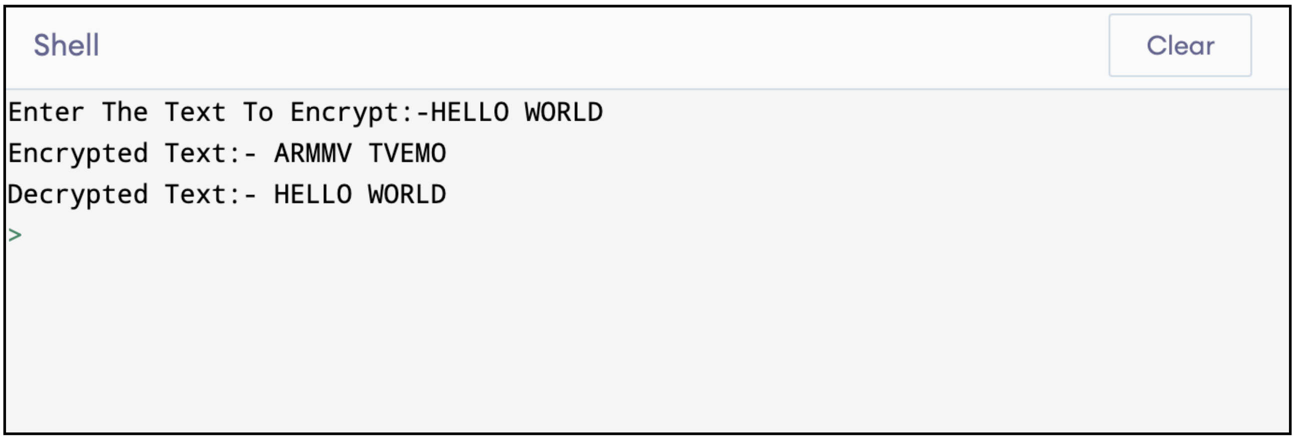
Problem Statement:- Encrypt and Decrypt using Affine Cipher, C=(aP+b)mod N, a=3,b=5.

Code:-

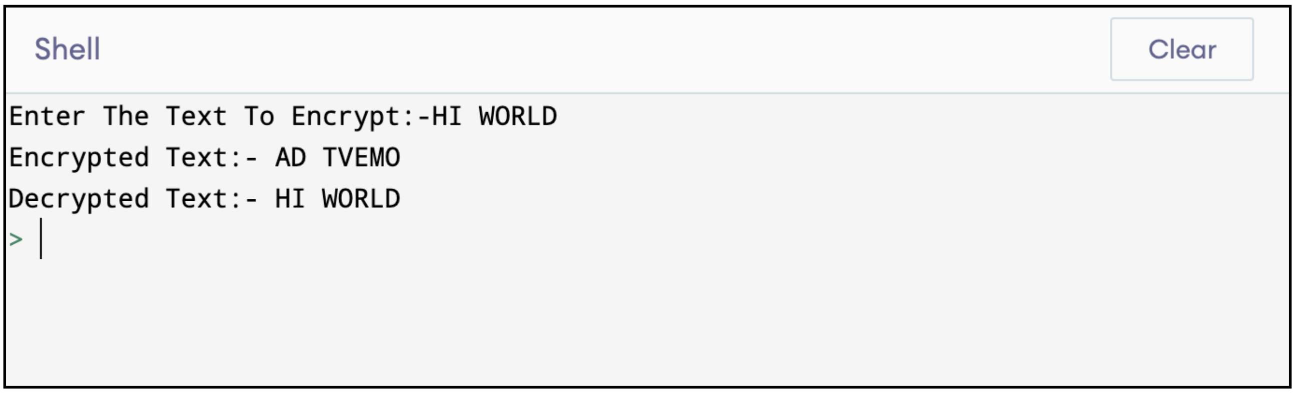
```
def affine_encrypt(text, a, b):
    encrypted_text = ''
    for char in text:
        if char.isalpha():
            ascii_code = ord(char) - ord('A')
            encrypted_ascii_code = (a * ascii_code + b) % 26
            encrypted_char = chr(encrypted_ascii_code + ord('A'))
            encrypted_text += encrypted_char
        else:
            encrypted_text += char
    return encrypted_text

def affine_decrypt(text, a, b):
    decrypted_text = ''
    for x in range(1, 26):
        if (a * x) % 26 == 1:
            a_inverse = x
            break
    else:
            raise ValueError(f"a ({a}) and 26 are not coprime, so a doesn't have an
inverse.")
    for char in text:
        if char.isalpha():
            ascii_code = ord(char) - ord('A')
            decrypted_ascii_code = (a_inverse * (ascii_code - b)) % 26
            decrypted_char = chr(decrypted_ascii_code + ord('A'))
            decrypted_text += decrypted_char
        else:
            decrypted_text += char
    return decrypted_text

plaintext = input(‘Enter The Text To Encrypt:-’)
a = 3
b = 5
cipher_text = affine_encrypt(plaintext, a, b)
print(‘Encrypted Text:-‘,  cipher_text)
```

```
decrypted_text = affine_decrypt(cipher_text, a, b)
print(‘Decrypted Text:-‘, decrypted_text)
```

Result:-



```
Enter The Text To Encrypt:-HELLO WORLD
Encrypted Text:- ARMMV TVEMO
Decrypted Text:- HELLO WORLD
>
```



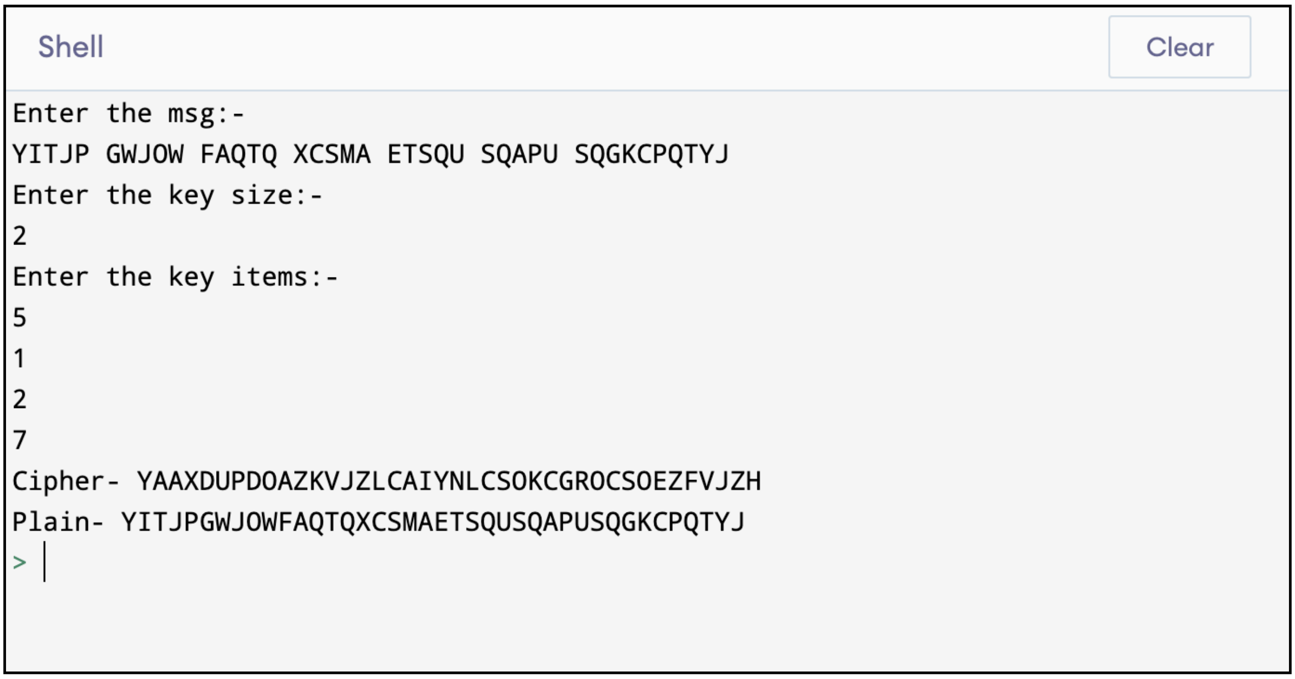
```
Enter The Text To Encrypt:-HI WORLD
Encrypted Text:- AD TVEMO
Decrypted Text:- HI WORLD
>
```

Problem Statement:- Decipher the message YITJP GWJOW FAQTQ XCSMA ETSQU SQAPU SQGKCPQTYJ with the help of Hill cipher with the inverse key (5 1
2 7).

Code:-

```
N = 2

def modular_inverse(n, b):
    r1, r2 = n, b
    t1, t2 = 0, 1
    while r2 > 0:
        q = r1 // r2
        r = r1 - q * r2
        r1 = r2
        r2 = r
        t = t1 - q * t2
        t1 = t2
        t2 = t
    if t1 < 0:
        t1 += n
    return t1

def encrypt(mat, key, n):
    res = ""
    for i in range(n):
        ans = 0
        for j in range(n):
            ans += mat[j] * key[i][j]
        ans = ans % 26
        res += chr(ans + ord('A'))
    return res

def get_cofactor(A, temp, p, q, n):
    i, j = 0, 0
    for row in range(n):
        for col in range(n):
            if row != p and col != q:
                temp[i][j] = A[row][col]
                if j == n - 1:
                    j = 0
                    i += 1
                else:
                    j += 1

def determinant(A, n):
    D = 0
    if n == 1:
        return A[0][0]
    temp = [[0 for _ in range(N)] for _ in range(N)]
    sign = 1
    for f in range(n):
        get_cofactor(A, temp, 0, f, n)
        D += sign * A[0][f] * determinant(temp, n - 1)
        sign = -sign
    return D

def adjoint(A, adj):
    if N == 1:
        adj[0][0] = 1
        return
    sign = 1
    temp = [[0 for _ in range(N)] for _ in range(N)]
    for i in range(N):
        for j in range(N):
            get_cofactor(A, temp, i, j, N)
            sign = 1 if (i + j) % 2 == 0 else -1
            adj[j][i] = sign * determinant(temp, N - 1)

def decrypt(mat, key, n):
    det = determinant(key, n)
    adj = [[0 for _ in range(n)] for _ in range(n)]
    adjoint(key, adj)
    inverse = modular_inverse(26, det)
    for i in range(n):
        for j in range(n):
            adj[i][j] = (adj[i][j] * inverse) % 26
            if adj[i][j] < 0:
                adj[i][j] += 26
    return encrypt(mat, adj, n)

def main():
    print("Enter the msg:- ")
    msg = input()
    print("Enter the key size:- ")
    n = int(input())
    key = [[0]*n for i in range(n)]
    print("Enter the key items:- ")
    for i in range(n):
        for j in range(n):
            key[i][j] = int(input())
    cipher = ""
    i = 0
    while i < len(msg):
        mat = [0]*n
        count = 0
        while i < len(msg) and count != n:
            if msg[i] == ' ':
                i += 1
            else:
                mat[count] = ord(msg[i]) - ord('A')
                i += 1
                count += 1
        cipher += encrypt(mat, key, n)
    print("Cipher- "+cipher)
    plain = ""
    i = 0
    while i < len(cipher):
        mat = [0]*n
        count = 0
        while i < len(cipher) and count < n:
            mat[count] = ord(cipher[i]) - ord('A')
            i += 1
            count += 1
        plain += decrypt(mat, key, n)
    print("Plain- "+ plain)

if __name__ == '__main__':
    main()
```

Result:-



```
Enter the msg:-
YITJP GWJOW FAQTQ XCSMA ETSQU SQAPU SQGKCPQTYJ
Enter the key size:-
2
Enter the key items:-
5
1
2
7
Cipher- YAAXDUPDOAZKVJZLCAIYNLCSOKCGROCSOEZFVJZH
Plain- YITJPGWJOWFAQTQXCSMAETSQUSQAPUSQGKCPQTYJ
>
```

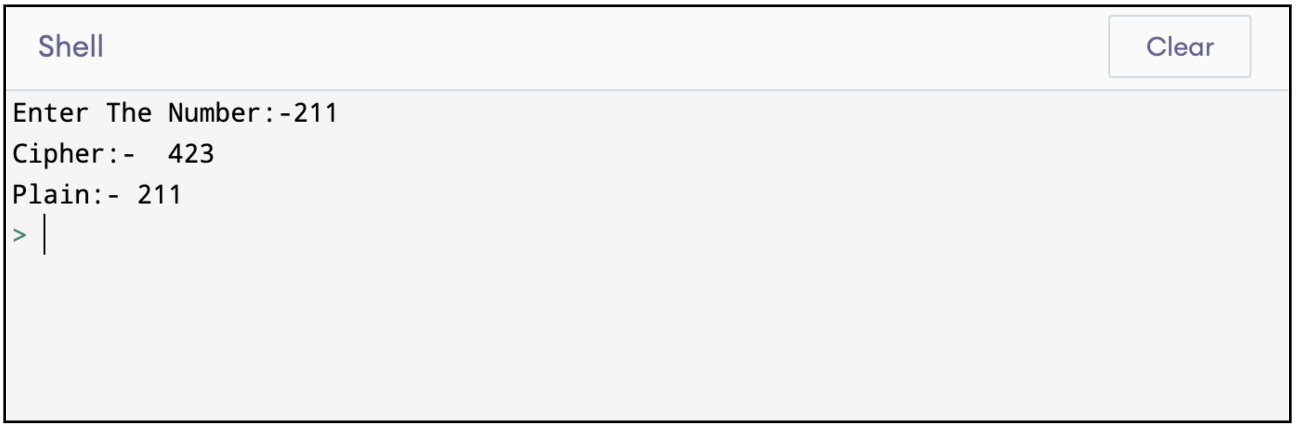
Problem Statement:- Encrypt and decrypt small numeral values using RSA Algorithm, say p=29, q=37, m=211.

Code:-

```
def findInverse(a: int, b: int) -> int:
    r1, r2 = b, a
    t1, t2 = 0, 1

    while r2 > 0:
        q = r1 // r2
        r = r1 - q * r2
        t = t1 - q * t2

        r1 = r2
        r2 = r

        t1 = t2
        t2 = t

    if t1 < 0:
        t1 += b

    if r1 == 1:
        return t1
    else:
        return 9999

def main():
    p = 53
    q = 59
    msg = int(input('Enter The Number:-'))
    phi = (p-1) * (q-1)
    n = p*q
    e = 9999
    d = 9999

    for i in range(2, phi):
        ans = findInverse(i, phi)
        if ans != 9999:
            e = i
            d = ans
```

```
                break

        if e == 9999:
            print("Not Possible")

        else:

            c = 1
            pp = None

            for i in range(1, e+1):
                c = (c*msg) % n

            print("Cipher:- ", c)

            pp = 1

            for i in range(1, d+1):
                pp = (pp*c) % n

            print("Plain:-", pp)

    if __name__ == '__main__':
        main()
```

Result:-



```
Enter The Number:-211
Cipher:-  423
Plain:- 211
>
```

Problem Statement:- Encrypting and decrypting plain text messages containing alphabets using their ASCII value, m="This is Computing Lab-1".

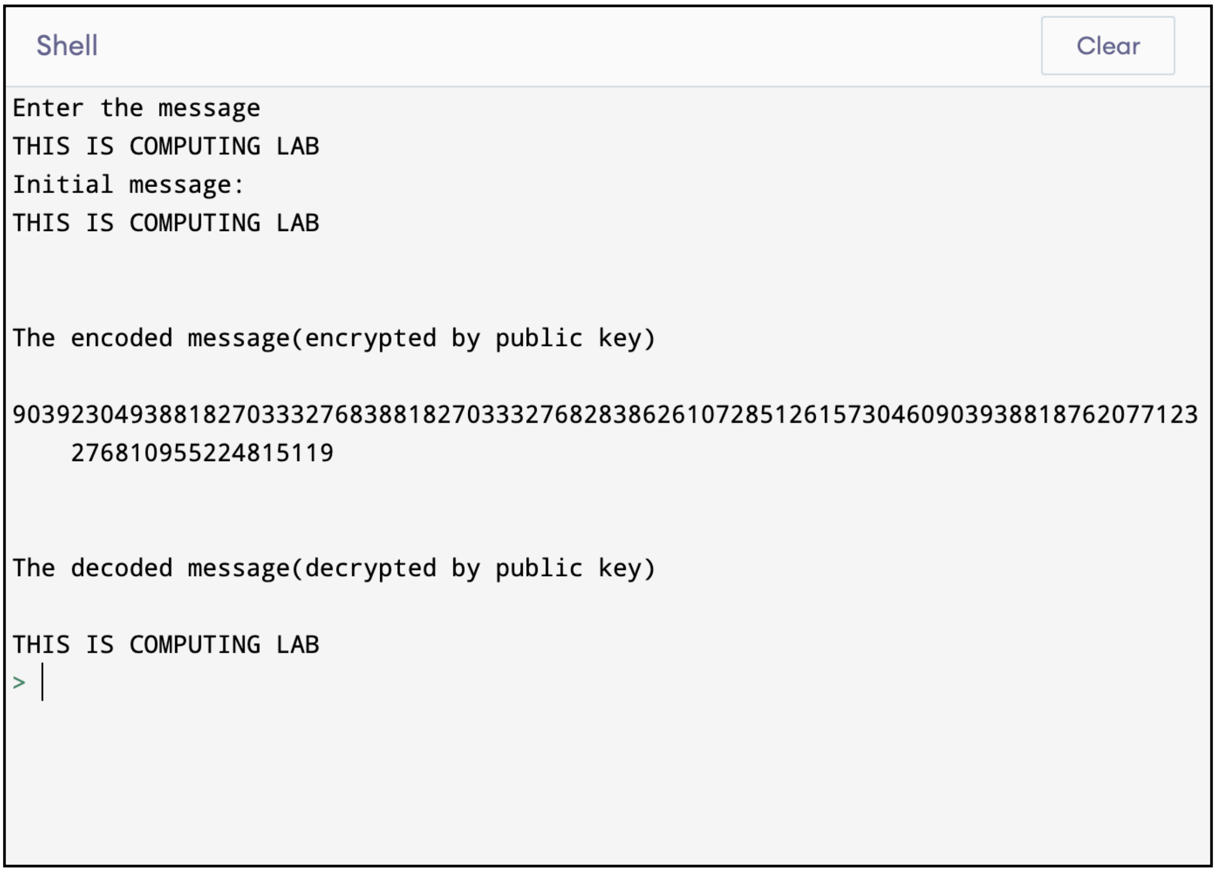
Code:-

```
import random
import math

prime = set()

public_key = None
private_key = None
n = None

def primefiller():
        seive = [True] * 250
        seive[0] = False
        seive[1] = False
        for i in range(2, 250):
                for j in range(i * 2, 250, i):
                        seive[j] = False

        for i in range(len(seive)):
                if seive[i]:
                        prime.add(i)

def pickrandomprime():
        global prime
        k = random.randint(0, len(prime) - 1)
        it = iter(prime)
        for _ in range(k):
                next(it)

        ret = next(it)
        prime.remove(ret)
        return ret


def setkeys():
        global public_key, private_key, n
        prime1 = pickrandomprime()
        prime2 = pickrandomprime()
```

```
    n = prime1 * prime2
    fi = (prime1 - 1) * (prime2 - 1)

    e = 2
    while True:
        if math.gcd(e, fi) == 1:
            break
        e += 1

    public_key = e

    d = 2
    while True:
        if (d * e) % fi == 1:
            break
        d += 1

    private_key = d

def encrypt(message):
    global public_key, n
    e = public_key
    encrypted_text = 1
    while e > 0:
        encrypted_text *= message
        encrypted_text %= n
        e -= 1
    return encrypted_text


# To decrypt the given number
def decrypt(encrypted_text):
    global private_key, n
    d = private_key
    decrypted = 1
    while d > 0:
        decrypted *= encrypted_text
        decrypted %= n
        d -= 1
    return decrypted
```

```
def encoder(message):
        encoded = []
        for letter in message:
                encoded.append(encrypt(ord(letter)))
        return encoded


def decoder(encoded):
        s = "
        for num in encoded:
                s += chr(decrypt(num))
        return s


if __name__ == '__main__':
        primefiller()
        setkeys()
        message = input("Enter the message\n")
        coded = encoder(message)

        print("Initial message:")
        print(message)
        print("\n\nThe encoded message(encrypted by public key)\n")
        print(".join(str(p) for p in coded))
        print("\n\nThe decoded message(decrypted by public key)\n")
        print(".join(str(p) for p in decoder(coded)))
```

Result:-



```
Enter the message
THIS IS COMPUTING LAB
Initial message:
THIS IS COMPUTING LAB


The encoded message(encrypted by public key)

90392304938818270333276838818270333276828386261072851261573046090393881876207712З
    276810955224815119


The decoded message(decrypted by public key)

THIS IS COMPUTING LAB
> 
```

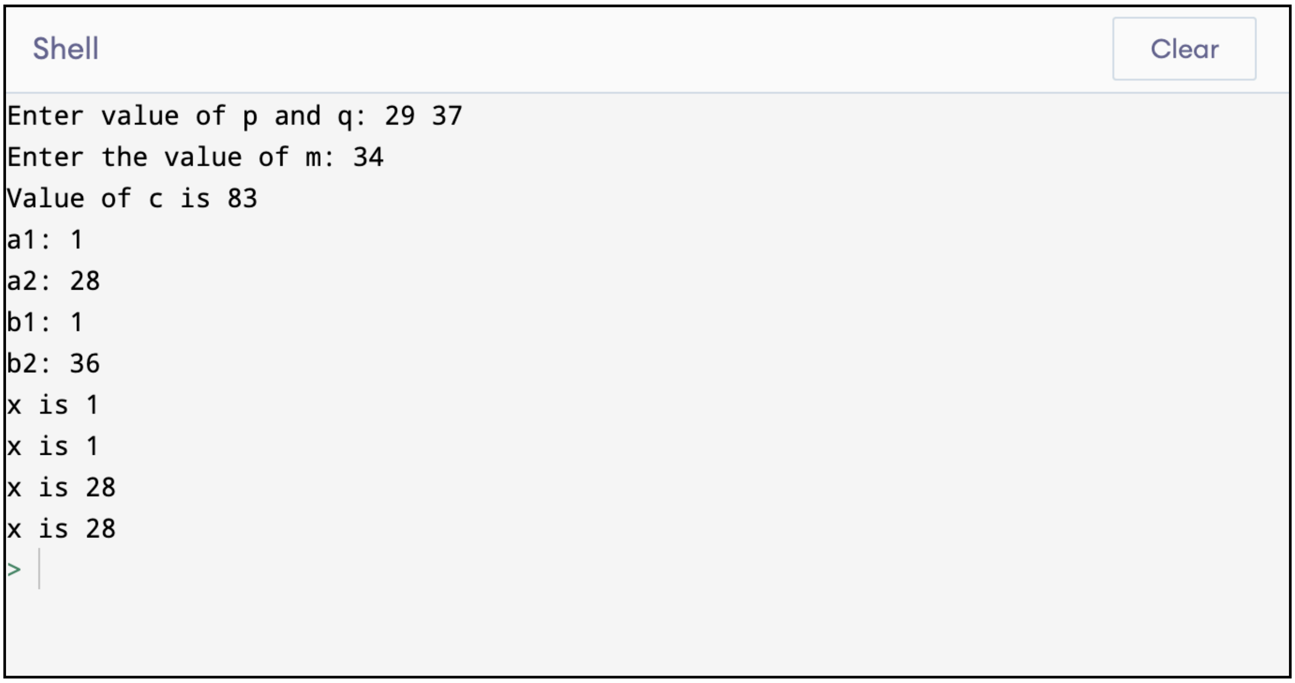
Problem Statement:- Encrypt and decrypt small numeral values using Rabin Cryptosystem Algorithm, say p=29, q=37, m=211.

Code:-

```
import math

def power(a, b, n):
    ans = 1
    for j in range(1, b+1):
        ans = (ans * a) % n
    return ans

def findMinX(num, rem, k):
    x = 1
    while True:
        for j in range(k):
            if x % num[j] != rem[j]:
                break
        if j == k-1:
            return x
        x += 1

def isprime(a):
    for i in range(2, int(math.sqrt(a))+1):
        if a % i == 0:
            return False
    return True

p, q = map(int, input("Enter value of p and q: ").split())

if p == q:
    print("Error")
    exit()

if not isprime(p) or not isprime(q):
    print("Error")
    exit()

n = p * q
e = 2
m = int(input("Enter the value of m: "))
```

```
c = power(m, 2, n)
print("Value of c is", c)

a1 = power(c, (p+1)//4, p)
a2 = p - a1

b1 = power(c, (q+1)//4, q)
b2 = q - b1

print("a1:", a1)
print("a2:", a2)
print("b1:", b1)
print("b2:", b2)

num = [p, q]
rem = [a1, b1]
k = 2
print("x is", findMinX(num, rem, k))

rem[0] = a1
rem[1] = b2
print("x is", findMinX(num, rem, k))

rem[0] = a2
rem[1] = b1
print("x is", findMinX(num, rem, k))

rem[0] = a2
rem[1] = b2
print("x is", findMinX(num, rem, k))
```

Result:-



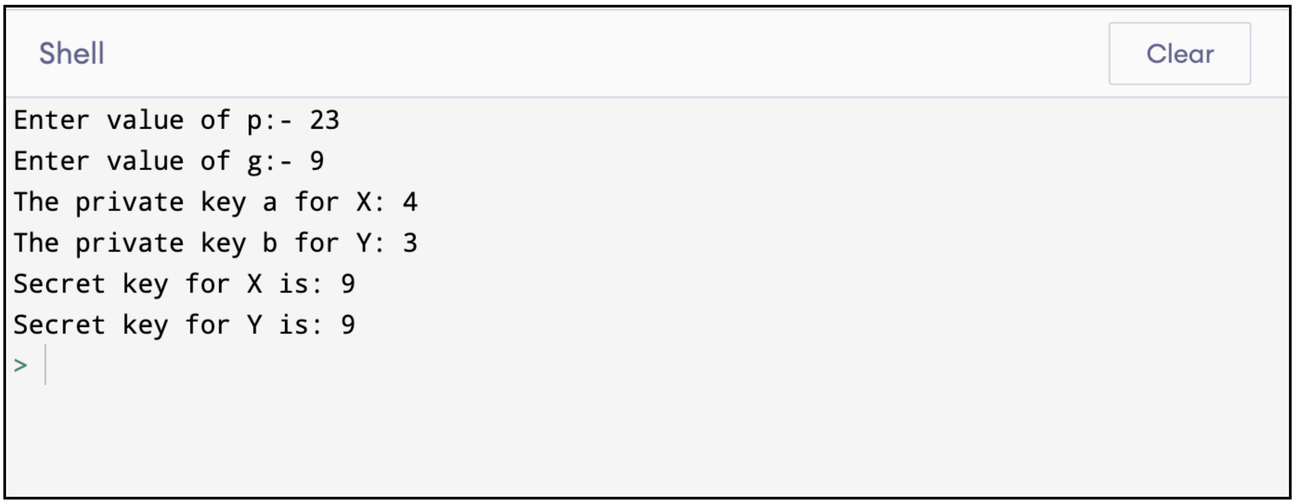
```
Enter value of p and q: 29 37
Enter the value of m: 34
Value of c is 83
a1: 1
a2: 28
b1: 1
b2: 36
x is 1
x is 1
x is 28
x is 28
>
```

Problem Statement:- Find the private and public keys for both Alice and Bob when p=23, g=9.

Code:-

```
def power(a, b, p):
    if b == 1:
        return a
    else:
        return pow(a, b, p)

P = int(input("Enter value of p:- "))
G = int(input("Enter value of g:- "))

a = 4
print("The private key a for X:", a)

x = power(G, a, P)

b = 3
print("The private key b for Y:", b)

y = power(G, b, P)

ka = power(y, a, P)
kb = power(x, b, P)

print("Secret key for X is:", ka)
print("Secret key for Y is:", kb)
```

Result:-



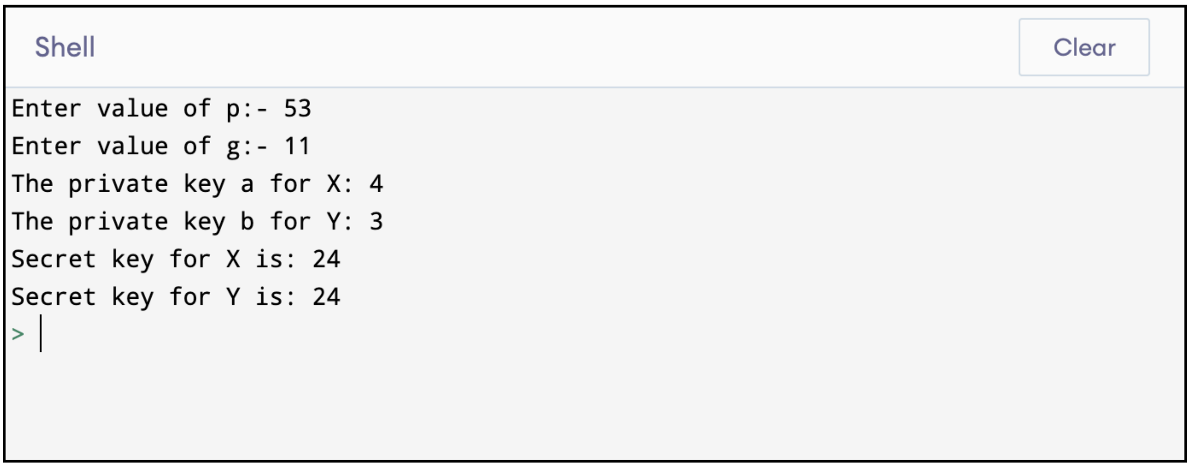
```
Enter value of p:- 23
Enter value of g:- 9
The private key a for X: 4
The private key b for Y: 3
Secret key for X is: 9
Secret key for Y is: 9
>
```

Problem Statement:- Find the private and public keys for both Alice and Bob when p=53, g=11.

Code:-

```
def power(a, b, p):
    if b == 1:
        return a
    else:
        return pow(a, b, p)

P = int(input("Enter value of p:- "))
G = int(input("Enter value of g:- "))

a = 4
print("The private key a for X:", a)

x = power(G, a, P)

b = 3
print("The private key b for Y:", b)

y = power(G, b, P)

ka = power(y, a, P)
kb = power(x, b, P)

print("Secret key for X is:", ka)
print("Secret key for Y is:", kb)
```

Result:-



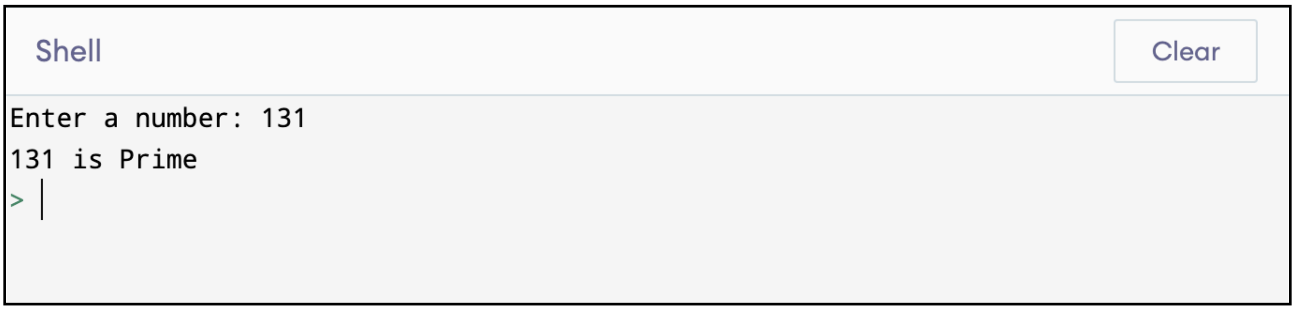
```
Enter value of p:- 53
Enter value of g:- 11
The private key a for X: 4
The private key b for Y: 3
Secret key for X is: 24
Secret key for Y is: 24
> 
```

Problem Statement:- Find whether a number N is prime or not using Miller-Rabin Method, take k=4.

Code:-

```
def findMandK(n):
    k = 0
    while n % 2 == 0:
        n //= 2
        k += 1
    return (n, k)

n = int(input("Enter a number: "))

m, k = findMandK(n-1)

t = 1

for i in range(1, m+1):
    t = (t*2) % n

if t == 1 or t == -1:
    print(n, "is Prime")
else:
    for i in range(1, k):
        t = (t*t) % n
    if t == -1 or t == n-1:
        print(n, "is Prime")
    else:
        print(n, "is Not Prime")
```

Result:-



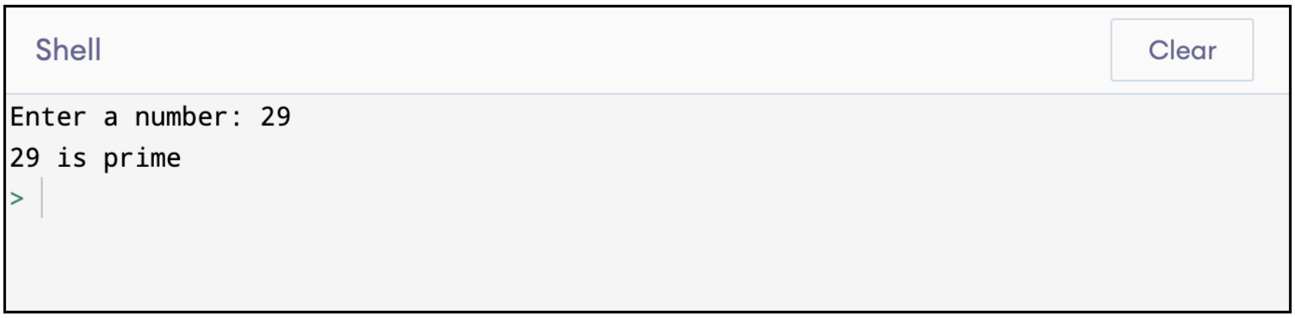
```
Enter a number: 131
131 is Prime
>
```

Problem Statement:- For a given number N check if it is prime or not using Pollard's Rho Algorithm.

Code:-

```
def gcd(a, b):
    gcd = 0
    for i in range(1, max(a, b)+1):
        if (a % i == 0) and (b % i == 0):
            gcd = i
    return gcd

def fx(x, c):
    return (x*x)+c

n = int(input("Enter a number: "))

for j in range(2, 4):
    x = y = j
    c = 1
    for i in range(1, 16):
        x = fx(x,c) % n
        y = fx(fx(y,c),c) % n

        g = gcd(abs(x-y), n)

        if g != 1 and g != n:
            print(n, "is not prime")
            exit(0)

print(n, "is prime")
```

Result:-



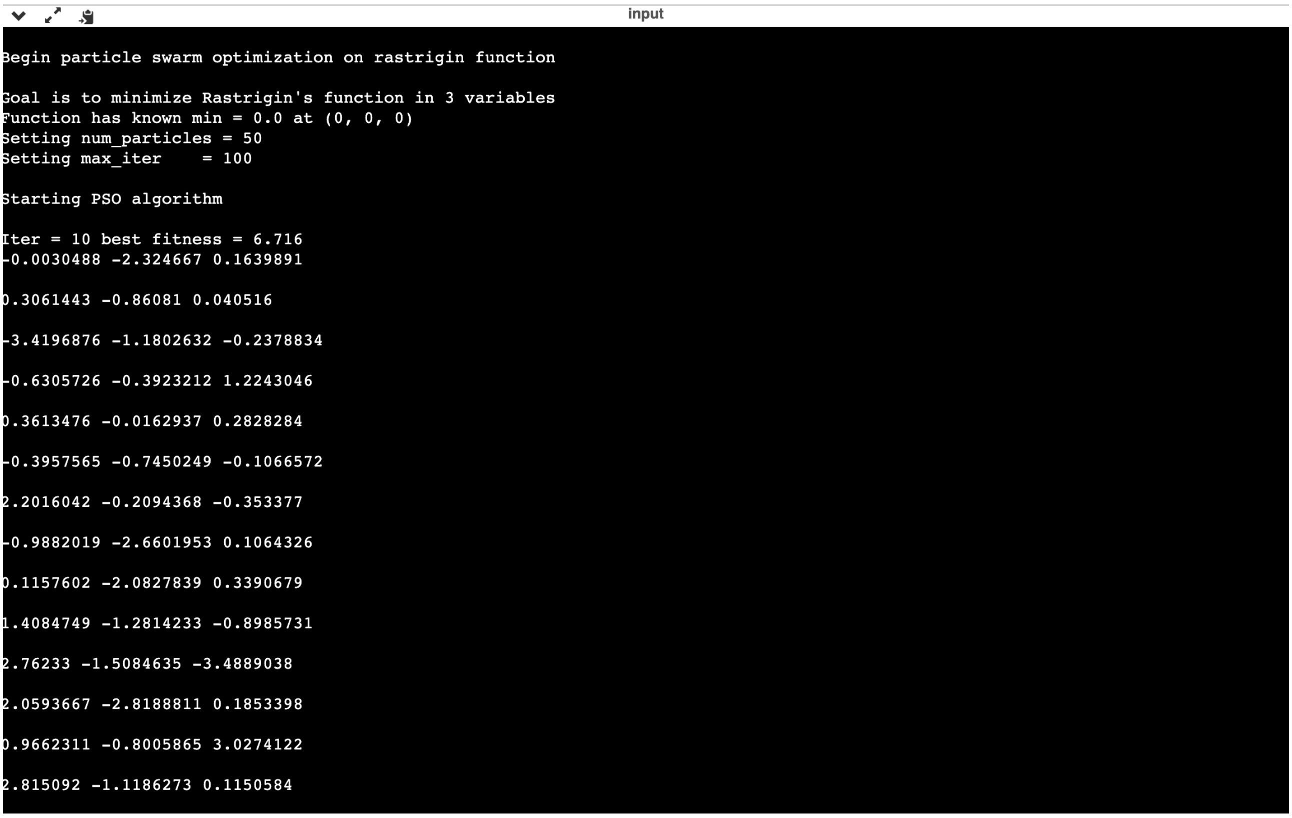
```
Enter a number: 29
29 is prime
>
```

Problem Statement:- Implement particle swarm optimization on rastrigin function.

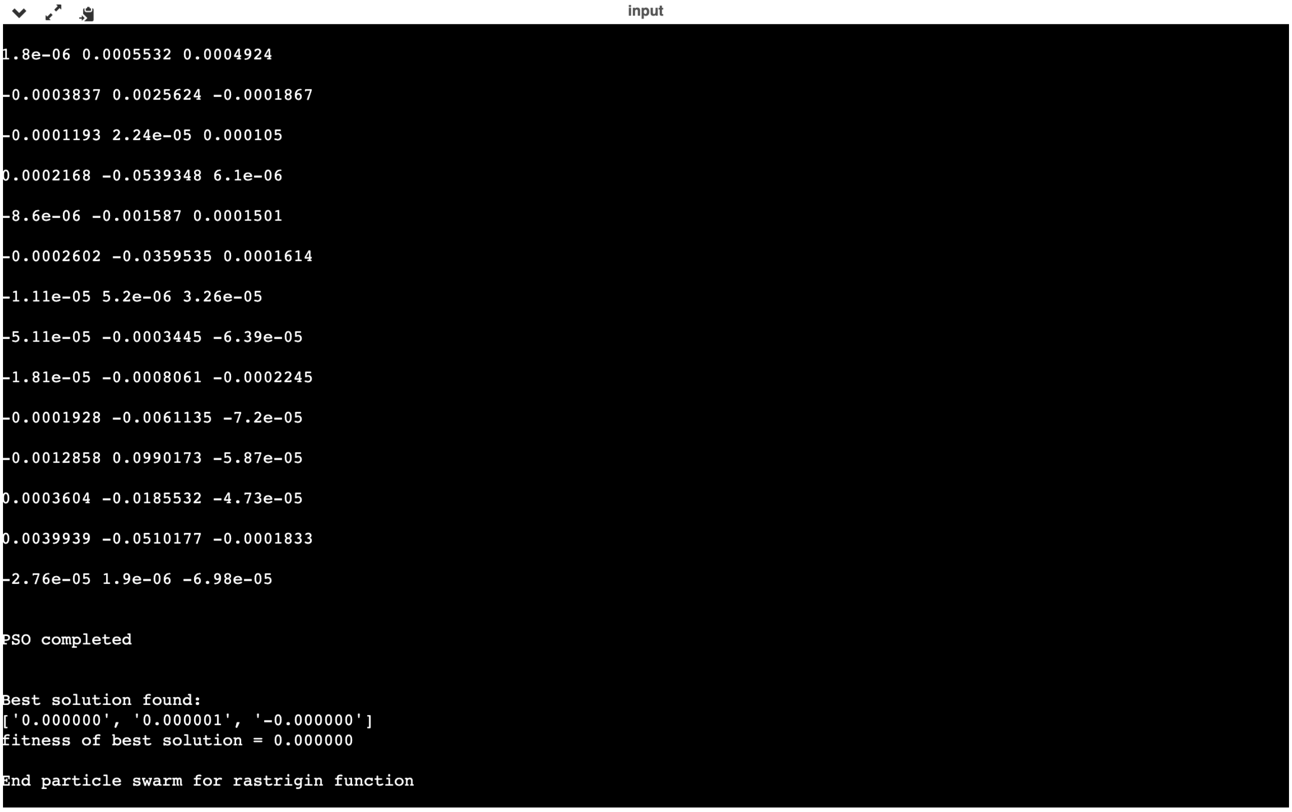
Code:-

```
import random
import math
import copy
import sys

def fitness_rastrigin(position):
  fitnessVal = 0.0
  for i in range(len(position)):
    xi = position[i]
    fitnessVal += (xi * xi) - (10 * math.cos(2 * math.pi * xi)) + 10
  return fitnessVal

class Particle:
  def __init__(self, fitness, dim, minx, maxx, seed):
    self.rnd = random.Random(seed)
    self.position = [0.0 for i in range(dim)]

    self.velocity = [0.0 for i in range(dim)]

    self.best_part_pos = [0.0 for i in range(dim)]

    for i in range(dim):
      self.position[i] = ((maxx - minx) *
        self.rnd.random() + minx)
      self.velocity[i] = ((maxx - minx) *
        self.rnd.random() + minx)

    self.fitness = fitness(self.position) # curr fitness

    self.best_part_pos = copy.copy(self.position)
    self.best_part_fitnessVal = self.fitness # best fitness

def pso(fitness, max_iter, n, dim, minx, maxx):
  w = 0.59
  c1 = 1.49445
  c2 = 1.49445
  rnd = random.Random(0)

  swarm = [Particle(fitness, dim, minx, maxx, i) for i in range(n)]

  best_swarm_pos = [0.0 for i in range(dim)]
  best_swarm_fitnessVal = sys.float_info.max

  for i in range(n):
    if swarm[i].fitness < best_swarm_fitnessVal:
      best_swarm_fitnessVal = swarm[i].fitness
      best_swarm_pos = copy.copy(swarm[i].position)

  Iter = 0
  while Iter < max_iter:

    if Iter % 10 == 0 and Iter > 1:
      print("Iter = " + str(Iter) + " best fitness = %.3f" % best_swarm_fitnessVal)
      for i in range(n):
        for j in range(3):
            print(round(swarm[i].position[j],7),end=" ");
        print("\n")
    for i in range(n):

      for k in range(dim):
        r1 = rnd.random()
        r2 = rnd.random()

        swarm[i].velocity[k] = (
                                  (w * swarm[i].velocity[k]) +
                                                               (c1  *  r1  *  (swarm[i].best_part_pos[k]  -
swarm[i].position[k])) +
                                  (c2 * r2 * (best_swarm_pos[k] -swarm[i].position[k]))
                                )


        if swarm[i].velocity[k] < minx:
          swarm[i].velocity[k] = minx
        elif swarm[i].velocity[k] > maxx:
          swarm[i].velocity[k] = maxx


      for k in range(dim):
        swarm[i].position[k] += swarm[i].velocity[k]

      swarm[i].fitness = fitness(swarm[i].position)

      if swarm[i].fitness < swarm[i].best_part_fitnessVal:
        swarm[i].best_part_fitnessVal = swarm[i].fitness
        swarm[i].best_part_pos = copy.copy(swarm[i].position)

      if swarm[i].fitness < best_swarm_fitnessVal:
        best_swarm_fitnessVal = swarm[i].fitness
        best_swarm_pos = copy.copy(swarm[i].position)

    Iter += 1
  return best_swarm_pos

print("\nBegin particle swarm optimization on rastrigin function\n")
dim = 3
fitness = fitness_rastrigin


print("Goal is to minimize Rastrigin's function in " + str(dim) + " variables")
print("Function has known min = 0.0 at (", end="")
for i in range(dim-1):
  print("0, ", end="")
print("0)")

num_particles = 50
max_iter = 100

print("Setting num_particles = " + str(num_particles))
print("Setting max_iter    = " + str(max_iter))
print("\nStarting PSO algorithm\n")

best_position = pso(fitness, max_iter, num_particles, dim, -10.0, 10.0)

print("\nPSO completed\n")
print("\nBest solution found:")
print(["%.6f"%best_position[k] for k in range(dim)])
fitnessVal = fitness(best_position)
print("fitness of best solution = %.6f" % fitnessVal)

print("\nEnd particle swarm for rastrigin function\n")
```

Result:-



```
Begin particle swarm optimization on rastrigin function

Goal is to minimize Rastrigin's function in 3 variables
Function has known min = 0.0 at (0, 0, 0)
Setting num_particles = 50
Setting max_iter    = 100

Starting PSO algorithm

Iter = 10 best fitness = 6.716
-0.0030488 -2.324667 0.1639891

0.3061443 -0.86081 0.040516

-3.4196876 -1.1802632 -0.2378834

-0.6305726 -0.3923212 1.2243046

0.3613476 -0.0162937 0.2828284

-0.3957565 -0.7450249 -0.1066572

2.2016042 -0.2094368 -0.353377

-0.9882019 -2.6601953 0.1064326

0.1157602 -2.0827839 0.3390679

1.4084749 -1.2814233 -0.8985731

2.76233 -1.5084635 -3.4889038

2.0593667 -2.8188811 0.1853398

0.9662311 -0.8005865 3.0274122

2.815092 -1.1186273 0.1150584
```



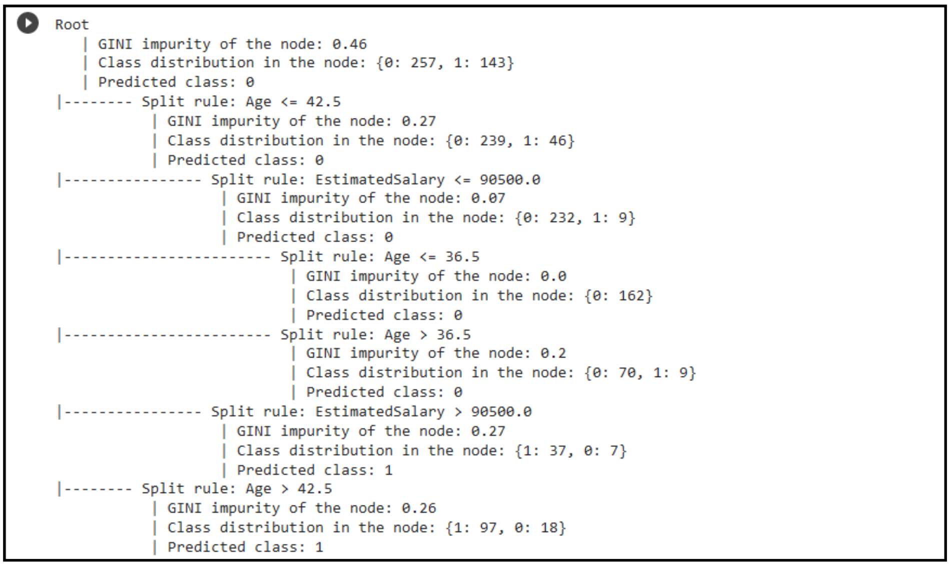
```
1.8e-06 0.0005532 0.0004924

-0.0003837 0.0025624 -0.0001867

-0.0001193 2.24e-05 0.000105

0.0002168 -0.0539348 6.1e-06

-8.6e-06 -0.001587 0.0001501

-0.0002602 -0.0359535 0.0001614

-1.11e-05 5.2e-06 3.26e-05

-5.11e-05 -0.0003445 -6.39e-05

-1.81e-05 -0.0008061 -0.0002245

-0.0001928 -0.0061135 -7.2e-05

-0.0012858 0.0990173 -5.87e-05

0.0003604 -0.0185532 -4.73e-05

0.0039939 -0.0510177 -0.0001833

-2.76e-05 1.9e-06 -6.98e-05


PSO completed


Best solution found:
['0.000000', '0.000001', '-0.000000']
fitness of best solution = 0.000000

End particle swarm for rastrigin function
```

Problem Statement:- Analyse the given dataset using Decision Tree Algorithm.

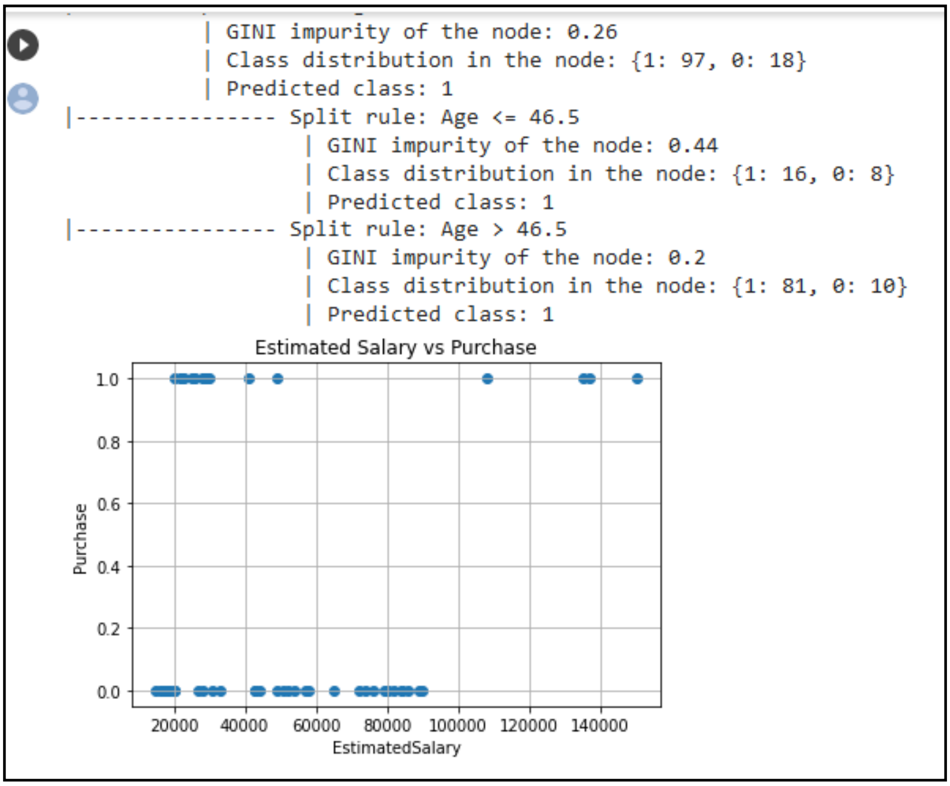
Code:-

```
from google.colab import files
uploaded = files.upload()
import pandas as pd
data = pd.read_csv('User_Data.csv')
data.head()
from sklearn import preprocessing
label_encoder = preprocessing.LabelEncoder()
import pandas as pd
import numpy as np
from collections import Counter
import matplotlib.pyplot as plt

class Node:

    def __init__(
        self,
        Y: list,
        X: pd.DataFrame,
        min_samples_split=None,
        max_depth=None,
        depth=None,
        node_type=None,
        rule=None
    ):
        self.Y = Y
        self.X = X

        self.min_samples_split = min_samples_split if min_samples_split else 20
        self.max_depth = max_depth if max_depth else 5

        self.depth = depth if depth else 0
        self.features = list(self.X.columns)
        self.node_type = node_type if node_type else 'root'
        self.rule = rule if rule else ""
        self.counts = Counter(Y)
        self.gini_impurity = self.get_GINI()
        counts_sorted = list(sorted(self.counts.items(), key=lambda item: item[1]))
        yhat = None
        if len(counts_sorted) > 0:
            yhat = counts_sorted[-1][0]
        self.yhat = yhat
        self.n = len(Y)
        self.left = None
        self.right = None
        self.best_feature = None
        self.best_value = None

    @staticmethod
    def GINI_impurity(y1_count: int, y2_count: int) -> float:
        if y1_count is None:
            y1_count = 0

        if y2_count is None:
            y2_count = 0

        n = y1_count + y2_count

        if n == 0:
            return 0.0
        p1 = y1_count / n
        p2 = y2_count / n
        gini = 1 - (p1 ** 2 + p2 ** 2)
        return gini

    @staticmethod
    def ma(x: np.array, window: int) -> np.array:
        return np.convolve(x, np.ones(window), 'valid') / window

    def get_GINI(self):
        y1_count, y2_count = self.counts.get(0, 0), self.counts.get(1, 0)
        return self.GINI_impurity(y1_count, y2_count)

    def best_split(self) -> tuple:
        df = self.X.copy()
        df['Y'] = self.Y
        GINI_base = self.get_GINI()
        max_gain = 0

        best_feature = None
        best_value = None
        for feature in self.features:
            Xdf = df.dropna().sort_values(feature)
            xmeans = self.ma(Xdf[feature].unique(), 2)

            for value in xmeans:
                left_counts = Counter(Xdf[Xdf[feature]<value]['Y'])
                right_counts = Counter(Xdf[Xdf[feature]>=value]['Y'])
                y0_left, y1_left, y0_right, y1_right = left_counts.get(0, 0),
left_counts.get(1, 0), right_counts.get(0, 0), right_counts.get(1, 0)
                gini_left = self.GINI_impurity(y0_left, y1_left)
                gini_right = self.GINI_impurity(y0_right, y1_right)
                n_left = y0_left + y1_left
                n_right = y0_right + y1_right
                w_left = n_left / (n_left + n_right)
                w_right = n_right / (n_left + n_right)
                wGINI = w_left * gini_left + w_right * gini_right
                GINIgain = GINI_base - wGINI
                if GINIgain > max_gain:
                    best_feature = feature
                    best_value = value
                    max_gain = GINIgain

        return (best_feature, best_value)

    def grow_tree(self):
        df = self.X.copy()
        df['Y'] = self.Y
        if (self.depth < self.max_depth) and (self.n >= self.min_samples_split):
            best_feature, best_value = self.best_split()

            if best_feature is not None:
                self.best_feature = best_feature
                self.best_value = best_value
                left_df, right_df = df[df[best_feature]<=best_value].copy(),
df[df[best_feature]>best_value].copy()
                left = Node(
                    left_df['Y'].values.tolist(),
                    left_df[self.features],
                    depth=self.depth + 1,
                    max_depth=self.max_depth,
                    min_samples_split=self.min_samples_split,
                    node_type='left_node',
                    rule=f"{best_feature} <= {round(best_value, 3)}"
                    )

                self.left = left
                self.left.grow_tree()

                right = Node(
                    right_df['Y'].values.tolist(),
                    right_df[self.features],
                    depth=self.depth + 1,
                    max_depth=self.max_depth,
                    min_samples_split=self.min_samples_split,
                    node_type='right_node',
                    rule=f"{best_feature} > {round(best_value, 3)}"
                    )

                self.right = right
                self.right.grow_tree()

    def print_info(self, width=4):
        const = int(self.depth * width ** 1.5)
        spaces = "-" * const

        if self.node_type == 'root':
            print("Root")
        else:
            print(f"|{spaces} Split rule: {self.rule}")
                               print(f"{'  '  *  const}         |  GINI  impurity  of  the  node:
{round(self.gini_impurity, 2)}")
        print(f"{' ' * const}   | Class distribution in the node: {dict(self.counts)}")
        print(f"{' ' * const}   | Predicted class: {self.yhat}")

    def print_tree(self):
        self.print_info()
        if self.left is not None:
            self.left.print_tree()

        if self.right is not None:
            self.right.print_tree()

    def predict(self, X:pd.DataFrame):
        predictions = []

        for _, x in X.iterrows():
            values = {}
            for feature in self.features:
                values.update({feature: x[feature]})

            predictions.append(self.predict_obs(values))

        return predictions

    def predict_obs(self, values: dict) -> int:
        cur_node = self
        while cur_node.depth < cur_node.max_depth:
            best_feature = cur_node.best_feature
            best_value = cur_node.best_value

            if cur_node.n < cur_node.min_samples_split:
                break

            if (values.get(best_feature) < best_value):
                if self.left is not None:
                    cur_node = cur_node.left
            else:
                if self.right is not None:
                    cur_node = cur_node.right

        return cur_node.yhat

if __name__ == '__main__':
            d = pd.read_csv("User_Data.csv")[['Age', 'EstimatedSalary', 'Gender', 
'Purchased']].dropna()
    d['Gender'] = label_encoder.fit_transform(d['Gender'])
    X = d[['Age', 'EstimatedSalary', 'Gender']]
    Y = d['Purchased'].values.tolist()
    root = Node(Y, X, max_depth=3, min_samples_split=100)
    root.grow_tree()
    root.print_tree()
    Xsubset = X.copy()
    Xsubset['yhat'] = root.predict(Xsubset)
    plt.scatter(Xsubset['EstimatedSalary'][:50],Xsubset['yhat'][:50])
    plt.grid(True)
    plt.xlabel('EstimatedSalary')
    plt.ylabel('Purchase')
    plt.title('Estimated Salary vs Purchase')
```

Result:-



```
Root
   | GINI impurity of the node: 0.46
   | Class distribution in the node: {0: 257, 1: 143}
   | Predicted class: 0
|-------- Split rule: Age <= 42.5
           | GINI impurity of the node: 0.27
           | Class distribution in the node: {0: 239, 1: 46}
           | Predicted class: 0
|--------------- Split rule: EstimatedSalary <= 90500.0
                  | GINI impurity of the node: 0.07
                  | Class distribution in the node: {0: 232, 1: 9}
                  | Predicted class: 0
|---------------------- Split rule: Age <= 36.5
                         | GINI impurity of the node: 0.0
                         | Class distribution in the node: {0: 162}
                         | Predicted class: 0
|---------------------- Split rule: Age > 36.5
                         | GINI impurity of the node: 0.2
                         | Class distribution in the node: {0: 70, 1: 9}
                         | Predicted class: 0
|--------------- Split rule: EstimatedSalary > 90500.0
                  | GINI impurity of the node: 0.27
                  | Class distribution in the node: {1: 37, 0: 7}
                  | Predicted class: 1
|-------- Split rule: Age > 42.5
           | GINI impurity of the node: 0.26
           | Class distribution in the node: {1: 97, 0: 18}
           | Predicted class: 1
```



```
           | GINI impurity of the node: 0.26
           | Class distribution in the node: {1: 97, 0: 18}
           | Predicted class: 1
|--------------- Split rule: Age <= 46.5
                  | GINI impurity of the node: 0.44
                  | Class distribution in the node: {1: 16, 0: 8}
                  | Predicted class: 1
|--------------- Split rule: Age > 46.5
                  | GINI impurity of the node: 0.2
                  | Class distribution in the node: {1: 81, 0: 10}
                  | Predicted class: 1
```

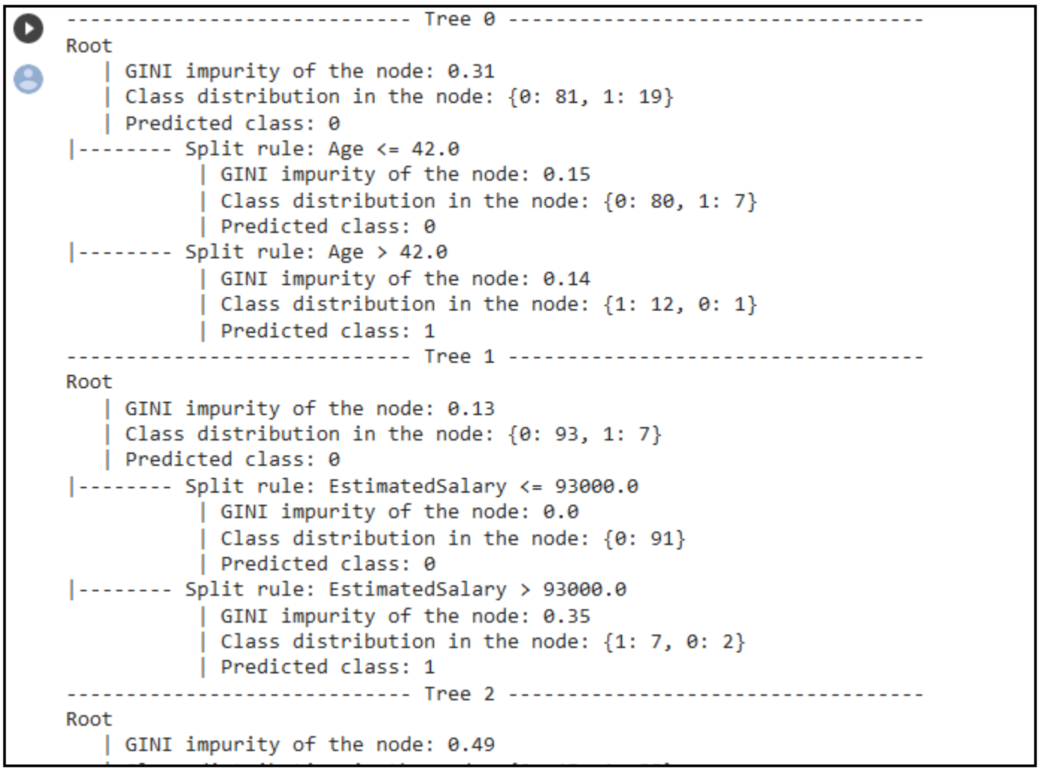
Problem Statement:- Analyse the given dataset using Naive Bayes Algorithm.

Code:-

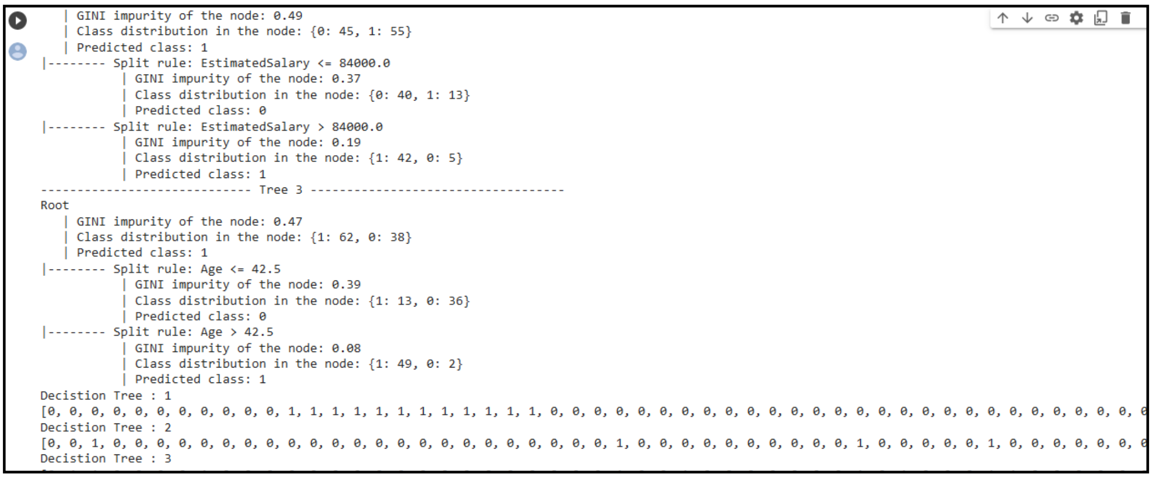
```
from google.colab import files
uploaded = files.upload()
import pandas as pd
data = pd.read_csv('User_Data.csv')
from sklearn import preprocessing
label_encoder = preprocessing.LabelEncoder()
import pandas as pd
import numpy as np
from collections import Counter

class Node:

    def __init__(
        self,
        Y: list,
        X: pd.DataFrame,
        min_samples_split=None,
        max_depth=None,
        depth=None,
        node_type=None,
        rule=None
    ):
        self.Y = Y
        self.X = X

        self.min_samples_split = min_samples_split if min_samples_split else 20
        self.max_depth = max_depth if max_depth else 5

        self.depth = depth if depth else 0
        self.features = list(self.X.columns)
        self.node_type = node_type if node_type else 'root'
        self.rule = rule if rule else ""
        self.counts = Counter(Y)
        self.gini_impurity = self.get_GINI()
        counts_sorted = list(sorted(self.counts.items(), key=lambda item: item[1]))
        yhat = None
        if len(counts_sorted) > 0:
            yhat = counts_sorted[-1][0]
        self.yhat = yhat
        self.n = len(Y)
        self.left = None
        self.right = None
        self.best_feature = None
        self.best_value = None

    @staticmethod
    def GINI_impurity(y1_count: int, y2_count: int) -> float:
        if y1_count is None:
            y1_count = 0

        if y2_count is None:
            y2_count = 0

        n = y1_count + y2_count

        if n == 0:
            return 0.0
        p1 = y1_count / n
        p2 = y2_count / n
        gini = 1 - (p1 ** 2 + p2 ** 2)
        return gini

    @staticmethod
    def ma(x: np.array, window: int) -> np.array:
        return np.convolve(x, np.ones(window), 'valid') / window

    def get_GINI(self):
        y1_count, y2_count = self.counts.get(0, 0), self.counts.get(1, 0)
        return self.GINI_impurity(y1_count, y2_count)

    def best_split(self) -> tuple:
        df = self.X.copy()
        df['Y'] = self.Y
        GINI_base = self.get_GINI()
        max_gain = 0

        best_feature = None
        best_value = None

        for feature in self.features:
            Xdf = df.dropna().sort_values(feature)
            xmeans = self.ma(Xdf[feature].unique(), 2)

            for value in xmeans:
                left_counts = Counter(Xdf[Xdf[feature]<value]['Y'])
                right_counts = Counter(Xdf[Xdf[feature]>=value]['Y'])
                y0_left, y1_left, y0_right, y1_right = left_counts.get(0, 0),
left_counts.get(1, 0), right_counts.get(0, 0), right_counts.get(1, 0)
                gini_left = self.GINI_impurity(y0_left, y1_left)
                gini_right = self.GINI_impurity(y0_right, y1_right)
                n_left = y0_left + y1_left
                n_right = y0_right + y1_right
                w_left = n_left / (n_left + n_right)
                w_right = n_right / (n_left + n_right)
                wGINI = w_left * gini_left + w_right * gini_right
                GINIgain = GINI_base - wGINI
                if GINIgain > max_gain:
                    best_feature = feature
                    best_value = value
                    max_gain = GINIgain

        return (best_feature, best_value)

    def grow_tree(self):
        df = self.X.copy()
        df['Y'] = self.Y
        if (self.depth < self.max_depth) and (self.n >= self.min_samples_split):
            best_feature, best_value = self.best_split()

            if best_feature is not None:
                self.best_feature = best_feature
                self.best_value = best_value
                left_df, right_df = df[df[best_feature]<=best_value].copy(),
df[df[best_feature]>best_value].copy()
                left = Node(
                    left_df['Y'].values.tolist(),
                    left_df[self.features],
                    depth=self.depth + 1,
                    max_depth=self.max_depth,
                    min_samples_split=self.min_samples_split,
                    node_type='left_node',
                    rule=f"{best_feature} <= {round(best_value, 3)}"
                    )
                self.left = left
                self.left.grow_tree()

                right = Node(
                    right_df['Y'].values.tolist(),
                    right_df[self.features],
                    depth=self.depth + 1,
                    max_depth=self.max_depth,
                    min_samples_split=self.min_samples_split,
                    node_type='right_node',
                    rule=f"{best_feature} > {round(best_value, 3)}"
                    )

                self.right = right
                self.right.grow_tree()

    def print_info(self, width=4):
        const = int(self.depth * width ** 1.5)
        spaces = "-" * const

        if self.node_type == 'root':
            print("Root")
        else:
            print(f"|{spaces} Split rule: {self.rule}")
                             print(f"{'  '  *  const}         |  GINI  impurity  of  the  node:
{round(self.gini_impurity, 2)}")
        print(f"{' ' * const}   | Class distribution in the node: {dict(self.counts)}")
        print(f"{' ' * const}   | Predicted class: {self.yhat}")

    def print_tree(self):
        self.print_info()
        if self.left is not None:
            self.left.print_tree()

        if self.right is not None:
            self.right.print_tree()

    def predict(self, X:pd.DataFrame):
        predictions = []

        for _, x in X.iterrows():
            values = {}
            for feature in self.features:
                values.update({feature: x[feature]})

            predictions.append(self.predict_obs(values))

        return predictions

    def predict_obs(self, values: dict) -> int:
        cur_node = self
        while cur_node.depth < cur_node.max_depth:
            best_feature = cur_node.best_feature
            best_value = cur_node.best_value

            if cur_node.n < cur_node.min_samples_split:
                break

            if (values.get(best_feature) < best_value):
                if self.left is not None:
                    cur_node = cur_node.left
            else:
                if self.right is not None:
                    cur_node = cur_node.right

        return cur_node.yhat

if __name__ == '__main__':
            d = pd.read_csv("User_Data.csv")[['Age', 'EstimatedSalary', 'Gender',
'Purchased']].dropna()
    d['Gender'] = label_encoder.fit_transform(d['Gender'])
    X = d[['Age', 'EstimatedSalary', 'Gender']]
    Y = d['Purchased'].values.tolist()
    # print(X.count())
    # print(len(Y))
    dt_lst = []
    for i in range(4):
      x_temp = X[100 * i : 100 * (i + 1)]
      y_temp = Y[100 * i : 100 * (i + 1)]
                    new_root_node = Node(y_temp, x_temp, max_depth = 3,
min_samples_split = 100)
      new_root_node.grow_tree()
      print(f"---------------------------- Tree {i} ----------------------------------")
```

```
  new_root_node.print_tree()
  dt_lst.append(new_root_node)
Xsubset = X.copy()
res_list = []
for j in range(4):
  val_res = dt_lst[j].predict(Xsubset.iloc[5 : 100, :])
  res_list.append(val_res)
_it = 0
for ele in res_list:
  _it += 1
  print(f"Decistion Tree : {_it}", sep = '\t')
  print(ele)
print("================== final result ===================")
res = []
for i in range(len(res_list[0])):
  ele = [res_list[j][i] for j in range(4)]
  e = 0 if ele.count(0) > ele.count(1) else 1
  res.append(e)
print(res)
print("=================== actual values =================")
print(Y[5:100])
```

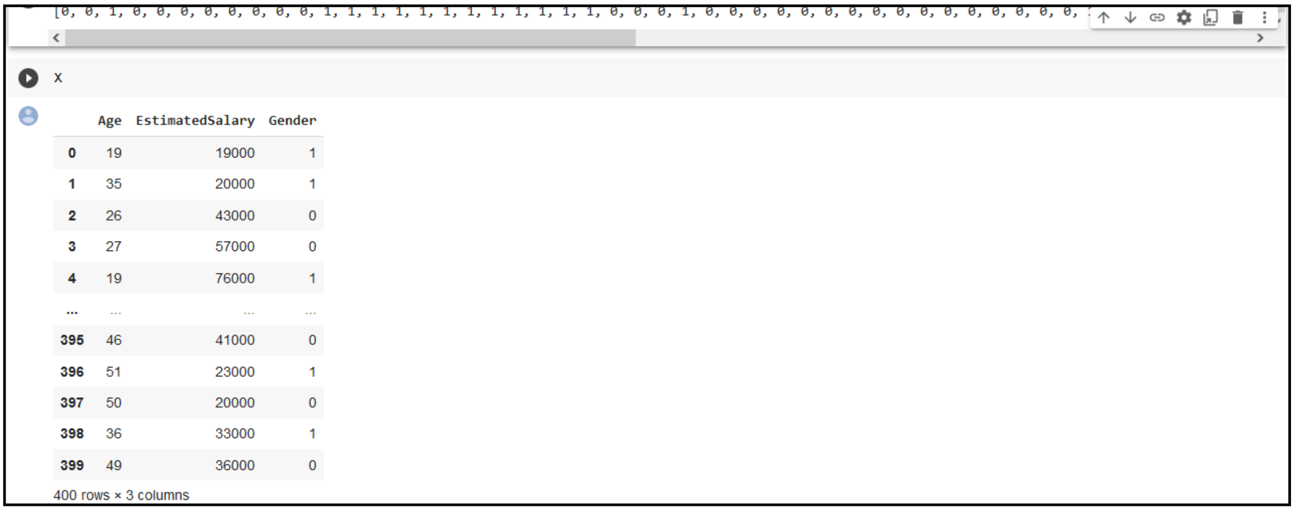
Result:-



```
------------------------------ Tree 0 ------------------------------------
Root
   | GINI impurity of the node: 0.31
   | Class distribution in the node: {0: 81, 1: 19}
   | Predicted class: 0
|-------- Split rule: Age <= 42.0
           | GINI impurity of the node: 0.15
           | Class distribution in the node: {0: 80, 1: 7}
           | Predicted class: 0
|-------- Split rule: Age > 42.0
           | GINI impurity of the node: 0.14
           | Class distribution in the node: {1: 12, 0: 1}
           | Predicted class: 1
------------------------------ Tree 1 ------------------------------------
Root
   | GINI impurity of the node: 0.13
   | Class distribution in the node: {0: 93, 1: 7}
   | Predicted class: 0
|-------- Split rule: EstimatedSalary <= 93000.0
           | GINI impurity of the node: 0.0
           | Class distribution in the node: {0: 91}
           | Predicted class: 0
|-------- Split rule: EstimatedSalary > 93000.0
           | GINI impurity of the node: 0.35
           | Class distribution in the node: {1: 7, 0: 2}
           | Predicted class: 1
------------------------------ Tree 2 ------------------------------------
Root
   | GINI impurity of the node: 0.49
```



```
   | GINI impurity of the node: 0.49
   | Class distribution in the node: {0: 45, 1: 55}
   | Predicted class: 1
|-------- Split rule: EstimatedSalary <= 84000.0
           | GINI impurity of the node: 0.37
           | Class distribution in the node: {0: 40, 1: 13}
           | Predicted class: 0
|-------- Split rule: EstimatedSalary > 84000.0
           | GINI impurity of the node: 0.19
           | Class distribution in the node: {1: 42, 0: 5}
           | Predicted class: 1
------------------------------ Tree 3 ------------------------------------
Root
   | GINI impurity of the node: 0.47
   | Class distribution in the node: {1: 62, 0: 38}
   | Predicted class: 1
|-------- Split rule: Age <= 42.5
           | GINI impurity of the node: 0.39
           | Class distribution in the node: {1: 13, 0: 36}
           | Predicted class: 0
|-------- Split rule: Age > 42.5
           | GINI impurity of the node: 0.08
           | Class distribution in the node: {1: 49, 0: 2}
           | Predicted class: 1
Decistion Tree : 1
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
Decistion Tree : 2
[0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0
Decistion Tree : 3
```



```
[0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
X
```

	Age	EstimatedSalary	Gender
0	19	19000	1
1	35	20000	1
2	26	43000	0
3	27	57000	0
4	19	76000	1
...	...	...	...
395	46	41000	0
396	51	23000	1
397	50	20000	0
398	36	33000	1
399	49	36000	0

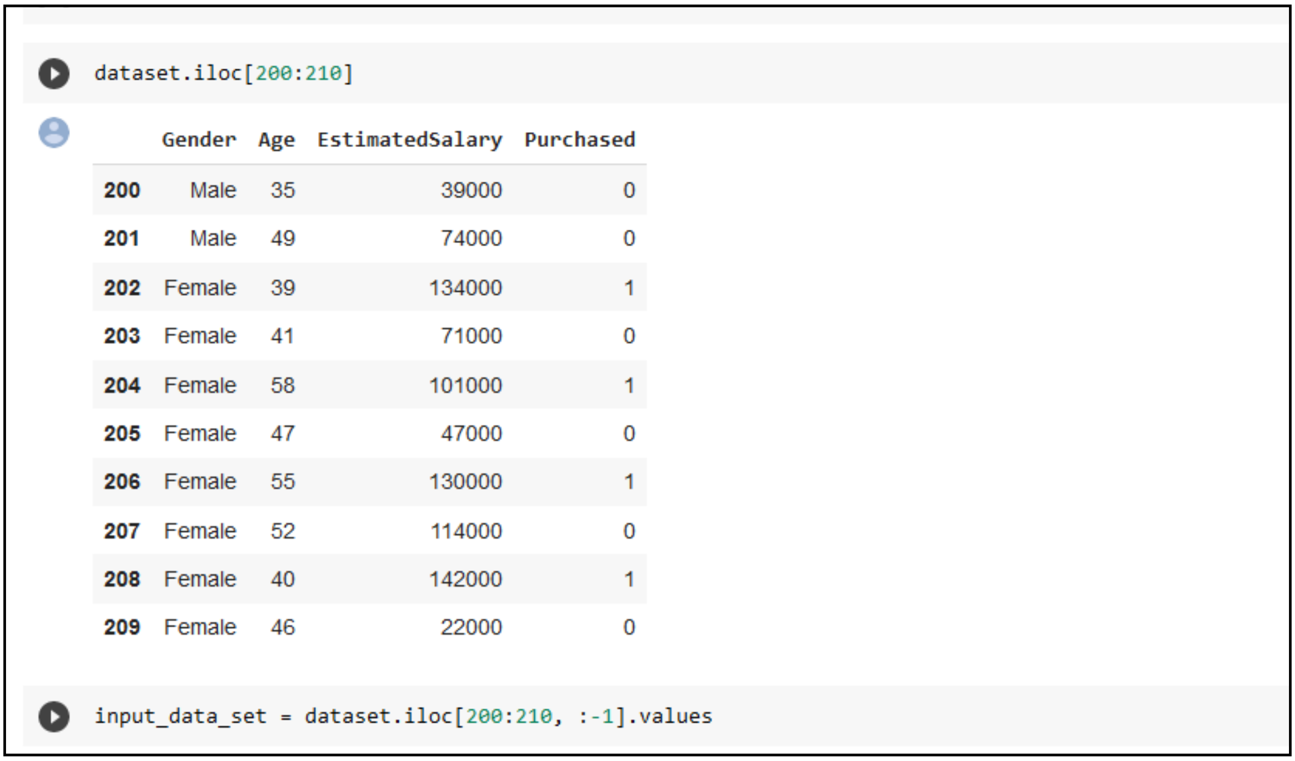
400 rows × 3 columns

Problem Statement:- Analyse the given dataset using Random Forest Algorithm.

Code:-

```
from google.colab import files
uploaded = files.upload()
import pandas as pd
import numpy as np
data = pd.read_csv('User_Data.csv')
data.head()
data.columns
dataset = data[['Gender', 'Age', 'EstimatedSalary', 'Purchased']]
dataset
dataset.columns
dataset_details = {}
purchase_count = len(dataset['Purchased'] == 1)
no_purchase_count = len(dataset['Purchased'] == 0)

for attr in dataset.columns:
  dataset_details[attr] = {}
  unique_lst = pd.unique(dataset[attr])
  total_count = len(dataset[attr])
  attr_lst = dataset[attr].values
  for a in unique_lst:
    a_count = list(attr_lst).count(a)
    df_1 = dataset[dataset[attr] == a]
    df_2 = df_1[dataset['Purchased'] == 0]
    df_3 = df_1[dataset['Purchased'] == 1]
    if type(a) != 'str':
      a = str(a)
    dataset_details[attr][a] = a_count / len(dataset['Purchased'])
    dataset_details[attr][a + "/1"] = len(df_3) / purchase_count
    dataset_details[attr][a + "/0"] = len(df_2)/ no_purchase_count
def naiveBayes(data_lst, attr_lst, dataset_details, outcome_lst):
  result = []
  for val in data_lst:
    ## for no
    prob = 0
    res = 0
    temp_sum = dataset_details["Purchased"]['0']
    for i, j in enumerate(attr_lst):
      temp_sum *= dataset_details[j][str(val[i]) + "/0"]
    res = 0
    prob = temp_sum
    temp_sum_1 = dataset_details["Purchased"]['0']
    for i, j in enumerate(attr_lst):
      temp_sum_1 *= dataset_details[j][str(val[i]) + "/1"]
    if temp_sum_1 > prob:
      res =  1
      prob = temp_sum_1
    result.append({'res' : res, 'prob' : prob / (temp_sum + temp_sum_1)})
  return result
columndata = list(dataset.columns[:-1])
outcome_lst = 'Purchased'
dataset.iloc[200:210]
input_data_set = dataset.iloc[200:210, :-1].values
result = naiveBayes(input_data_set, columndata, dataset_details, outcome_lst)
for res in result:
  print(res)
print("============================================= DataSet
Details ============================")
print(dataset_details)
print("=====================================================
==================================")
```

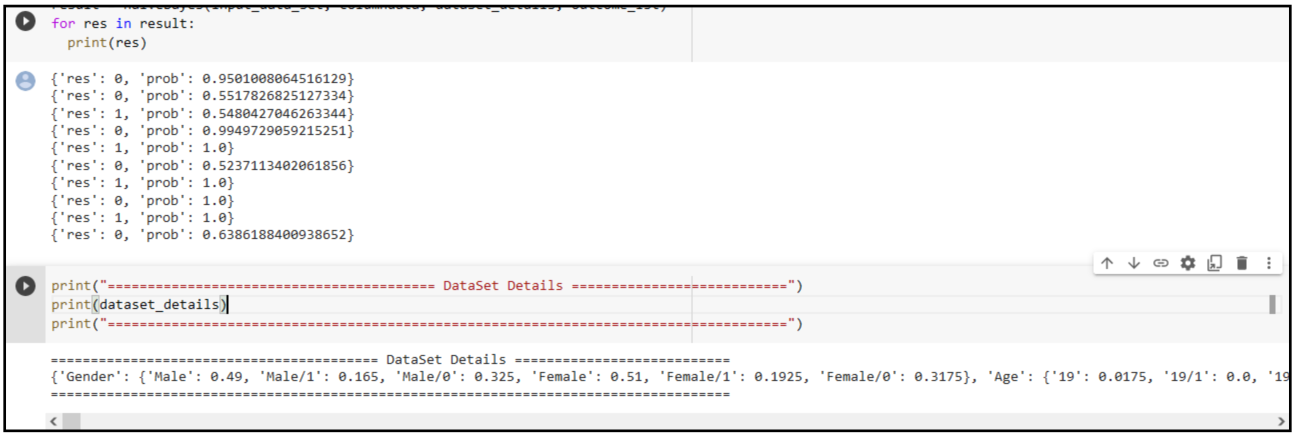
Result:-



```
dataset.iloc[200:210]
```

	Gender	Age	EstimatedSalary	Purchased
200	Male	35	39000	0
201	Male	49	74000	0
202	Female	39	134000	1
203	Female	41	71000	0
204	Female	58	101000	1
205	Female	47	47000	0
206	Female	55	130000	1
207	Female	52	114000	0
208	Female	40	142000	1
209	Female	46	22000	0

```
input_data_set = dataset.iloc[200:210, :-1].values
```



```
for res in result:
  print(res)
```

```
{'res': 0, 'prob': 0.9501008064516129}
{'res': 0, 'prob': 0.5517826825127334}
{'res': 1, 'prob': 0.5480427046263344}
{'res': 0, 'prob': 0.9949729059215251}
{'res': 1, 'prob': 1.0}
{'res': 0, 'prob': 0.5237113402061856}
{'res': 1, 'prob': 1.0}
{'res': 0, 'prob': 1.0}
{'res': 1, 'prob': 1.0}
{'res': 0, 'prob': 0.6386188400938652}
```

```
print("========================================= DataSet Details ===========================")
print(dataset_details)
print("===================================================================================")
```

```
========================================= DataSet Details ===========================
{'Gender': {'Male': 0.49, 'Male/1': 0.165, 'Male/0': 0.325, 'Female': 0.51, 'Female/1': 0.1925, 'Female/0': 0.3175}, 'Age': {'19': 0.0175, '19/1': 0.0, '19
===================================================================================
```

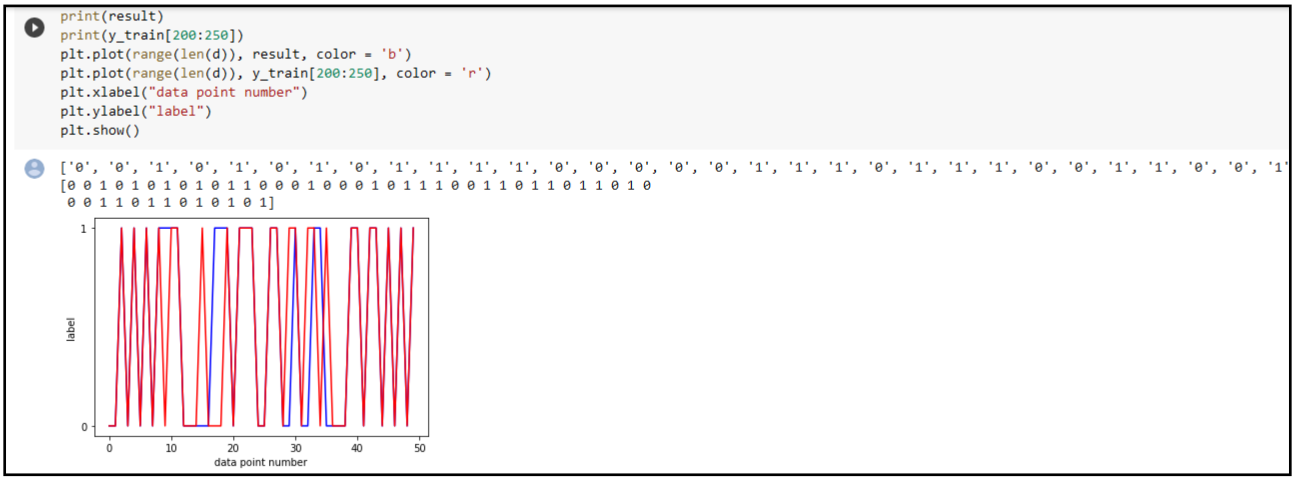
Problem Statement:- Analyse the given dataset using KNN Algorithm.

Code:-

```
from google.colab import files
uploaded = files.upload()
import pandas as pd
import numpy as np
dataset = pd.read_csv('User_Data.csv')
dataset.head()
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
dataset['Gender'] = label_encoder.fit_transform(dataset['Gender'])
dataset.head()
dataset.count()
x_train, y_train = dataset.iloc[:300, 1:-1].values, dataset.iloc[:300, -1].values
x_train[0]
x_train = np.array(x_train).astype('float32')
x_train -= np.mean(x_train)
x_train /= np.std(x_train)
def get_norm(dataset : any, dataset_target : any, datapoint : any, p : int,k : int) -> any:
  distance_arry = []
  for i,data in enumerate(dataset):
    dist = 0
    for j, ele in enumerate(data):
      dist += (datapoint[j] - ele) ** p
    dist **= 1/p
    if len(distance_arry)  < k:
      distance_arry.append([dist, dataset_target[i]])
      continue
    max_index = 0
    max_value = -1
    for m, ele in enumerate(distance_arry):
      if distance_arry[m][0] > max_value:
        max_value = distance_arry[m][0]
        max_index = m
    if max_value > dist:
      # print(distance_arry)
      # print("== i ==", i , "== dataset - target -, ", dataset_target[i])
      distance_arry[max_index] = [dist, dataset_target[i]]
  return distance_arry
y_train[0]
```

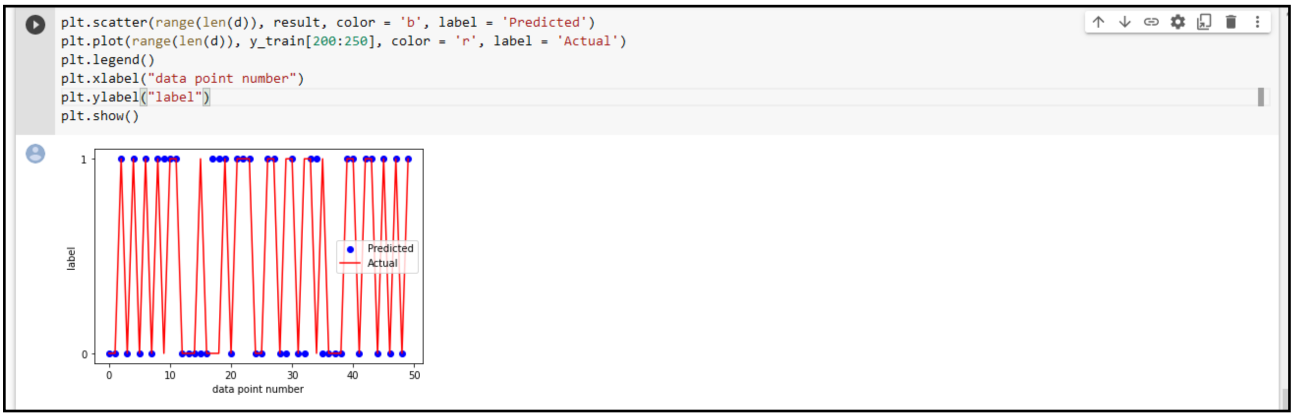
```
result_dist = get_norm(x_train, y_train, x_train[20], 2, 4)
def find_result(result_dist : any) -> int:
  dict_count = {}
  res = -1
  max_value = 0
  for i in result_dist:
    i[1] = str(i[1])
    if i[1] not in result_dist:
      dict_count[i[1]] = 1
    else:
      dict_count[i[1]] += 1
    if dict_count[i[1]] > max_value:
      res = i[1]
  return res
import matplotlib.pyplot as plt
d = x_train[200:250]
result = []
for i in d:
  result.append(find_result(get_norm(x_train, y_train, i, 2, 5)))
print(result)
print(y_train[200:250])
plt.plot(range(len(d)), result, color = 'b')
plt.plot(range(len(d)), y_train[200:250], color = 'r')
plt.xlabel("data point number")
plt.ylabel("label")
plt.show()
plt.scatter(range(len(d)), result, color = 'b', label = 'Predicted')
plt.plot(range(len(d)), y_train[200:250], color = 'r', label = 'Actual')
plt.legend()
plt.xlabel("data point number")
plt.ylabel("label")
plt.show()
```

Result:-



```
print(result)
print(y_train[200:250])
plt.plot(range(len(d)), result, color = 'b')
plt.plot(range(len(d)), y_train[200:250], color = 'r')
plt.xlabel("data point number")
plt.ylabel("label")
plt.show()
```

```
['0', '0', '1', '0', '1', '0', '1', '0', '1', '1', '1', '1', '0', '0', '0', '0', '0', '1', '1', '1', '0', '1', '1', '1', '0', '0', '1', '1', '0', '0', '1'
[0 0 1 0 1 0 1 0 1 0 1 1 0 0 0 1 0 0 0 1 0 1 1 1 0 0 1 1 0 1 1 0 1 1 0 1 0
 0 0 1 1 0 1 1 0 1 0 1 0 1]
```



```
plt.scatter(range(len(d)), result, color = 'b', label = 'Predicted')
plt.plot(range(len(d)), y_train[200:250], color = 'r', label = 'Actual')
plt.legend()
plt.xlabel("data point number")
plt.ylabel("label")
plt.show()
```

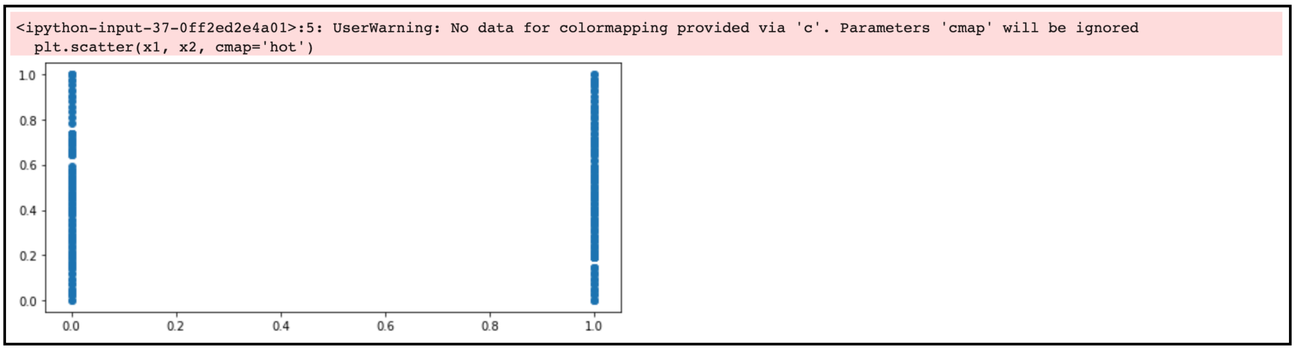
Problem Statement:- Analyse the given dataset using SVM Algorithm.

Code:-

```
import pandas as pd
df = pd.read_csv('User_Data.csv')
df

for column in df.columns:
  bool_series = pd.isnull(df[column])
  print()
  print(bool_series)

df = df.drop('User ID', axis=True)

normalized_df=(df['Age']-df['Age'].min())/(df['Age'].max()-df['Age'].min())

df = df.drop('Age', axis=True)

df = df.drop('EstimatedSalary', axis=True)

df['Age'] = normalized_df

for i in range(len(df)):
    if df.loc[i, "Gender"]=='Female':
      df.loc[i, "Gender"] = 1
    else:
      df.loc[i, "Gender"] = 0

x_points = []
for i in range(len(df)):
  temp = []
  temp.append(df.loc[i, "Gender"])
  temp.append(df.loc[i, "Age"])
  x_points.append(temp)

y_points = []
for i in range(len(df)):
  temp = []
  temp.append(df.loc[i, "Purchased"])
  y_points.append(temp)
```

```
x1 = []
for i in x_points:
  x1.append(i[0])
x2 = []
for i in x_points:
  x2.append(I[1])

import numpy as np
from matplotlib import pyplot as plt

# Scatter plot
plt.scatter(x1, x2, cmap='hot')

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

Result:-



```
<ipython-input-37-0ff2ed2e4a01>:5: UserWarning: No data for colormapping provided via 'c'. Parameters 'cmap' will be ignored
  plt.scatter(x1, x2, cmap='hot')
```

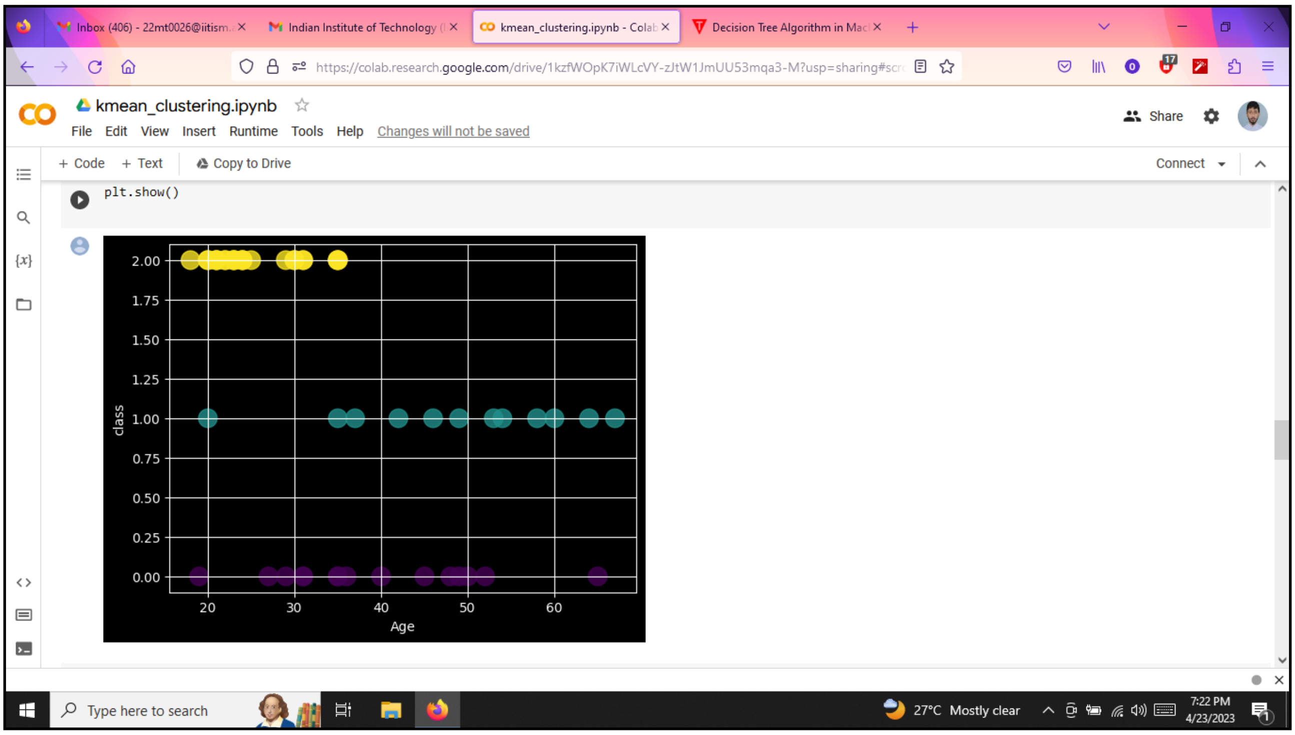
<u>Problem Statement</u>:- Analyse the given dataset using K-Means Algorithm.

<u>Code</u>:-

```
from google.colab import files
data = files.upload()
import pandas as pd
data = pd.read_csv('Mall_Customers.csv')
data.head()
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
data['Gender'] = le.fit_transform(data['Gender'])
data.head()
import numpy as np
def euclidean(point, data):
   return np.sqrt(np.sum((point - data)**2, axis=1))
import matplotlib.pyplot as plt

class KMeanClustering:

  def __init__(self, x_train, number_of_cluster):
      self.K = number_of_cluster
      self.max_iterations = 100
      self.num_examples, self.num_features = x_train.shape
      self.plot_figure = True

  def initialize_random_centroids(self, X):
      centroids = np.zeros((self.K, self.num_features))
      for k in range(self.K):
         centroid = X[np.random.choice(range(self.num_examples))]
         centroids[k] = centroid
      return centroids

  def create_cluster(self, X, centroids):
      clusters = [[] for _ in range(self.K)]
      for point_idx, point in enumerate(X):
         closest_centroid = np.argmin(
            np.sqrt(np.sum((point-centroids)**2, axis=1))
         )
         clusters[closest_centroid].append(point_idx)
      return clusters

  def calculate_new_centroids(self, cluster, X):
      centroids = np.zeros((self.K, self.num_features))
      for idx, cluster in enumerate(cluster):
          new_centroid = np.mean(X[cluster], axis=0)
          centroids[idx] = new_centroid
      return centroids

  def predict_cluster(self, clusters, X):
      y_pred = np.zeros(self.num_examples)
      for cluster_idx, cluster in enumerate(clusters):
          for sample_idx in cluster:
              y_pred[sample_idx] = cluster_idx
      return y_pred
  def plot_fig(self, X, y):
      fig = plt.scatter(X[:, 0], X[:, 1], c=y)
      fig.show()

  def fit(self, X):
      centroids = self.initialize_random_centroids(X)
      for _ in range(self.max_iterations):
          clusters = self.create_cluster(X, centroids)
          previous_centroids = centroids
          centroids = self.calculate_new_centroids(clusters, X)
          diff = centroids - previous_centroids
          if not diff.any():
              break
      y_pred = self.predict_cluster(clusters, X)
      return y_pred
data.head()
dataset = data.iloc[:50,1:].values

kmeancluster = KMeanClustering(dataset, 3)
y_pred = kmeancluster.fit(dataset)
import matplotlib.pyplot as plt
plt.style.use('dark_background')
plt.scatter(dataset[:, 1:2],y_pred, c = y_pred, s = 200, alpha = 0.8)
plt.grid(True)
data.columns
plt.xlabel(data.columns[2])
plt.ylabel("class")
plt.show()
plt.scatter(dataset[:, 2:3], y_pred, c = y_pred, s = 200, alpha = 0.8)
data.columns
```

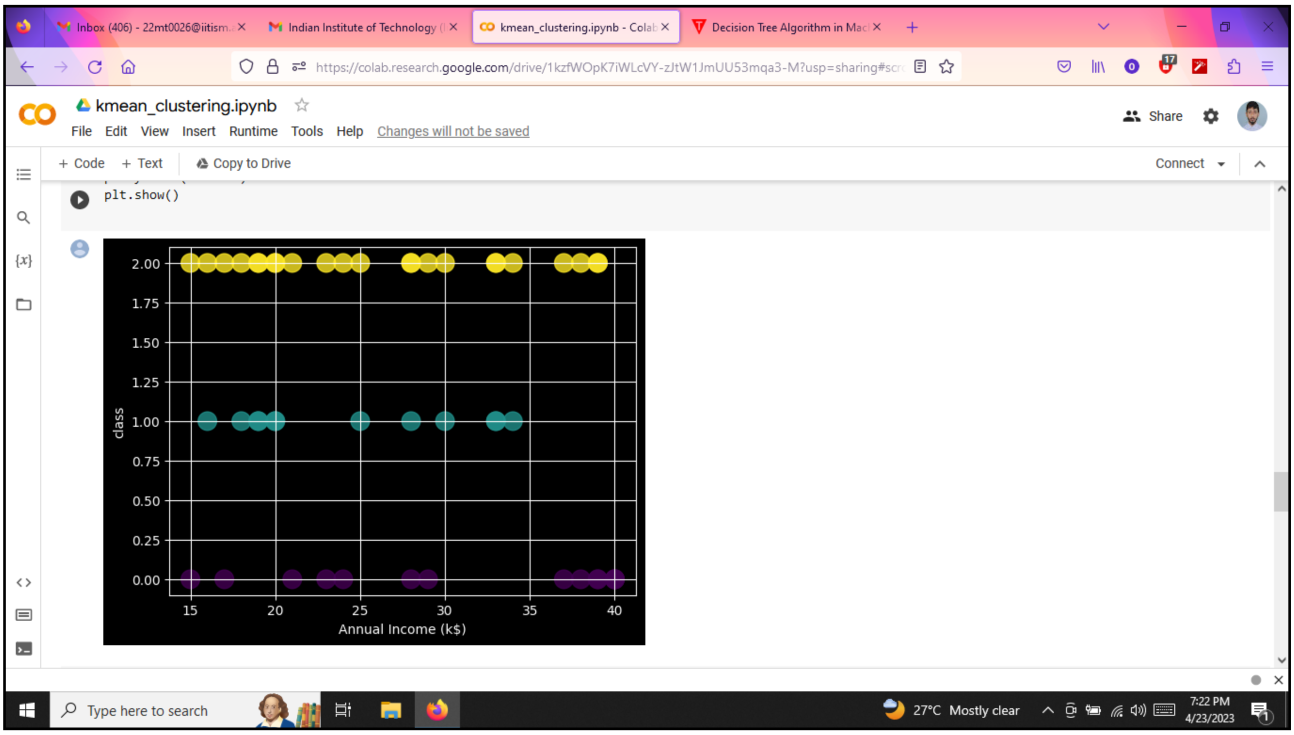
```
plt.xlabel(data.columns[3])
plt.grid(True)
plt.ylabel("class")
plt.show()
plt.scatter(dataset[:, 3:4], y_pred, c = y_pred, s = 200, alpha = 0.8)
plt.grid(True)
plt.xlabel(data.columns[4])
plt.ylabel("class")
plt.show()
plt.scatter(dataset[:, 0:1], y_pred, c = y_pred, s = 200, alpha = 0.8)
data.columns
plt.grid(True)
plt.xlabel(data.columns[1])
plt.ylabel("class")
plt.show()
```

Result:-

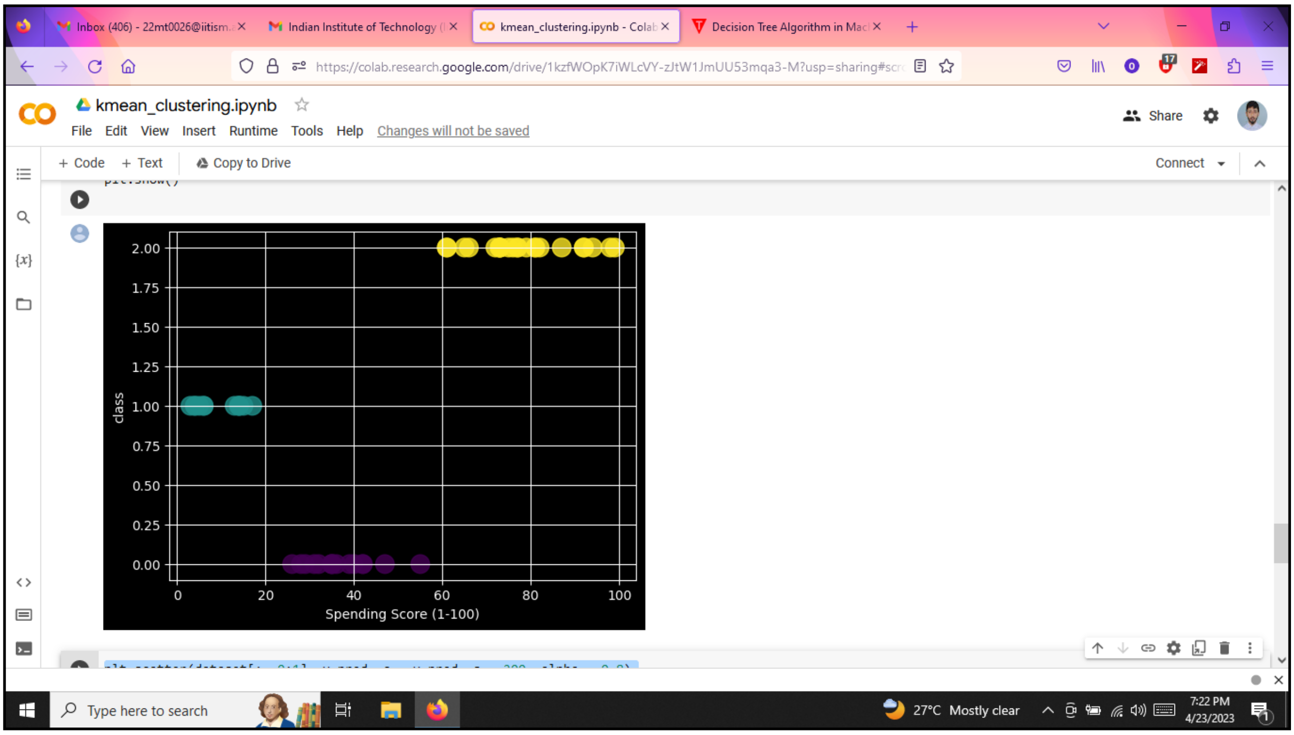


```
plt.show()
```

Inbox (406) - 22mt0026@iitism...
Indian Institute of Technology
kmean_clustering.ipynb - Colab
Decision Tree Algorithm in Mac
https://colab.research.google.com/drive/1kzfWOpK7iWLcVY-zJtW1JmUU53mqa3-M?usp=sharing#scr
kmean_clustering.ipynb
File Edit View Insert Runtime Tools Help Changes will not be saved
Share
+ Code + Text
Copy to Drive
Connect
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
class
20
30
40
50
60
Age
Type here to search
27°C Mostly clear
7:22 PM
4/23/2023


Inbox (406) - 22mt0026@iitism...
Indian Institute of Technology (...
kmean_clustering.ipynb - Colab
Decision Tree Algorithm in Mac...
https://colab.research.google.com/drive/1kzfWOpK7iWLcVY-zJtW1JmUU53mqa3-M?usp=sharing#scr
kmean_clustering.ipynb
File Edit View Insert Runtime Tools Help Changes will not be saved
Share
+ Code + Text Copy to Drive
Connect
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
class
15
20
25
30
35
40
Annual Income (k$)
Type here to search
27°C Mostly clear
7:22 PM
4/23/2023

```
plt.show()
```


Inbox (406) - 22mt0026@iitism...
Indian Institute of Technology (...
kmean_clustering.ipynb - Colab
Decision Tree Algorithm in Mac...
https://colab.research.google.com/drive/1kzfWOpK7iWLcVY-zJtW1JmUU53mqa3-M?usp=sharing#scr
kmean_clustering.ipynb
File Edit View Insert Runtime Tools Help Changes will not be saved
Share
+ Code + Text Copy to Drive
Connect
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
class
0
20
40
60
80
100
Spending Score (1-100)
Type here to search
27°C Mostly clear
7:22 PM
4/23/2023

```
plt.show()
```

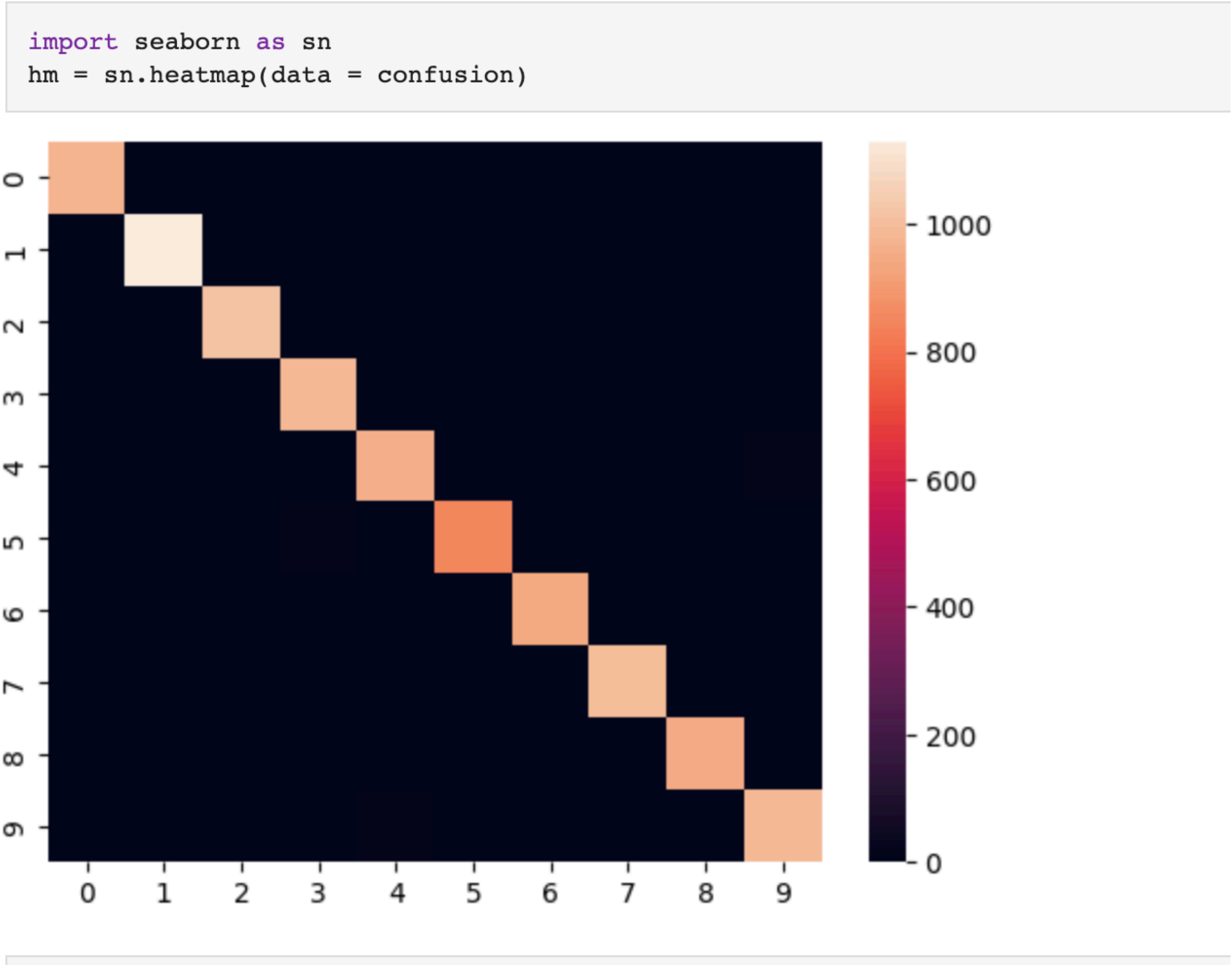
Problem Statement:- Implement Digit Recogniser using Neural Network

Code:-

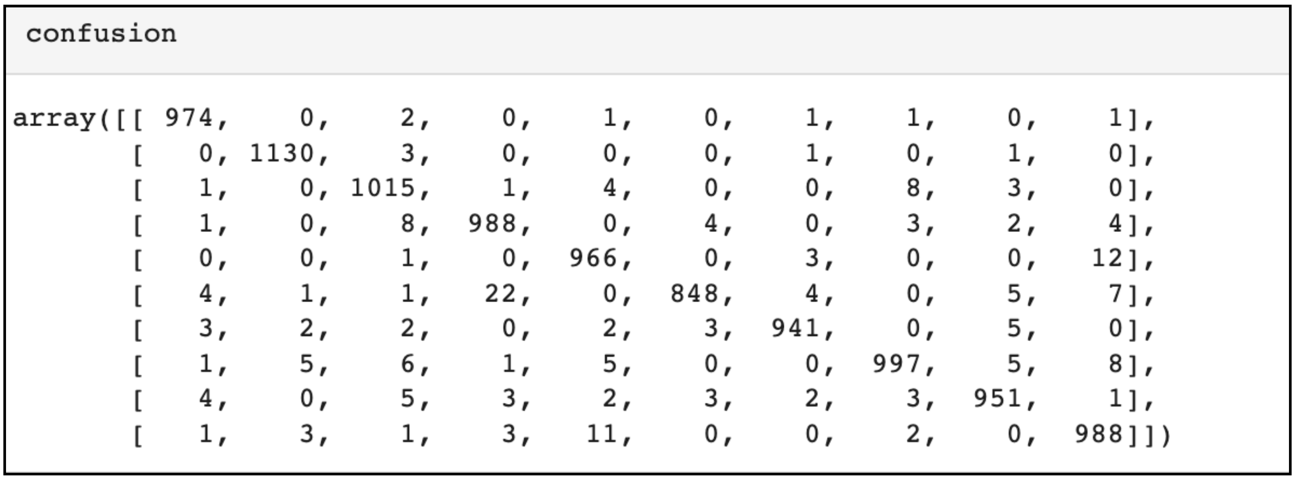
```
import tensorflow as tf
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

x_train = x_train.reshape(x_train.shape[0], 196, 4, 1)
x_test = x_test.reshape(x_test.shape[0], 196, 4, 1)
input_shape = (196, 4, 1)

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')

x_train /= 255
x_test /= 255

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, Dropout, Flatten, MaxPooling2D
model = Sequential()
model.add(Conv2D(196, kernel_size=(3,3), input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(128, activation=tf.nn.relu))
model.add(Dropout(0.2))
model.add(Dense(10,activation=tf.nn.softmax))

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
model.fit(x=x_train,y=y_train, epochs=10)

model.evaluate(x_test, y_test)

import matplotlib.pyplot as plt
image_index = 2853
plt.imshow(x_test[image_index].reshape(28, 28),cmap='Greys')
predict = x_test[image_index].reshape(28,28)
pred = model.predict(x_test[image_index].reshape(1, 196, 4, 1))
print(pred.argmax())

from sklearn.metrics import confusion_matrix
```

```
import numpy as np

test_predictions = model.predict(x_test)
confusion = confusion_matrix(y_test, np.argmax(test_predictions,axis=1))

confusion

import seaborn as sn
hm = sn.heatmap(data = confusion)
```

Result:-

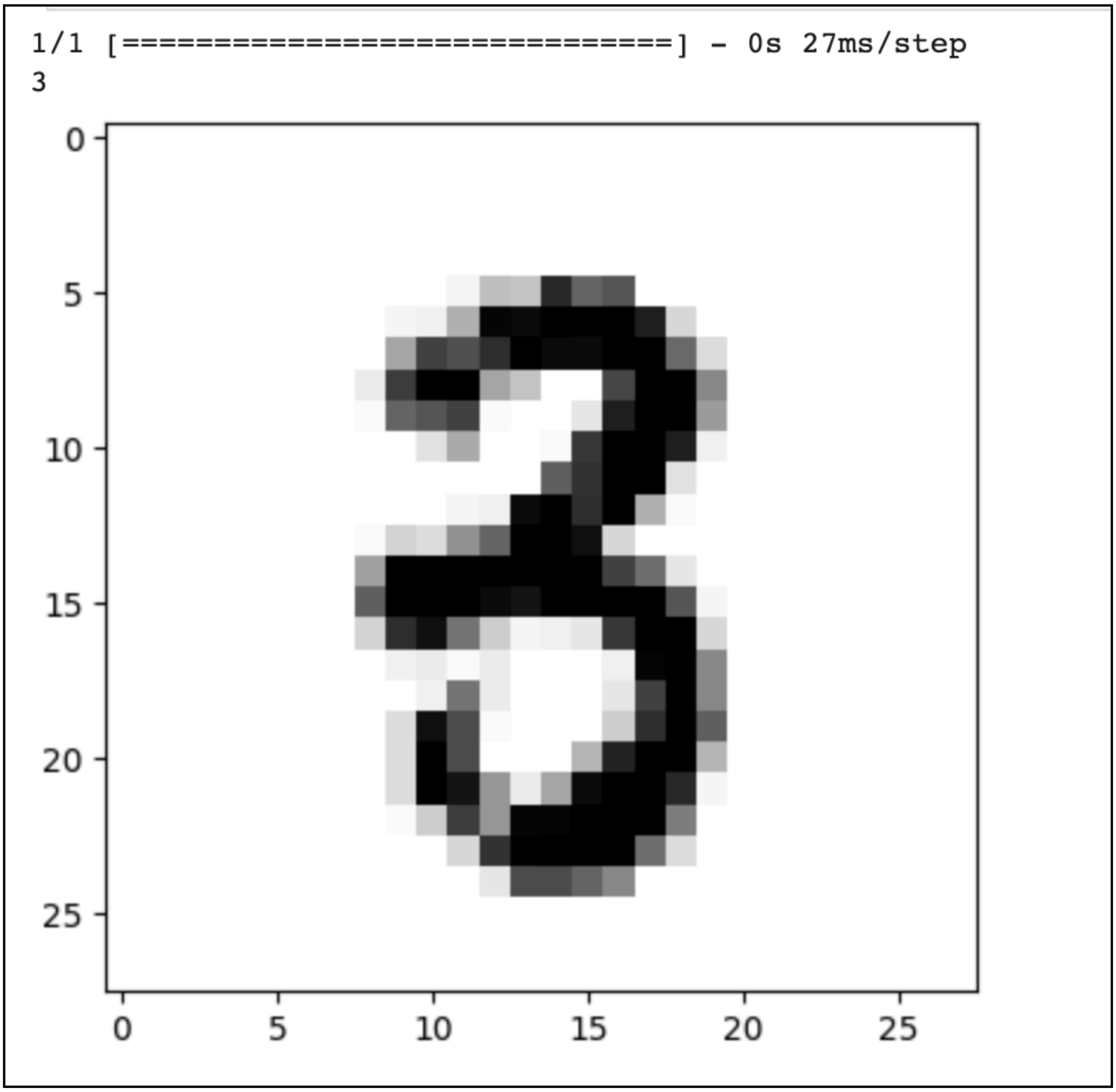


```
import seaborn as sn
hm = sn.heatmap(data = confusion)
```



```
confusion
```

```
array([[ 974,    0,    2,    0,    1,    0,    1,    1,    0,    1],
       [   0, 1130,    3,    0,    0,    0,    1,    0,    1,    0],
       [   1,    0, 1015,    1,    4,    0,    0,    8,    3,    0],
       [   1,    0,    8,  988,    0,    4,    0,    3,    2,    4],
       [   0,    0,    1,    0,  966,    0,    3,    0,    0,   12],
       [   4,    1,    1,   22,    0,  848,    4,    0,    5,    7],
       [   3,    2,    2,    0,    2,    3,  941,    0,    5,    0],
       [   1,    5,    6,    1,    5,    0,    0,  997,    5,    8],
       [   4,    0,    5,    3,    2,    3,    2,    3,  951,    1],
       [   1,    3,    1,    3,   11,    0,    0,    2,    0,  988]])
```



```
1/1 [==============================] - 0s 27ms/step
3
```