In [27]:

```
#Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
%matplotlib inline
```

In [28]:

```
#Importing the dataset and Extracting the Independent and Dependent variable
companies=pd.read_csv('E:/Likhita/MBA- IT/Second Sem/PDA/Linear Regression/1000_Companies.csv')
X = companies.iloc[:,:-1].values
Y = companies.iloc[:,4].values
companies.head()
```

Out[28]:

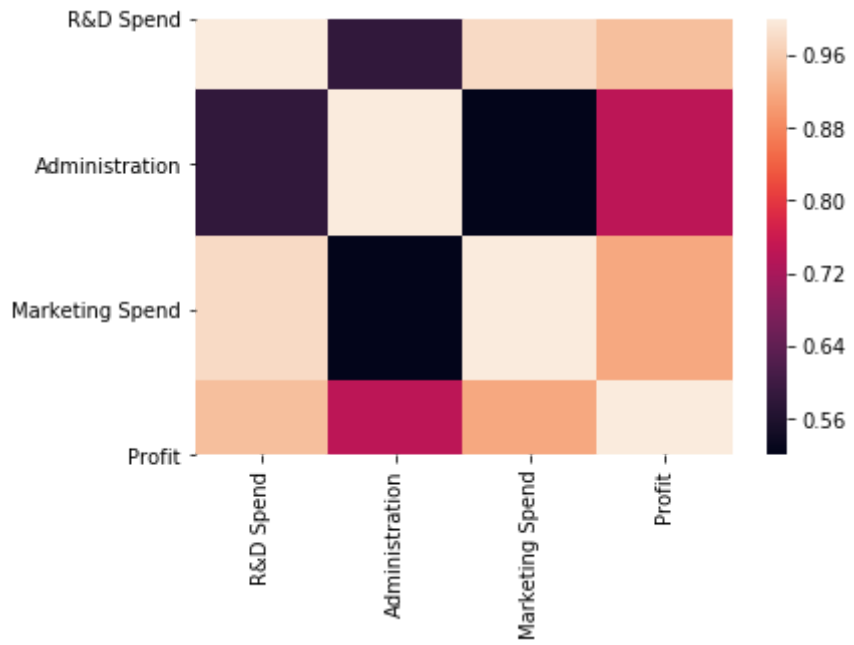
	R&D Spend	Administration	Marketing Spend	State	Profit
0	165349.20	136897.80	471784.10	New York	192261.83
1	162597.70	151377.59	443898.53	California	191792.06
2	153441.51	101145.55	407934.54	Florida	191050.39
3	144372.41	118671.85	383199.62	New York	182901.99
4	142107.34	91391.77	366168.42	Florida	166187.94

In [29]:

```
#Data Visualisation
#Building the Correlation matrix
sns.heatmap(companies.corr())
```

Out[29]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2ab85555e48>
```



In [30]:

```
  #encoding categorial data
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
labelencoder = LabelEncoder()
X[:,3] = labelencoder.fit_transform(X[:,3])


onehotencoder = OneHotEncoder(categorical_features = [3])
X = onehotencoder.fit_transform(X).toarray()
```

```
C:\dell\Anaconda3\lib\site-packages\sklearn\preprocessing\_encoders.py:41
5: FutureWarning: The handling of integer data will change in version 0.2
2. Currently, the categories are determined based on the range [0, max(val
ues)], while in the future they will be determined based on the unique val
ues.
If you want the future behaviour and silence this warning, you can specify
"categories='auto'".
In case you used a LabelEncoder before this OneHotEncoder to convert the c
ategories to integers, then you can now use the OneHotEncoder directly.
  warnings.warn(msg, FutureWarning)
C:\dell\Anaconda3\lib\site-packages\sklearn\preprocessing\_encoders.py:45
1: DeprecationWarning: The 'categorical_features' keyword is deprecated in
version 0.20 and will be removed in 0.22. You can use the ColumnTransforme
r instead.
  "use the ColumnTransformer instead.", DeprecationWarning)
```

In [31]:

```
#Avoiding the Dummy VAriable Trap
X=X[:,1:]
```

In [32]:

```
#Slitting the dataset into the training set and test set
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,Y,test_size=0.2,random_state=0)
```

In [33]:

```
#Fitting Multiple Linear Regression to the Training set
from sklearn.linear_model import LinearRegression
regressor=LinearRegression()
regressor.fit(X_train,y_train)
```

Out[33]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=F
alse)
```

In [34]:

```
#Predicting the Test set results
y_pred = regressor.predict(X_test)
y_pred
```

Out[34]:

```
array([ 89790.61532916,  88427.07187361,  94894.67836972, 175680.8672561 ,
        83411.73042089, 110571.90200074, 132145.22936439,  91473.37719687,
       164597.05380606,  53222.82667402,  66950.1905099 , 150566.43987004,
       126915.20858596,  59337.85971051, 177513.91053061,  75316.28143051,
       118248.14406603, 164574.40699901, 170937.2898107 , 182069.11645084,
       118845.0325269 ,  85669.95112229, 180992.59396143,  84145.08220145,
       105005.83769214, 101233.56772747,  53831.07669092,  56881.41475225,
        68896.39346905, 210040.00765882, 120778.72270894, 111724.87157654,
       101487.90541518, 137959.02649624,  63969.95996744, 108857.91214126,
       186014.72531987, 171442.64130747, 174644.26529204, 117671.49128195,
        96731.37857433, 165452.25779409, 107724.34331255,  50194.54176914,
       116513.89532179,  58632.48986821, 158416.46827609,  78541.4852161 ,
       159727.66671743, 131137.87699644, 184880.70924515, 174609.08266879,
        93745.66352059,  78341.13383418, 180745.90439079,  84461.61490552,
       142900.90602903, 170618.44098396,  84365.09530839, 105307.3716218 ,
       141660.07290786,  52527.34340443, 141842.9626416 , 139176.27973195,
        98294.52669666, 113586.86790969, 126754.21895489, 152135.51985561,
        58864.51658955, 174285.57361129, 124624.04380784, 169065.77658978,
        91279.3319821 , 156170.37268962,  84307.26579366,  77877.75223097,
       120414.02421346,  93380.44273242, 139020.62514121, 143604.67103572,
       171148.30815368, 140082.97050131, 106369.71689747, 155641.43851387,
       140030.10330037, 110172.87893525,  69672.98677565,  88148.52068041,
       140133.59925093, 148479.09537887, 157916.63505257,  58532.94863142,
        93707.3842239 , 112646.37475705,  56556.18943661, 107414.89996181,
       147352.80227752, 152144.10104034, 167808.11701783, 118750.25230713,
       120763.27666701, 139029.95295662, 157527.90934119, 121962.0621496 ,
        87091.32399737, 104792.91384334,  95335.22679185, 178389.52287436,
       181942.63776381, 109831.34945506, 165254.03344096, 167806.06491902,
       158002.44642543, 174782.86900956, 170196.77102698,  52302.18161612,
       176938.11595789, 104751.83583865,  82710.31528806, 138890.52767844,
       144274.74675425, 161679.0183644 , 169662.05445895, 120450.9231013 ,
       158880.70799546, 110213.73252824, 169674.51532366,  60760.61300842,
       159036.99629068, 158169.44286047, 174511.70494474, 156294.79927783,
       103714.37583212,  85635.96237574, 141603.54878757, 165917.69156979,
       121182.03641977, 170751.87883893, 100505.77549412,  82097.51033128,
       178643.18879842, 101790.48384579,  70507.40958622,  90250.04230089,
        61247.4996268 ,  68912.17534521,  72775.81613476, 176914.08873124,
        89704.69244931, 129209.43730015,  92672.90938383,  88133.59175044,
       172836.33021618,  60893.62070015, 169015.8944601 , 166450.24453204,
       165425.54476415, 102170.5169499 , 181594.57928215,  73702.57942562,
        91267.42979669, 135791.54160195,  64922.802573  ,  71775.70235726,
        60603.91401516, 184288.61041915, 176286.69585945, 158907.75687038,
       141359.32216438, 154611.17928321,  58549.58863233,  90618.58407899,
       152885.51163925, 168398.05223805,  72485.3627454 , 116064.24350667,
        80087.80697209, 149828.90896188, 116806.9595737 , 130191.48845161,
       174534.42670328, 293584.45948282, 146270.83174788, 150646.69178013,
        86107.47782247,  69967.20842246,  70096.78368774,  69033.69170769,
       120666.75708063,  89677.68014064, 166824.27091662, 125514.76626409,
        67209.67687467, 140930.69427701, 118544.30490695, 165897.61905906,
       168655.48652552, 147009.66805048, 141396.22104146, 109086.5063484
9])
```

In [35]:

```
#Calculating the Coefficinets
print(regressor.coef_)
```

```
[-8.80536598e+02 -6.98169073e+02  5.25845857e-01  8.44390881e-01
  1.07574255e-01]
```

In [36]:

```
#calculating the intercept
print(regressor.intercept_)
```

```
-51035.22972401607
```

In [37]:

```
#Calculating the R squared value
from sklearn.metrics import r2_score
r2_score(y_test,y_pred)
```

Out[37]:

```
0.9112695892268822
```