In [83]:
```
import numpy as np
import tensorflow as tf
from tensorflow import keras
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

In [84]:
```
df=pd.read_csv('downloads/insurance_data.csv')
df.head()
```

Out[84]:

	age	affordability	bought_insurance
0	22	1	0
1	25	0	1
2	47	1	0
3	52	1	1
4	46	1	1

Preprocessing

Scaling

In [85]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(df[['age','affordability']],df.bought_insurance,test_size=0.2,ran
```

In [39]:
```
len(X_train)
```

Out[39]: 21

In [6]:
```
X_train
```

Out[6]:

	age	affordability
0	22	1
13	29	1
6	55	1
17	58	1
24	50	1
19	18	0
25	54	1
16	25	1
20	21	1
3	52	1
7	60	1
1	25	0
5	56	0
26	23	1
8	62	0
18	19	1
12	27	0
23	45	1
22	40	0
15	55	1
4	46	1

In [40]:
```
X_test
```

Out[40]:

	age	affordability
2	47	1
10	18	1
21	26	1
11	28	1
14	46	0
9	61	1

In [41]:
```
y_train
```

Out[41]:
```
0     0
13    1
6     1
17    0
24    1
19    0
25    1
16    0
20    0
3     1
7     0
1     1
5     0
26    0
8     1
18    1
12    0
23    1
22    0
15    1
4     1
Name: bought_insurance, dtype: int64
```

In [42]:
```
y_test
```

Out[42]:
```
2     0
10    1
21    0
11    0
14    1
9     1
Name: bought_insurance, dtype: int64
```

In [10]:
```
# Scaling the age you can divide the age by 100
X_train_scaled=X_train.copy()
X_train_scaled['age']=X_train_scaled['age']/100
```

```
X_test_scaled=X_test.copy()
X_test_scaled['age']=X_test_scaled['age']/100
```

In [11]:
```
X_train_scaled
```

Out[11]:

	age	affordability
0	0.22	1
13	0.29	1
6	0.55	1
17	0.58	1
24	0.50	1
19	0.18	0
25	0.54	1
16	0.25	1
20	0.21	1
3	0.52	1
7	0.60	1
1	0.25	0
5	0.56	0
26	0.23	1
8	0.62	0
18	0.19	1
12	0.27	0
23	0.45	1
22	0.40	0
15	0.55	1
4	0.46	1

In [50]:
```
X_test_scaled
```

Out[50]:

	age	affordability
2	0.47	1
10	0.18	1
21	0.26	1
11	0.28	1
14	0.46	0
9	0.61	1

Creating a tensorflow model

In [133…
```
model=keras.Sequential([
    keras.layers.Dense(1,input_shape=(2,),activation='sigmoid',kernel_initializer='ones',bias_initializer='zeros
])

model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
model.fit(X_train_scaled,y_train,epochs=50)
```

```
Epoch 1/50
1/1 [==============================] - 1s 818ms/step - loss: 0.7596 - accuracy: 0.5238
Epoch 2/50
1/1 [==============================] - 0s 11ms/step - loss: 0.7592 - accuracy: 0.5238
Epoch 3/50
1/1 [==============================] - 0s 18ms/step - loss: 0.7587 - accuracy: 0.5238
Epoch 4/50
1/1 [==============================] - 0s 11ms/step - loss: 0.7583 - accuracy: 0.5238
Epoch 5/50
1/1 [==============================] - 0s 15ms/step - loss: 0.7579 - accuracy: 0.5238
Epoch 6/50
1/1 [==============================] - 4s 4s/step - loss: 0.7574 - accuracy: 0.5238
Epoch 7/50
1/1 [==============================] - 0s 43ms/step - loss: 0.7570 - accuracy: 0.5238
Epoch 8/50
1/1 [==============================] - 0s 8ms/step - loss: 0.7566 - accuracy: 0.5238
Epoch 9/50
1/1 [==============================] - 0s 8ms/step - loss: 0.7562 - accuracy: 0.5238
Epoch 10/50
1/1 [==============================] - 0s 8ms/step - loss: 0.7557 - accuracy: 0.5238
Epoch 11/50
1/1 [==============================] - 0s 11ms/step - loss: 0.7553 - accuracy: 0.5238
Epoch 12/50
1/1 [==============================] - 0s 8ms/step - loss: 0.7549 - accuracy: 0.5238
Epoch 13/50
1/1 [==============================] - 0s 7ms/step - loss: 0.7545 - accuracy: 0.5238
Epoch 14/50
1/1 [==============================] - 0s 7ms/step - loss: 0.7540 - accuracy: 0.5238
Epoch 15/50
1/1 [==============================] - 0s 8ms/step - loss: 0.7536 - accuracy: 0.5238
Epoch 16/50
1/1 [==============================] - 0s 8ms/step - loss: 0.7532 - accuracy: 0.5238
Epoch 17/50
1/1 [==============================] - 0s 14ms/step - loss: 0.7528 - accuracy: 0.5238
Epoch 18/50
1/1 [==============================] - 0s 10ms/step - loss: 0.7524 - accuracy: 0.5238
Epoch 19/50
1/1 [==============================] - 0s 8ms/step - loss: 0.7519 - accuracy: 0.5238
```

```
Epoch 20/50
1/1 [==============================] - 0s 8ms/step - loss: 0.7515 - accuracy: 0.5238
Epoch 21/50
1/1 [==============================] - 0s 7ms/step - loss: 0.7511 - accuracy: 0.5238
Epoch 22/50
1/1 [==============================] - 0s 9ms/step - loss: 0.7507 - accuracy: 0.5238
Epoch 23/50
1/1 [==============================] - 0s 8ms/step - loss: 0.7503 - accuracy: 0.5238
Epoch 24/50
1/1 [==============================] - 0s 7ms/step - loss: 0.7499 - accuracy: 0.5238
Epoch 25/50
1/1 [==============================] - 0s 8ms/step - loss: 0.7495 - accuracy: 0.5238
Epoch 26/50
1/1 [==============================] - 0s 8ms/step - loss: 0.7491 - accuracy: 0.5238
Epoch 27/50
1/1 [==============================] - 0s 10ms/step - loss: 0.7487 - accuracy: 0.5238
Epoch 28/50
1/1 [==============================] - 0s 10ms/step - loss: 0.7482 - accuracy: 0.5238
Epoch 29/50
1/1 [==============================] - 0s 13ms/step - loss: 0.7478 - accuracy: 0.5238
Epoch 30/50
1/1 [==============================] - 0s 11ms/step - loss: 0.7474 - accuracy: 0.5238
Epoch 31/50
1/1 [==============================] - 0s 17ms/step - loss: 0.7470 - accuracy: 0.5238
Epoch 32/50
1/1 [==============================] - 0s 8ms/step - loss: 0.7466 - accuracy: 0.5238
Epoch 33/50
1/1 [==============================] - 0s 7ms/step - loss: 0.7462 - accuracy: 0.5238
Epoch 34/50
1/1 [==============================] - 0s 7ms/step - loss: 0.7458 - accuracy: 0.5238
Epoch 35/50
1/1 [==============================] - 0s 8ms/step - loss: 0.7454 - accuracy: 0.5238
Epoch 36/50
1/1 [==============================] - 0s 9ms/step - loss: 0.7450 - accuracy: 0.5238
Epoch 37/50
1/1 [==============================] - 0s 8ms/step - loss: 0.7446 - accuracy: 0.5238
Epoch 38/50
1/1 [==============================] - 0s 7ms/step - loss: 0.7442 - accuracy: 0.5238
```

```
Epoch 39/50
1/1 [==============================] - 0s 8ms/step - loss: 0.7438 - accuracy: 0.5238
Epoch 40/50
1/1 [==============================] - 0s 9ms/step - loss: 0.7434 - accuracy: 0.5238
Epoch 41/50
1/1 [==============================] - 0s 9ms/step - loss: 0.7430 - accuracy: 0.5238
Epoch 42/50
1/1 [==============================] - 0s 7ms/step - loss: 0.7427 - accuracy: 0.5238
Epoch 43/50
1/1 [==============================] - 0s 7ms/step - loss: 0.7423 - accuracy: 0.5238
Epoch 44/50
1/1 [==============================] - 0s 7ms/step - loss: 0.7419 - accuracy: 0.5238
Epoch 45/50
1/1 [==============================] - 0s 7ms/step - loss: 0.7415 - accuracy: 0.5238
Epoch 46/50
1/1 [==============================] - 0s 7ms/step - loss: 0.7411 - accuracy: 0.5238
Epoch 47/50
1/1 [==============================] - 0s 8ms/step - loss: 0.7407 - accuracy: 0.5238
Epoch 48/50
1/1 [==============================] - 0s 7ms/step - loss: 0.7403 - accuracy: 0.5238
Epoch 49/50
1/1 [==============================] - 0s 7ms/step - loss: 0.7399 - accuracy: 0.5238
Epoch 50/50
1/1 [==============================] - 0s 6ms/step - loss: 0.7396 - accuracy: 0.5238
```

Out[133]:
```
<keras.callbacks.History at 0x23743d28ee0>
```

In [52]:
```
model.evaluate(X_test_scaled,y_test)
```

```
1/1 [==============================] - 0s 25ms/step - loss: 0.7489 - accuracy: 0.3333
```

Out[52]:
```
[0.7488711476325989, 0.3333333432674408]
```

In [24]:
```
X_test_scaled
```

Out[24]:

	age	affordability
2	0.47	1
10	0.18	1
21	0.26	1
11	0.28	1
14	0.46	0
9	0.61	1

In [53]:
```
model.predict(X_test_scaled)           # 5/6 instance this model predicted well
```

Out[53]:
```
1/1 [==============================] - 0s 82ms/step
array([[0.6509947 ],
       [0.42012927],
       [0.48466185],
       [0.5009615 ],
       [0.39733383],
       [0.7464823 ]], dtype=float32)
```

In [54]:
```
y_test
```

Out[54]:
```
2     0
10    1
21    0
11    0
14    1
9     1
Name: bought_insurance, dtype: int64
```

In [121…
```
#final weights of w1, w2 , bias
coef,intercept=model.get_weights()
coef,intercept
```

Out[121]:
```
(array([[3.2608867],
        [1.0073922]], dtype=float32),
 array([-1.9165945], dtype=float32))
```

In [56]:
```
# implement the neural network without using tensorflow

def sigmoid(x):
    import math
    return 1/(1+math.exp(-x))
sigmoid(8)
```

Out[56]:
```
0.9996646498695336
```

In [122…
```
# Predict the result which almost similar to the result of keras
def prediction_function(age,affordability):
    weighted_sum=coef[0]*age+coef[1]*affordability+intercept
    return sigmoid(weighted_sum)
```

In [58]:
```
prediction_function(.47,1)
```

Out[58]:
```
0.6509947281565442
```

In [59]:
```
prediction_function(.18,1)
```

Out[59]:
```
0.42012927315172927
```

Gradient descent function

In [60]:
```
def log_loss(y_true,y_predicted):
    epsilon=1e-15
    y_predicted_new=[max(i,epsilon) for i in y_predicted]
    y_predicted_new=[min(i,1-epsilon) for i in y_predicted_new]
    y_predicted_new=np.array(y_predicted_new)
    return -np.mean(y_true*np.log(y_predicted_new)+(1-y_true)*np.log(1-y_predicted_new))
```

In [61]:
```
def sigmoid_numpy(X):
    return 1/(1+np.exp(-X))
sigmoid_numpy(np.array([12,0,1]))
```

Out[61]:
```
array([0.99999386, 0.5       , 0.73105858])
```

Write a neural network code to compare with keras results of insurance data

In [113…
```
class myNN:
    def __init__(self):
        self.w1=1
        self.w2=1
        self.bias=0

    def fit(self,X,y,epochs,loss_thresold):
        self.w1,self.w2,self.bias=self.gradient_descent(X['age'],X['affordability'],y,epochs,loss_thresold)

    def predict(self,X_test):
        weighted_sum= self.w1*X_test['age']+self.w2*X_test['affordability']+self.bias
        return sigmoid_numpy(weighted_sum)

    def gradient_descent(self,age,affordability,y_true,epochs,loss_thresold):
    #we,w2,bias
        w1=w2=1
        bias=0
        rate=0.5
        n=len(age)

        for i in range(epochs):
            wieghted_sum=w1*age+w2*affordability+bias
```

```
            y_predicted=sigmoid_numpy(wieghted_sum)
            loss=log_loss(y_true,y_predicted)
            w1d=(1/n)*np.dot(np.transpose(age),(y_predicted-y_true))
            w2d=(1/n)*np.dot(np.transpose(affordability),(y_predicted-y_true))

            bias_d=np.mean(y_predicted-y_true)

            w1=w1-rate*w1d
            w2=w2-rate*w2d
            bias=bias-rate*bias_d

            if i%50==0:
                print(f'Epoch:{i},w1:{w1},w2:{w2},bias:{bias},loss:{loss}')

            if loss<=loss_thresold:
                break
        return w1,w2,bias
```

In [72]:

In [123…
```
customModel=myNN()
customModel.fit(X_train_scaled, y_train,epochs=500,loss_thresold=0.6184)
```

```
Epoch:0,w1:0.9653985199198097,w2:0.9276540870265453,bias:-0.10947703314470893,loss:0.759599300965634
Epoch:50,w1:1.1010960404071617,w2:0.7970280273172278,bias:-0.8740786970777803,loss:0.6431810945906878
Epoch:100,w1:1.4114417038747153,w2:0.9003588399367809,bias:-1.0844691264418924,loss:0.6370565188162108
Epoch:150,w1:1.6991149232822982,w2:0.9498592101711977,bias:-1.2417757610281204,loss:0.6326286930626427
Epoch:200,w1:1.9598741387130207,w2:0.9756195759081225,bias:-1.3686746014130726,loss:0.6292203352498184
Epoch:250,w1:2.193823135418373,w2:0.9907212007064669,bias:-1.4758896512729063,loss:0.6265492108387684
Epoch:300,w1:2.4027474601115335,w2:1.0009237112032021,bias:-1.5688777809870271,loss:0.6244433021014762
Epoch:350,w1:2.5889724735558826,w2:1.0087725143774808,bias:-1.6506876567161701,loss:0.6227782161862067
Epoch:400,w1:2.754877958714612,w2:1.0153901212859267,bias:-1.723221159211406,loss:0.6214589829929645
Epoch:450,w1:2.902702951976107,w2:1.0212690095462507,bias:-1.7878026078077278,loss:0.6204119206442096
```

In [124…
```
customModel.predict(X_test_scaled)
```

Out[124]:
```
2     0.647303
10    0.432399
21    0.492622
11    0.507781
14    0.389447
9     0.737242
dtype: float64
```

In [125…
```
coef, intercept
```

Out[125]:
```
(array([[3.2608867],
        [1.0073922]], dtype=float32),
 array([-1.9165945], dtype=float32))
```

In [131…
```
# Write a gradient descent function to compare the result with keras result
```

In [127…
```
def gradient_descent(age,affordability,y_true,epochs,loss_thresold):
    #we,w2,bias
    w1=w2=1
    bias=0
    rate=0.5
    n=len(age)

    for i in range(epochs):
        wieghted_sum=w1*age+w2*affordability+bias
        y_predicted=sigmoid_numpy(wieghted_sum)
        loss=log_loss(y_true,y_predicted)
        w1d=(1/n)*np.dot(np.transpose(age),(y_predicted-y_true))
        w2d=(1/n)*np.dot(np.transpose(affordability),(y_predicted-y_true))

        bias_d=np.mean(y_predicted-y_true)

        w1=w1-rate*w1d
        w2=w2-rate*w2d
        bias=bias-rate*bias_d
        print(f'Epoch:{i},w1:{w1},w2:{w2},bias:{bias},loss:{loss}')

        if loss<=loss_thresold:
```

```
            break
    return w1,w2,bias
```

In [132…]
```
gradient_descent(X_train_scaled['age'],X_train_scaled['affordability'],y_train,10,0.6184)
```

```
Epoch:0,w1:0.9653985199198097,w2:0.9276540870265453,bias:-0.10947703314470893,loss:0.759599300965634
Epoch:1,w1:0.9370979826487914,w2:0.8669835961266957,bias:-0.20303988713488086,loss:0.7255269363244933
Epoch:2,w1:0.914586250285506,w2:0.8172046369298321,bias:-0.2820493834042744,loss:0.7012153994623181
Epoch:3,w1:0.8972039012132778,w2:0.7771761245929059,bias:-0.34820967447656526,loss:0.6843627680774081
Epoch:4,w1:0.8842361344719337,w2:0.7455936187977191,bias:-0.40334046819548475,loss:0.6729367732776856
Epoch:5,w1:0.8749831969554591,w2:0.7211433292672716,bias:-0.44920616272476555,loss:0.6653073606163848
Epoch:6,w1:0.868803942191361,w2:0.702601076620092,bias:-0.48741269952300087,loss:0.6602556471933526
Epoch:7,w1:0.8651365336399749,w2:0.688882739263888,bias:-0.5193606432233379,loss:0.6569154612350001
Epoch:8,w1:0.8635034321621546,w2:0.6790605442448461,bias:-0.5462362053147177,loss:0.6546934287642662
Epoch:9,w1:0.8635072226108902,w2:0.6723589261731341,bias:-0.5690239398443738,loss:0.6531935725224463
```

Out[132]:
```
(0.8635072226108902, 0.6723589261731341, -0.5690239398443738)
```

In [130…]
```
coef, intercept
```

Out[130]:
```
(array([[3.2608867],
        [1.0073922]], dtype=float32),
 array([-1.9165945], dtype=float32))
```

We could be able to replicate the tensor flow values with a plain python code as below tensor flow- w1=3.2608867 w2=1.0073922 bias=1.9165945 plain python - w1=3.2549412168887453 w2=1.0360136437098828 bias=1.9421720797036695