Import data¶

In [1]:

```
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[1]:

	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 [2]:

```
dataset.shape
```

Out[2]:

```
(50, 5)
```

Visualize the data¶

In [3]:

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

Out[3]:

```
<seaborn.axisgrid.PairGrid at 0x25cb1bcaf70>
```

Split into x and y¶

In [4]:

```
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 [5]:

```
# 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 into train and test¶

In [6]:

```
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 [10]:

```
def initialize_parameters(x):
    w1=np.random.randn(5,x.shape[1])
    w2=np.random.randn(5,5)
    w3=np.random.randn(1,5)
    b1=np.random.randn(5,1)
    b2=np.random.randn(5,1)
    b3=np.random.randn(1,1)
    return w1,w2,w3,b1,b2,b3
def sigmoid_act(x):
    s= 1/(1+np.exp(-x))
    return s
```

```python
def forward_path(x_train,w1,b1,w2,b2,w3,b3):
    #Hidden1 layer
    z1=np.matmul(w1,x_train.T)+b1
    A1=sigmoid_act(z1)
    #Hidden2 layer
    z2=np.matmul(w2,A1)+b2
    A2=sigmoid_act(z2)
    #output layer
    z3=np.matmul(w3,A2)+b3
    #Linear activation function at output layer
    A3=z3
    return A3,A2,A1
def compute_cost(y_pred,y_train):
    cost=np.mean((y_pred-y_train)**2)
    return cost
def backward_path(A3,A2,A1,w3,y_train,x_train):
    #output layer
    dA3=(2*(A3-y_train.T))/len(y_train) #gradient of loss wrt A2
    dz3=dA3 #gradient of loss wrt z2
    dw3=np.matmul(dz3,A2.T)
    db3=dz3.sum(axis=1)

    #Hidden layer 2
    dA2=np.matmul(w3.T,dz3)
    dz2=np.multiply(dA2,A2*(1-A2))
    dw2=np.matmul(dz2,A1.T)
    db2=dz2.sum(axis=1)

    #Hidden layer 1
    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 dw3,dw2,dw1,db3,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 using Gradient Descent¶

In [11]:

```
num_iter=500
learning_rate=0.001
his1=[]
w1,w2,w3,b1,b2,b3=initialize_parameters(x_train)
for i in range(num_iter):
    A3,A2,A1=forward_path(x_train,w1,b1,w2,b2,w3,b3)
    y_pred=A3
    cost=compute_cost(y_pred,y_train)
    dw3,dw2,dw1,db3,db2,db1=backward_path(A3,A2,A1,w3,y_train,x_train)
    w3,b3=update_parameters(w3,b3,dw3,db3,learning_rate)
    w2,b2=update_parameters(w2,b2,dw2,db2,learning_rate)
    w1,b1=update_parameters(w1,b1,dw1,db1,learning_rate)
    his1.append(cost)
```

In [12]:

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

Out[12]:

```
[<matplotlib.lines.Line2D at 0x25cb9a2f760>]
```

In [13]:

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

```
[[ 2.22982405 -0.84111873 -0.75268661]
 [ 1.25059415  0.27643916 -0.90666439]
 [ 0.01762027  0.86689346  0.18241872]
 [ 3.38296661 -0.35027431 -0.4544435 ]
 [-0.51040494 -0.24583232 -2.80302061]]
[[-0.16422378]
 [-1.57928538]
 [ 0.65783283]
 [-0.14624383]
 [-0.40244009]]
```

In [14]:

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

```
[[-4.01516602e-01  6.93388382e-01  2.19444722e-03 -2.61795864e-01
  -2.44558983e+00]
 [-4.47187119e-02 -6.72529284e-01  1.22359596e+00  6.36321648e-01
  -5.57945236e-01]
```

```
 [ 1.04665515e+00  1.44557943e+00 -2.17045722e+00 -5.78790461e-01
  -8.74205630e-01]
 [-6.93212288e-01 -8.34432972e-01  6.60880016e-02 -9.43299501e-01
   2.30353917e-01]
 [-1.19847637e+00  1.40777058e+00 -9.69312338e-02 -4.02938434e-01
  -1.18038453e-01]]
[[-0.64870749]
 [ 1.62770386]
 [ 0.91377687]
 [ 0.77065244]
 [-0.43957049]]
```

In [15]:

```
print(w3)
print(b3)
```

```
[[ 0.30322598  0.94401006 -1.67252669 -0.25705086 -0.91137719]]
[[0.01316571]]
```

Prediction¶

In [16]:

```
def pred(w1,w2,w3,b1,b2,b3,x_test):
    A3,A2,A1=forward_path(x_test,w1,b1,w2,b2,w3,b3)
    return A3
```

In [17]:

```
y_pred=pred(w1,w2,w3,b1,b2,b3,x_test)
print(y_pred)
```

```
[[-0.2728263  -0.09631945 -0.03444146 -0.09588367 -0.13820144 -0.15752294
  -0.13594885 -0.12136283 -0.12834389 -0.09080897]]
```

Rescalling dataset¶

In [18]:

```
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)
```

Comaparing test and predicted values¶

In [19]:

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

```
   Original profit  Predicted profit
0         81229.06     101126.574498
1        156122.51     108169.389034
2        155752.60     110638.387763
3        105008.31     108186.777079
4        124266.90     106498.252078
5        108733.99     105727.303309
6        141585.52     106588.132797
7        134307.35     107170.130856
8         69758.98     106891.578948
9         97483.56     108389.262917
```

Error calculation¶

In [20]:

```
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: 24709.155760441685
RMSE: 29196.888089000906
```

Training using Stochastic Gradient descent¶

In [30]:

```
def initialize_parameters(x):
    w1=np.random.randn(5,x.shape[1])
    w2=np.random.randn(5,5)
    w3=np.random.randn(1,5)
    b1=np.random.randn(5,1)
    b2=np.random.randn(5,1)
    b3=np.random.randn(1,1)
    return w1,w2,w3,b1,b2,b3
```

```python
def sigmoid_act(x):
    s= 1/(1+np.exp(-x))
    return s
def forward_path(x_train,w1,b1,w2,b2,w3,b3):
    #Hidden layer 1
    z1=np.reshape(np.matmul(w1,x_train.T),(b1.shape))+b1
    A1=sigmoid_act(z1)

    #Hidden layer 2
    z2=np.matmul(w2,A1)+b2
    A2=sigmoid_act(z2)

    #Outer layer
    z3=np.matmul(w3,A2)+b3
    A3=z3
    return A3,A2,A1
def compute_cost(y_pred,y_train):
    cost=np.mean((y_pred-y_train)**2)
    return cost
def backward_path(A3,A2,A1,w3,y_train,x_train):
    #output layer
    dA3=2*(A3-y_train.T)
    dz3=dA3
    dw3=np.matmul(dz3,A2.T)
    db3=dz3.sum(axis=1)

    #Hidden layer 2
    dA2=np.matmul(w3.T,dz3)
    dz2=np.multiply(dA2,A2*(1-A2))
    dw2=np.matmul(dz2,A1.T)
    db2=dz2.sum(axis=1)

    #Hidden layer 1
    dA1=np.matmul(w2.T,dz2)
    dz1=np.multiply(dA1,A1*(1-A1))
    dw1=dz1*x_train
    db1=dz1.sum(axis=1)
    return dw3,dw2,dw1,db3,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 [33]:

```
num_iter=500
learning_rate=0.001
his1=[]
w1,w2,w3,b1,b2,b3=initialize_parameters(x_train)
for i in range(num_iter):
    cost=0
    for j in range(len(x_train)):
        A3,A2,A1 = forward_path(x_train[j],w1,b1,w2,b2,w3,b3)
        y_pred=A3
        cost+=compute_cost(y_pred,y_train[j])
        
dw3,dw2,dw1,db3,db2,db1=backward_path(A3,A2,A1,w3,y_train[j],x_train[j])
        w3,b3=update_parameters(w3,b3,dw3,db3,learning_rate)
        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 [34]:

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

Out[34]:

```
[<matplotlib.lines.Line2D at 0x25cba207340>]
```

In [35]:

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

```
[[-0.01200262  1.42795631 -0.91774738]
 [-1.55594329 -0.93373708 -0.97464145]
 [-0.77781382 -1.43913585 -0.12934494]
 [-0.82890471 -0.67729867 -0.30060691]
 [ 0.78203697 -0.06624352  0.45125684]]
[[ 1.56778746]
 [ 0.26605122]
 [-0.43268025]
 [ 0.07065172]
 [ 0.92805018]]
```

In [36]:

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

```
[[-0.6466405   1.16558483 -1.02166882  0.58952838 -0.54872043]
 [ 0.3724921   0.38980482  0.47698183  0.79457453  1.44370987]
 [-0.23835624  0.77508018  0.45490067  2.00058434 -1.6936975 ]
 [ 1.62029082  1.68874557  0.40411368  0.80110928 -0.20670633]
```

```
 [-0.18681959 -0.08920247  1.26645946  0.31853325 -0.07715246]]
[[ 0.3330803 ]
 [ 0.35824501]
 [-0.50495584]
 [-1.62762572]
 [ 0.34957563]]
```

In [37]:

```
print(w3)
print(b3)
```

```
[[ 5.86932272e-01  1.82962880e+00 -2.59546298e+00 -1.97715883e+00
  -1.36353499e-03]]
[[0.7142939]]
```

Prediction¶

In [38]:

```
def pred2(w1,w2,w3,b1,b2,b3,x_test):
    y_pred=np.zeros(y_test.shape)
    for i in range(len(x_test)):
        A3,A2,A1=forward_path(x_test[i],w1,b1,w2,b2,w3,b3)
```

```
        y_pred[i]=A3
    return y_pred
```

In [39]:

```
y_pred2=pred2(w1,w2,w3,b1,b2,b3,x_test)
print(y_pred2)
print(y_pred2.shape)
```

```
[[-0.80817589]
 [ 0.10065632]
 [ 1.13349487]
 [ 0.76817616]
 [ 0.53120029]
 [ 0.30134011]
 [ 0.31414422]
 [ 0.35181499]
 [-1.08949205]
 [-0.66949912]]
(10, 1)
```

Rescalling data¶

In [40]:

```
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 [41]:

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

```
   Original profit  Predicted profit
0         81229.06      79765.546095
1        156122.51     116028.935272
2        155752.60     157240.312007
3        105008.31     142663.699683
4        124266.90     133208.105896
5        108733.99     124036.435834
6        141585.52     124547.333545
```

```
7        134307.35     126050.438317
8         69758.98      68540.726813
9         97483.56      85298.899210
```

Error Calculations¶

In [42]:

```
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: 14364.185416827726
RMSE: 19627.314553007396
```

In [ ]: