Code:-

```
import numpy as np
from tabulate import tabulate
import pandas as pd
def convert(x):
    values = []
    for i in range(len(x)):
        v = int(x[i])
        values.append(v)
    return values
def generate_data_matrix(k, intial):
    arr = 2 ** k
    a = []
    for i in range(intial, arr):
        b = bin(i)
        b = b[2:]
        b = b.zfill(k)
        a.append(list(b))
    data = np.array(a, dtype=int)
    return data
def generator_parity(parity):
    p = generate_data_matrix(parity, 0)
    p = list(map(list, p))
    p_matrix = p.copy()
    for i in range(len(p)):
        ones = list(p[i]).count(1)
        if ones < parity - 1:
            p_matrix.remove(p[i])
    return p_matrix
def generator_matrix1(n, k):
    """Generation of generator matrix"""
    I = np.identity(k, dtype=int)
    parity = n - k
    p_matrix = generator_parity(parity)
    I = list(map(list, I))
    I_copy = I.copy()
    p = p_matrix.copy()
    emp = []
    empI = []
    for j in range(len(p)):
        if p[j].count(1) >= 2:
            emp.append(p[j])
    form = input('Enter the generator matrix format:-')
    if 'PI' == form.upper():
        print('The generator matrix is in PI')
        print()
        for i in range(k):
            num_of_ones = list(I_copy[i]).count(1)
            if num_of_ones == 1:
                a = emp[i] + I_copy[i]
                empI.append(a)
                p_matrix.remove(p[i])
    elif 'IP' == form.upper():
        print('The generator matrix is in IP')
        print()
        for i in range(k):
            num_of_ones = list(I_copy[i]).count(1)
            if num_of_ones == 1:
                a = I_copy[i] + emp[i]
                empI.append(a)
                p_matrix.remove(p[i])
    I1 = np.array(empI)
    return I1, emp, form
def generator2(h,k):
    f = check_form(h, parity)
    s = Extract(h, k, f)
    I = np.identity(k, dtype=int)
    I = list(map(list, I))
    I_copy = I.copy()
    empI = []
    if 'PI' == f.upper():
        for i in range(k):
            a = s[i] + I_copy[i]
            empI.append(a)
        return empI
    elif 'IP' == form.upper():
        for i in range(k):
            a = I_copy[i] + s[i]
            empI.append(a)
        return  empI,f
def H_matrix(p,form):
    pt = np.transpose(p)
    I = np.identity(n - k, dtype=int)
    I = list(map(list, I))
    pt= list(map(list, pt))
    P = pt.copy()
    I_copy = I.copy()
    empI = []
    if 'PI' == form.upper():
        for i in range(n-k):
            a =  I_copy[i] + P[i]
            empI.append(a)
        print()
    elif 'IP' == form.upper():
        for i in range(n-k):
            a = P[i] + I_copy[i]
            empI.append(a)
        print()
    return empI
def words(data,form):
    """genearte words"""
    emp = []
    SUB = str.maketrans("0123456789", "₀₁₂₃₄₅₆₇₈₉")
    for i in range(1, data + 1):
        if n - k == data:
            emp.append('S' + str(i).translate(SUB))
        else:
            emp.append('e' + str(i).translate(SUB))
    if 'IP' == form.upper():
        emp = sorted(emp, reverse=True)
    return emp
def check_form(g1,a):
    global form
    s1 = [item[-a:] for item in g1]
    s2 = [item[:a] for item in g1]
```

```
    s1 = list(map(list, s1))
    s2 = list(map(list, s2))
    I = np.identity(a, dtype=int)
    I = list(map(list, I))

    if a == k:
        for i in range(len(I)):
            for j in range(len(g1)):
                if s1[j] == I[i]:
                    form = 'pi'
                elif s2[j] == I[i]:
                    form = 'ip'
    else:
        for i in range(len(I)):
            for j in range(len(g1)):
                if s1[j] == I[i]:
                    form = 'ip'
                elif s2[j] == I[i]:
                    form = 'pi'
    return form
def Extract(lst, parity, form):
    """extract the parity matrix"""
    if parity == n-k:
        if 'IP' == form.upper():
            print('The generator matrix is in IP')
            s = [item[-parity:] for item in lst]
            return s
        elif 'PI' == form.upper():
            print('The generator matrix is in PI')
            s = [item[0:parity] for item in lst]
            return s
    else:
        if 'IP' == form.upper():
            print('The generator matrix is in IP')
            s = [item[0:parity] for item in lst]
            s = np.transpose(s)
            s = list(map(list,s))
            return s
        elif 'PI' == form.upper():
            print('The generator matrix is in PI')

            s = [item[-parity:] for item in lst]
            s = np.transpose(s)
            s = list(map(list, s))
            return s
def display(data):
    """displaying the data"""
    if dict != type(data):
        data = np.array(data)
        data = np.transpose(data)
        D = words(len(data), form)
        data = list(map(list,data))
        data = dict(zip(D, data))
    df = pd.DataFrame(data)
    df = tabulate(df, headers='keys', tablefmt='fancy_grid')
    return df, data
def decoding(n,h):
    emp = []
    s = [1] + [0]*(n-1)
    for i in range(n):
        a = np.roll(s,i)
        a = list(a)
        emp.append(a)
    emp.append([0]*n)
    ht = np.transpose(h)
    ht = list(map(list,ht))
    a = [0]*parity
    ht.append(a)
    return emp, ht
def syndrome(h,e):
    global s
    r = list(input('Enter the recieved code:-'))
    r = convert(r)
    ht = np.transpose(h)
    ht = list(map(list,ht))
    s= []
    emp =[]
    for i in range(len(r)):
        if r[i] == 1:
            s.append(ht[i])
    list1 = [0]*len(ht[0])
    for i in range(len(s)):
        list2 = s[i]
        emp= [a^b for a,b in zip(list1,list2)]
        list1 = emp
    if emp == [0]*parity:
        print('No error')
        return r
    elif emp != [0]*parity:
        err = []
        for i in range(len(ht)):
            if emp == ht[i]:
                err = e[i]
        print('for Syndrome =',string(emp),'the error code
is',string(err))

        print('The error is in ',err.index(1)+1,'-bit')
        C = [a^b for a,b in zip(err,r)]
        return C
def string(T):
    st =''
    for i in range(len(T)):
        v = str(T[i])
        st = st + v
    return st
print('''plz select what you want to do:-
      1 = to use the genearator and H matrix which is there
      2 = to create a new generator and H matrix
      3 = to enter the values of generator matrix and create H
matrix
      4 = to enter the values of H matrix and create generator
matrix''')
print()
q = int(input('Enter the option number you want: -'))
global n, k, g, p,form,h,decode,d
if 1 == q:
    n = 7
    k = 4
    parity = n - k
    g = [[1, 0, 0, 0, 1, 1, 0], [0, 1, 0, 0, 1, 1, 1], [0, 0, 1, 0, 1, 1, 1],
```

```
[0, 0, 0, 1, 1, 0, 1]]
    h = [[1,0,1,1,1,0,0],[1,1,1,0,0,1,0],[0,1,1,1,0,0,1]]
    form = 'ip'
    decode, ht = decoding(n, h)
    a, b = display(decode)
    a1, b1 = display(ht)
    di = b | b1
    d, _ = display(di)
elif 2 == q:
    n, k = map(int, input('Enter the number of  code bit and data
bit:-').split())
    print('The linear code is of ','(',n,',', k,')')
    parity = n - k
    g, p, form = generator_matrix1(n, k)
    h = H_matrix(p, form)
    decode, ht = decoding(n, h)
    a, b = display(decode)
    a1, b1 = display(ht)
    di = b | b1
    d, _ = display(di)
elif 3 == q:
    n, k = map(int, input('Enter the number of  code bit and data
bit:-').split())
    print('The linear code is of ', '(', n, ',', k, ')')
    parity = n - k
    print()
    g_matrix = [input('Enter the generator matrix with space :-
').split() for _ in range(k)]
    g = []
    for i in range(len(g_matrix)):
        a = convert(g_matrix[i])
        g.append(a)
    g1 = g.copy()
    form = check_form(g1, k)
    print('The generator matrix is in', form.upper(), 'form\n')
    p = Extract(g, parity, form)
    g = np.array(g)
    h = H_matrix(p, form)
    decode, ht = decoding(n, h)
    a, b = display(decode)
    a1, b1 = display(ht)
    di = b | b1
    d, _ = display(di)
elif 4 == q:
    n, k = map(int, input('Enter the number of  code bit and data
bit:-').split())
    print('The linear code is of ', '(', n, ',', k, ')')
    parity = n - k
    print()
    h_matrix = [input('Enter the H matrix with space :-').split() for
_ in range(parity)]
    h = []
    for i in range(len(h_matrix)):
        a = convert(h_matrix[i])
        h.append(a)
    g2,form = generator2(h, k)
    print('The Generator matrix is in', form.upper(), 'form\n')
    g = np.array(g2)
    decode, ht = decoding(n, h)
    a, b = display(decode)
    a1, b1 = display(ht)
    di = b | b1
    d, _ = display(di)
g1 = tabulate(g, tablefmt='fancy_grid')
h1 = tabulate(h, tablefmt='fancy_grid')
print('--'*50, end='\n')
print('The GENERATOR MATRIX IS ')
print(g1)
print('--'*50, end='\n')
print('The H MATRIX IS :-')
print(h1)
print('The Decoding Table is')
print(d)
i = 0
while i < 3:
    print('For ',i,'-bit error')
    r = syndrome(h, decode)
    print('The recieved Code is',string(r))
    i = i +1
    print('--' * 50, end='\n')
print('Made by varad patil')
```

```
plz select what you want to do:-
        1 = to use the genearator and H matrix which is there
        2 = to create a new generator and H matrix
        3 = to enter the values of generator matrix and create H matrix
        4 = to enter the values of H matrix and create generator matrix

Enter the option number you want: -1
```

The GENERATOR MATRIX IS

1	0	0	0	1	1	0
0	1	0	0	1	1	1
0	0	1	0	1	1	1
0	0	0	1	1	0	1

The H MATRIX IS :-

1	0	1	1	1	0	0
1	1	1	0	0	1	0
0	1	1	1	0	0	1

The Decoding Table is

	e_7	e_6	e_5	e_4	e_3	e_2	e_1	S_3	S_2	S_1
0	1	0	0	0	0	0	0	1	1	0
1	0	1	0	0	0	0	0	0	1	1
2	0	0	1	0	0	0	0	1	1	1
3	0	0	0	1	0	0	0	1	0	1
4	0	0	0	0	1	0	0	1	0	0
5	0	0	0	0	0	1	0	0	1	0
6	0	0	0	0	0	0	1	0	0	1
7	0	0	0	0	0	0	0	0	0	0

```
For  0 -bit error
Enter the recieved code:-1000110
No error
The recieved Code is 1000110
--------------------------------------------------
For  1 -bit error
Enter the recieved code:-0000110
for Syndrome = 110 the error code is 1000000
The error is in  1 -bit
The recieved Code is 1000110
--------------------------------------------------
For  2 -bit error
Enter the recieved code:-1000000
for Syndrome = 110 the error code is 1000000
The error is in  1 -bit
The recieved Code is 0000000
--------------------------------------------------
Made by varad patil
```