In [1]:
```
import pandas as pd
```

In [2]:
```
url = "Credit_Risk_Prediction.csv"
data = pd.read_csv(url)
data
```

Out[2]:

	Age	Loan	MonthlyPaid	Gender	Credit Risk Prediction
0	25	586000	2250	m	Yes
1	63	869000	4950	f	No
2	49	665000	4500	m	Yes
3	66	245000	4700	m	No
4	21	282000	1000	f	Yes
...	...	...	...	...	...
294	62	662000	600	f	No
295	40	195000	3850	f	Yes
296	22	215000	1000	f	Yes
297	59	349000	2150	m	No
298	34	533000	3150	f	Yes

299 rows × 5 columns

In [3]:
```
#Convert the unit of Loan and MonthlyPaid to (k)
data["Loan"] = data["Loan"] / 1000
data["MonthlyPaid"] = data["MonthlyPaid"] / 1000
```

In [4]:
```
data["Gender"] = data["Gender"].map({"m":1 , "f":0})
data["Credit Risk Prediction"] = data["Credit Risk Prediction"].map({"Yes":1 , "No"
```

In [5]:
```
from sklearn.model_selection import train_test_split

x = data[["Age","Loan","MonthlyPaid","Gender"]]
y = data["Credit Risk Prediction"]

x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2,random_state=87)
x_train = x_train.to_numpy()
x_test = x_test.to_numpy()
```

Feature Scaler

In [6]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
```

```
scaler.fit(x_train)
x_train = scaler.transform(x_train)
x_test = scaler.transform(x_test)
```

In [7]:
```
import numpy as np
```

In [8]:
```
w = np.array([1,2,3,4])
b = 1
```

In [9]:
```
def sigmoid(z):
    return 1/(1+np.exp(-z))
```

In [11]:
```
# y_pred = (w * x_train).sum(axis = 1) + b
z = (w * x_train).sum(axis = 1) + b

sigmoid(z)
```

Out[11]:
```
array([9.99746762e-01, 8.79277770e-01, 9.57586723e-01, 5.63990514e-03,
       6.62933641e-01, 9.94401452e-01, 9.99685627e-01, 9.85627444e-01,
       3.03202418e-02, 1.46912377e-03, 3.58665440e-03, 9.94819355e-01,
       4.77775989e-01, 8.00938174e-01, 9.99974360e-01, 5.12543280e-01,
       3.11106489e-02, 9.99997786e-01, 7.68328909e-01, 9.59556870e-01,
       7.69492546e-01, 9.50523810e-01, 4.77907459e-02, 9.99855398e-01,
       9.45187751e-01, 9.99999006e-01, 6.92083093e-01, 9.98493887e-01,
       1.53110462e-02, 9.89785957e-01, 2.38305508e-01, 2.26852978e-01,
       9.99897426e-01, 6.48460841e-01, 9.86194641e-01, 2.63321341e-01,
       7.77172131e-02, 3.38939487e-01, 8.90133248e-01, 9.99980645e-01,
       9.98495219e-01, 1.88875537e-02, 9.68547667e-01, 5.75954750e-03,
       4.40748702e-01, 9.33133328e-01, 9.69420605e-01, 9.94509348e-01,
       9.96934886e-01, 9.95206129e-01, 8.99889176e-05, 5.86536504e-03,
       4.34843513e-01, 9.98454253e-01, 7.16122568e-01, 9.97730791e-01,
       2.93111212e-02, 3.27139771e-02, 9.99144962e-01, 3.94554615e-02,
       1.15083208e-03, 5.03710317e-02, 9.99935258e-01, 9.98754039e-01,
       7.13515124e-04, 9.88792380e-01, 9.99992462e-01, 9.96881225e-01,
       1.55814131e-01, 9.97565581e-01, 2.60744285e-04, 6.37214306e-01,
       1.24691932e-03, 2.76215049e-01, 1.82280727e-01, 3.60381944e-01,
       9.32905308e-01, 1.04840399e-02, 3.09907933e-04, 9.99612984e-01,
       7.12487175e-01, 9.99894788e-01, 6.20724279e-06, 8.43311204e-04,
       9.99943210e-01, 9.98871382e-01, 1.14180323e-01, 7.56988000e-02,
       5.33908216e-02, 9.98325967e-01, 1.22432263e-02, 9.99984228e-01,
       9.94464439e-01, 2.68684563e-01, 4.05597606e-01, 2.22388874e-05,
       9.98724788e-01, 9.99276446e-01, 9.89312841e-01, 7.26130596e-01,
       9.87947241e-01, 9.99839463e-01, 5.45704576e-03, 7.38718849e-05,
       1.63590091e-04, 9.95438910e-01, 7.98635042e-02, 9.92998946e-01,
       4.48914029e-02, 6.37859082e-01, 2.01445625e-01, 4.47689140e-03,
       9.95507205e-01, 5.30421793e-01, 9.98456731e-01, 9.16436651e-01,
       2.82403962e-03, 9.29822693e-01, 2.85005414e-02, 9.98961873e-01,
       1.56711902e-03, 1.23704798e-02, 5.04035521e-05, 8.42330924e-01,
       9.99652876e-01, 6.49660256e-04, 9.96119243e-01, 9.99525604e-01,
       6.76932627e-04, 7.44594405e-02, 9.98918592e-01, 6.36048346e-03,
       9.99736668e-01, 9.99991855e-01, 6.96142036e-02, 7.81548775e-01,
       9.96832296e-01, 6.14317840e-02, 2.35229771e-02, 1.34752739e-04,
       9.99543796e-01, 3.91624095e-04, 1.26255756e-01, 3.77418168e-01,
       9.99900578e-01, 9.99996671e-01, 2.62861508e-03, 9.96519439e-01,
       2.01781256e-02, 9.94828506e-01, 9.99962596e-01, 1.81377065e-02,
       9.68471341e-01, 9.99995357e-01, 2.07498085e-02, 9.99983209e-01,
       9.99987040e-01, 4.95599377e-01, 9.99868961e-01, 6.58493104e-01,
       9.82429360e-01, 1.51109160e-03, 9.10739228e-01, 9.45751817e-02,
       9.10884546e-01, 1.43996535e-05, 1.83219393e-02, 9.99979787e-01,
       2.45130652e-05, 1.43312142e-02, 9.84594107e-01, 9.98640759e-01,
       6.43077066e-01, 3.39790815e-01, 9.93878951e-01, 9.77417528e-01,
       9.86813575e-01, 9.55730816e-01, 9.01532648e-01, 3.75531033e-02,
       6.99969668e-01, 5.15866682e-03, 9.99954421e-01, 6.87858246e-01,
       9.45063767e-03, 7.13685360e-02, 9.99968458e-01, 6.54695994e-01,
       9.21526622e-01, 3.46011835e-03, 8.84330178e-01, 9.68409352e-01,
       6.76225943e-03, 3.74588338e-02, 7.38649760e-03, 9.59824974e-01,
       6.46502446e-05, 1.39443525e-03, 9.64219237e-01, 4.78293064e-01,
       2.26050848e-01, 9.99973410e-01, 4.56385322e-03, 9.89355605e-01,
       5.13422430e-04, 3.76804755e-01, 2.33205088e-01, 9.99561352e-01,
       9.78651882e-01, 9.50466049e-03, 8.03151654e-01, 9.63040599e-01,
       9.98862528e-01, 7.88222439e-01, 9.98929410e-01, 5.12505939e-01,
       1.28971683e-04, 6.62787333e-03, 5.54051469e-01, 1.12907626e-01,
       9.99158514e-01, 3.33685189e-05, 9.39622574e-01, 9.88924796e-01,
```

```
       9.75546130e-02, 9.46616916e-01, 9.99943552e-01, 9.44403774e-01,
       2.50382395e-02, 6.20952325e-03, 4.06877562e-01, 5.04876916e-02,
       9.99789019e-01, 6.27774473e-01, 6.54187940e-01, 5.29215155e-05,
       9.16333183e-01, 9.54473888e-01, 9.98384949e-01])
```

Binary Cross Entropy

In [12]:
```
# cost-> when y = 1 , -log(y_pred)
# cost-> when y = 0 , -log(1 - y_pred)

# cost = -y * log(y_pred) - (1-y)*log(1-y_pred)
y_pred = sigmoid(z)
cost = -y_train * np.log(y_pred) - (1-y_train)*np.log(1-y_pred)
cost.mean()
```

Out[12]:
```
np.float64(2.527873380328441)
```

In [13]:
```
def compute_cost(x,y,w,b):
    z = (w * x).sum(axis = 1) + b
    y_pred = sigmoid(z)
    cost = -y * np.log(y_pred) - (1-y)*np.log(1-y_pred)
    cost = cost.mean()

    return cost
```

In [14]:
```
compute_cost(x_train,y_train,w,b)
```

Out[14]:
```
np.float64(2.527873380328441)
```

Gradient Descent

In [16]:
```
# w1_gradient = 2*x1*(w*x+b - y)
#              = 2*x1*(y_pred - y)
```

In [18]:
```
w = np.array([1,2,3,4])
b = 1
z = (w * x_train).sum(axis = 1) + b
y_pred = sigmoid(z)
w_gradient = np.zeros(x_train.shape[1])
b_gradient = (y_pred - y_train).mean()

for i in range(x_train.shape[1]):
    w_gradient[i] = (2 * x_train[:,i] * (y_pred - y_train)).mean()

w_gradient,b_gradient
```

Out[18]:
```
(array([0.73190218, 0.56026874, 0.13089824, 0.5866118 ]),
 np.float64(0.045103162866522335))
```

In [19]:
```
def compute_gradient(x,y,w,b):
    z = (w * x).sum(axis = 1) + b
    y_pred = sigmoid(z)
```

```
    w_gradient = np.zeros(x.shape[1])
    b_gradient = (y_pred - y).mean()

    for i in range(x_train.shape[1]):
        w_gradient[i] = (2 * x[:,i] * (y_pred - y)).mean()

    return w_gradient,b_gradient
```

In [20]:
```
compute_gradient(x_train,y_train,w,b)
```

Out[20]:
```
(array([0.73190218, 0.56026874, 0.13089824, 0.5866118 ]),
 np.float64(0.045103162866522335))
```

In [24]:
```
w = np.array([1,2,3,4])
b = 1
learning_rate = 1

w_gradient,b_gradient = compute_gradient(x_train,y_train,w,b)
print(compute_cost(x_train,y_train,w,b))

w = w - w_gradient * learning_rate
b = b - b_gradient * learning_rate

print(compute_cost(x_train,y_train,w,b))
```

```
2.527873380328441
1.935758241659671
```

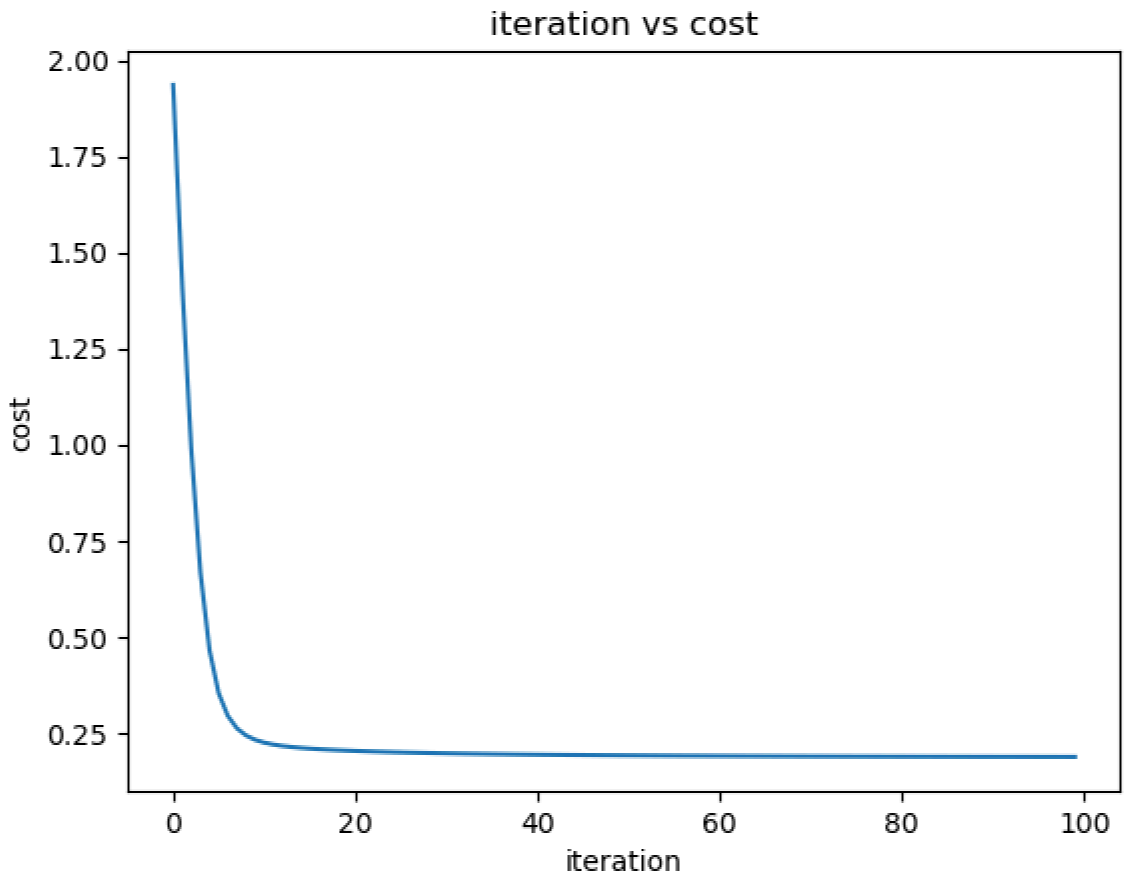
In [25]:
```
def gradient_descent(x,y,w_init,b_init,cost_function,gradient_function,learning_rat
    c_record = []
    w_record = []
    b_record = []

    w = w_init
    b = b_init

    for i in range(run_iter):
        w_gradient,b_gradient = compute_gradient(x,y,w,b)
        w = w - w_gradient * learning_rate
        b = b - b_gradient * learning_rate
        cost = compute_cost(x,y,w,b)

        c_record.append(cost)
        w_record.append(w)
        b_record.append(b)

        if i % p_iter == 0:
            print(f"Iteration {i:5}: Cost {cost:.2f}: w {w}: b {b:.2f} :w_gradient
    return w,b,c_record,w_record,b_record
```

In [27]:
```
w_init = np.array([1,2,3,4])
b_init = 1
learning_rate = 1
run_iter = 10000

w_final,b_final,c_record,w_record,b_record = gradient_descent(x_train,y_train,w_ini
```

```
Iteration     0: Cost 1.94: w [0.26809782 1.43973126 2.86910176 3.4133882 ]: b 0.95
:w_gradient [0.73190218 0.56026874 0.13089824 0.5866118 ]: b_gradient 0.05
Iteration  1000: Cost 0.19: w [-4.47976308 -2.64655716  2.64581782  0.71832414]: b
0.04 :w_gradient [ 1.79417840e-07  1.19112916e-07 -1.17409796e-07 -2.54335334e-08]:
b_gradient 0.00
Iteration  2000: Cost 0.19: w [-4.47977905 -2.64656776  2.64582828  0.7183264 ]: b
0.04 :w_gradient [ 2.52596874e-12  1.67693892e-12 -1.65292048e-12 -3.58061326e-13]:
b_gradient 0.00
Iteration  3000: Cost 0.19: w [-4.47977905 -2.64656776  2.64582828  0.7183264 ]: b
0.04 :w_gradient [ 4.36656754e-16  2.07295826e-16 -1.96031011e-16 -3.43751062e-17]:
b_gradient 0.00
Iteration  4000: Cost 0.19: w [-4.47977905 -2.64656776  2.64582828  0.7183264 ]: b
0.04 :w_gradient [ 4.36656754e-16  2.07295826e-16 -1.96031011e-16 -3.43751062e-17]:
b_gradient 0.00
Iteration  5000: Cost 0.19: w [-4.47977905 -2.64656776  2.64582828  0.7183264 ]: b
0.04 :w_gradient [ 4.36656754e-16  2.07295826e-16 -1.96031011e-16 -3.43751062e-17]:
b_gradient 0.00
Iteration  6000: Cost 0.19: w [-4.47977905 -2.64656776  2.64582828  0.7183264 ]: b
0.04 :w_gradient [ 4.36656754e-16  2.07295826e-16 -1.96031011e-16 -3.43751062e-17]:
b_gradient 0.00
Iteration  7000: Cost 0.19: w [-4.47977905 -2.64656776  2.64582828  0.7183264 ]: b
0.04 :w_gradient [ 4.36656754e-16  2.07295826e-16 -1.96031011e-16 -3.43751062e-17]:
b_gradient 0.00
Iteration  8000: Cost 0.19: w [-4.47977905 -2.64656776  2.64582828  0.7183264 ]: b
0.04 :w_gradient [ 4.36656754e-16  2.07295826e-16 -1.96031011e-16 -3.43751062e-17]:
b_gradient 0.00
Iteration  9000: Cost 0.19: w [-4.47977905 -2.64656776  2.64582828  0.7183264 ]: b
0.04 :w_gradient [ 4.36656754e-16  2.07295826e-16 -1.96031011e-16 -3.43751062e-17]:
b_gradient 0.00
```

In [28]:

```python
import matplotlib.pyplot as plt
import numpy as np

# plt.plot(np.arange(0,20000),c_hist)
plt.plot(np.arange(0,100),c_record[:100])
plt.title("iteration vs cost")
plt.xlabel("iteration")
plt.ylabel("cost")
plt.show()
```



In [29]:
```
z = (w_final * x_test).sum(axis=1) + b
y_pred = sigmoid(z)
y_pred
```

Out[29]:
```
array([6.58568640e-01, 7.71853697e-04, 2.79875044e-03, 4.47956102e-06,
       9.97377090e-01, 1.23771340e-01, 9.95664661e-01, 9.99990372e-01,
       2.98632593e-01, 9.50302519e-02, 1.12614274e-02, 9.99781379e-01,
       1.42645207e-02, 9.94754096e-01, 7.30841403e-01, 1.32404580e-05,
       1.07233493e-04, 9.99472624e-01, 9.99954840e-01, 9.96380743e-01,
       6.64077317e-01, 4.99788163e-02, 5.06884222e-02, 9.68489725e-01,
       2.78896599e-03, 8.40149783e-01, 9.99787868e-01, 9.98325439e-01,
       9.24694710e-01, 7.98512514e-02, 4.73730812e-06, 9.99997554e-01,
       9.95670190e-01, 9.99949057e-01, 4.92431371e-03, 9.04189843e-03,
       7.18926197e-02, 9.99967136e-01, 9.98461130e-01, 9.92944327e-01,
       1.13353679e-04, 7.49687714e-05, 9.98975476e-01, 9.95351770e-01,
       9.75037728e-01, 9.96044544e-01, 9.99745558e-01, 2.59098334e-01,
       9.95680166e-01, 3.32259819e-02, 9.98422809e-01, 3.91755946e-01,
       9.94469776e-01, 6.41516274e-02, 9.85849895e-01, 7.88053827e-01,
       4.98763311e-02, 9.51600917e-01, 1.14275428e-02, 2.62034667e-03])
```

In [30]:
```
y_pred = np.where(y_pred>0.5,1,0)
y_pred
```

Out[30]:
```
array([1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0,
       0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1,
       1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0])
```

In [31]:
```
(y_pred == y_test).sum() / len(y_test)
```

Out[31]: np.float64(0.9833333333333333)

In [33]:
```
acc = (y_pred == y_test).sum() / len(y_test) * 100
print(f"Accuracy: {acc:.2f} %")
```

```
Accuracy: 98.33 %
```

Check data here

In [52]:
```
age = 65
loan = 300000 / 1000 #unit(k)
monthlypaid = 3500 / 1000 #unit(k)
gender = 0  #female

x_realData = np.array([[age,loan,monthlypaid,gender]])
x_realData = scaler.transform(x_realData)
y_realData = (x_realData * w_final).sum(axis = 1) + b_final
y_realData = sigmoid(y_realData)

if y_realData > 0.5:
    print("Congratulations, you may get the loan")
else:
    print("Sorry, your loan are rejected")
```

```
Sorry, your loan are rejected
```

In [ ]: