Import data¶

In [32]:

```
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[32]:

	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 [33]:

```
dataset.shape
```

Out[33]:

```
(50, 5)
```

Visualize the data¶

In [34]:

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

Out[34]:

```
<seaborn.axisgrid.PairGrid at 0x203172fe3d0>
```

Split data into x and y¶

In [35]:

```
import numpy as np
x=np.array(dataset[["R&D Spend"]])
```

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

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

Standardize the dataset¶

In [36]:

```
# 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, 1)
(50, 1)
```

Split the data into train and test¶

In [37]:

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

In [ ]:

Training using Gradient Descent¶

In [38]:

```
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
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
```

In [39]:

```
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 [40]:

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

Out[40]:

```
[<matplotlib.lines.Line2D at 0x20319dab820>]
```

In [41]:

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

```
[[-0.06490524]
 [-0.94953102]
 [ 0.0379391 ]
 [-0.86839446]
 [-0.61278253]]
[[ 1.55490262]
 [-0.41206396]
 [-0.4497486 ]
 [-1.22247557]
 [-0.68285193]]
```

In [42]:

```python
print(w2)
```

```
print(b2)
```

```
[[-0.01073475  1.00921716 -0.02094204  0.45931606  1.1458824 ]
 [ 2.14090821  0.72456906  1.09791638 -2.76412118  0.25004372]
 [ 0.28560674  1.0297897   1.14227671 -0.2679987   2.41814081]
 [-0.65986361 -0.69898264  0.24936261 -0.50677519 -1.65087437]
 [ 1.67067143  0.46931577 -0.92582994  0.1160587   0.24519178]]
[[-0.16013001]
 [ 0.38692063]
 [-0.99092365]
 [ 1.01659364]
 [-0.59458359]]
```

In [43]:

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

```
[[ 0.25154061  1.3381161  -1.62912013  0.16236396 -0.13255369]]
[[-0.32737911]]
```

Prediction¶

In [44]:

```
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 [45]:

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

```
[[-0.13128794  0.08040998  0.20145175 -0.0897833   0.11182845  0.14602598
   0.12481821  0.19566887 -0.25839916 -0.08397857]]
```

Rescalling dataset¶

In [46]:

```
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 [47]:

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

```
   Original profit  Predicted profit
0        108733.99     106774.108303
1        152211.77     115221.084659
2        192261.83     120050.782219
3        108552.04     108430.188154
4        155752.60     116474.715393
5        166187.94     117839.234095
6        156122.51     116993.020964
7        191792.06     119820.039067
8         81005.76     101702.232760
9        111313.02     108661.803372
```

Error calculations¶

In [48]:

```
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: 33335.925653253784
RMSE: 41968.33739056794
```

Traning using Stochastic gradiennt descent¶

In [49]:

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

```python
    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
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 [55]:

```
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 [58]:

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

Out[58]:

```
[<matplotlib.lines.Line2D at 0x2031a12b3d0>]
```

In [59]:

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

```
[[ 0.26262679]
 [-0.37259068]
 [ 2.17489721]
 [ 0.42670935]
 [-1.57426967]]
[[ 0.76415167]
 [ 0.01825898]
 [ 0.15398544]
 [-0.60578455]
 [-1.09473013]]
```

In [60]:

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

```
[[ 0.46133679 -0.0731024   2.22410746  0.21183576 -1.06170263]
 [ 0.91211761 -0.23039018  0.49716979 -0.45963911  1.78173947]
 [ 1.77687893  0.11601866  0.73584832  0.16143609 -0.83111002]
 [ 0.73340366  1.18562623 -1.96115617 -0.1345312   0.30581892]
 [-0.25921424  1.79969667  0.50566739 -0.16854131 -0.02473292]]
[[-0.96049271]
 [-0.66875797]
 [ 2.19145744]
 [-0.40060649]
 [ 1.73642667]]
```

In [61]:

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

```
[[ 2.25118929 -0.50475782 -0.90499256 -1.81516683  1.60455536]]
[[-0.76009889]]
```

Predcition¶

In [62]:

```python
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 [63]:

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

```
[[-0.09591431]
 [ 0.9594644 ]
 [ 1.10380923]
 [ 0.20512646]
 [ 1.01557682]
 [ 1.0593445 ]
 [ 1.03405514]
 [ 1.10031225]
 [-0.91532749]
 [ 0.24506868]]
(10, 1)
```

Rescalling data¶

In [64]:

```
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 values¶

In [65]:

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

```
   Original profit  Predicted profit
0        108733.99     108185.554252
1        152211.77     150296.307878
2        192261.83     156055.822707
```

3	108552.04	120197.407006
4	155752.60	152535.254126
5	166187.94	154281.631888
6	156122.51	153272.558954
7	191792.06	155916.289609
8	81005.76	75490.081274
9	111313.02	121791.144848

Error calculations¶

In [66]:

```
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: 12015.845116672488
RMSE: 17429.79556764171
```

In [ ]: