Import data¶

In [222]:

```
import pandas as pd
path="C:\\Users\\TANUJA HARISH\\Desktop\\ML and DL Summer
Internship\\50_Startups.csv"
dataset=pd.read_csv(path)
dataset.head()
```

Out[222]:

	R&D Spend	Administration	Marketing Spend	State	Profit
0	165349.20	136897.80	471784.10	New York	192261.83
1	162597.70	151377.59	443898.53	California	191792.06
2	153441.51	101145.55	407934.54	Florida	191050.39
3	144372.41	118671.85	383199.62	New York	182901.99
4	142107.34	91391.77	366168.42	Florida	166187.94

In [223]:

```
dataset.shape
```

Out[223]:

```
(50, 5)
```

Visualize the data¶

In [224]:

```
import seaborn as sns
sns.pairplot(dataset)
```

Out[224]:

```
<seaborn.axisgrid.PairGrid at 0x1a0e1260eb0>
```

Split into x and y¶

In [225]:

```
import numpy as np
x=np.array(dataset.iloc[:,0:3])
```

```
y=np.array(dataset[["Profit"]])
print(x.shape)
print(y.shape)
```

```
(50, 3)
(50, 1)
```

standardize the dataset¶

In [226]:

```
# without using in-built lib
x_norm=(x-x.mean())/x.std()
y_norm=(y-y.mean())/y.std()
print(x_norm.shape)
print(y_norm.shape)
```

```
(50, 3)
(50, 1)
```

Split data into train and test¶

In [227]:

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=
train_test_split(x_norm,y_norm,test_size=0.2)
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(40, 3)
(10, 3)
(40, 1)
(10, 1)
```

In [228]:

```
def initialize_parameters(x):
    w1=np.random.randn(5,x.shape[1])
    w2=np.random.randn(1,5)
    b1=np.random.randn(5,1)
    b2=np.random.randn(1,1)
    return w1,w2,b1,b2
def sigmoid_act(x):
    s= 1/(1+np.exp(-x))
    return s
def forward_path(x_train,w1,b1,w2,b2):
    #Hidden layer
```

```
    z1=np.matmul(w1,x_train.T)+b1
    A1=sigmoid_act(z1)
    #output layer
    z2=np.matmul(w2,A1)+b2
    #Linear activation function at output layer
    A2=z2
    return A2,A1
def compute_cost(y_pred,y_train):
    cost=np.mean((y_pred-y_train)**2)
    return cost
def backward_path(A2,A1,w2,y_train,x_train):
    #output layer
    dA2=(2*(A2-y_train.T))/len(y_train) #gradient of loss wrt A2
    dz2=dA2 #gradient of loss wrt z2
    dw2=np.matmul(dz2,A1.T)
    db2=dz2.sum(axis=1)

    #first layer
    dA1=np.matmul(w2.T,dz2)
    dz1=np.multiply(dA1,A1*(1-A1))
    dw1=np.matmul(dz1,x_train)
    db1=dz1.sum(axis=1)
    return dw2,dw1,db2,db1
def update_parameters(w,b,dw,db,learning_rate):
    w=w-learning_rate*dw
    db=np.reshape(db,b.shape)
    b=b-learning_rate*db
    return w,b
```

Training uisng batch gradient descent¶

In [229]:

```
num_iter=500
learning_rate=0.001
his1=[]
w1,w2,b1,b2=initialize_parameters(x_train)
for i in range(num_iter):
    A2,A1=forward_path(x_train,w1,b1,w2,b2)
    y_pred=A2
    cost=compute_cost(y_pred,y_train)
    dw2,dw1,db2,db1=backward_path(A2,A1,w2,y_train,x_train)
    w2,b2=update_parameters(w2,b2,dw2,db2,learning_rate)
    w1,b1=update_parameters(w1,b1,dw1,db1,learning_rate)
    his1.append(cost)
```

In [230]:

```
import matplotlib.pyplot as plt
plt.plot(his1)
```

Out[230]:

```
[<matplotlib.lines.Line2D at 0x1a0e1dc08b0>]
```

In [231]:

```
print(w1)
print(b1)
```

```
[[ 0.51582139 -0.03898305 -0.83515171]
 [-2.25409591  1.60478305  0.79489813]
 [ 1.14970789  0.24402845  1.56439212]
 [ 0.62454372  0.12590519  0.32885814]
 [-0.24260171 -0.41469506  0.80190617]]
[[-1.07623212]
 [-0.68808689]
 [-0.02853039]
 [-1.46408837]
 [ 0.72554767]]
```

In [232]:

```
print(w2)
print(b2)
```

```
[[ 0.48945482 -0.3007637  -0.88792272 -0.4044172   0.53737571]]
[[0.25549305]]
```

Prediction¶

In [233]:

```
def pred(w1,w2,b1,b2,x_test):
    A2,A1=forward_path(x_test,w1,b1,w2,b2)
    return A2
```

In [234]:

```
y_pred=pred(w1,w2,b1,b2,x_test)
print(y_pred)
```

```
[[ 0.05662249 -0.53331337  0.2370734  -0.3344759  -0.41039638 -0.44264871
  -0.28249889  0.17567684 -0.02924356 -0.33872981]]
```

Rescale data¶

In [235]:

```
y_test_rescalled=(y_test*y.std())+y.mean()
y_pred_rescalled=(y_pred*y.std())+y.mean()
y_pred_rescalled=y_pred_rescalled.T
print(y_pred_rescalled.shape)
print(y_test_rescalled.shape)
```

```
(10, 1)
(10, 1)
```

Comparing test and predicted values¶

In [236]:

```
comp=pd.DataFrame(np.c_[y_test_rescalled,y_pred_rescalled],columns=["Origin
al profit","Predicted profit"])
print(comp)
```

```
   Original profit  Predicted profit
0         96479.51     114271.937951
1        192261.83      90732.858100
2         77798.83     121472.124614
3        108733.99      98666.688782
4        156991.12      95637.379313
5        191050.39      94350.476469
6        129917.04     100740.627453
7        105733.54     119022.335288
8         96778.92     110845.789436
9        124266.90      98496.953076
```

Error calculations¶

In [237]:

```
from sklearn.metrics import mean_squared_error,mean_absolute_error
from math import sqrt
print("MAE:",mean_absolute_error(y_test_rescalled,y_pred_rescalled))
print("RMSE:",np.sqrt(mean_squared_error(y_test_rescalled,y_pred_rescalled)
))
```

```
MAE: 41341.767409624954
RMSE: 52572.22339459676
```

Training using stochastic gradient descent¶

In [238]:

```
def initialize_parameters(x):
    w1=np.random.randn(5,x.shape[1])
    w2=np.random.randn(1,5)
    b1=np.random.randn(5,1)
    b2=np.random.randn(1,1)
```

```
    return w1,w2,b1,b2
def sigmoid_act(x):
    s= 1/(1+np.exp(-x))
    return s
def forward_path(x_train,w1,b1,w2,b2):
    #Hidden layer
    z1=np.reshape(np.matmul(w1,x_train.T),(b1.shape))+b1
    A1=sigmoid_act(z1)
    #output layer
    z2=np.matmul(w2,A1)+b2
    #Linear activation function at output layer
    A2=z2
    return A2,A1
def compute_cost(y_pred,y_train):
    cost=np.mean((y_pred-y_train)**2)
    return cost
def backward_path(A2,A1,w2,y_train,x_train):
    #output layer
    dA2=2*(A2-y_train.T) #gradient of loss wrt A2
    dz2=dA2 #gradient of loss wrt z2
    dw2=np.matmul(dz2,A1.T)
    db2=dz2.sum(axis=1)

    #first layer
    dA1=np.matmul(w2.T,dz2)
    dz1=np.multiply(dA1,A1*(1-A1))
    dw1=dz1*x_train
    db1=dz1.sum(axis=1)
    return dw2,dw1,db2,db1
def update_parameters(w,b,dw,db,learning_rate):
    w=w-learning_rate*dw
    db=np.reshape(db,b.shape)
    b=b-learning_rate*db
    return w,b
```

In [239]:

```
num_iter=500
learning_rate=0.001
his1=[]
w1,w2,b1,b2=initialize_parameters(x_train)
for i in range(num_iter):
    cost=0
    for j in range(len(x_train)):
        A2, A1 = forward_path(x_train[j],w1,b1,w2,b2)
        y_pred=A2
        cost+=compute_cost(y_pred,y_train[j])
        dw2,dw1,db2,db1=backward_path(A2,A1,w2,y_train[j],x_train[j])
        w2,b2=update_parameters(w2,b2,dw2,db2,learning_rate)
        w1,b1=update_parameters(w1,b1,dw1,db1,learning_rate)
    his1.append(cost/len(x_train))
```

In [240]:

```
plt.plot(his1)
```

Out[240]:

```
[<matplotlib.lines.Line2D at 0x1a0e1e28820>]
```

In [241]:

```
print(w1)
print(b1)
```

```
[[ 0.16027353  0.31090904  0.57133215]
 [-1.90550738 -0.84853485  0.15047891]
 [ 1.67378072 -0.17830375  0.10854694]
 [-0.36188786  0.10080839  0.18829129]
 [ 0.98946951  0.63507377 -0.05002533]]
[[ 1.1188528 ]
 [-1.27735928]
 [ 0.03157532]
 [ 0.92321531]
 [-1.45210957]]
```

In [242]:

```
print(w2)
print(b2)
```

```
[[ 2.13831301 -2.03246075  1.91064995  0.14787418  0.25138376]]
[[-1.31002193]]
```

Prediction¶

In [243]:

```
def pred2(w1,w2,b1,b2,x_test):
    y_pred=np.zeros(y_test.shape)
    for i in range(len(x_test)):
        A2,A1=forward_path(x_test[i],w1,b1,w2,b2)
        y_pred[i]=A2
    return y_pred
```

In [244]:

```
y_pred2=pred2(w1,w2,b1,b2,x_test)
print(y_pred2)
print(y_pred2.shape)
```

```
[[-0.64909131]
 [ 1.8761466 ]
 [-0.97209962]
 [-0.07339642]
 [ 1.13916227]
 [ 1.54953075]
 [ 0.80569327]
 [-0.02713373]
 [-0.41264801]
 [ 0.38370066]]
(10, 1)
```

rescalling data¶

In [245]:

```
y_test_rescalled=(y_test*y.std())+y.mean()
y_pred2_rescalled=(y_pred2*y.std())+y.mean()
print(y_pred2_rescalled.shape)
print(y_test_rescalled.shape)
```

```
(10, 1)
(10, 1)
```

Comparing test and predicted value¶

In [246]:

```
comp=pd.DataFrame(np.c_[y_test_rescalled,y_pred2_rescalled],columns=["Origi
nal profit","Predicted profit"])
print(comp)
```

```
   Original profit  Predicted profit
0         96479.51      86113.193146
1        192261.83     186872.920029
```

```
2         77798.83      73224.811955
3        108733.99     109084.042495
4        156991.12     157466.447129
5        191050.39     173840.594173
6        129917.04     144160.672939
7        105733.54     110929.973828
8         96778.92      95547.536613
9        124266.90     127322.711023
```

Error calculations¶

In [247]:

```
from sklearn.metrics import mean_squared_error,mean_absolute_error
from math import sqrt
print("MAE:",mean_absolute_error(y_test_rescalled,y_pred2_rescalled))
print("RMSE:",np.sqrt(mean_squared_error(y_test_rescalled,y_pred2_rescalled
)))
```

```
MAE: 6209.1681497174795
RMSE: 8334.784650980793
```

In [ ]:

In [ ]: