ALBUKHARY INTERNATIONAL UNIVERSITY

Artificial Intelligence

LAB EIGHT (b)

Multiple Linear Regression

Multiple Linear Regression is one of the important regression algorithms which models the linear relationship between a single dependent continuous variable and more than one independent variable.

Problem Description:

We have a dataset of 50 start-up companies. This dataset contains five main information: R&D Spend, Administration Spend, Marketing Spend, State, and Profit for a financial year. Our goal is to create a model that can easily determine which company has a maximum profit, and which is the most affecting factor for the profit of a company.

Below are the main steps of deploying the MLR model:

- Data Pre-processing Steps
- Fitting the MLR model to the training set
- Predicting the result of the test set

1. Data Pre-processing Step

Importing libraries:

In [1]:

```
# importing libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

Importing dataset:

In [2]:

```
dataset = pd.read_csv('50_Startups.csv')
```

In [3]:

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50 entries, 0 to 49
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   R&D Spend        50 non-null     float64
 1   Administration   50 non-null     float64
 2   Marketing Spend  50 non-null     float64
 3   State            50 non-null     object 
 4   Profit           50 non-null     float64
dtypes: float64(4), object(1)
memory usage: 2.1+ KB
```

In [4]:

```
dataset.shape
```

Out[4]:

```
(50, 5)
```

In [5]:

```
dataset.describe()
```

Out[5]:

	R&D Spend	Administration	Marketing Spend	Profit
count	50.000000	50.000000	50.000000	50.000000
mean	73721.615600	121344.639600	211025.097800	112012.639200
std	45902.256482	28017.802755	122290.310726	40306.180338
min	0.000000	51283.140000	0.000000	14681.400000
25%	39936.370000	103730.875000	129300.132500	90138.902500
50%	73051.080000	122699.795000	212716.240000	107978.190000
75%	101602.800000	144842.180000	299469.085000	139765.977500
max	165349.200000	182645.560000	471784.100000	192261.830000

In [6]:

```
dataset.head()
```

Out[6]:

	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 [7]:

```
dataset
```

13	91992.39	135495.07	252664.93	California	134307.35
14	119943.24	156547.42	256512.92	Florida	132602.65
15	114523.61	122616.84	261776.23	New York	129917.04
16	78013.11	121597.55	264346.06	California	126992.93
17	94657.16	145077.58	282574.31	New York	125370.37
18	91749.16	114175.79	294919.57	Florida	124266.90
19	86419.70	153514.11	0.00	New York	122776.86
20	76253.86	113867.30	298664.47	California	118474.03
21	78389.47	153773.43	299737.29	New York	111313.02
22	73994.56	122782.75	303319.26	Florida	110352.25
23	67532.53	105751.03	304768.73	Florida	108733.99
24	77044.01	99281.34	140574.81	New York	108552.04

Extracting dependent and independent Variables:

In [10]:

```
#Extracting Independent and dependent Variable
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
```

In [11]:

```
print(X)
```

```
[[165349.2 136897.8 471784.1 'New York']
 [162597.7 151377.59 443898.53 'California']
 [153441.51 101145.55 407934.54 'Florida']
 [144372.41 118671.85 383199.62 'New York']
 [142107.34 91391.77 366168.42 'Florida']
 [131876.9 99814.71 362861.36 'New York']
 [134615.46 147198.87 127716.82 'California']
 [130298.13 145530.06 323876.68 'Florida']
 [120542.52 148718.95 311613.29 'New York']
 [123334.88 108679.17 304981.62 'California']
 [101913.08 110594.11 229160.95 'Florida']
 [100671.96 91790.61 249744.55 'California']
 [93863.75 127320.38 249839.44 'Florida']
 [91992.39 135495.07 252664.93 'California']
 [119943.24 156547.42 256512.92 'Florida']
 [114523.61 122616.84 261776.23 'New York']
 [78013.11 121597.55 264346.06 'California']
 [94657.16 145077.58 282574.31 'New York']
 [91749.16 114175.79 294919.57 'Florida']
 [86419.7 153514.11 0.0 'New York']
 [76253.86 113867.3 298664.47 'California']
 [78389.47 153773.43 299737.29 'New York']
 [73994.56 122782.75 303319.26 'Florida']
 [67532.53 105751.03 304768.73 'Florida']
 [77044.01 99281.34 140574.81 'New York']
 [64664.71 139553.16 137962.62 'California']
 [75328.87 144135.98 134050.07 'Florida']
 [72107.6 127864.55 353183.81 'New York']
 [66051.52 182645.56 118148.2 'Florida']
 [65605.48 153032.06 107138.38 'New York']
 [61994.48 115641.28 91131.24 'Florida']
 [61136.38 152701.92 88218.23 'New York']
 [63408.86 129219.61 46085.25 'California']
 [55493.95 103057.49 214634.81 'Florida']
 [46426.07 157693.92 210797.67 'California']
 [46014.02 85047.44 205517.64 'New York']
 [28663.76 127056.21 201126.82 'Florida']
 [44069.95 51283.14 197029.42 'California']
 [20229.59 65947.93 185265.1 'New York']
 [38558.51 82982.09 174999.3 'California']
 [28754.33 118546.05 172795.67 'California']
 [27892.92 84710.77 164470.71 'Florida']
 [23640.93 96189.63 148001.11 'California']
 [15505.73 127382.3 35534.17 'New York']
 [22177.74 154806.14 28334.72 'California']
 [1000.23 124153.04 1903.93 'New York']
 [1315.46 115816.21 297114.46 'Florida']
 [0.0 135426.92 0.0 'California']
 [542.05 51743.15 0.0 'New York']
 [0.0 116983.8 45173.06 'California']]
```

In [12]:

```
y
```

Out[12]:

```
array([192261.83, 191792.06, 191050.39, 182901.99, 166187.94, 156991.1
2,
       156122.51, 155752.6 , 152211.77, 149759.96, 146121.95, 144259.4
,
       141585.52, 134307.35, 132602.65, 129917.04, 126992.93, 125370.3
7,
       124266.9 , 122776.86, 118474.03, 111313.02, 110352.25, 108733.9
9,
       108552.04, 107404.34, 105733.54, 105008.31, 103282.38, 101004.6
4,
        99937.59,  97483.56,  97427.84,  96778.92,  96712.8 ,  96479.5
1,
        90708.19,  89949.14,  81229.06,  81005.76,  78239.91,  77798.8
3,
        71498.49,  69758.98,  65200.33,  64926.08,  49490.75,  42559.7
3,
        35673.41,  14681.4 ])
```

Encoding Categorical Data:

As we have one categorical variable (State), which cannot be directly applied to the model, so we will encode it. To encode the categorical variable into numbers, we will use the LabelEncoder class. But it is not sufficient because it still has some relational order, which may create a wrong model. So in order to remove this problem, we will use OneHotEncoder.

In [13]:

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [3])],
                       remainder='passthrough')
X = np.array(ct.fit_transform(X))
```

In [15]:

```
print(X)
```

```
[[0.0 0.0 1.0 165349.2 136897.8 471784.1]
 [1.0 0.0 0.0 162597.7 151377.59 443898.53]
 [0.0 1.0 0.0 153441.51 101145.55 407934.54]
 [0.0 0.0 1.0 144372.41 118671.85 383199.62]
 [0.0 1.0 0.0 142107.34 91391.77 366168.42]
 [0.0 0.0 1.0 131876.9 99814.71 362861.36]
 [1.0 0.0 0.0 134615.46 147198.87 127716.82]
 [0.0 1.0 0.0 130298.13 145530.06 323876.68]
 [0.0 0.0 1.0 120542.52 148718.95 311613.29]
 [1.0 0.0 0.0 123334.88 108679.17 304981.62]
 [0.0 1.0 0.0 101913.08 110594.11 229160.95]
 [1.0 0.0 0.0 100671.96 91790.61 249744.55]
 [0.0 1.0 0.0 93863.75 127320.38 249839.44]
 [1.0 0.0 0.0 91992.39 135495.07 252664.93]
 [0.0 1.0 0.0 119943.24 156547.42 256512.92]
 [0.0 0.0 1.0 114523.61 122616.84 261776.23]
 [1.0 0.0 0.0 78013.11 121597.55 264346.06]
 [0.0 0.0 1.0 94657.16 145077.58 282574.31]
 [0.0 1.0 0.0 91749.16 114175.79 294919.57]
 [0.0 0.0 1.0 86419.7 153514.11 0.0]
 [1.0 0.0 0.0 76253.86 113867.3 298664.47]
 [0.0 0.0 1.0 78389.47 153773.43 299737.29]
 [0.0 1.0 0.0 73994.56 122782.75 303319.26]
 [0.0 1.0 0.0 67532.53 105751.03 304768.73]
 [0.0 0.0 1.0 77044.01 99281.34 140574.81]
 [1.0 0.0 0.0 64664.71 139553.16 137962.62]
 [0.0 1.0 0.0 75328.87 144135.98 134050.07]
 [0.0 0.0 1.0 72107.6 127864.55 353183.81]
 [0.0 1.0 0.0 66051.52 182645.56 118148.2]
 [0.0 0.0 1.0 65605.48 153032.06 107138.38]
 [0.0 1.0 0.0 61994.48 115641.28 91131.24]
 [0.0 0.0 1.0 61136.38 152701.92 88218.23]
 [1.0 0.0 0.0 63408.86 129219.61 46085.25]
 [0.0 1.0 0.0 55493.95 103057.49 214634.81]
 [1.0 0.0 0.0 46426.07 157693.92 210797.67]
 [0.0 0.0 1.0 46014.02 85047.44 205517.64]
 [0.0 1.0 0.0 28663.76 127056.21 201126.82]
 [1.0 0.0 0.0 44069.95 51283.14 197029.42]
 [0.0 0.0 1.0 20229.59 65947.93 185265.1]
 [1.0 0.0 0.0 38558.51 82982.09 174999.3]
 [1.0 0.0 0.0 28754.33 118546.05 172795.67]
 [0.0 1.0 0.0 27892.92 84710.77 164470.71]
 [1.0 0.0 0.0 23640.93 96189.63 148001.11]
 [0.0 0.0 1.0 15505.73 127382.3 35534.17]
 [1.0 0.0 0.0 22177.74 154806.14 28334.72]
 [0.0 0.0 1.0 1000.23 124153.04 1903.93]
 [0.0 1.0 0.0 1315.46 115816.21 297114.46]
 [1.0 0.0 0.0 0.0 135426.92 0.0]
 [0.0 0.0 1.0 542.05 51743.15 0.0]
 [1.0 0.0 0.0 0.0 116983.8 45173.06]]
```

Splitting the dataset into the Training set and Test set

In [16]:

```
# Splitting the dataset into training 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)
```

2. Fitting our MLR model to the Training set:

In [17]:

```
#Fitting the MLR model to the training set:
from sklearn.linear_model import LinearRegression
regressor= LinearRegression()
regressor.fit(x_train, y_train)
```

Out[17]:

```
LinearRegression()
```

3. Prediction of Test set results

The last step for our model is checking the performance of the model. We will do it by predicting the test set result. For prediction, we will create a y_pred vector. Below is the code for it:

In [18]:

```
#Predicting the Test set result;
y_pred= regressor.predict(x_test)
```

In [19]:

```
print(y_pred)
```

```
[103015.20159794 132582.27760815 132447.73845174  71976.09851258
 178537.48221057 116161.24230168  67851.69209677  98791.73374686
 113969.43533014 167921.06569552]
```

In [20]:

```
print(y_test)
```

```
[103282.38 144259.4  146121.95  77798.83 191050.39 105008.31  81229.06
  97483.56 110352.25 166187.94]
```

In the above output, we have predicted result set and test set. We can check model performance by comparing these two value index by index. For example, the first index has a predicted value of 103015 $profitandtest/realvalueof103282$ profit. The difference is only of 267$, which is a good prediction, so, finally, our model is completed here.

We can also check the score for training dataset and test dataset. Below is the code for it:

In [21]:

```
print('Train Score: ', regressor.score(x_train, y_train))
print('Test Score: ', regressor.score(x_test, y_test))
```

```
Train Score:  0.9501847627493607
Test Score:  0.9347068473282364
```

The above score tells that our model is 95% accurate with the training dataset and 93% accurate with the test dataset.

In [22]:

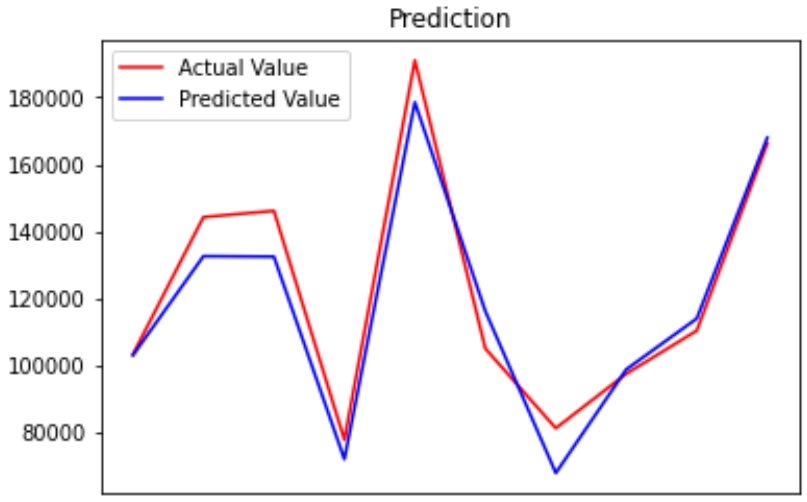
```
import sklearn.metrics as sm

# Compute performance metrics
print("Linear regressor performance:")
print("R2 score =", round(sm.r2_score(y_test, y_pred), 2))
```

```
Linear regressor performance:
R2 score = 0.93
```

In [23]:

```
fig, ax = plt.subplots()
#visualizing the Test set results
plt.plot(y_test, color="red", label="Actual Value")
plt.plot(y_pred, color="blue", label="Predicted Value")
ax.legend(loc=2); # upper left corner
ax.set_xticks([])
plt.title("Prediction")
#plt.xlabel("Years of Experience")
#plt.ylabel("Salary")
plt.show()
```



In [ ]:

```
```