Step 1

In [586]:
```
import pandas as pd
df=pd.read_csv(r"C:\Users\hasan\Downloads\datasets\datasets\penguins.csv")
df
```

Out[586]:

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex	year
0	Adelie	Torgersen	39.1	18.7	181.0	3750.0	male	2007
1	Adelie	Torgersen	39.5	17.4	186.0	3800.0	female	2007
2	Adelie	Torgersen	40.3	18.0	195.0	3250.0	female	2007
3	Adelie	Torgersen	NaN	NaN	NaN	NaN	NaN	2007
4	Adelie	Torgersen	36.7	19.3	193.0	3450.0	female	2007
...	...	...	...	...	...	...	...	...
339	Chinstrap	Dream	55.8	19.8	207.0	4000.0	male	2009
340	Chinstrap	Dream	43.5	18.1	202.0	3400.0	female	2009
341	Chinstrap	Dream	49.6	18.2	193.0	3775.0	male	2009
342	Chinstrap	Dream	50.8	19.0	210.0	4100.0	male	2009
343	Chinstrap	Dream	50.2	18.7	198.0	3775.0	female	2009

344 rows × 8 columns

Step 2

In [324]:
```
df.describe()
```

Out[324]:

	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	year
count	342.000000	342.000000	342.000000	342.000000	344.000000
mean	43.921930	17.151170	200.915205	4201.754386	2008.029070
std	5.459584	1.974793	14.061714	801.954536	0.818356
min	32.100000	13.100000	172.000000	2700.000000	2007.000000
25%	39.225000	15.600000	190.000000	3550.000000	2007.000000
50%	44.450000	17.300000	197.000000	4050.000000	2008.000000
75%	48.500000	18.700000	213.000000	4750.000000	2009.000000
max	59.600000	21.500000	231.000000	6300.000000	2009.000000

In [325]:
```
df.isna().sum()
```

Out[325]:
```
species               0
island                0
bill_length_mm        2
bill_depth_mm         2
flipper_length_mm     2
body_mass_g           2
sex                  11
year                  0
dtype: int64
```

In [326]:
```
df.dropna(axis=0,inplace=True)
df.isna().sum()
```

Out[326]:
```
species              0
island               0
bill_length_mm       0
bill_depth_mm        0
flipper_length_mm    0
body_mass_g          0
sex                  0
year                 0
dtype: int64
```

In [327]:
```
df.dtypes
```
Out[327]:
```
species               object
island                object
bill_length_mm       float64
bill_depth_mm        float64
flipper_length_mm    float64
body_mass_g          float64
sex                   object
year                   int64
dtype: object
```

In [328]:
```
df.species.unique()
```
Out[328]:
```
array(['Adelie', 'Gentoo', 'Chinstrap'], dtype=object)
```

In [329]:
```
df.island.unique()
```
Out[329]:
```
array(['Torgersen', 'Biscoe', 'Dream'], dtype=object)
```

In [330]:
```
df.sex.unique()
```
Out[330]:
```
array(['male', 'female'], dtype=object)
```

Step 3

In [331]:
```
df.loc[df['species']=="Adelie", 'species'] = 1
df.loc[df['species']=="Gentoo", 'species'] = 2
df.loc[df['species']=="Chinstrap", 'species'] = 3
```

In [332]:
```
#df.loc[df['island']=="Torgersen", 'island'] = 1
#df.loc[df['island']=="Biscoe", 'island'] = 2
#df.loc[df['island']=="Dream", 'island'] = 3
```

In [333]:
```
df.loc[df['island']=="Torgersen", 'island'] = 1
df.loc[df['island']=="Biscoe", 'island'] = 0
df.loc[df['island']=="Dream", 'island'] = 0
```

In [334]:
```
df.loc[df['sex']=="male", 'sex'] = 1
df.loc[df['sex']=="female", 'sex'] = 2
```

In [335]:
```
print(df.species.unique())
print(df.island.unique())
print(df.sex.unique())
```
```
[1 2 3]
[1 0]
[1 2]
```

In [336]:
```
df['island'] = df['island'].astype("int")
df['species'] = df['species'].astype(int)
df['sex'] = df['sex'].astype("int")
```

In [337]:
```
df.dtypes
```
Out[337]:
```
species                int32
island                 int32
bill_length_mm       float64
bill_depth_mm        float64
flipper_length_mm    float64
body_mass_g          float64
sex                    int32
year                   int64
dtype: object
```

Step 4

In [338]:
```
non_categorial_col=("bill_length_mm", "bill_depth_mm", "flipper_length_mm", "body_mass_g")
for i in non_categorial_col:
    maximum=df[i].max()
    minimum=df[i].min()
    print(f"{i}: Max is {maximum} and Min is {minimum}")
```

```
bill_length_mm: Max is 59.6 and Min is 32.1
bill_depth_mm: Max is 21.5 and Min is 13.1
flipper_length_mm: Max is 231.0 and Min is 172.0
body_mass_g: Max is 6300.0 and Min is 2700.0
```

In [339]:
```
for i in non_categorial_col:
    df[i]=(df[i]-df[i].min())/(df[i].max()-df[i].min())
```

In [340]:
```
for i in non_categorial_col:
    maximum=df[i].max()
    minimum=df[i].min()
    print(f"{i}: Max is {maximum} and Min is {minimum}")
```

```
bill_length_mm: Max is 1.0 and Min is 0.0
bill_depth_mm: Max is 1.0 and Min is 0.0
flipper_length_mm: Max is 1.0 and Min is 0.0
body_mass_g: Max is 1.0 and Min is 0.0
```

Step 5,6,7,8

In [341]:
```
df.drop("year",axis=1,inplace=True)
```

In [479]:
```
import numpy as np
def train_test_split(df):
    train_index = np.random.rand(len(df)) < 0.8
    train_data = df[train_index]
    test_data = df[~train_index]
    train_x=train_data.drop("island",axis=1)
    test_x=test_data.drop("island",axis=1)
    train_y=train_data["island"]
    test_y=test_data["island"]
    return(train_x,train_y,test_x,test_y)
```

In [480]:
```
train_x,train_y,test_x,test_y=train_test_split(df)

print(train_x.shape)
print(train_y.shape)
print(test_x.shape)
print(test_y.shape)
```

```
(263, 6)
(263,)
(70, 6)
(70,)
```

Step 9

In [597]:
```
import numpy as np

class LogisticRegression:

    def __init__(self, itr, learning_rate):
        self.learning_rate = learning_rate
        self.itr = itr
        self.weights = None
        self.bias = None
        self.loss = []
        self.we = []
        self.bi = []

    def sigmoid(self, z):
        return(1/(1+ np.exp(-z)))

    def cost(self,train_y, y_predicted):
        return(-(1/train_y.shape[0])*(np.sum(train_y*np.log(y_predicted+ 1e-8)+ (1-train_y) *np.log(1-y_predicted+ 1e-8))))

    def gradient_descent(self,x_train,train_y, y_predicted):
        delta = y_predicted- train_y
        dW=np.dot(x_train.T,delta)/x_train.shape[0]
        db =np.sum(y_predicted- train_y)/train_y.shape[0]
        return (dW,db)

    def fit(self, x_train, train_y):
        self.weights=np.ones(x_train.shape[1])
        self.bias= 0
        for i in range(self.itr):
            z = np.dot(x_train,self.weights.T) +self.bias
            y_predicted= self.sigmoid(z)
            dW,db= self.gradient_descent(x_train,train_y, y_predicted)
            self.weights=self.weights-(self.learning_rate*dW)
            self.bias=self.bias-(db*self.learning_rate)
            cost=self.cost(train_y, y_predicted)

            self.loss.append(cost)
            print(f"For Iteration {i} the Loss is {round(self.loss[i],4)}.")
            self.we.append(self.weights)
            self.bi.append(self.bias)

    def predict(self, x_train):
        z=np.dot(x_train,self.weights.T)+ self.bias
        y_predicted=self.sigmoid(z)
        tmp=np.ones(x_train.shape[0])
        for i in range(x_train.shape[0]):
            if y_predicted[i]<.5:
                tmp[i]=0
            else:
                tmp[i]=1

        return(tmp)
```

In [541]:
```
def accuracy(y,y_predicted):
    acc=0
    n=y_predicted.shape[0]
    y_mat = np.array(y)
    for i in range(n):
        #print(y_mat[i],y_hat[0,i])
        if y_mat[i]==y_predicted[i]:
            acc+=1
        else:
            continue
    return(round(acc/n,2))
```

Model 1

In [608]:
```
model1=LogisticRegression(learning_rate=.003, itr=5000)
model1.fit(train_x,train_y)
#t=model1.predict(test_x)
```

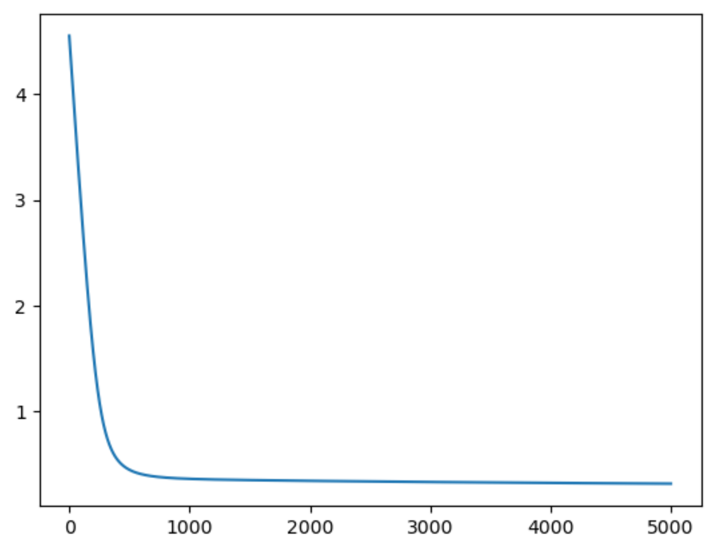
```
For Iteration 0 the Loss is 4.5526.
For Iteration 1 the Loss is 4.5358.
For Iteration 2 the Loss is 4.519.
For Iteration 3 the Loss is 4.5022.
For Iteration 4 the Loss is 4.4855.
For Iteration 5 the Loss is 4.4687.
For Iteration 6 the Loss is 4.4519.
For Iteration 7 the Loss is 4.4352.
For Iteration 8 the Loss is 4.4184.
For Iteration 9 the Loss is 4.4017.
For Iteration 10 the Loss is 4.3849.
For Iteration 11 the Loss is 4.3682.
For Iteration 12 the Loss is 4.3514.
For Iteration 13 the Loss is 4.3347.
For Iteration 14 the Loss is 4.318.
For Iteration 15 the Loss is 4.3013.
For Iteration 16 the Loss is 4.2846.
For Iteration 17 the Loss is 4.2679.
For Iteration 18 the Loss is 4.2512.
```

In [609]:
```
accuracy(test_y,model1.predict(test_x))
```

Out[609]: 0.87

In [610]:
```
import matplotlib.pyplot as plt
plt.plot(model1.loss)
```

Out[610]: [<matplotlib.lines.Line2D at 0x26102bb6a90>]



In [611]:
```
import pickle
with open('Hasan_Hussain_assignment1_part_1', 'wb') as files:
    pickle.dump(model1, files)
```

In [612]:
```
with open('Hasan_Hussain_assignment1_part_1' , 'rb') as f:
    lr1= pickle.load(f)

accuracy(test_y,lr1.predict(test_x))
```

Out[612]: 0.87

Model 2

In [574]:
```
train_x,train_y,test_x,test_y=train_test_split(df)
```

In [575]:
```
model2=LogisticRegression(learning_rate=.001, itr=2000)
model2.fit(train_x,train_y)
#model.predict(test_x)
```

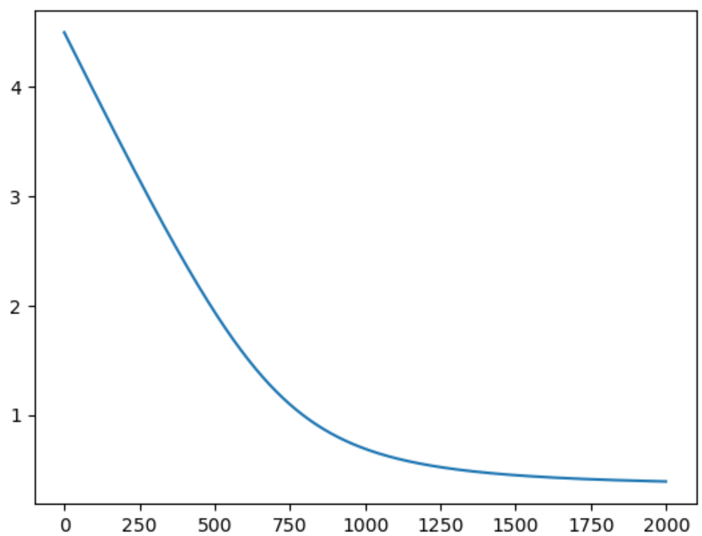
```
For Iteration 40 the Loss is 4.2779.
For Iteration 41 the Loss is 4.2724.
For Iteration 42 the Loss is 4.2669.
For Iteration 43 the Loss is 4.2614.
For Iteration 44 the Loss is 4.2559.
For Iteration 45 the Loss is 4.2505.
For Iteration 46 the Loss is 4.245.
For Iteration 47 the Loss is 4.2395.
For Iteration 48 the Loss is 4.234.
For Iteration 49 the Loss is 4.2285.
For Iteration 50 the Loss is 4.223.
For Iteration 51 the Loss is 4.2175.
For Iteration 52 the Loss is 4.2121.
For Iteration 53 the Loss is 4.2066.
For Iteration 54 the Loss is 4.2011.
For Iteration 55 the Loss is 4.1956.
For Iteration 56 the Loss is 4.1901.
For Iteration 57 the Loss is 4.1846.
For Iteration 58 the Loss is 4.1792.
For Iteration 59 the Loss is 4.1737.
```

In [576]:
```
accuracy(test_y,model2.predict(test_x))
```

Out[576]: 0.87

In [577]:
```
plt.plot(model2.loss)
```

Out[577]: [<matplotlib.lines.Line2D at 0x261023faf70>]



In [578]:
```
import pickle
with open('pkl_file2', 'wb') as files:
    pickle.dump(model2, files)
```

In [579]:
```
with open('pkl_file2' , 'rb') as f:
    lr2 = pickle.load(f)

accuracy(test_y,lr2.predict(test_x))
```

Out[579]: 0.87

Model 3

In [580]:
```
train_x,train_y,test_x,test_y=train_test_split(df)
```

In [581]:
```
model3=LogisticRegression(learning_rate=.4,itr=10000)
model3.fit(train_x,train_y)
#model.predict(test_x)
```

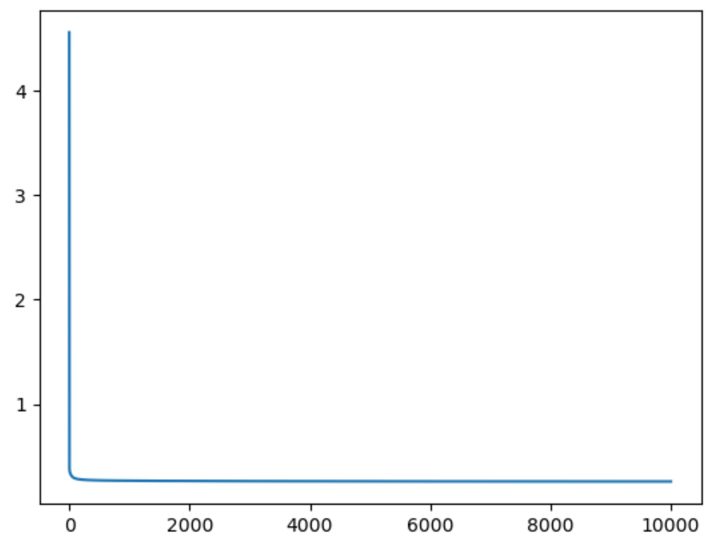
```
For Iteration 7596 the Loss is 0.2608.
For Iteration 7597 the Loss is 0.2608.
For Iteration 7598 the Loss is 0.2608.
For Iteration 7599 the Loss is 0.2608.
For Iteration 7600 the Loss is 0.2608.
For Iteration 7601 the Loss is 0.2608.
For Iteration 7602 the Loss is 0.2608.
For Iteration 7603 the Loss is 0.2608.
For Iteration 7604 the Loss is 0.2608.
For Iteration 7605 the Loss is 0.2608.
For Iteration 7606 the Loss is 0.2608.
For Iteration 7607 the Loss is 0.2608.
For Iteration 7608 the Loss is 0.2608.
For Iteration 7609 the Loss is 0.2608.
For Iteration 7610 the Loss is 0.2608.
For Iteration 7611 the Loss is 0.2608.
For Iteration 7612 the Loss is 0.2608.
For Iteration 7613 the Loss is 0.2608.
For Iteration 7614 the Loss is 0.2608.
For Iteration 7615 the Loss is 0.2608
```

In [582]:
```
accuracy(test_y,model3.predict(test_x))
```

Out[582]: 0.86

In [583]:
```
plt.plot(model3.loss)
```

Out[583]: [<matplotlib.lines.Line2D at 0x261026225b0>]



In [584]:
```
import pickle
with open('pkl_file3', 'wb') as files:
    pickle.dump(model3, files)
```

In [585]:
```
with open('pkl_file3' , 'rb') as f:
    lr3 = pickle.load(f)

accuracy(test_y,lr3.predict(test_x))
```

Out[585]: 0.86

Best Model Parameters

In [589]:
```
model1.bi
```

Out[589]:
```
[-0.002542204096210659,
 -0.005083935650023809,
 -0.007625187317765646,
 -0.010165951641590665,
 -0.012706221047763838,
 -0.015245987844919933,
 -0.01778524422229979,
 -0.020323982247963363,
 -0.02286219386697939,
 -0.025399870899591483,
 -0.027937005039360488,
 -0.030473587851282992,
 -0.03300961076988576,
 -0.035545065097296026,
 -0.03807994200128742,
 -0.04061423251330147,
 -0.04314792752644453,
 -0.04568101779345996,
 -0.04821349392467555,
```

In [596]:
```
model1.we
```

Out[596]:
```
[array([0.99521622, 0.99882567, 0.99885875, 0.99868819, 0.99887888,
        0.99613319]),
 array([0.99043303, 0.99765149, 0.99771776, 0.99737656, 0.99775792,
        0.99226707]),
 array([0.98565045, 0.99647746, 0.99657703, 0.9960651 , 0.99663713,
        0.98840164]),
 array([0.98086849, 0.99530358, 0.99543658, 0.99475383, 0.9955165 ,
        0.98453693]),
 array([0.97608716, 0.99412986, 0.9942964 , 0.99344274, 0.99439603,
        0.98067295]),
 array([0.97130646, 0.9929563 , 0.99315649, 0.99213184, 0.99327574,
        0.97680969]),
 array([0.96652641, 0.99178289, 0.99201687, 0.99082114, 0.99215562,
        0.97294719]),
 array([0.96174702, 0.99060966, 0.99087753, 0.98951062, 0.99103567,
        0.96908544]),
 array([0.95696829, 0.98943658, 0.98973848, 0.98820031, 0.9899159 ,
        0.96522446]),
 array([0.95219025, 0.98826368, 0.98859973, 0.9868902 , 0.98879631,
```

In [ ]: