Ex.No.: 1.|(a)	IMPLEMENT SEARCH STRATEGIES OF 8-PUZZLE
Date:	

AIM:

Toimplement8-puzzle problem using searching strategies.

ALGORITHM:

Step1:Start the program.

Step2: Create the class name as priorityQueue.

Step3: Define the function as push and operation for ordering the list in queue.

Step4: Calculate the cost of the nodes.

Step5: Call the method for solve the puzzle.

Step6: Stop the program.

PROGRAM:

```
import copy
from heapq import heappush, heappop
n = 3
rows = [ 1, 0, -1, 0 ]
cols = [ 0, -1, 0, 1 ]
class priorityQueue:
    def __init__(self):
        self.heap = []
    def push(self, key):
        heappush(self.heap, key)
    def pop(self):
        return heappop(self.heap)
    def empty(self):
        if not self.heap:
            return True
        else:
            return False
class nodes:
    def __init__(self, parent, mats, empty_tile_posi,
                costs, levels):
        self.parent = parent
        self.mats = mats
        self.empty_tile_posi = empty_tile_posi
        self.costs = costs
        self.levels = levels
    def __lt__(self, nxt):
        return self.costs < nxt.costs
def calculateCosts(mats, final) -> int:
    count = 0
    for i in range(n):
        for j in range(n):
            if ((mats[i][j]) and
                (mats[i][j] != final[i][j])):
                count += 1
    return count


def newNodes(mats, empty_tile_posi, new_empty_tile_posi,
            levels, parent, final) -> nodes:
    new_mats = copy.deepcopy(mats)
    x1 = empty_tile_posi[0]
    y1 = empty_tile_posi[1]
    x2 = new_empty_tile_posi[0]
    y2 = new_empty_tile_posi[1]
    new_mats[x1][y1], new_mats[x2][y2] = new_mats[x2][y2], new_mats[x1][y1]
    costs = calculateCosts(new_mats, final)
    new_nodes = nodes(parent, new_mats, new_empty_tile_posi,
                    costs, levels)
    return new_nodes


def printMatsrix(mats):

    for i in range(n):
        for j in range(n):
            print("%d " % (mats[i][j]), end = " ")
        print()
def isSafe(x, y):
    return x >= 0 and x < n and y >= 0 and y < n
def printPath(root):
    if root == None:
        return
    printPath(root.parent)
    printMatsrix(root.mats)
    print()
def solve(initial, empty_tile_posi, final):
    pq = priorityQueue()
    costs = calculateCosts(initial, final)
    root = nodes(None, initial,
   empty_tile_posi, costs, 0)
    pq.push(root)
   while not pq.empty():
        minimum = pq.pop()

        # If the min. is ans node
       if minimum.costs == 0:
            printPath(minimum)
            return
        for i in range(n):
            new_tile_posi = [
                minimum.empty_tile_posi[0] + rows[i],
                minimum.empty_tile_posi[1] + cols[i], ]

            if isSafe(new_tile_posi[0], new_tile_posi[1]):
child = newNodes(minimum.mats,
                                minimum.empty_tile_posi,
                                new_tile_posi,
                                minimum.levels + 1,  minimum
   pq.push(child)

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

# Final configuration that can be solved
# Value 0 is taken as an empty space
final = [ [ 1, 2, 3 ],
        [ 5, 8, 6 ],
        [ 0, 7, 4 ] ]

# Blank tile coordinates in the
# initial configuration
empty_tile_posi = [ 1, 2 ]

# Method call for solving the puzzle
solve(initial, empty_tile_posi, final)
```

RESULT:

Thustheprogramtoimplementthe 8 puzzle programwasexecutedand the output was verified Successfull

Ex.No.: 1(b)	**IMPLEMENT SEARCH STRATEGIES OF 4- QUEENS PROBLEM**
Date:	

AIM:

To implementation of 8 queens problem using search Strategies.

ALGORITHM:

Step1:Start the program

Step2:Define the function for the board using defprintSolution(board):

Step3:Implement the iteration inside the function.

Step4:Check the condition if a queen can be placed on board as a row and column wise.

Step5:Using diagonal value the queens are placed in a board without intersecting each other.

Step6: Check whether the queen are placed in a board or not.

Step 7:Return the value.

Step8:Stop the program.

PROGRAM:

```
globalN
N =4
 defprintSolution(board):
    fori inrange(N):
        forj inrange(N):
            print(board[i][j],end=' ')
        print()
 defisSafe(board, row, col):
fori inrange(col):
        ifboard[row][i] ==1:
            returnFalse
fori, j inzip(range(row, -1, -1), range(col, -1, -1)):
        ifboard[i][j] ==1:
            returnFalse
fori, j inzip(range(row, N, 1), range(col, -1, -1)):
        ifboard[i][j] ==1:
            returnFalse
 returnTrue

defsolveNQUtil(board, col):
    ifcol >=N:
        returnTrue
fori inrange(N):

        ifisSafe(board, i, col):

            board[i][col] =1

            ifsolveNQUtil(board, col +1) ==True:
                returnTrue
board[i][col] =0
returnFalse
defsolveNQ():
    board =[ [0, 0, 0, 0],
              [0, 0, 0, 0],
              [0, 0, 0, 0],
              [0, 0, 0, 0]
             ]

    ifsolveNQUtil(board, 0) ==False:
        print("Solution does not exist")
        returnFalse

    printSolution(board)
    returnTrue
solveNQ()
```

Output:

```
0 0 Q 0
Q 0 0 0
0 0 0 Q
0 Q 0 0
```

RESULT:
Thus the above 4 Queens problem using searching strategies program is executed and the output was verified successfully.

Ex.No.:1(c)	IMPLEMENT SEARCH STRATEGIES OF CRYPTARITHMETIC.
Date:	

AIM:

Toimplementcryptarithmetic problem using searching strategies.

ALGORITHM:

```
Step1:Start the program.
Step2: Create the function as issolvable().
Step3: Define the function and  solve the hash function.
Step4: Encrypte the data using hash function .
Step5: Check the condition.
    if issolvable():
        print("yes")
    else:
        print("No")
Step6: Stop the program.
```

PROGRAM:

```
def isSolvable(words, result):
    mp = [-1]*(26)
    Hash = [0]*(26)
    CharAtfront = [0]*(26)
    uniq = ""
    for word in range(len(words)):
      for i in range(len(words[word])):
         ch = words[word][i]
         Hash[ord(ch) - ord('A')] += pow(10, len(words[word]) - i - 1)
         if mp[ord(ch) - ord('A')] == -1:
            mp[ord(ch) - ord('A')] = 0
            uniq += str(ch)
```

```
                if i == 0 and len(words[word]) > 1:
                    CharAtfront[ord(ch) - ord('A')] = 1
        for i in range(len(result)):
            ch = result[i]
            Hash[ord(ch) - ord('A')] -= pow(10, len(result) - i - 1)
            if mp[ord(ch) - ord('A')] == -1:
                mp[ord(ch) - ord('A')] = 0
                uniq += str(ch)
            if i == 0 and len(result) > 1:
                CharAtfront[ord(ch) - ord('A')] = 1


        mp = [-1]*(26)
        return True
    def solve(words, i, S, mp, used, Hash, CharAtfront):


        if i == len(words):
            return S == 0
        ch = words[i]
        val = mp[ord(words[i]) - ord('A')]
        if val != -1:
           return solve(words, i + 1, S + val * Hash[ord(ch) - ord('A')], mp, used,
Hash, CharAtfront)
        x = False
        for l in range(10):


            if CharAtfront[ord(ch) - ord('A')] == 1 and l == 0:
                continue
            if used[l] == 1:
                continue


              mp[ord(ch) - ord('A')] = l
        used[l] = 1
        x |= solve(words, i + 1, S + l * Hash[ord(ch) - ord('A')], mp, used, Hash, CharAtfront)
        mp[ord(ch) - ord('A')] = -1
        used[l] = 0
        return x

arr = [ "SIX", "SEVEN", "SEVEN" ]
S = "TWENTY"
if isSolvable(arr, S):
    print("YES")
else:
    print("NO")
```

OUTPUT:

```
YES
```

RESULT:

Thustheprogramtoimplement the cryptarithmetic problem using search strategieswasexecutedand the output was verifiedSuccessfully.

Ex.No.:2	IMPLEMENT A* AND MEMORY BOUNDED A* ALGORITHMS
Date:	

AIM:

Toimplementan A*algorithm usingpython.

ALGORITHM:

1. Addstart nodeto list
2. Foralltheneighbouringnodes,findtheleast costFnode
3. Switchto theclosed list
   - For8nodesadjacenttothecurrentnode
   - If thenodeisnot reachable,ignoreit.Else
     - Ifthenodeisnotontheopenlist,moveittotheopenlistandcalculate f,g,h.
     - Ifthenodeisontheopenlist,checkifthepathitoffersislessthanthecurrentpath and changeto it if it does so.
4. Stopworkingwhen
   - Youfindthedestination
   - Youcannotfind thedestination goingthroughallpossiblepoints

PROGRAM:

```
fromcollectionsi

mportdequeclas

sGraph:

   #exampleofadjacencylist(orr

   athermap)#adjacency_list =

   {

   #'A':[('B',1),('C',

   3),('D',7)],#'B':

   [('D', 5)],

   #'

   C

   ':
```

```
 [(
 '
 D
 ',
 1
 2
 )]
 #
 }
 definit(self,adjacency_li
     st):self.adjacency_list
     =adjacency_list
 def
      get_neighbors
      (self,
      v):returnself.a
      djacency_list[
      v]

#heuristicfunctionwithequalvaluesfo
 rallnodesdefh(self, n):
     H={'A':1,'B':1, 'C':1,'D':1}
     returnH[n]
 defa_star_algorithm(self,start_node,stop_node):
     # open_list is a list of nodes which have been visited, but
     who's neighbors#haven't all been inspected,starts off with
     the startnode
     # closed_list is a list of nodes which
     have been visited#and who's
```

```
neighborshavebeen inspected
open_list =
set([start_node
])closed_list=
set([])
#gcontainscurrentdistancesfromstart_nodetoalloth
er nodes#the default value (if it's notfound in
themap) is +infinity
g={}
g[start_node]=0
#parentscontainsanadjacencymapo
fallnodesparents= {}
parents[start_node
]=start_nodewhile
len(open_list)>0:
   n =None
   # find a node with the lowest value of f() -
   evaluation functionforv in open_list:
      ifn==Noneorg[v] +self.h(v)
         <g[n]+self.h(n):n =v;
   ifn ==None:
      print('Pathdo
      esnotexist!')r
      eturnNone
   #if thecurrent nodeis thestop_node
```

```
# then we begin reconstructin the path from it to
the start_nodeifn ==stop_node:
    reconst
    _path=[
    ]whilep
    arents[n
    ]!=n:
        reconst_p
        ath.appen
        d(n)n
        =parents[
        n]
    reconst_path.append(st
    art_node)reconst_path.
    reverse()
    print('Pathfound:{}'.format(re
    const_path))returnreconst_pat
    h
#forallneighborsofthecurre
ntnodedofor(m,weight)inse
lf.get_neighbors(n):
    #ifthecurrentnodeisn'tinbothopen_listandclo
    sed_list#add it to open_list andnoten as it's
    parent
    if m not in open_list and m not
        in
        closed_list:open_list.add(m)
        parents[m] =n
        g[m]=g[n]+weight
      # otherwise, check if it's quicker to first
      visit n, then m#and if it is, update parent
      dataandgdata
      # and if the node was in the closed_list, move
      it to open_listelse:
        ifg[m]>g[
          n]+wei
          ght:g[
          m]     =
          g[n]   +
          weight
          parents
          [m] =n
          if m in
            closed_l
            ist:close
            d_list.re
            move(m
            )open_li
            st.add(m
            )

    #removenfromtheopen_list,andaddittoclo
    sed_list# because all of his neighbors
    were inspectedopen_list.remove(n)
    closed_list
  .add(n)print('
  Pathdoesnot
```

```
        exist!')return
        None
adjacency_list={'A':[('B',1),('C',3),('D',7)],'B':[('D',5)],'C':[('D',12)]}
graph1 =
Graph(adjacency_l
ist)graph1.a_star_a
lgorithm('A','D')
```

OUTPUT:

```
Pathfound:['A','B', 'D']
```

RESULT:

ThustheprogramtoimplementA*algorithmusingpythonwasexecutedandverified successfully.

Ex.No.:3	IMPLEMENT MINIMAX ALGORITHM FOR GAME PLAYING (ALPHA-BETA PRUNING)
Date:	

AIM:

Toimplement minmax algorithm for game playing.

ALGORITHM:

Step1: Start the program**.**

Step2: Define the function for minmax problem and pass the argument.

Step3: Check the player score .

Step4: Check the level of the player and find the maximum and minimum value of the player

Step5: Call the function . And print the result.

Step6: Stop the program.

PROGRAM:

```
import math


def minimax (curDepth, nodeIndex,

maxTurn, scores,

targetDepth):


  # base case : targetDepth reached

  if (curDepth == targetDepth):

      return scores[nodeIndex]


  if (maxTurn):

      return max(minimax(curDepth + 1, nodeIndex * 2,

      False, scores, targetDepth),

      minimax(curDepth + 1, nodeIndex * 2 + 1,

      False, scores, targetDepth))
  else:
      return min(minimax(curDepth + 1, nodeIndex * 2,
      True, scores, targetDepth),
      minimax(curDepth + 1, nodeIndex * 2 + 1,
      True, scores, targetDepth))

  # Driver code
scores = [3, 5, 2, 9, 12, 5, 23, 23]
treeDepth = math.log(len(scores), 2)
print("The optimal value is : ", end = "")
print(minimax(0, 0, True, scores, treeDepth))
```

OUTPUT:

```
The optimal value is : 12
```

RESULT:

Thustheprogramtoimplement the min-max algorithm wasexecutedand the output was verified Successfully

Ex.no:4	**IMPLEMENT CONSTRAINT SATISFACTION PROBLEMSFOR MAP COLORING**
Date:	

AIM:

Toimplement map coloring technique using constraint satisfaction problem.

ALGORITHM:

Step1:Start the program.

Step2: Define the function for coloring.

Step3: Identify the colors for suitable states.

Step4: Fill the color to the state under specific condition.

Step5: Stop the program.

PROGRAM:

```
colors = ['Red', 'Blue', 'Green', 'Yellow', 'Black']

states = ['Andhra', 'Karnataka', 'TamilNadu', 'Kerala']

neighbors = {}
neighbors['Andhra'] = ['Karnataka', 'TamilNadu']
neighbors['Karnataka'] = ['Andhra', 'TamilNadu', 'Kerala']
neighbors['TamilNadu'] = ['Andhra', 'Karnataka', 'Kerala']
neighbors['Kerala'] = ['Karnataka', 'TamilNadu']

colors_of_states = {}

def promising(state, color):
    for neighbor in neighbors.get(state):
        color_of_neighbor = colors_of_states.get(neighbor)
```

```
                if color_of_neighbor == color:
                    return False

            return True

        def get_color_for_state(state):
            for color in colors:
                if promising(state, color):
                    return color

        def main():
            for state in states:
                colors_of_states[state] = get_color_for_state(state)
print (colors_of_states)
        main()
```

OUTPUT:

```
{'Andhra': 'Red', 'Karnataka': 'Blue', 'TamilNadu': 'Green', 'Kerala': 'Red'}
```

RESULT:

Thustheprogramtoimplementmap coloring technique using constraint satisfaction problem wasexecutedand the output was verified Successfully

Ex.No.: 5	IMPLEMENT PROPOSITIONAL MODEL CHECKING ALGORITHMS
Date:	

AIM:

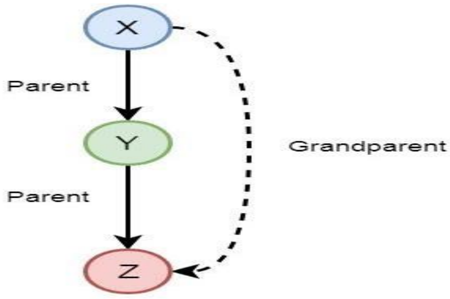
To develop a small KB using prolog to answer simple queries.

ALGORITHM:

InPrologsyntax,wecanwrite–

mother(X,Y):-parent(X,Y),female(X).
sister(X,Y):-parent(Z,X),parent(Z,Y),female(X),X\==Y.

Soifwewanttomakeagrandparentrelationship,thatcanbeformedasfollows–



Grandparent Relationship

Wecanalsocreatesomeotherrelationshipslikewife,uncle,etc.Wecanwritetherelationshipsas given below –

- grandparent(X,Y):-parent(X,Z),parent(Z,Y).
- grandmother(X,Z):-mother(X,Y),parent(Y,Z).
- grandfather(X,Z):-father(X,Y),parent(Y,Z).
- wife(X,Y):-parent(X,Z),parent(Y,Z),female(X),male(Y).
- uncle(X,Z):-brother(X,Y),parent(Y,Z).

Soletuswriteaprologprogramtoseethisinaction.Herewewillalsoseethetracetotrace-outtheexecution

PROGRAM:

```
female(pam). female(liz).

female(pat).
 female(ann).
male(jim).
```

```
 male(bob).
male(tom).
male(peter).
parent(pam,bob).
 parent(tom,bob).
parent(tom,liz).
parent(bob,ann).

parent(bob,pat).

parent(pat,jim).

parent(bob,peter).

parent(peter,jim).

mother(X,Y):- parent(X,Y),female(X).

father(X,Y):-parent(X,Y),male(X).

sister(X,Y):-parent(Z,X),parent(Z,Y),female(X),X\==Y.

brother(X,Y):-parent(Z,X),parent(Z,Y),male(X),X\==Y.

grandparent(X,Y):-parent(X,Z),parent(Z,Y).

grandmother(X,Z):-mother(X,Y),parent(Y,Z).

grandfather(X,Z):-father(X,Y),parent(Y,Z).

wife(X,Y):-parent(X,Z),parent(Y,Z),female(X),male(Y). uncle(X,Z):-
brother(X,Y),parent(Y,Z).
```

OUTPUT:

```
|?-[family_ext].

compilingD:/TPProlog/Sample_Codes/family_ext.plforbytecode...

D:/TPProlog/Sample_Codes/family_ext.plcompiled,27linesread-
4646byteswritten,10ms

|?-uncle(X,Y).



X


=


p

e

t

e

r

Y

=

j

i

m

?


;


no

|?-grandparent(X,Y).


X

=
p
a
m
Y
=
a
n
n
?
;

X
=
p
a
m
Y
=
p
a
t
?
;

X=pam
Y=peter?;

X=tom
Y=ann ?;

X
=
```

t

o

m

Y

=

p

a

t

?

;

X=tom

Y=peter?;

X=pam

Y=peter?;

X=tom

Y=ann ?;

X

=

t

o

m

Y

=

p

a

```
t
?

;

X=tom
Y=peter?;
X=bob
Y =jim?;

X=bob
Y =jim?;

no
|?-wife(X,Y).

X=pam
Y =tom ?;

X=pat
Y=peter?;

(15ms)no
|?
```

RESULT:

Thus the program to develop a simple query using prolog was executed and verified successfully.

Ex.No.:6(a)	**IMPLEMENT FORWARD CHAINING STRATEGIES**
Date:	

AIM:

ToimplementBFSalgorithmusingpython.

ALGORITHM:

1. Pickanynode,visit theadjacentunvisited vertex,markit asvisited,displayit,andinsertitina queue.
2. If there arenoremaining adjacentverticesleft,removethefirstvertex from thequeue.
3. Repeatstep 1 and step 2until thequeueis emptyor thedesired nodeis found.

PROGRAM:

```
graph={
  'A':['B','C'],
  'B':['D', 'E'],
  'C':['F'],
  'D': [],
          ]
          }

visited=[] #Listtokeep
trackofvisitednodes.queue=[]
                    #Initializeaqueue
defbfs(visited,
  graph,node):v
  isited.append
  (node)queue.
  append(node)
  whilequeue:
    s = queue.pop(0)print(s,end="")


    for neighbor in graph[s]:
      if neighbor not in visited:
        visited.appe
        nd(neighbou
        r)queue.app
        end(neighbo
        ur)
# Driver
Codebfs
(visited,
graph,'A
')
```

OUTPUT:

```
ABC D EF
```

RESULT:

Thus the program to implement BFS algorithm using python was executed and verified successfully

Ex.No.:6(b)	**IMPLEMENT BACKWARD CHAINING STRATEGIES**
Date:	

AIM:

To implement DFS algorithm using python.

ALGORITHM:

1. Pick any node.If it is unvisited,mark it as visited and recuron all its adjacent nodes.
2. Repeat until all the nodes are visited,or the node to be searched is found.

PROGRAM:

```
#Using a Python dictionary to act as an
adjacency
listgraph={'A':['B','C'],'B':['D',
'E'],'C':['F'],'D': [],'E':['F'],'F': []}
visited=set()#Settokeeptrackofvisitedno
des.defdfs(visited,graph, node):
  if node not in visited:print (node)visited.add(node)
    for neighbour in
      graph[node]:dfs
      (visited,graph,n
      eighbour)
#Driver
Codedfs
(visited,
graph,'A
')
```

OUTPUT:

```
A

B

D

E

F

C
```

RESULT:

Thus the program to implement DFS algorithm using python was executed and verified successfully.

Ex.No.:7	IMPLEMENTATION OF NAÏVE BAYESIAN CLASSIFIER FOR CREDIT CARD ANALYSIS
Date:	

AIM:

ToimplementNaïveBayesianclassifierforcreditcardanalysisandcomputetheaccuracywithfew datasets.

ALGORITHM:

ThisNaiveBayesisbrokendownint

o5parts:1:SeparateByClass.

2:SummarizeDataset.

3:SummarizeData ByClass.

4:GaussianProbabilityDensi

tyFunction.5:Class

Probabilities.

PROGRAM:

Importingpackages:

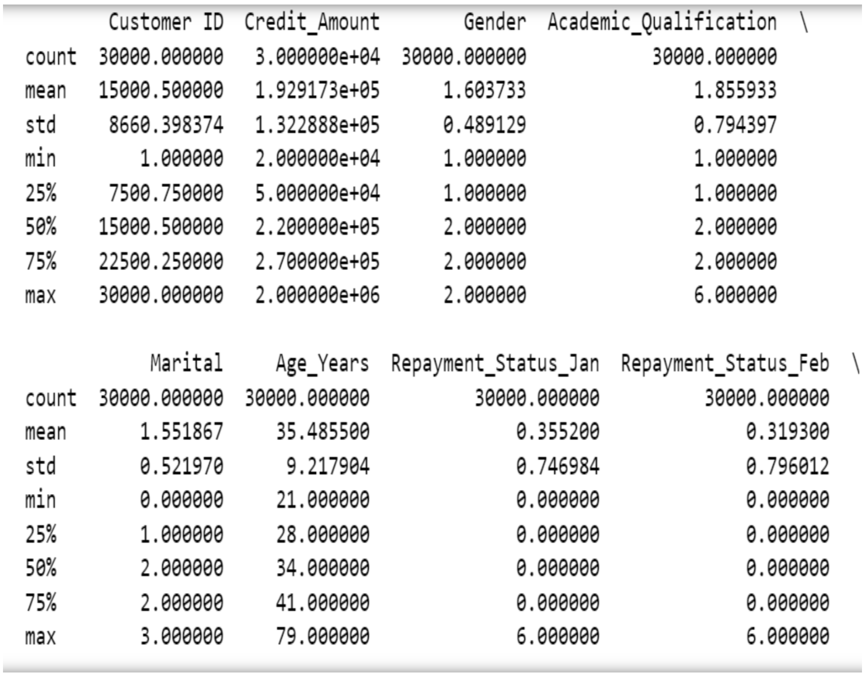
```
importnumpyasnpimportpandasaspd

fromscipy.statsi

mportrandintimp

ortpandas as pd

import      matplotlib.pyplot      as      pltfrom      pandas      import

set_optionplt.style.use('ggplot')

from sklearn.model_selection import

train_test_splitfromsklearn.linear_mode

limportLogisticRegressionfromsklearn.

feature_selection importRFE

fromsklearn.model_selectionimportKFold
fromsklearn.model_selectionimportGridSearchCV
fromsklearn.model_selectionimportRandomiz
edSearchCVfromsklearn.preprocessingimport
StandardScaler
fromsklearn.pipelineimportPipeline
from sklearn.ensemble import
RandomForestClassifierimportxgboost
as xgb
fromxgboostimportXGBClassifier
fromsklearn.naive_bayesimportGaussianNB
fromsklearn.model_selectionimportcros
s_val_scorefromsklearn.metrics import
confusion_matrix
fromsklearn.neighborsimportKNeighb
orsClassifierfrom sklearn.tree import
DecisionTreeClassifierfromsklearn.ens
embleimportExtraTreesClassifier
fromsklearn.feature_selectionimportSelect
FromModelfromsklearn import metrics
importwarnings
warnings.filterwarnings("ignore",category=Fu
tureWarning)fromsklearn.metrics
importclassification_report

BankCreditCard = pd.read_csv("E:\BALA\AI\Lab
programs\pgms\BankCreditCard.csv")print(f'Theshapeof thedataframe is
{BankCreditCard.shape}')
print()
print(BankCredi
tCard.info())prin
```

```
t()

BankCreditCard.replace(to_replace='?',value=np.NaN,i
nplace=True)print(BankCreditCard.describe(include='al
l'))

print()

print(BankCreditCard['Credit_Amount'].value_counts())print(Ba
nkCreditCard.isnull().sum())

importseabornassns
sns.countplot(x='Credit_Amount',data=BankCreditCard,
linewidth=3)plt.show()

BankCreditCard[['Customer     ID',     'Credit_Amount',     'Gender',
                                        'Academic_Qualification','Marital','Age_Years','Jan
_Bill_Amount', 'Feb_Bill_Amount']].hist(bins=50,figsize=(15,8))
plt.show()

BankCreditCard['Credit_Amount'].fillna(BankCreditCard['Credit_Amount'].mode()[
0], inplace=True)
BankCreditCard['Jan_Bill_Amount'].fillna(BankCreditCard['Jan_Bill_Amount'].mod
e()[0], inplace=True)
X=BankCreditCard.drop(['Mar
ital'],axis=1)y=BankCreditCar
d.Marital
X
=X[['Repayment_Status_Jan','Repayment_Status_Feb','Repayment_Status_March','R
epayment_ Status_April','Repayment_Status_May']]
X_train,X_test,y_train,y_test=train_test_split(X,y,test_
size=0.2)NB_classifier= GaussianNB()
```

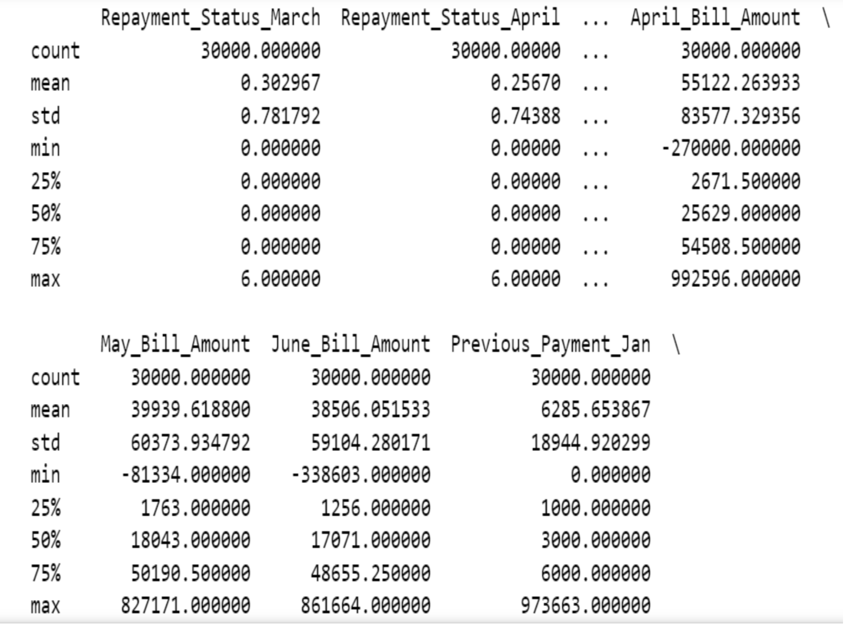
```
NB_classifier.fit(X_train,y
_train)y_predict =
NB_classifier.predict(X_tes
t)cm=confusion_matrix(y_t
est,y_predict)

sns.heatmap(cm, annot=True,
cmap='Blues')print(classificati
on_report(y_test,y_predict))
```

OUTPUT:

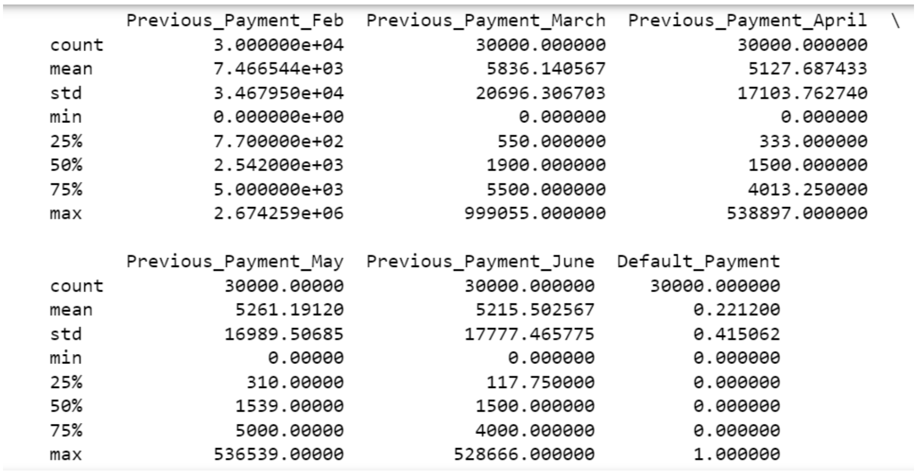
```
 Theshapeofthedataframeis(30000,25)

<class
'pandas.core.frame.DataFram
e'>RangeIndex:         30000
entries,        0        to
29999Datacolumns(total25col
umns):
 #Column                     Non-NullCountDtype
---------------------------------------------------------  ---------
  0     CustomerID              30000    non-null   int64
  1     Credit_Amount           30000    non-null   float64
  2     Gender                  30000    non-null   int64
  3     Academic_Qualification  30000    non-null   int64
  4     Marital                 30000    non-null   int64
  5     Age_Years               30000    non-null   int64
  6     Repayment_Status_Jan    30000    non-null   int64
  7     Repayment_Status_Feb    30000    non-null   int64
  8     Repayment_Status_March  30000    non-null   int64
  9     Repayment_Status_April  30000    non-null   int64
  10Repayment_Status_May        30000    non-null   int64
  11Repayment_Status_June       30000    non-null   int64
  12Jan_Bill_Amount             30000    non-null   float64
  13Feb_Bill_Amount             30000    non-null   float64
  14March_Bill_Amount           30000    non-null   float64
  15April_Bill_Amount           30000    non-null   float64
  16May_Bill_Amount             30000    non-null   float64
  17June_Bill_Amount            30000    non-null   float64
  18Previous_Payment_Jan        30000    non-null   float64
  19Previous_Payment_Feb        30000    non-null   float64
  20Previous_Payment_March      30000    non-null   float64
  21Previous_Payment_April      30000    non-null   float64
  22Previous_Payment_May        30000    non-null   float64
  23Previous_Payment_June       30000    non-null   float64
  24Default_Payment             30000    non-null   int64
dtypes: float64(13),
int64(12)memoryusage:
5.7MB
None
```



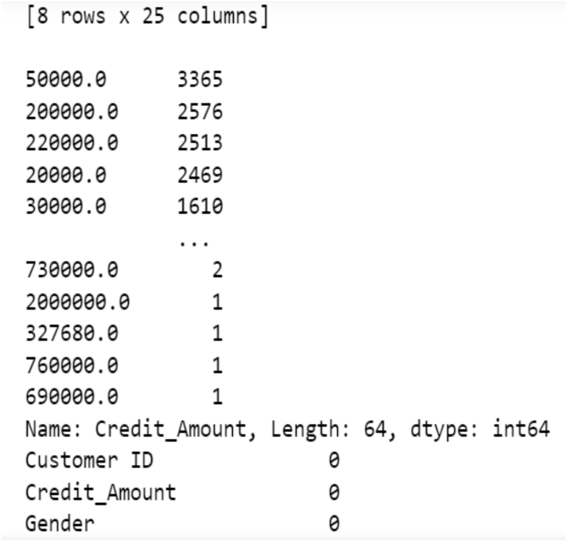
```
        Customer ID  Credit_Amount        Gender  Academic_Qualification  \
count  30000.000000   3.000000e+04  30000.000000            30000.000000
mean   15000.500000   1.929173e+05      1.603733                1.855933
std     8660.398374   1.322888e+05      0.489129                0.794397
min        1.000000   2.000000e+04      1.000000                1.000000
25%     7500.750000   5.000000e+04      1.000000                1.000000
50%    15000.500000   2.200000e+05      2.000000                2.000000
75%    22500.250000   2.700000e+05      2.000000                2.000000
max    30000.000000   2.000000e+06      2.000000                6.000000

            Marital     Age_Years  Repayment_Status_Jan  Repayment_Status_Feb  \
count  30000.000000  30000.000000          30000.000000          30000.000000
mean       1.551867     35.485500              0.355200              0.319300
std        0.521970      9.217904              0.746984              0.796012
min        0.000000     21.000000              0.000000              0.000000
25%        1.000000     28.000000              0.000000              0.000000
50%        2.000000     34.000000              0.000000              0.000000
75%        2.000000     41.000000              0.000000              0.000000
max        3.000000     79.000000              6.000000              6.000000
```



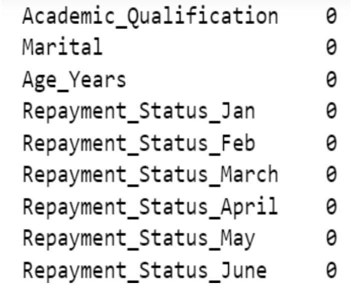
```
       Repayment_Status_March  Repayment_Status_April  ...  April_Bill_Amount  \
count            30000.000000             30000.00000  ...       30000.000000
mean                 0.302967                 0.25670  ...       55122.263933
std                  0.781792                 0.74388  ...       83577.329356
min                  0.000000                 0.00000  ...     -270000.000000
25%                  0.000000                 0.00000  ...        2671.500000
50%                  0.000000                 0.00000  ...       25629.000000
75%                  0.000000                 0.00000  ...       54508.500000
max                  6.000000                 6.00000  ...      992596.000000

       May_Bill_Amount  June_Bill_Amount  Previous_Payment_Jan  \
count     30000.000000      30000.000000          30000.000000
mean      39939.618800      38506.051533           6285.653867
std       60373.934792      59104.280171          18944.920299
min      -81334.000000    -338603.000000              0.000000
25%        1763.000000       1256.000000           1000.000000
50%       18043.000000      17071.000000           3000.000000
75%       50190.500000      48655.250000           6000.000000
max      827171.000000     861664.000000         973663.000000
```



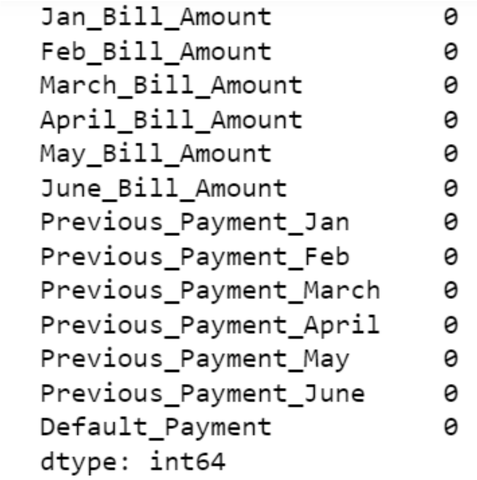
```
       Previous_Payment_Feb  Previous_Payment_March  Previous_Payment_April  \
count          3.000000e+04            30000.000000            30000.000000   
mean           7.466544e+03             5836.140567             5127.687433   
std            3.467950e+04            20696.306703            17103.762740   
min            0.000000e+00                0.000000                0.000000   
25%            7.700000e+02              550.000000              333.000000   
50%            2.542000e+03             1900.000000             1500.000000   
75%            5.000000e+03             5500.000000             4013.250000   
max            2.674259e+06           999055.000000           538897.000000   

       Previous_Payment_May  Previous_Payment_June  Default_Payment  
count           30000.00000           30000.000000     30000.000000  
mean             5261.19120            5215.502567         0.221200  
std             16989.50685           17777.465775         0.415062  
min                 0.00000               0.000000         0.000000  
25%               310.00000             117.750000         0.000000  
50%              1539.00000            1500.000000         0.000000  
75%              5000.00000            4000.000000         0.000000  
max            536539.00000          528666.000000         1.000000  
```



```
[8 rows x 25 columns]

50000.0      3365
200000.0     2576
220000.0     2513
20000.0      2469
30000.0      1610
             ... 
730000.0        2
2000000.0       1
327680.0        1
760000.0        1
690000.0        1
Name: Credit_Amount, Length: 64, dtype: int64
Customer ID               0
Credit_Amount             0
Gender                    0
```

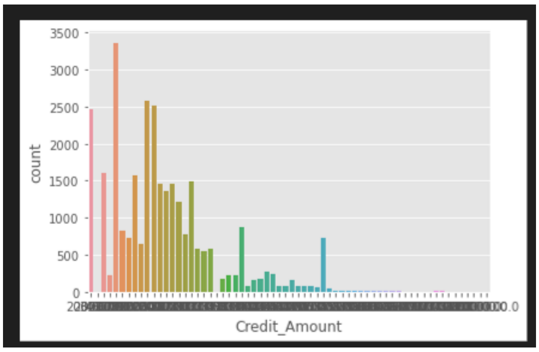


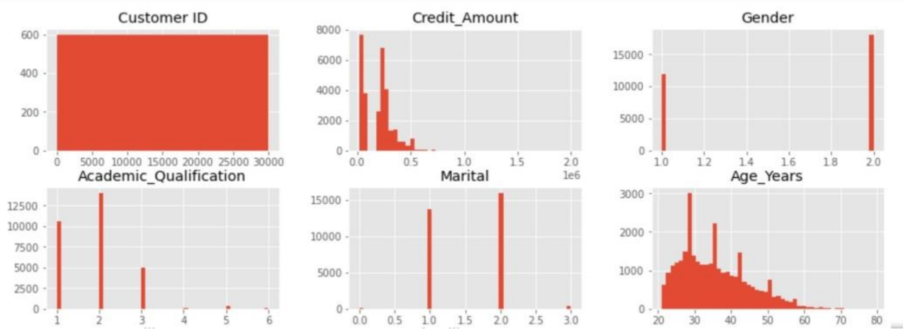
```
Academic_Qualification    0
Marital                   0
Age_Years                 0
Repayment_Status_Jan      0
Repayment_Status_Feb      0
Repayment_Status_March    0
Repayment_Status_April    0
Repayment_Status_May      0
Repayment_Status_June     0
```

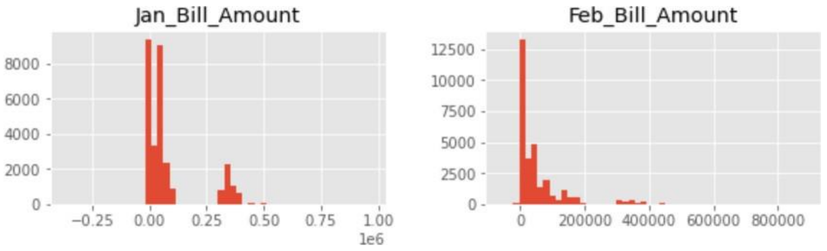


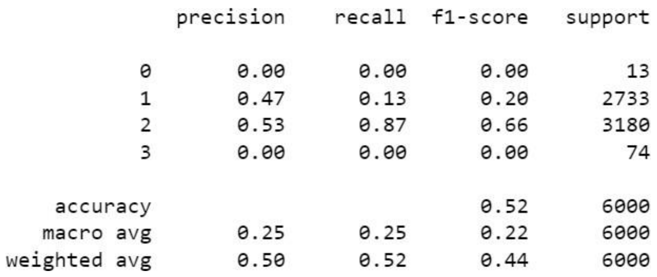
```
Jan_Bill_Amount            0
Feb_Bill_Amount            0
March_Bill_Amount          0
April_Bill_Amount          0
May_Bill_Amount            0
June_Bill_Amount           0
Previous_Payment_Jan       0
Previous_Payment_Feb       0
Previous_Payment_March     0
Previous_Payment_April     0
Previous_Payment_May       0
Previous_Payment_June      0
Default_Payment            0
dtype: int64
```

ExploratoryDataSet:

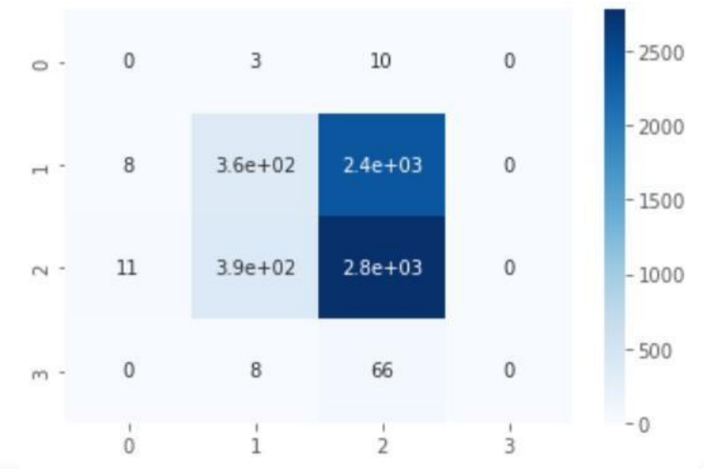








```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        13
           1       0.47      0.13      0.20      2733
           2       0.53      0.87      0.66      3180
           3       0.00      0.00      0.00        74

    accuracy                           0.52      6000
   macro avg       0.25      0.25      0.22      6000
weighted avg       0.50      0.52      0.44      6000
```



RESULT:

Thus the program to implement naïveBayesian classifier for credit card analysis and to compute the accuracy with few data sets was performed successfully.

Ex.No.:8	**INFERENCETHROUGH PYTHON**
Date:	

AIM:

TocalculatethestatisticalInferencethroughpython.

ALGORITHM:

1. Entirelybasedonyourpriorexperience,withoutanyobserveddata,— i.e.basedonp(theta)

   a.k.a**prior**,whichisanon-statisticianapproachto things.
2. Thesecondwayis thefrequentistmethod,inwhichweanticipatehow rareitistoobservethis outcome if the hypothesis is true, i.e. p(data | theta) a.k.a **likelihood**, which means wearesimplyrelyingon observed dataand nothingelse.
3. Finally, we have Bayesian inference, which uses **both** our prior knowledge p(theta) andour observed data to construct a distribution of probable posteriors. So one key differencebetweenfrequentistand Bayesianinferenceisour priorknowledge,i.e. p(theta).

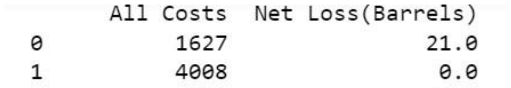
PROGRAM

```
importseabornassns

importmatplotli

b.pyplotaspltimp

ort numpyas np

importpandasaspd

data=pd.read_csv('E:\BALA\AI\Labprograms\pgms\database.csv')

cols_of_interest=['AccidentDate/Time','AccidentState','PipelineLocation','LiquidTyp
e','NetLoss(Barrels)','AllCosts']

data=data[cols_of_interest]
```

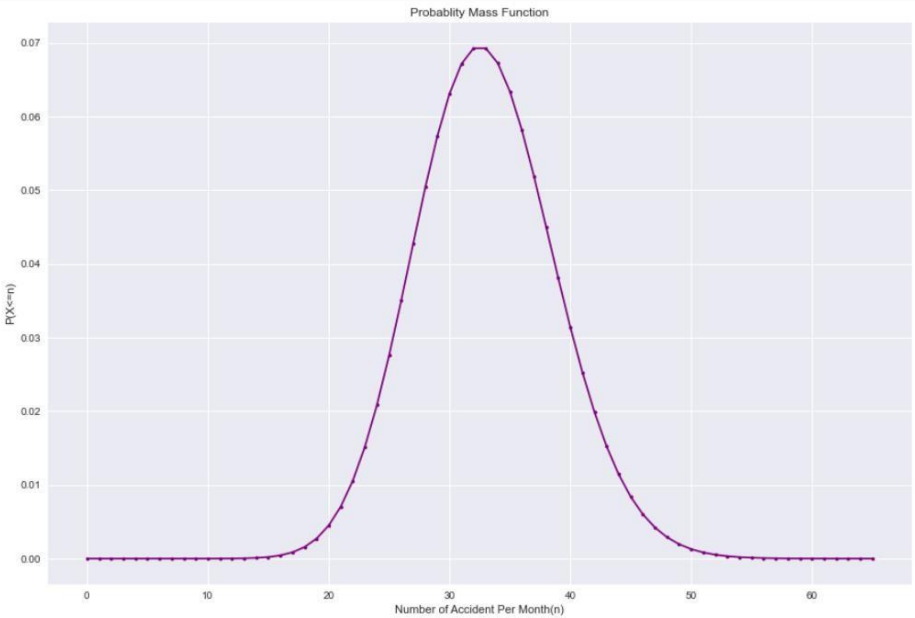
```
data_summary=print(data[['All Costs','Net
Loss(Barrels)']])data['All Costs']=data['All
Costs']/1000000plt.style.use('seaborn')
sns.boxplot(data['AllCosts'],data=data)
plt.title('Cost
sofAccident')
plt.show()
plt.close()
sns.boxplot(data['NetLoss(Barrels
)'],data=data)plt.title('NetLoss
(Barrels)')
plt.show()
data['Accident Date/Time']=pd.to_datetime(data['Accident
Date/Time'])totaltimespan=np.max(data['AccidentDate/Time'])-
np.min(data['AccidentDate/Time'])totaltime_hour=(totaltimespan.days*2
4+totaltimespan.seconds/(3600))totaltime_month=(totaltimespan.days+to
taltimespan.seconds/(3600*24))*12/365Imda_h=len(data)/totaltime_hour
Imda_m=len(data)/totaltime_month
print('Estimated no.of Accident per
hour:{}'.format(Imda_h))print('Estimatedno.ofAc
cidentpermonth:{}'.format(Imda_m))importmath
importmatplotli
b.pyplotaspltX=
{}
foriinrange(66):
   X[i]=math.pow(2.71828,-
1*33)*math.pow(33,i)/math.factorial(i)
p_Poission=pd.DataFrame(X.items(),columns=['X','PX'])plt.style
.use('seaborn')
```

```
fig=plt.subplots(figsize=(15,10))plt.plot(p_Poission['X'],p_Poission['P
X'],marker='.',color =
'purple',linestyle='solid')plt.xlabel('NumberofAccident Per Month(n)')
plt.ylabel('P(X<=n)')
plt.title('ProbablityM
assFunction')plt.show
()
plt.close()
defcdf(data):
    n=len(da
    ta)x=np.
    sort(data
    )y=np.ar
    ange(1,n
    +1)/nretu
    rn x,y
np.random.seed(42)sample_Poission=n
p.random.poisson(33,10000)x,y=cdf(sa
mple_Poission)fig=plt.subplots(figsize
=(15,10))
plt.plot(x,y,marker=",alpha=0.5,color='purple',lin
estyle='solid')plt.xlabel('Number of Accident Per
Month(n)')plt.ylabel('P(X<=n)')
plt.title('CummulativeDistributi
onFunction')plt.show()
```

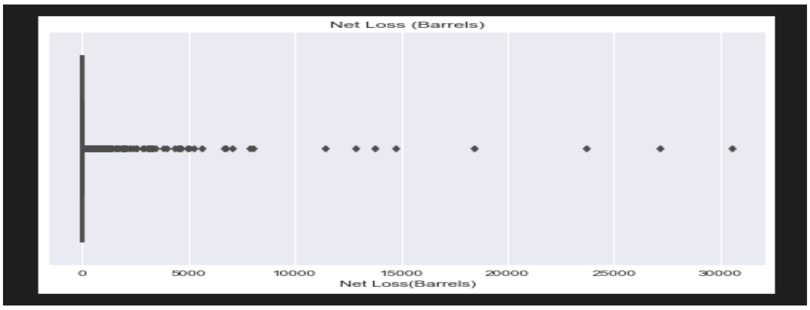
OUTPUT:



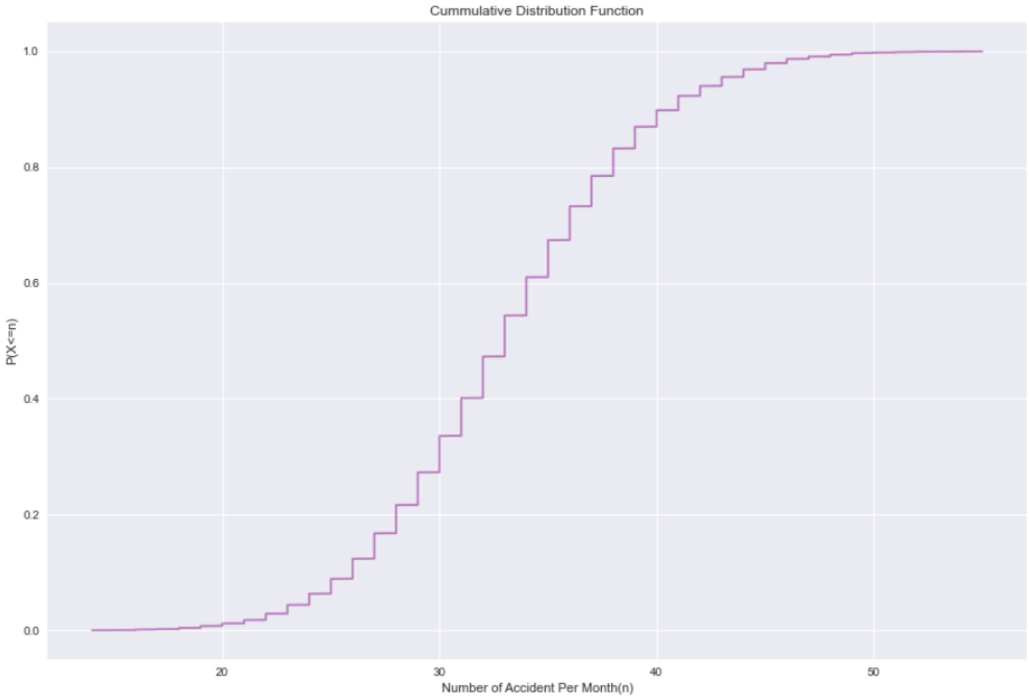
```
      All Costs  Net Loss(Barrels)
0          1627               21.0
1          4008                0.0
```



0
200
400
600
800
All Costs



Estimated no.of Accident per hour:0.04540255169379675Estimatedno.of Accidentpermonth:33.14386273647162



RESULT:

Thus the statistical Inference was calculated successfully using python.

Ex.No.:9	**USER-DEFINEDTYPESINSQLSERVER**
Date:	

AIM:

Toevaluatetheperformanceoflinearregression,logisticregression,naïvebayesa dSVMbasedprediction models forheart diseasediagnosis.

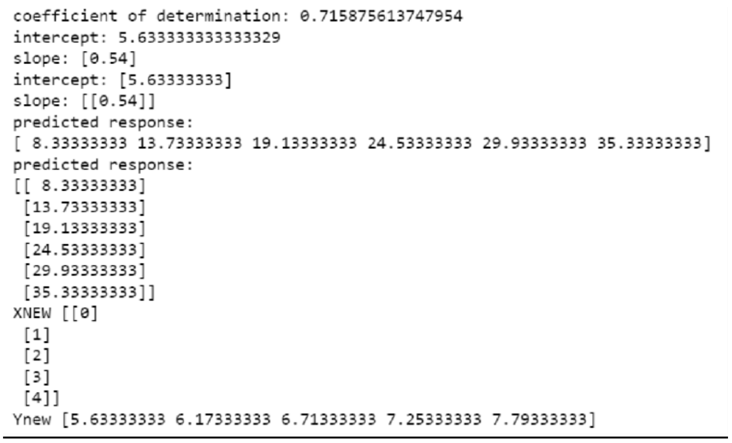
PROGRAM

LINEARREGRESSION

```
import numpyas np
fromsklearn.linear_modelimportLinea
rRegressionx=np.array([5,15,25,35,
45,55]).reshape((-1,1))
y=np.array([5, 20, 14, 32, 22, 38])
model =
LinearRegress
ion()model.fit
(x,y)
model=LinearRegressi
on().fit(x,y)r_sq=mod
el.score(x,y)
print(f"coefficientofdeterminat
ion:{r_sq}")print(f"intercept:
{model.intercept_}")print(f"slo
pe:{model.coef_}")
new_model = LinearRegression().fit(x,
y.reshape((-1,
1)))print(f"intercept:{new_model.intercept_
}")
print(f"slope:
```

```
{new_model.coef_}")y_pr
ed =
model.predict(x)print(f"pr
edicted
response:\n{y_pred}")
y_pred=model.intercept_+mod
el.coef_*xprint(f"predictedresp
onse:\n{y_pred}")


x_new =
np.arange(5).reshape((-
1,
1))print("XNEW",x_ne
w)
y_new=model.predict(x_new)print("Ynew",y_new)
```

OUTPUT:



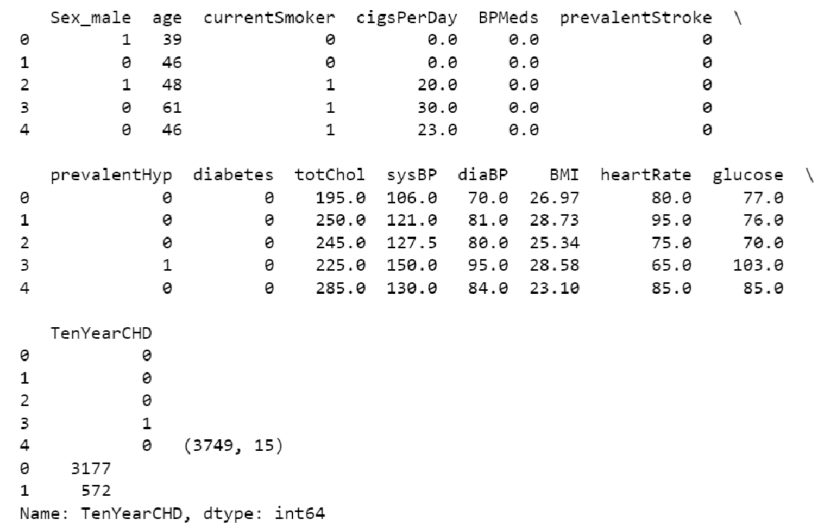
```
coefficient of determination: 0.715875613747954
intercept: 5.633333333333329
slope: [0.54]
intercept: [5.63333333]
slope: [[0.54]]
predicted response:
[ 8.33333333 13.73333333 19.13333333 24.53333333 29.93333333 35.33333333]
predicted response:
[[ 8.33333333]
 [13.73333333]
 [19.13333333]
 [24.53333333]
 [29.93333333]
 [35.33333333]]
XNEW [[0]
 [1]
 [2]
 [3]
 [4]]
Ynew [5.63333333 6.17333333 6.71333333 7.25333333 7.79333333]
```

LOGISTICREGRESSION

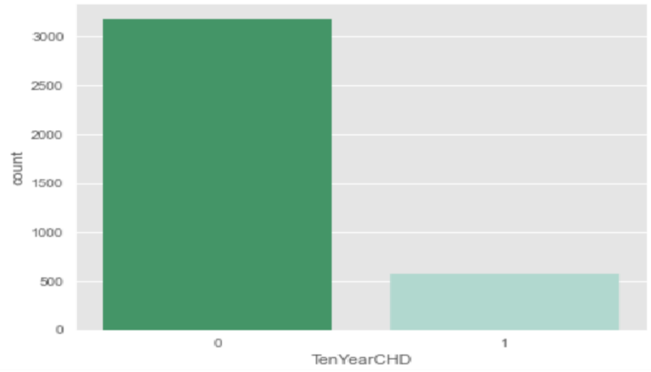
```
importpandasaspdimport pylab as plimportnumpyasnp
import scipy.optimize as optimportstatsmodels.apiassm
fromsklearnimportpreprocessing
fromsklearn.metricsimportja
ccard_score'exec(%matplotl
ib inline)'
import
matplotlib.pyplot
as
pltimportmatplotl
ib.mlabasmlabim
portseabornas sn
disease_df=pd.read_csv("E:\BALA\AI\Labprograms\pgms\l
ogistic.csv")disease_df.drop(['education'], inplace = True,
axis =
1)disease_df.rename(columns={'male':'Sex_male'},inplace=
True)
# removing NaN / NULL
valuesdisease_df.dropna(axis =
0, inplace =
True)print(disease_df.head(),
disease_df.shape)print(disease_d
f.TenYearCHD.value_counts())p
lt.figure(figsize=(7, 5))
sn.countplot(x ='TenYearCHD', data = disease_df, palette
="BuGn_r" )plt.show()

laste =
disease_df['TenYearCHD'
].plot()plt.show(laste)
X=np.asarray(disease_df[['age','Sex_male','cigsPerDay','totChol','sysBP','gl
ucose']])y=np.asarray(disease_df['TenYearCHD'])
#normalizationofthedataset
X =
preprocessing.StandardScaler().fit(X).tran
sform(X)#Train-and-Test -Split
fromsklearn.model_selectionimporttrain_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3,
random_state = 4)print('Train set:', X_train.shape,y_train.shape)
print('Testset:', X_test.shape,  y_test.shape)
fromsklearn.linear_modelimportLogisti
cRegressionlogreg=LogisticRegression(
)
logreg.fit(X_train
,y_train)y_pred=l
```

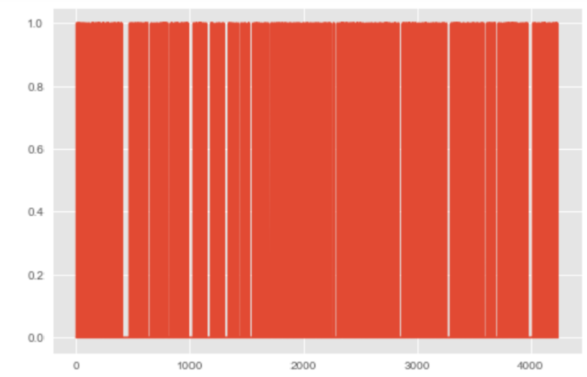
```
ogreg.predict(X_t

est)#Evaluation

and accuracy

fromsklearn.metricsimportja

ccard_scoreprint('')

print('Accuracy of the model in jaccard similarity score is = ',

jaccard_score(y_test, y_pred))fromsklearn.metrics import

confusion_matrix,classification_report

cm=confusion_matrix(y_test, y_pred)

conf_matrix=pd.DataFrame(data=cm,columns=['Predicted:0','Predicted:1'],index=['
Actual:0','Actual:1'])


plt.figure(figsize=(8,5))

sn.heatmap(conf_matrix, annot = True, fmt = 'd',

cmap = "Greens")plt.show()

print('The details for confusion matrix is =')print (classification_report(y_test, y_pred))
```

OUTPUT:

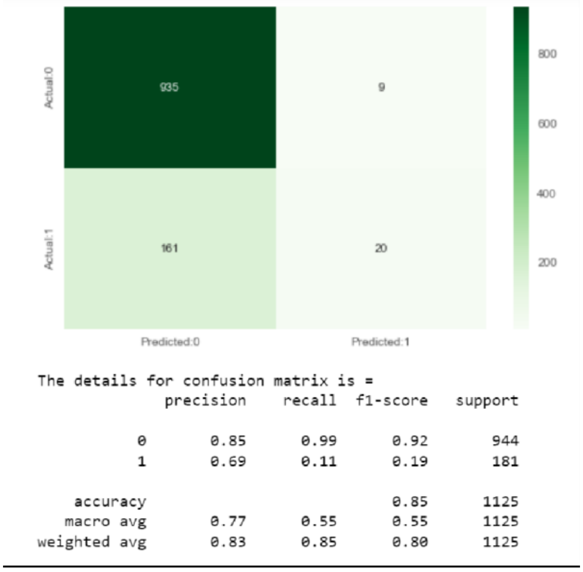


```
   Sex_male  age  currentSmoker  cigsPerDay  BPMeds  prevalentStroke  \
0         1   39              0         0.0     0.0                0
1         0   46              0         0.0     0.0                0
2         1   48              1        20.0     0.0                0
3         0   61              1        30.0     0.0                0
4         0   46              1        23.0     0.0                0

   prevalentHyp  diabetes  totChol  sysBP  diaBP    BMI  heartRate  glucose  \
0             0         0    195.0  106.0   70.0  26.97       80.0     77.0
1             0         0    250.0  121.0   81.0  28.73       95.0     76.0
2             0         0    245.0  127.5   80.0  25.34       75.0     70.0
3             1         0    225.0  150.0   95.0  28.58       65.0    103.0
4             0         0    285.0  130.0   84.0  23.10       85.0     85.0

   TenYearCHD
0           0
1           0
2           0
3           1
4           0   (3749, 15)
0    3177
1     572
Name: TenYearCHD, dtype: int64
```





```
Train set: (2624, 6) (2624,)
Test set: (1125, 6) (1125,)

Accuracy of the model in jaccard similarity score is =  0.10526315789473684
```



```
The details for confusion matrix is =
              precision    recall  f1-score   support

           0       0.85      0.99      0.92       944
           1       0.69      0.11      0.19       181

    accuracy                           0.85      1125
   macro avg       0.77      0.55      0.55      1125
weighted avg       0.83      0.85      0.80      1125
```

SVMBASEDPREDICTIONMODELS

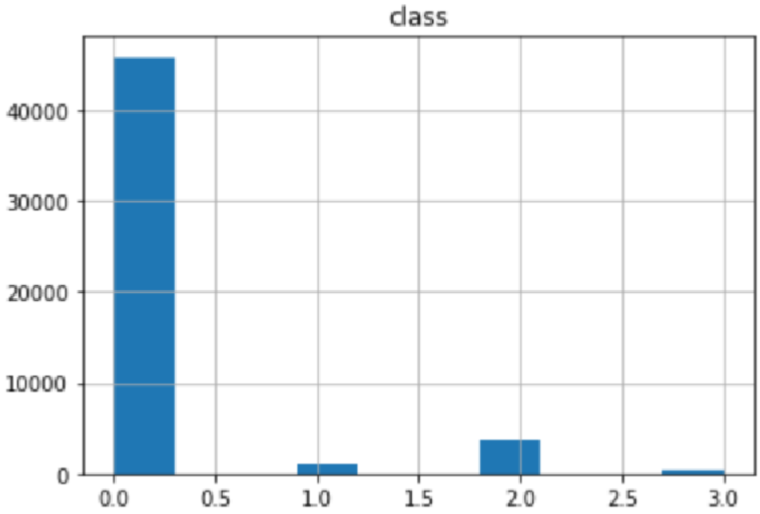
```
import
warningswarnings.filte
rwarnings('ignore')imp
ort numpyas np
importmatplotli
b.pyplotaspltimp
ortsklearn
from sklearn.svm import
LinearSVC,SVCfromsklear
n.svm importSVC
fromsklearn.preprocessingimportStandardScaler
fromsklearn.model_selectionimportlearning_curve,Gri
dSearchCVimportseabornas sn
importpandasaspd
signals = pd.read_csv("E:\BALA\AI\Lab programs\pgms\DS1_signals.csv",
header=None)labels = pd.read_csv("E:\BALA\AI\Lab
programs\pgms\DS1_labels.csv", header=None)print("*"*50)
print("SignalsInfo:")print("*"*50)print(signals.info())print("*"*50)print("Label
s Info:")print("*"*50)print(labels.info())print("*"*50)signals.head()
print("ColumnNumb
erofNaN's")forcol in
signals.columns:
   if
      signals[col].isnull().su
      m() >
      0:print(col,signals[col]
      .isnull().sum())

joined_data=signals.join(labels,rsuffix="_signals",lsuffix
="_labels")joined_data.columns = [i for i in
range(180)]+['class']cor_mat=joined_data.corr()
print('*'*50)
print('Top10highpositivelycorrelatedfeatures')p
rint('*'*50)print(cor_mat['class'].sort_values(asc
ending=False).head(10)) print('*'*50)
print('Top10highnegativelycorrelat
edfeatures')print('*'*50)print(cor_m
at['class'].sort_values().head(10))
%matplotlibinline
frompandas.plottingimportsc
atter_matrixfeatures=[79,80,
78,77]
scatter_matrix(joined_data[features],figsize=(20,15),c=joined_data['class'],
alpha=0.5);print('-'*20)
print('Class\t%')print('-'*20)
print(joined_data['class'].value_counts()/len(joined_data))
joined_data.hist('class');
print('-'*20)
fromsklearn.model_selectionimportStratifiedShuffleSplit
split1 = StratifiedShuffleSplit(n_splits=1,
test_size=0.2,random_state=42)fortrain_index,test_indexins
plit1.split(joined_data,joined_data['class']):
    strat_train_set =
    joined_data.loc[train_index]strat_test
    _set =
    joined_data.loc[test_index]strat_feat
    ures_train =
    strat_train_set.drop('class',
    1)labels_train= strat_train_set['class']

scaler=StandardScaler()
std_features=scaler.fit_transform(strat_fe
atures_train)svc_param_grid ={'C':[10],
'gamma':[0.1,1,10]}
svc=SVC(kernel='rbf',decision_function_shape='ovo',random_state=42,max
_iter=500)svc_grid_search = GridSearchCV(svc,svc_param_grid, cv=3,
scoring="f1_macro")svc_grid_search.fit(std_features,
labels_train)train_accuracy=svc_grid_search.best_score_
print('Model\t\tBestparams\
t\tBestscore')print("-"*50)
print("SVC\t\t",svc_grid_search.best_params_,tra
in_accuracy)features_test=
strat_test_set.drop('class', 1)
labels_test=strat_test_set['class']
```

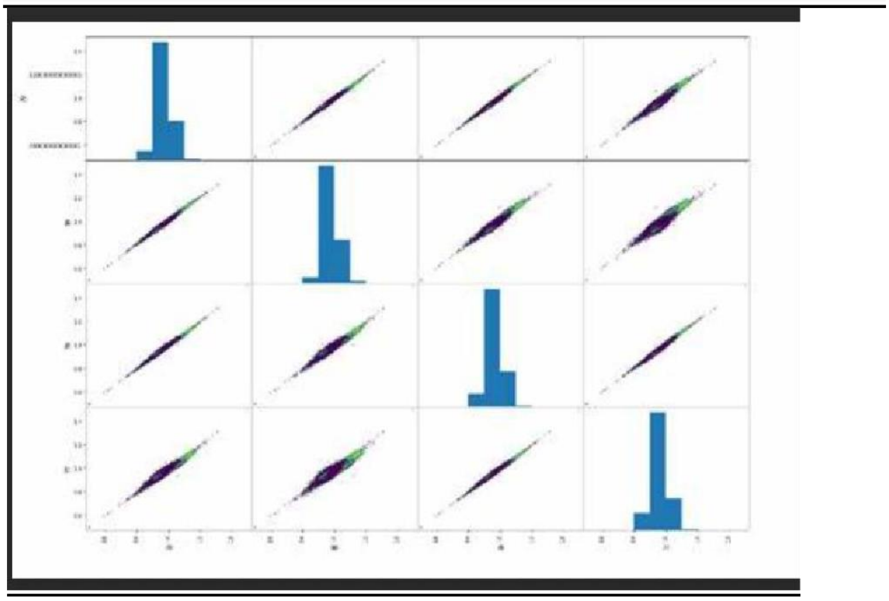
```
std_features=scaler.fit_transform(f
eatures_test)svc_grid_search.fit(st
d_features,
labels_test)test_accuracy=svc_gri
d_search.best_score_print('Model\
t\tBest params\t\tBest
score')print("-"*50)
print("SVC\t\t",svc_grid_search.best_params_,te
st_accuracy)print("TrainAccuracy:
"+str(train_accuracy))
print("TestAccuracy: "+str(test_accuracy))
```

OUTPUT:

```
        ****************************************
        ***********SignalsInfo:
        ***************************************************
        <class
        'pandas.core.frame.DataF
        rame'>RangeIndex:5100
        2entries,0to51001
        Columns:180ent
        ries,0to179dtype
        s:float64(180)
        memoryus
        age:70.0M
        BNone

        ****************************************
        ***********LabelsInfo:
        ***************************************************
        <class
        'pandas.core.frame.DataF
        rame'>RangeIndex:
        51002    entries,    0    to
        51001Datacolumns (total
        1 columns):
         #ColumnNon-Null CountDtype
--  ------- ---------  ------
         0   0      51002
        non-
        nullint64dtypes:i
        nt64(1)
        memoryusa
        ge:398.6K
        BNone
        ****************************************
        ***********ColumnNumber ofNaN's
        ****************************************
        ***********Top10 high
        positivelycorrelatedfeatures
        ***************************************************
        class    1.000000
        79       0.322446
        80       0.320138
        78       0.318702
        77       0.311504
```

```
81      0.310178
76      0.302628
82      0.292991
75      0.291687
98      0.285491
Name:class, dtype:float64
*****************************************
***********Top10 high
negativelycorrelatedfeatures
*****************************************************
153   -0.090500
154   -0.090206
152   -0.089958
155   -0.089625
156   -0.089017
157   -0.088890
151   -0.088853
158   -0.088647
150   -0.087771
159   -0.087768
Name:class,dtype:float64

Class       %
-------------
0    0.898475
2    0.074272
1    0.019137
3    0.008117
Name:class,dtype:float64
-------------
Model                Bestparams                    Bestscore
---------------------------------
SVC                   {'C':10, 'gamma':0.1}0.9104871061578681
Model                Bestparams                    Bestscore
---------------------------------
SVC                   {'C':10, 'gamma':0.1}0.8343809959585644
TrainAccuracy:0.9104871061578681
TestAccuracy:0.8343809959585644
```





RESULT:
Thus the performance of linear regression, logisticregression, naïvebayes and SVM based prediction models for heart disease diagnosis was evaluated successfully.