Code: -

```
"""linear code experiment"""
import numpy as np
from tabulate import tabulate
import pandas as pd
def generate_data_matrix(k, intial):
    """Generation of data matrix"""
    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)a
    return data
def generator_parity(parity):
    """Generation of parity matrix"""
    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_matrix(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 code_generation(data,g):
    """code matrix generation"""
    drow = data.shape[0]
    gcol = g.shape[1]
    global C
    if data.shape[1] == g.shape[0]:
        C = np.zeros((data.shape[0], g.shape[1]), dtype=int)
        for row in range(drow):
            for col in range(gcol):
                for elt in range(len(g)):
                    C[row, col] ^= data[row, elt] * g[elt,
col]
        return C
    else:
        return "Sorry, cannot multiply A and B."
def minimum_weight(c):
    """calculate min hamming weight"""
    num_of_ones = []
    c = list(map(list, c))
    for i in range(len(c)):
        a = list(c[i]).count(1)
        num_of_ones.append(a)
    num_of_ones.remove(0)
    return num_of_ones
def error(ones):
    """Error correction and detectiion capabality"""
    global t

    dmin = min(ones)
    err_detect = dmin - 1
    if dmin%2 == 0:
        print('dmin is an even number')
        t = (dmin - 2)/2
    elif dmin%2 ==1:
        print('dmin is an odd number')
        t = (dmin - 1) / 2
    return err detect, t
def words(data,form):
    """genearte words"""
    emp = []
    SUB = str.maketrans("0123456789", "0123456789")
    for i in range(1, data + 1):
        if k == data:
            emp.append('D' + str(i).translate(SUB))
        elif n - k == data:
            emp.append('P' + str(i).translate(SUB))
        else:
            emp.append('C' + str(i).translate(SUB))
    if 'IP' == form.upper():
        emp = sorted(emp, reverse=True)
    return emp
def parity_eqn(p,form):
    """generate parity eqn"""

    result = []
    last = []
    emp = words(k,form)
    parity = words(n-k,form)


    p = np.array(p)
    p = np.transpose(p)
    emp1 = list(map(list, p))

    for j in range(p.shape[0]):
        b = emp1[j]
        a = [item1 * item2 for item1, item2 in zip(emp, b)]
        a1 = a.copy()
        for i in range(len(a)):
            if '' == a[i]:
                a1.pop(i)
        result.append(a1)
    for j in range(len(result)):
        a = result[j]
        res = '+'.join(str(a[i]) for i in
range(len(result[j])))
        last.append(res)

    parity1 = dict(zip(parity, last))
    return parity1, emp


def convert(x):
    """convert str list to int list"""
    values = []
    for i in range(len(x)):
        v = int(x[i])
        values.append(v)
    return values


def Extract(lst, parity, form):
    """extract the parity matrix"""
    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


def check_form(g1,k):
    global form
    s1 = [item[-k:] for item in g1]
    s2 = [item[:k] for item in g1]
    I = np.identity(k, dtype=int)
    I = list(map(list, I))
    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'
```

```
    return form


def convert_to_list(arr):
    """convert to list_string """
    lst = list(map(list, arr))
    flatList = [item for elem in lst for item in elem]
    return flatList


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

print('''plz select what you want to do:-
         1 = to use the genearator and data matrix which is
there
         2 = to create a new generator and data matrix
         3 = to enter the values of generator and data
matrix from the user''')
print('\n')
q = int(input('Enter the option number you want: -'))

print()

global n, k, initial, data, g, p,form
if 1 == q:
    print('''which linear code you want of (6,3) or (7,4)
             Enter a for (6,3)
             Enter b for another(6,3)
             Enter c for (7,4)''')

    a = input('Enter linear code : -')

    if a.lower() == 'a':
        g = [[1, 0, 0, 1, 1, 0], [0, 1, 0, 0, 1, 1], [0, 0,
1, 1, 0, 1]]
        k = 3
        intial = 0
        n = 6
        data = generate_data_matrix(k, intial)
        parity = n - k
        p = Extract(g, parity, 'ip')
        g = np.array(g)
        form = 'ip'
    elif a.lower() == 'b':
        n = 6
        k = 3
        parity = n - k
        d = [[0, 0, 0], [0, 0, 1], [0, 1, 0], [0, 1, 1], [1,
0, 0], [1, 0, 1], [1, 1, 0], [1, 1, 1]]
        g = [[1, 0, 0, 1, 0, 1], [0, 1, 0, 0, 1, 1], [0, 0,
1, 1, 1, 1]]
        g1 = g.copy()
        form = check_form(g1, k)
        print('The generator matrix is in', form.upper(),
'form\n')
        p = Extract(g, parity, form)
        data = np.array(d)
        g = np.array(g)

    elif a.lower() == 'c':
        k = 4
        intial = 0
        n = 7
        data = generate_data_matrix(k, intial)
        g = [[1, 0, 0, 0, 1, 1, 1], [0, 1, 0, 0, 1, 0, 1],
[0, 0, 1, 0, 0, 1, 1], [0, 0, 0, 1, 1, 1, 0]]
        parity = n - k
        p = Extract(g, parity, 'ip')
        g = np.array(g)
        form = 'ip'

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,')')
    intial = int(input('enter The starting  decimal number:-
'))
    data = generate_data_matrix(k, intial)
    g, p, form = generator_matrix(n, k)
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, ')')
    print("\n")
    d = [input('Enter the data matrix with space :-
').split() for _ in range(2**k)]
    g_matrix = [input('Enter the generator matrix with space
:-').split() for _ in range(k)]

    data = []
    g = []
    parity = n - k
    for i in range(len(d)):
        a = convert(d[i])
        data.append(a)

    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)
    data = np.array(data)
    g = np.array(g)
rate = round(k/n,2)
c = code_generation(data, g)
ones = minimum_weight(c)
err_detect, t = error(ones)
eqn, D = parity_eqn(p, form)

data, dict1 = display(data)
c, dict2 = display(c)
merge = dict1|dict2
final, _ = display(merge)
g = tabulate(g, tablefmt='fancy_grid')
p = tabulate(eqn.items(), headers=['PARITY', 'DATA'],
tablefmt='fancy_grid')
print('--'*100, end='\n')
print('The DATA MATRIX IS :-')
print(data)
print('--'*100, end='\n')
print('The GENERATOR MATRIX IS ')
print(g)
print('--'*100, end='\n')
print('The CODE is :-')
print(c)
print('The minimum Hamming weight is = ', min(ones))
print('The error detection capability :- ', err_detect)
print('The error correction capability :- ', t)
print('The Code rate is', rate)
print('The Code efficiency is ', round(rate*100, 2))
print('The Parity eqn are')
print(p)
print('The final table is :- ')
print(final)
print('made by Varad Patil 120A2036')
```

Result:- For (6,3)

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


Enter the option number you want: -1

which linear code you want of (6,3) or (7,4)
            Enter a for (6,3)
            Enter b for another(6,3)
            Enter c for (7,4)
Enter linear code : -a
The generator matrix is in IP
dmin is an odd number
```

The DATA MATRIX IS :-

	D_3	D_2	D_1
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

The GENERATOR MATRIX IS

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

The minimum Hamming weight is = 3
The error detection capability :- 2
The error correction capability :- 1.0
The Code rate is 0.5
The Code efficiency is 50.0
The Parity eqn are

PARITY	DATA
D_3	D_3+D_1
D_2	D_3+D_2
D_1	D_2+D_1

The CODE is :-

	C_6	C_5	C_4	C_3	C_2	C_1
0	0	0	0	0	0	0
1	0	0	1	1	0	1
2	0	1	0	0	1	1
3	0	1	1	1	1	0
4	1	0	0	1	1	0
5	1	0	1	0	1	1
6	1	1	0	1	0	1
7	1	1	1	0	0	0

The final table is :-

	D_3	D_2	D_1	C_6	C_5	C_4	C_3	C_2	C_1
0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	1	1	0	1
2	0	1	0	0	1	0	0	1	1
3	0	1	1	0	1	1	1	1	0
4	1	0	0	1	0	0	1	1	0
5	1	0	1	1	0	1	0	1	1
6	1	1	0	1	1	0	1	0	1
7	1	1	1	1	1	1	0	0	0

made by Varad Patil 120A2036

For (7,4)

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

which linear code you want of (6,3) or (7,4)
            Enter a for (6,3)
            Enter b for another(6,3)
            Enter c for (7,4)
Enter linear code : -c
The generator matrix is in IP
dmin is an odd number
```

The DATA MATRIX IS :-

	D_4	D_3	D_2	D_1
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

The GENERATOR MATRIX IS

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

The minimum Hamming weight is = 3
The error detection capability :- 2
The error correction capability :- 1.0
The Code rate is 0.57
The Code efficiency is 57.0
The Parity eqn are

PARITY	DATA
P_3	$D_4+D_3+D_1$
P_2	$D_4+D_2+D_1$
P_1	$D_4+D_3+D_2$

The CODE is :-

	C_7	C_6	C_5	C_4	C_3	C_2	C_1
0	0	0	0	0	0	0	0
1	0	0	0	1	1	1	0
2	0	0	1	0	0	1	1
3	0	0	1	1	1	0	1
4	0	1	0	0	1	0	1
5	0	1	0	1	0	1	1
6	0	1	1	0	1	1	0
7	0	1	1	1	0	0	0
8	1	0	0	0	1	1	1
9	1	0	0	1	0	0	1
10	1	0	1	0	1	0	0
11	1	0	1	1	0	1	0
12	1	1	0	0	0	1	0
13	1	1	0	1	1	0	0
14	1	1	1	0	0	0	1
15	1	1	1	1	1	1	1

The final table is :-

	D_4	D_3	D_2	D_1	C_7	C_6	C_5	C_4	C_3	C_2	C_1
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1	1	1	0
2	0	0	1	0	0	0	1	0	0	1	1
3	0	0	1	1	0	0	1	1	1	0	1
4	0	1	0	0	0	1	0	0	1	0	1
5	0	1	0	1	0	1	0	1	0	1	1
6	0	1	1	0	0	1	1	0	1	1	0
7	0	1	1	1	0	1	1	1	0	0	0
8	1	0	0	0	1	0	0	0	1	1	1
9	1	0	0	1	1	0	0	1	0	0	1
10	1	0	1	0	1	0	1	0	1	0	0
11	1	0	1	1	1	0	1	1	0	1	0
12	1	1	0	0	1	1	0	0	0	1	0
13	1	1	0	1	1	1	0	1	1	0	0
14	1	1	1	0	1	1	1	0	0	0	1
15	1	1	1	1	1	1	1	1	1	1	1

made by Varad Patil 120A2036