate 'ψ' is : "))
Pi=float(input("Shortage cost 'π' is : ") )

if(P==0 and Pi==0): #MODEL 1
  print("Model 1")
  Q= math.sqrt(2*A*D/Ic)

elif (Pi==0): #MODEL 2
  print("Model 2")
  Q=math.sqrt((2*A*D*P)/((P-D)*Ic))

elif (P==0): #MODEL 3
  print("Model 3")
  Q= math.sqrt((2*A*D*(Ic+Pi))/(Ic*Pi))

else : #MODEL 4
  print("Model 4")
  Q=math.sqrt((2*A*D*P*(Ic+Pi))/(Ic*Pi*(P-D)))

Q=round(Q,3) #round off for 3places

# Run example
print("The EOQ  value is " ,Q)
```



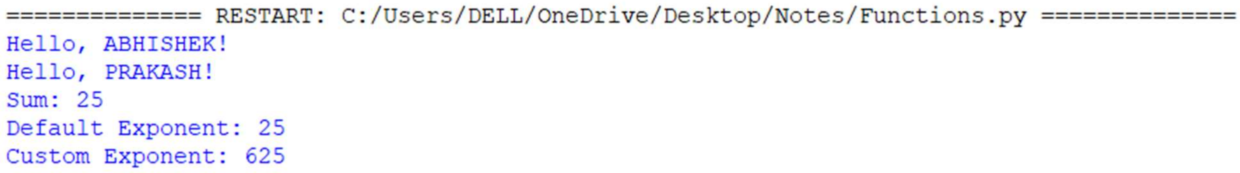
```
= RESTART: C:\Users\DELL\OneDrive\Desktop\Notes\EOQ Model.py
Ordering cost 'A' is : 30
Demand rate 'λ' is : 8000
Inventory carrying cost 'Ic' is : 3
Production rate 'ψ' is : 0
Shortage cost 'π' is : 0
Model 1
The EOQ  value is  400.0
>>> 
============== RESTART: C:\Users\DELL\OneDrive\Desktop\Notes\EOQ Model.py ==============
Ordering cost 'A' is : 16
Demand rate 'λ' is : 1200
Inventory carrying cost 'Ic' is : 0.24
Production rate 'ψ' is : 1500
Shortage cost 'π' is : 0
Model 2
The EOQ  value is  894.427
>>> 
============== RESTART: C:\Users\DELL\OneDrive\Desktop\Notes\EOQ Model.py ==============
Ordering cost 'A' is : 30
Demand rate 'λ' is : 8000
Inventory carrying cost 'Ic' is : 3
Production rate 'ψ' is : 0
Shortage cost 'π' is : 5
Model 3
The EOQ  value is  505.964
>>> 
============== RESTART: C:\Users\DELL\OneDrive\Desktop\Notes\EOQ Model.py ==============
Ordering cost 'A' is : 30
Demand rate 'λ' is : 8000
Inventory carrying cost 'Ic' is : 3
Production rate 'ψ' is : 13000
Shortage cost 'π' is : 5
Model 4
The EOQ  value is  815.843
```

PRACTICAL 2

Objective : Write a program using Function in Python.

CODE :

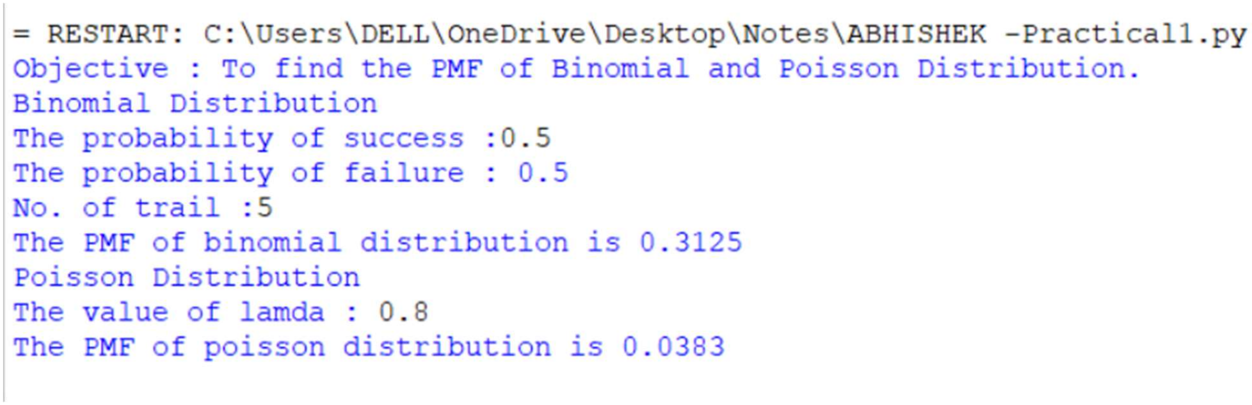
```
# Define a simple function
def name(name):
    print(f"Hello, {name}!")
# Call the function
name("ABHISHEK")
name("PRAKASH")

# Define a function with a return value
def sum(a, b):
    add = a + b
    return add
# Call the function and store the result in a variable
sum_result = sum(10, 15)
print("Sum:", sum_result)

# Function with default parameter value
def power(base, exponent=2):
    """This function calculates the power of a number with an optional
exponent."""
    output = base ** exponent
    return output


# Call the function with and without providing the exponent
default_value = power(5) # take default exponent value as 2
Custom_value = power(5, 4) #if change the exponent value
print("Default Exponent:", default_value)
print("Custom Exponent:", Custom_value)
```



```
============== RESTART: C:/Users/DELL/OneDrive/Desktop/Notes/Functions.py ==============
Hello, ABHISHEK!
Hello, PRAKASH!
Sum: 25
Default Exponent: 25
Custom Exponent: 625
```

Q . Find the PMF of Binomial and Poisson distribution using Function.

```
print("Objective : To find the PMF of Binomial and Poisson Distribution.")

import math
def bino(x): #Binomial Distribution
    print("Binomial Distribution")
    p=float(input("The probability of success :"))
    print("The probability of failure :" ,1-p)
    n=int(input("No. of trail :"))
    PMF=math.factorial(n)*(p**x)*((1-p)**(n-x))/(math.factorial(n-x)
        *math.factorial(x))
    #PMF of Poission = n!*p^x*(1-p)^n-x /(n-x)!*x!
    print("The PMF of binomial distribution is" ,PMF)

def poisson(x): #Poisson Distribution
    print("Poisson Distribution")
    lamda=float(input("The value of lamda : "))
    e=2.7182
    P=round((e**(-lamda)*lamda**x)/math.factorial(x),4)
    #PMF of Poission =(e^-lamda*lamda^x)/x! and round of by 4 decimal
    print("The PMF of poisson distribution is" ,P)

bino(2)
poisson(3)
```



```
= RESTART: C:\Users\DELL\OneDrive\Desktop\Notes\ABHISHEK -Practical1.py
Objective : To find the PMF of Binomial and Poisson Distribution.
Binomial Distribution
The probability of success :0.5
The probability of failure : 0.5
No. of trail :5
The PMF of binomial distribution is 0.3125
Poisson Distribution
The value of lamda : 0.8
The PMF of poisson distribution is 0.0383
```

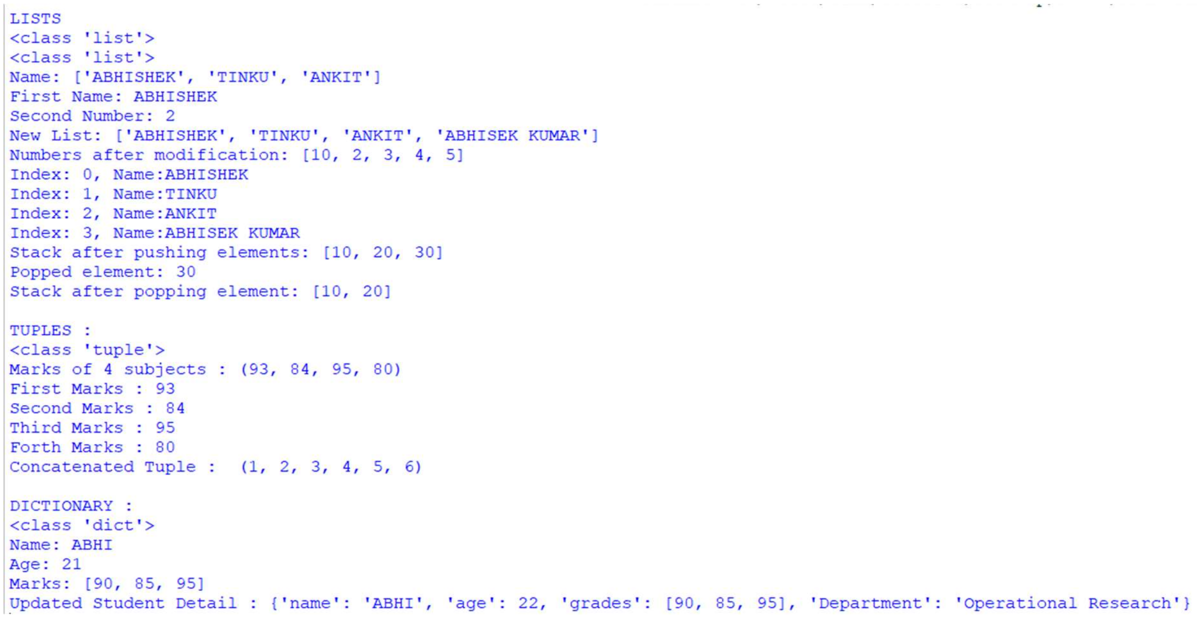
PRACTICAL 3

Q. Data Structures : List, Tuple and Dictionary in Python.

CODE :

```
# Lists
print("LISTS")
Name = ['ABHISHEK', 'TINKU', 'ANKIT']
numbers = [1, 2, 3, 4, 5]
print(type(Name))
print(type(numbers))

# Accessing elements in a list
print("Name:", Name)
print("First Name:", Name[0])
print("Second Number:", numbers[1])

# Modifying a list
Name.append('ABHISEK KUMAR')
numbers[0] = 10

print("New List:", Name)
print("Numbers after modification:", numbers)

#enumerate in List
for index,value in enumerate(Name):
    print(f"Index: {index}, Name:{value}")

#List as a Stack
stack = [] #empty list to act as a stack
stack.append(10)
stack.append(20)
stack.append(30)

# Displaying the current state of the stack
print("Stack after pushing elements:", stack)
# Popping elements from the stack
popped_element = stack.pop()
print("Popped element:", popped_element)
# Displaying the updated state of the stack
print("Stack after popping element:", stack)
print("")

# Tuples
print("TUPLES : ")
Marks = (93, 84, 95, 80)
print(type(Marks))
# Accessing elements in a tuple
```

```python
print("Marks of 4 subjects :", Marks)
print("First Marks :", Marks[0])
print("Second Marks :", Marks[1])
print("Third Marks :", Marks[2])
print("Forth Marks :", Marks[3])

# Modification in Tuple is not possible because Tuples are Immutable
# Marks[0] = 50  TypeError: 'tuple' object does not support item assignment

#Concatenation
tuple1=(1,2,3)
tuple2=(4,5,6)
tuple3=tuple1 + tuple2
print("Concatenated Tuple : " ,tuple3)

# Dictionaries
print("")
print("DICTIONARY : ")
Student = {'name': 'ABHI', 'age': 21, 'grades': [90,85,95]}
print(type(Student))

# Accessing Values
print("Name:", Student['name'])
print("Age:", Student['age'])
print("Marks:", Student['grades'])

# Modifying a dictionary
Student['age'] = 22
Student['Department'] = 'Operational Research'
print("Updated Student Detail :", Student)
```



```
LISTS
<class 'list'>
<class 'list'>
Name: ['ABHISHEK', 'TINKU', 'ANKIT']
First Name: ABHISHEK
Second Number: 2
New List: ['ABHISHEK', 'TINKU', 'ANKIT', 'ABHISEK KUMAR']
Numbers after modification: [10, 2, 3, 4, 5]
Index: 0, Name:ABHISHEK
Index: 1, Name:TINKU
Index: 2, Name:ANKIT
Index: 3, Name:ABHISEK KUMAR
Stack after pushing elements: [10, 20, 30]
Popped element: 30
Stack after popping element: [10, 20]

TUPLES :
<class 'tuple'>
Marks of 4 subjects : (93, 84, 95, 80)
First Marks : 93
Second Marks : 84
Third Marks : 95
Forth Marks : 80
Concatenated Tuple :  (1, 2, 3, 4, 5, 6)

DICTIONARY :
<class 'dict'>
Name: ABHI
Age: 21
Marks: [90, 85, 95]
Updated Student Detail : {'name': 'ABHI', 'age': 22, 'grades': [90, 85, 95], 'Department': 'Operational Research'}
```

PRACTICAL 4

Objective: Write a program of calculating Correlation and Regression.

Regression:

CODE :

```
from scipy import stats
x=[5,9,15,6,8,10]
y=[2,5,6,5,6,5]

slope, intercept,r,p, std_error=stats.linregress(x,y)
a=round(intercept,4)
b=round(slope,4)
print('The intercept is',a)
print('The slope is' , b)
print('The  correlation coefficient is' , round(r,4))
print('The value P is' , round(p,4))

def myfun(x):
    y=round((a+b*x),4)
    print('The required equation is y=',a,'+' ,b,'x')
    print('Y=',y)
myfun(5)
```

OUTPUT :

```
The intercept is 2.4668
The slope is 0.2679
The  correlation coefficient is 0.6452
The value P is 0.1665
The required equation is y= 2.4668 + 0.2679 x
Y= 3.8063
```

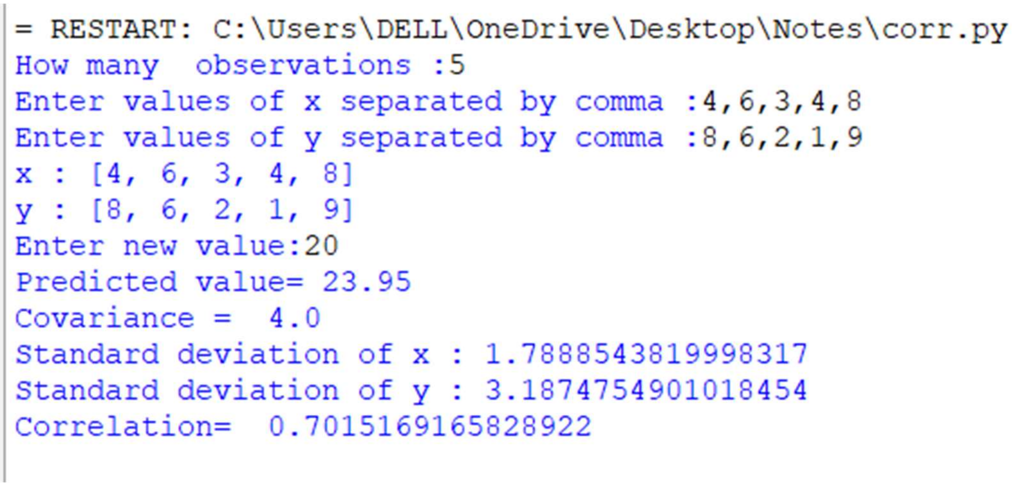
Correlation :

CODE :

```
import math

#taking input values from user
n=int(input("How many  observations :"))
x=list(map(int,input("Enter values of x separated by comma :").split(",")))
y=list(map(int,input("Enter values of y separated by comma :").split(",")))
print("x :",x)
print("y :",y)

#calculating mean of x and y sets
mean_x=sum(x)/len(x)
mean_y=sum(y)/len(y)
numerator=0
denominator=0

#calculating slope by finding numerator and denominator separately
for i in range(n):
    numerator+=(x[i]-mean_x)*(y[i]-mean_y)
    denominator+=(x[i]-mean_x)**2
slope=numerator/denominator
intercept=mean_y-slope*mean_x

#prediction calculation-regression
new_value=int(input("Enter new value:"))
predicted=slope*new_value+intercept
print("Predicted value=",predicted)

#covariance
covariance=numerator/n
print("Covariance = ",covariance)
sx=0
sy=0
for i in range(n):
    sx+=(x[i]-mean_x)**2
    sy+=(y[i]-mean_y)**2
```

```
#finding
std_x=math.sqrt(sx/n)
std_y=math.sqrt(sy/n)
corr=covariance/(std_x*std_y)
print("Standard deviation of x :",std_x)
print("Standard deviation of y :",std_y)
print("Correlation= ",corr)
```

OUTPUT :



```
= RESTART: C:\Users\DELL\OneDrive\Desktop\Notes\corr.py
How many  observations :5
Enter values of x separated by comma :4,6,3,4,8
Enter values of y separated by comma :8,6,2,1,9
x : [4, 6, 3, 4, 8]
y : [8, 6, 2, 1, 9]
Enter new value:20
Predicted value= 23.95
Covariance =  4.0
Standard deviation of x : 1.7888543819998317
Standard deviation of y : 3.1874754901018454
Correlation=  0.7015169165828922
```

PRACTICAL 5

Objective: Goodness of fit of a Distribution.

1. **Goodness of fit test of Binomial distribution fitting.**
2. **Goodness of fit test of Poisson distribution fitting.**
3. **Goodness of fit test of Normal distribution fitting.**

Q. A survey of 800 families with 4 children each revealed the following distribution.

No. of Boys	0	1	2	3	4
No. of Girls	**4**	**3**	**2**	**1**	**0**
No. of family	**32**	**178**	**290**	**236**	**64**

Is this result consistent with the hypothesis that male and female births are equally probable? (#Binomial Distribution)

CODE :

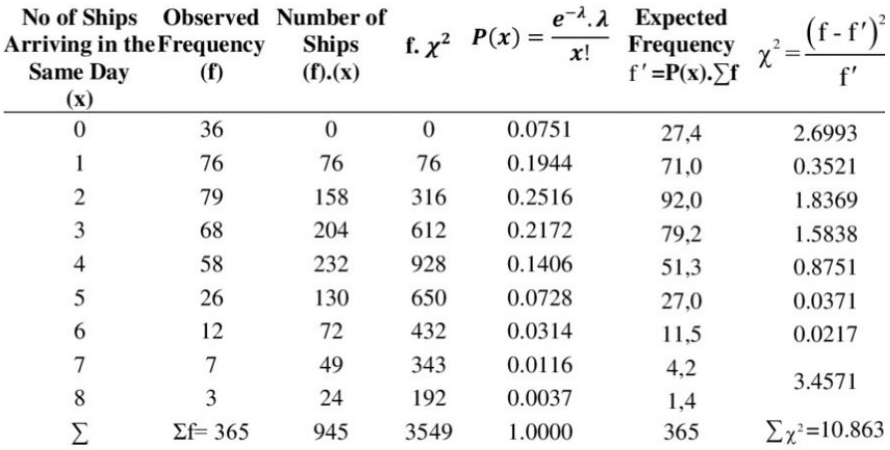
```
import math
#to find binomial distribution
def binom_pmf(x,n,p):
    c= math.factorial(n)/(math.factorial (n-x) *math.factorial(x))
    prob=c* (p**x) * ((1-p) ** (n-x))
    return prob

print("No. of boys  : [0, 1, 2, 3, 4]" )
print("No. of girls : [4, 3, 2, 1, 0]" )

#pmf of binomial
p=[]
x=[0,1,2,3,4]
for i in x:
    p.append(binom_pmf(i,4,0.5))
print ("Probability : ", p)      #output [0.0625, 0.25, 0.375, 0.25, 0.0625]

#expectedfrequency
Ex=[]
N=800
for i in p:
    Ex.append (N*i)
print ("No. of Family :", Ex)     #output [50.0, 200.0, 300.0, 200.0, 50.0]
```

```
#for expected chi square value
N=[32,178,290,236,64]
chi=0
for i in range(5):
    chi=chi + (((N[i]-Ex[i])**2)/(Ex[i]))
chi=round(chi,4)
print ("The Expected chi square value is : " ,chi,".")  #output19.6333

#for tabulated chi square value
import scipy.stats
Ec=scipy.stats.chi2.ppf(1-0.05,4)
Ec=round(Ec,4)

#for checking the test
print("Tabulated value of chi square : ",Ec,".")
if (chi <= Ec):
    print ("Accept the Hypothesis Test.")
else:
    print ("Rejected the Hypothesis Test.")
```

OUTPUT :

```
No. of boys  : [0, 1, 2, 3, 4]
No. of girls : [4, 3, 2, 1, 0]
Probability :  [0.0625, 0.25, 0.375, 0.25, 0.0625]
No. of Family : [50.0, 200.0, 300.0, 200.0, 50.0]
The Expected chi square value is :  19.6333 .
Tabulated value of chi square :  9.4877 .
Rejected the Hypothesis Test.
```

Q. Goodness of fit test of Poisson distribution fitting.



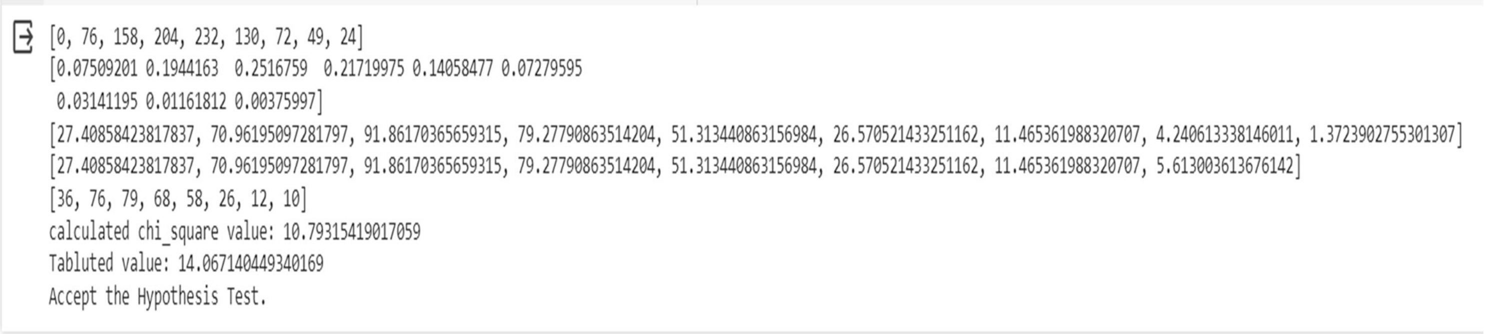
No of Ships Arriving in the Same Day (x)	Observed Frequency (f)	Number of Ships (f).(x)	f. χ^2	$P(x) = \frac{e^{-\lambda}.\lambda}{x!}$	Expected Frequency $f' = P(x).\sum f$	$\chi^2 = \frac{(f - f')^2}{f'}$
0	36	0	0	0.0751	27,4	2.6993
1	76	76	76	0.1944	71,0	0.3521
2	79	158	316	0.2516	92,0	1.8369
3	68	204	612	0.2172	79,2	1.5838
4	58	232	928	0.1406	51,3	0.8751
5	26	130	650	0.0728	27,0	0.0371
6	12	72	432	0.0314	11,5	0.0217
7	7	49	343	0.0116	4,2	3.4571
8	3	24	192	0.0037	1,4	
$\sum$	Σf= 365	945	3549	1.0000	365	$\sum \chi^2$=10.863

CODE :

```
# Goodness of fit test of poisson distribution fitting

import numpy as np
from scipy.stats import poisson,chisquare

x=[0,1,2,3,4,5,6,7,8]

f=[36,76,79,68,58,26,12,7,3]
fx=[]

for i in range(9):

  fx.append(x[i]*f[i])

print("f(x) : ",fx)

l=sum(fx)/sum(f) # l = 2.589041095890411

pmf=poisson.pmf(x,l)
print("The pmf of poisson distribution : ",pmf)

#Expected Frequency f' = P(x)*Sum(f) = pmf*365

npx=[]

for i in pmf:

  npx.append(365*i)
print("Expected Frequency f’ :",npx)

#combining last two rows , since they are less than 5
y=npx[7]+npx[8]
npx.pop(8)
npx.pop(7)
npx.append(y)
print(("New Expected Frequency f :"npx)
y=f[7]+f[8]
f.pop(8)
f.pop(7)
f.append(y)
print(("New f(x) :"f)


#checking for chi square

chi=0

for i in range(8):

  chi=chi+(((f[i]-npx[i])**2)/(npx[i]))
print("Calculated chi square value :",chi)

import scipy.stats
c=scipy.stats.chi2.ppf(1-0.05,7)
print("Tabluted chi square value:",c)
```

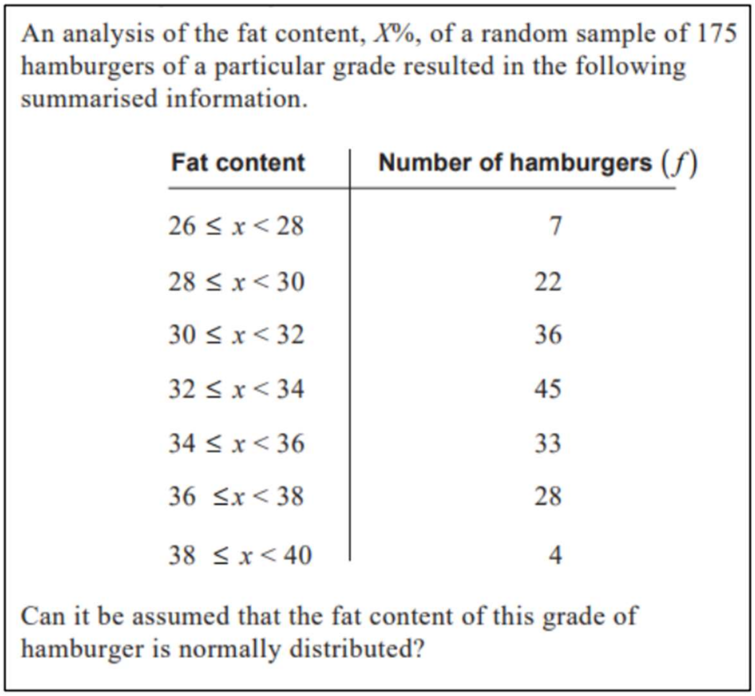
```
#checking the test
if (chi <= c):
    print ("Accept the Hypothesis Test.")
else:
    print ("Rejected the Hypothesis Test.")
```

OUTPUT :



```
[0, 76, 158, 204, 232, 130, 72, 49, 24]
[0.07509201 0.1944163  0.2516759  0.21719975 0.14058477 0.07279595
 0.03141195 0.01161812 0.00375997]
[27.40858423817837, 70.96195097281797, 91.86170365659315, 79.27790863514204, 51.313440863156984, 26.570521433251162, 11.465361988320707, 4.240613338146011, 1.3723902755301307]
[27.40858423817837, 70.96195097281797, 91.86170365659315, 79.27790863514204, 51.313440863156984, 26.570521433251162, 11.465361988320707, 5.613003613676142]
[36, 76, 79, 68, 58, 26, 12, 10]
calculated chi_square value: 10.79315419017059
Tabluted value: 14.067140449340169
Accept the Hypothesis Test.
```

Q. Goodness of fit test of normal distribution fitting.



An analysis of the fat content, $X\%$, of a random sample of 175 hamburgers of a particular grade resulted in the following summarised information.

Fat content	Number of hamburgers (f)
$26 \leq x < 28$	7
$28 \leq x < 30$	22
$30 \leq x < 32$	36
$32 \leq x < 34$	45
$34 \leq x < 36$	33
$36 \leq x < 38$	28
$38 \leq x < 40$	4

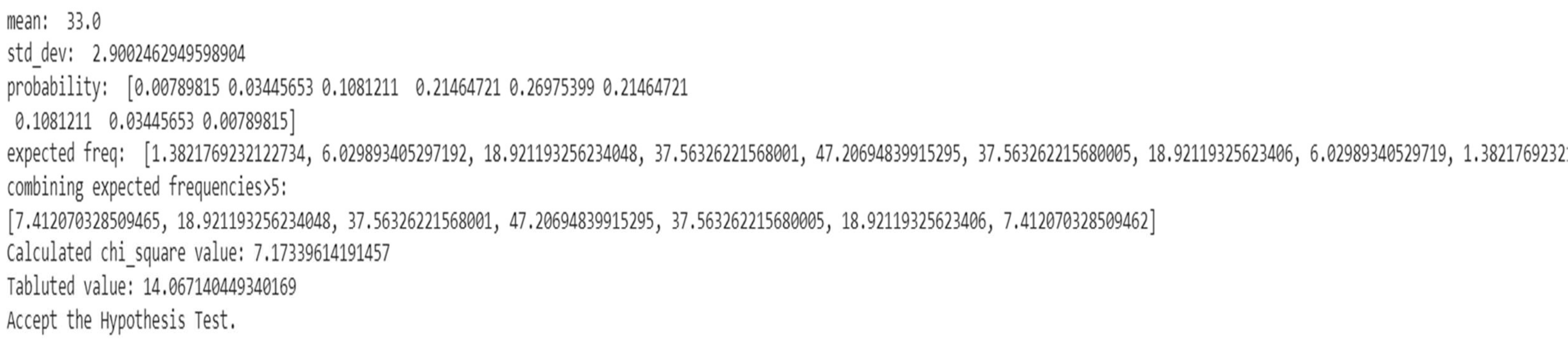
Can it be assumed that the fat content of this grade of hamburger is normally distributed?

CODE :

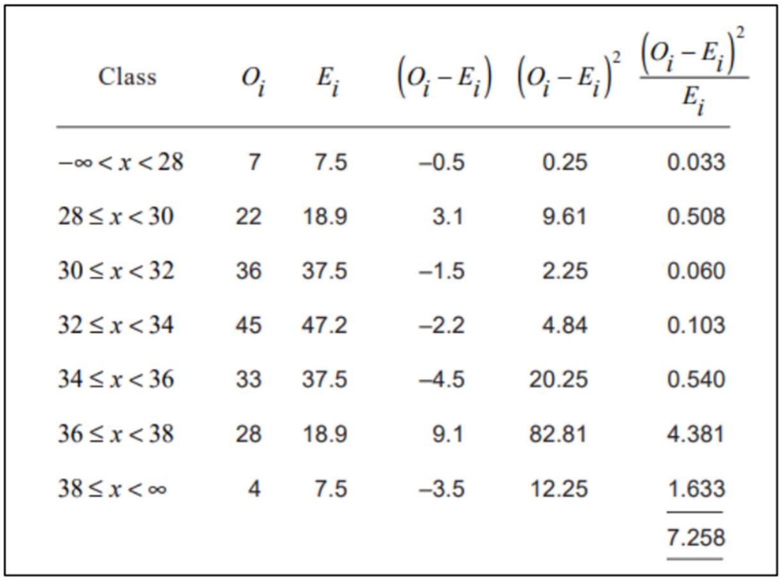
```
import numpy as np
from scipy.stats import chi2, norm

def weighted_mean(midpoints, freq):
    return np.average(midpoints, weights=freq)
```

```
midpoints=[27, 29, 31, 33, 35, 37, 39]
freq=[7, 22, 36, 45, 33, 28, 4]
mean = weighted_mean(midpoints, freq)
print("mean: ",mean)
std_dev = np.sqrt(np.average((midpoints - np.average(midpoints,
weights=freq))**2, weights=freq))
print("std_dev: ",std_dev)
lower_bound=[-np.inf, 26, 28, 30, 32, 34, 36, 38, 40]
upper_bound=[ 26, 28, 30, 32, 34, 36, 38, 40, np.inf]
probability = norm.cdf(upper_bound, loc=mean, scale=std_dev) -
norm.cdf(lower_bound, loc=mean, scale=std_dev)
print("probability: ",probability)
# Calculate the expected frequency
expected_freq=[]
for i in probability:
    expected_freq.append(175*i)
print("expected freq: ",expected_freq)

# Combining classes >5
x=expected_freq[0]+expected_freq[1]
y=expected_freq[7]+expected_freq[8]
expected_freq.pop(0)
expected_freq.pop(7)
expected_freq[0]=x
expected_freq[6]=y
print("combining expected frequencies>5: ")
print(expected_freq)

#performing chi
chi=0
for i in range(7):
  chi=chi+(((freq[i]-expected_freq[i])**2)/(expected_freq[i]))

import scipy.stats
c=scipy.stats.chi2.ppf(1-0.05,7)
print("Calculated chi_square value:",chi)
print("Tabluted value:",c)
if (chi <= c):
  print ("Accept the Hypothesis Test.")
else :
  print ("Reject the Hypothesis Test.")
```

OUTPUT :



```
mean:  33.0
std_dev:  2.9002462949598904
probability:  [0.00789815 0.03445653 0.1081211  0.21464721 0.26975399 0.21464721
 0.1081211  0.03445653 0.00789815]
expected freq:  [1.3821769232122734, 6.029893405297192, 18.921193256234048, 37.56326221568001, 47.20694839915295, 37.563262215680005, 18.92119325623406, 6.02989340529719, 1.3821769232
combining expected frequencies>5:
[7.412070328509465, 18.921193256234048, 37.56326221568001, 47.20694839915295, 37.563262215680005, 18.92119325623406, 7.412070328509462]
Calculated chi_square value: 7.17339614191457
Tabluted value: 14.067140449340169
Accept the Hypothesis Test.
```



Class	O_i	E_i	$(O_i - E_i)$	$(O_i - E_i)^2$	$\frac{(O_i - E_i)^2}{E_i}$
$-\infty < x < 28$	7	7.5	–0.5	0.25	0.033
$28 \le x < 30$	22	18.9	3.1	9.61	0.508
$30 \le x < 32$	36	37.5	–1.5	2.25	0.060
$32 \le x < 34$	45	47.2	–2.2	4.84	0.103
$34 \le x < 36$	33	37.5	–4.5	20.25	0.540
$36 \le x < 38$	28	18.9	9.1	82.81	4.381
$38 \le x < \infty$	4	7.5	–3.5	12.25	1.633
					7.258

PRACTICAL 6

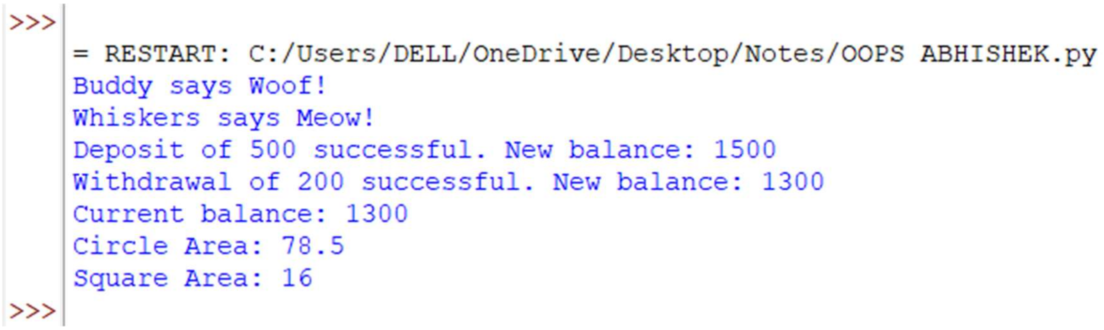
Objective: Write a program showing concepts of OOPs.

CODE :

```
# Class definition
class Animal:
    def __init__(self, name):
        self.name = name
    def speak(self):
        pass

# Inheritance: Dog is a subclass of Animal
class Dog(Animal):
    def speak(self):
        return f"{self.name} says Woof!"

# Inheritance: Cat is a subclass of Animal
class Cat(Animal):
    def speak(self):
        return f"{self.name} says Meow!"

# Polymorphism: Function that works with objects of different classes
def animal_sound(animal):
    return animal.speak()

# Encapsulation: Using private attributes and methods
class BankAccount:
    def __init__(self, balance=0):
        self.__balance = balance
    def deposit(self, amount):
        if amount > 0:
            self.__balance += amount
            print(f"Deposit of {amount} successful. New balance:
{self.__balance}")
        else:
            print("Invalid deposit amount.")

    def withdraw(self, amount):
        if 0 < amount <= self.__balance:
            self.__balance -= amount
            print(f"Withdrawal of {amount} successful. New balance:
{self.__balance}")
        else:
            print("Invalid withdrawal amount or insufficient funds.")

    def get_balance(self):
        return self.__balance
```

```python
# Data Abstraction: Abstracting away the implementation details
class Shape:
    def area(self):
        pass
# Data Abstraction: Implementation of the Shape class
class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius
    def area(self):
        return 3.14 * self.radius**2
class Square(Shape):
    def __init__(self, side):
        self.side = side
    def area(self):
        return self.side**2
# Creating objects and demonstrating OOP concepts
dog = Dog("Buddy")
cat = Cat("Whiskers")
print(animal_sound(dog))  # Polymorphism
print(animal_sound(cat))  # Polymorphism

account = BankAccount(1000)
account.deposit(500)  # Encapsulation
account.withdraw(200)  # Encapsulation
print(f"Current balance: {account.get_balance()}")  # Encapsulation

circle = Circle(5)
square = Square(4)
print(f"Circle Area: {circle.area()}")  # Data Abstraction
print(f"Square Area: {square.area()}")  # Data Abstraction
```

OUTPUT :



```
>>>
    = RESTART: C:/Users/DELL/OneDrive/Desktop/Notes/OOPS ABHISHEK.py
    Buddy says Woof!
    Whiskers says Meow!
    Deposit of 500 successful. New balance: 1500
    Withdrawal of 200 successful. New balance: 1300
    Current balance: 1300
    Circle Area: 78.5
    Square Area: 16
>>>
```

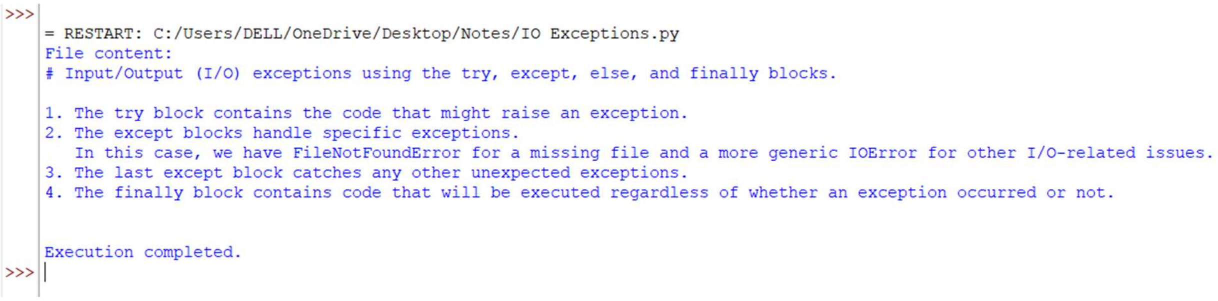
PRACTICAL 7

Objective: Write a program of I/O Exception.

CODE :

```
try:
    # Attempt to open a file for reading
    file_path = "python.txt"
    with open(file_path, 'r') as file:
        content = file.read()
        print(f"File content: {content}")

except FileNotFoundError:
    # Handle the specific exception for a missing file
    print(f"Error: The file '{file_path}' does not exist.")

except IOError as e:
    # Handle more generic I/O exceptions
    print(f"An I/O error occurred: {e}")

except Exception as e:
    # Handle any other unexpected exceptions
    print(f"An unexpected error occurred: {e}")

finally: # Code in this block will be executed no matter what, whether
an exception occurred or not
    print("Execution completed.")
```

OUTPUT:



```
>>>
= RESTART: C:/Users/DELL/OneDrive/Desktop/Notes/IO Exceptions.py
File content:
# Input/Output (I/O) exceptions using the try, except, else, and finally blocks.

1. The try block contains the code that might raise an exception.
2. The except blocks handle specific exceptions.
   In this case, we have FileNotFoundError for a missing file and a more generic IOError for other I/O-related issues.
3. The last except block catches any other unexpected exceptions.
4. The finally block contains code that will be executed regardless of whether an exception occurred or not.


Execution completed.
>>> |
```

PRACTICAL 8

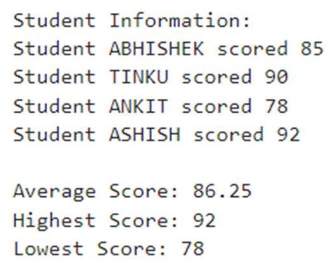
Objective: Write a program using LIBRARY.

➢ NUMPY :

CODE :

```
import numpy as np
# Function to calculate the average score
def calculate_average(scores):
    return np.mean(scores)

# Function to find the highest score
def find_highest_score(scores):
    return np.max(scores)

# Function to find the lowest score
def find_lowest_score(scores):
    return np.min(scores)

# Function to display student information
def display_student_info(names, scores):
    for i in range(len(names)):
        print(f"Student {names[i]} scored {scores[i]}")
# College scenario using NumPy
student_names = ['ABHISHEK', 'TINKU', 'ANKIT', 'ASHISH']
student_scores = np.array([85, 90, 78, 92])

# Display student information
print("Student Information:")
display_student_info(student_names, student_scores)
# Calculate and display average score
average_score = calculate_average(student_scores)
print(f"\nAverage Score: {average_score:.2f}")

# Find and display the highest and lowest scores
highest_score = find_highest_score(student_scores)
lowest_score = find_lowest_score(student_scores)
print(f"Highest Score: {highest_score}")
print(f"Lowest Score: {lowest_score}")
```

OUTPUT:



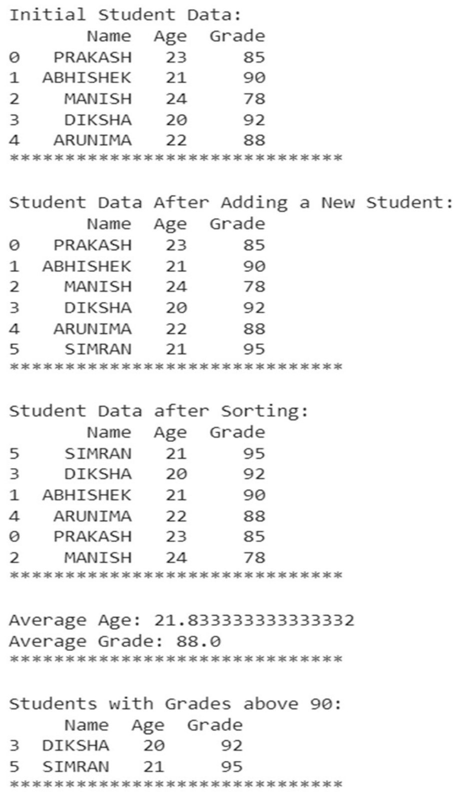
```
Student Information:
Student ABHISHEK scored 85
Student TINKU scored 90
Student ANKIT scored 78
Student ASHISH scored 92

Average Score: 86.25
Highest Score: 92
Lowest Score: 78
```

➢ PANDA :

CODE :

```
import pandas as pd
# Create a dictionary with student data
data = {
    'Name': ['PRAKASH', 'ABHISHEK', 'MANISH', 'DIKSHA', 'ARUNIMA'],
    'Age': [23, 21, 24, 20, 22],
    'Grade': [85, 90, 78, 92, 88]
}

# Create a DataFrame from the dictionary
df = pd.DataFrame(data)

# Display the initial student data
print("Initial Student Data:")
print(df)
print(30*"*")

# Add a new student to the DataFrame
new_student = {'Name': 'SIMRAN', 'Age': 21, 'Grade': 95}
df = df.append(new_student, ignore_index=True)

# Display the updated student data
print("\nStudent Data After Adding a New Student:")
print(df)
print(30*"*")

# Sorting the DataFrame based on 'Marks' in descending order
sorted_data = df.sort_values(by='Grade', ascending=False)
# Display the sorted student data
print("\nStudent Data after Sorting:")
print(sorted_data)
print(30*"*")

# Calculate average age and grade
average_age = df['Age'].mean()
average_grade = df['Grade'].mean()
print("\nAverage Age:", average_age)
print("Average Grade:", average_grade)
print(30*"*")


# Filter students with a grade above a certain threshold
threshold = 90
high_grades_df = df[df['Grade'] > threshold]

print(f"\nStudents with Grades above {threshold}:")
print(high_grades_df)
print(30*"*")
```

OUTPUT:



```
Initial Student Data:
       Name  Age  Grade
0   PRAKASH   23     85
1  ABHISHEK   21     90
2    MANISH   24     78
3    DIKSHA   20     92
4   ARUNIMA   22     88
******************************

Student Data After Adding a New Student:
       Name  Age  Grade
0   PRAKASH   23     85
1  ABHISHEK   21     90
2    MANISH   24     78
3    DIKSHA   20     92
4   ARUNIMA   22     88
5    SIMRAN   21     95
******************************

Student Data after Sorting:
       Name  Age  Grade
5    SIMRAN   21     95
3    DIKSHA   20     92
1  ABHISHEK   21     90
4   ARUNIMA   22     88
0   PRAKASH   23     85
2    MANISH   24     78
******************************

Average Age: 21.833333333333332
Average Grade: 88.0
******************************

Students with Grades above 90:
     Name  Age  Grade
3  DIKSHA   20     92
5  SIMRAN   21     95
******************************
```

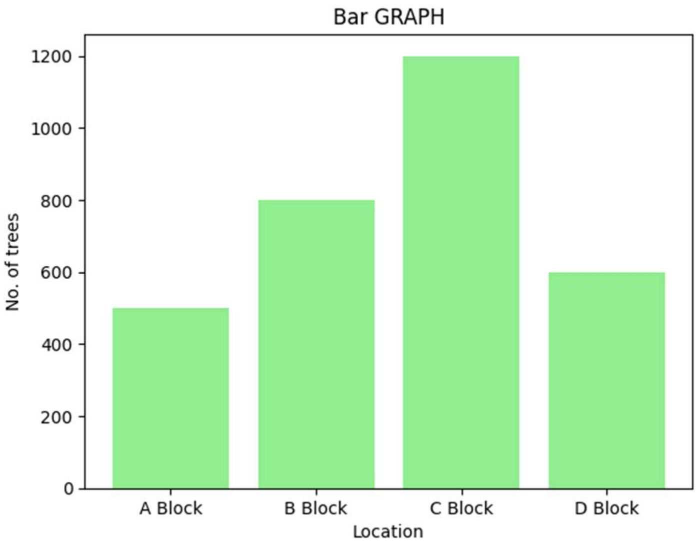
➢ MATPLOTLIB :

1.BAR GRAPH

CODE :

```
import matplotlib.pyplot as plt
# Number of Trees data
Location = ['A Block', 'B Block', 'C Block', 'D Block']
Number_of_Tress = [500, 800, 1200, 600]
# Plotting the bar chart
plt.bar(Location, Number_of_Tress, color='lightgreen')
plt.xlabel('Location')
plt.ylabel('No. of trees')
plt.title('Bar GRAPH')
# Display the bar chart
plt.show()
```

OUTPUT:

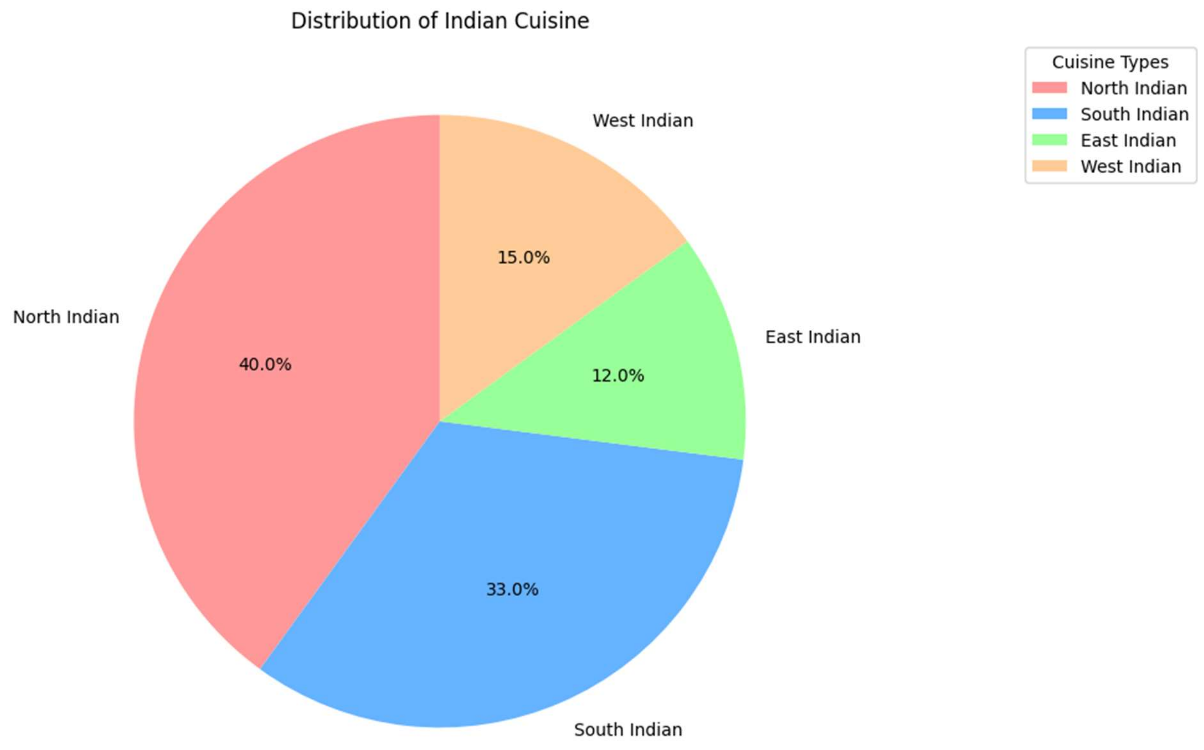


2. PIE CHART

CODE :

```
import matplotlib.pyplot as plt

# Data for the pie chart
cuisine_labels = ['North Indian', 'South Indian', 'East Indian', 'West Indian']
cuisine_percentages = [40, 33, 12, 15]

# Colors for each cuisine
colors = ['#ff9999', '#66b3ff', '#99ff99', '#ffcc99']

# Plotting the pie chart
plt.figure(figsize=(8, 8))
plt.pie(cuisine_percentages, labels=cuisine_labels, autopct='%1.1f%%',
startangle=90, colors=colors)

# Title and legend
plt.title('Distribution of Indian Cuisine')
plt.legend(title='Cuisine Types', loc='upper right',
bbox_to_anchor=(1, 0, 0.5, 1))

# Display the pie chart
plt.show()
```

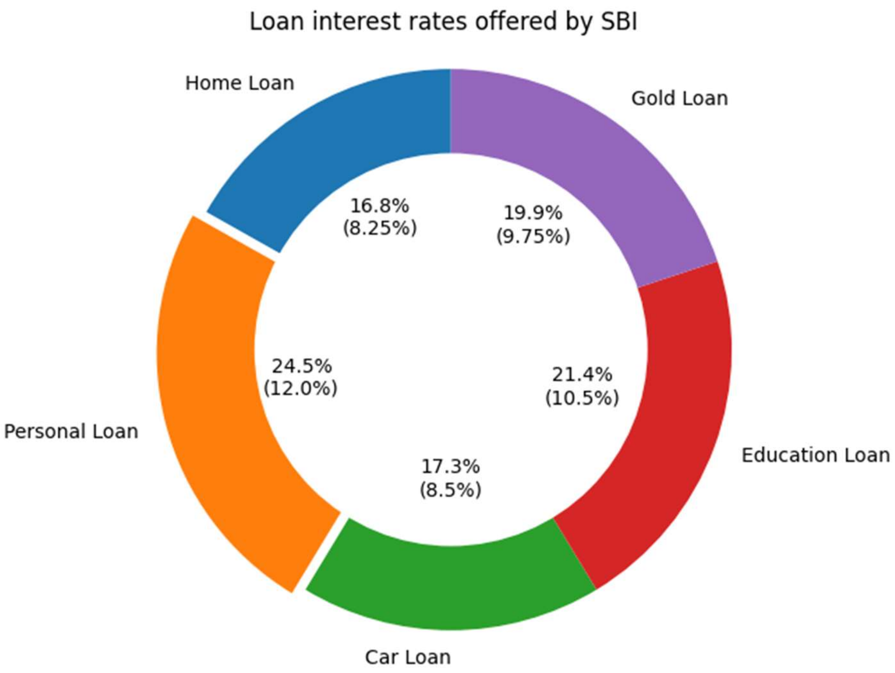
OUTPUT:



CODE :

```
import matplotlib.pyplot as plt

# Loan interest rates offered by SBI
loan_types = ['Home Loan', 'Personal Loan', 'Car Loan', 'Education
Loan', 'Gold Loan']
interest_rates = [8.25, 12.0, 8.5, 10.5, 9.75]  # Example interest
rates (replace with actual rates)

explode = (0,0.05,0,0,0) #Highlight high loan interest

fig1, ax1 = plt.subplots()
wedges, texts, autotexts = ax1.pie(interest_rates, labels=loan_types,
autopct='%1.1f%%',startangle=90, pctdistance=0.5, explode=explode)

# Adding exact interest rate values
for i, (text, autotext) in enumerate(zip(texts, autotexts)):
    percentage = autotext.get_text()
    exact_value = interest_rates[i]
    autotext.set_text(f'{percentage}\n({exact_value}%)\n')

#Draw circle
centre_circle = plt.Circle((0,0),0.70,fc='white')
fig = plt.gcf()
fig.gca().add_artist(centre_circle)
# Equal aspect ratio ensures that pie is drawn as a circle
ax1.axis('equal')
plt.tight_layout()
plt.title('Loan interest rates offered by SBI')
plt.show()
```

OUTPUT:

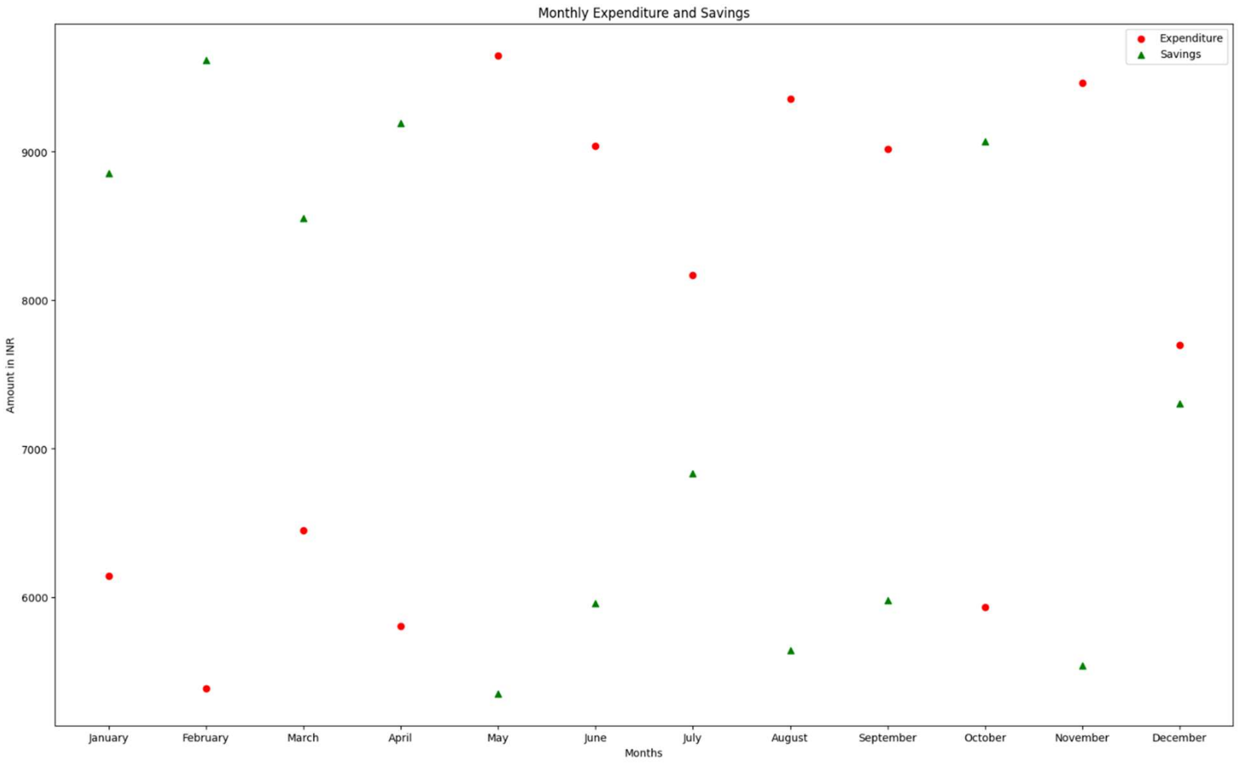


3.Scatter Plot

CODE :

```
import matplotlib.pyplot as plt
import numpy as np
# Generate example data
months = ['January', 'February', 'March', 'April', 'May', 'June',
'July', 'August', 'September', 'October', 'November', 'December']
monthly_income = 15000
expenditure = np.random.uniform(5000, 10000, 12)  # Random expenditure
between 15,000 and 25,000 INR
savings = monthly_income - expenditure
# Plotting the scatter plot
plt.figure(figsize=(20, 12))
plt.scatter(months, expenditure, label='Expenditure', color='red',
marker='o')
plt.scatter(months, savings, label='Savings', color='green',
marker='^')

# Adding labels and title
plt.title('Monthly Expenditure and Savings')
plt.xlabel('Months')
plt.ylabel('Amount in INR')

# Adding legend
plt.legend()

# Display the scatter plot
plt.show()
```

OUTPUT :


Monthly Expenditure and Savings
Expenditure
Savings
Amount in INR
January
February
March
April
May
June
July
August
September
October
November
December
Months