Multiple Linear Regression

In the previous topic, we have learned about Simple Linear Regression, where a single Independent/Predictor(X) variable is used to model the response variable (Y). But there may be various cases in which the response variable is affected by more than one predictor variable; for such cases, the Multiple Linear Regression algorithm is used.

Moreover, Multiple Linear Regression is an extension of Simple Linear regression as it takes more than one predictor variable to predict the response variable. We can define it as:

> *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.*

Example:

Prediction of CO_2 emission based on engine size and number of cylinders in a car.

Some key points about MLR:

- For MLR, the dependent or target variable(Y) must be the continuous/real, but the predictor or independent variable may be of continuous or categorical form.
- Each feature variable must model the linear relationship with the dependent variable.
- MLR tries to fit a regression line through a multidimensional space of data-points.

MLR equation:

In Multiple Linear Regression, the target variable(Y) is a linear combination of multiple predictor variables x_1, x_2, x_3, ...,x_n. Since it is an enhancement of Simple Linear Regression, so the same is applied for the multiple linear regression equation, the equation becomes:

```
Y= b<sub>0</sub>+b<sub>1</sub>x<sub>1</sub>+ b<sub>2</sub>x<sub>2</sub>+ b<sub>3</sub>x
```

Where,

Y= Output/Response variable

b_0, b_1, b_2, b_3 , b_n....= Coefficients of the model.

x_1, x_2, x_3, x_4,...= Various Independent/feature variable

Assumptions for Multiple Linear Regression:

- A **linear relationship** should exist between the Target and predictor variables.
- The regression residuals must be **normally distributed**.
- MLR assumes little or **no multicollinearity** (correlation between the independent variable) in data.

Implementation of Multiple Linear Regression model using Python:

To implement MLR using Python, we have below problem:

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.

Since we need to find the Profit, so it is the dependent variable, and the other four variables are independent variables. Below are the main steps of deploying the MLR model:

1. **Data Pre-processing Steps**
2. **Fitting the MLR model to the training set**
3. **Predicting the result of the test set**

Step-1: Data Pre-processing Step:

The very first step is data pre-processing, which we have already discussed in this tutorial. This process contains the below steps:

- **Importing libraries:** Firstly we will import the library which will help in building the model. Below is the code for it:

```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
```

- **Importing dataset:** Now we will import the dataset(50_CompList), which contains all the variables. Below is the code for it:

```
#importing datasets
data_set= pd.read_csv('50_CompList.csv')
```

Output: We will get the dataset as:

data_set - DataFrame

Index	R&D Spend	Administration	Marketing Spend	State	Profit
0	165349	136898	471784	New York	192262
1	162598	151378	443899	California	191792
2	153442	101146	407935	Florida	191050
3	144372	118672	383200	New York	182902
4	142107	91391.8	366168	Florida	166188
5	131877	99814.7	362861	New York	156991
6	134615	147199	127717	California	156123
7	130298	145530	323877	Florida	155753
8	120543	148719	311613	New York	152212
9	123335	108679	304982	California	149760
10	101913	110594	229161	Florida	146122
11	100672	91790.6	249745	California	144259
12	93863.8	127320	249839	Florida	141586
13	91992.4	135495	252665	California	134307

Format Resize Background color Column min/max Save and Close Close

In above output, we can clearly see that there are five variables, in which four variables are continuous and one is categorical variable.

- **Extracting dependent and independent Variables:**

```
#Extracting Independent and dependent Variable
x= data_set.iloc[:, :-1].values
y= data_set.iloc[:, 4].values
```

Output:

Out[5]:

```
array([[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']], dtype=object)
```

As we can see in the above output, the last column contains categorical variables which are not suitable to apply directly for fitting the model. So we need to encode this variable.

Encoding Dummy Variables:

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**, which will create the dummy variables. Below is code for it:

```
#Catgorical data
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
labelencoder_x= LabelEncoder()
x[:, 3]= labelencoder_x.fit_transform(x[:,3])
onehotencoder= OneHotEncoder(categorical_features= [3])
x= onehotencoder.fit_transform(x).toarray()
```

Here we are only encoding one independent variable, which is state as other variables are continuous.

Output:

x - NumPy array

	0	1	2	3	4	5
0	0	0	1	165349	136898	471784
1	1	0	0	162598	151378	443899
2	0	1	0	153442	101146	407935
3	0	0	1	144372	118672	383200
4	0	1	0	142107	91391.8	366168
5	0	0	1	131877	99814.7	362861
6	1	0	0	134615	147199	127717
7	0	1	0	130298	145530	323877
8	0	0	1	120543	148719	311613
9	1	0	0	123335	108679	304982
10	0	1	0	101913	110594	229161
11	1	0	0	100672	91790.6	249745
12	0	1	0	93863.8	127320	249839
13	1	0	0	91992.4	135495	252665

Format Resize Background color

As we can see in the above output, the state column has been converted into dummy variables (0 and 1). **Here each dummy variable column is corresponding to the one State**. We can check by comparing it with the original dataset. The first column corresponds to the **California State**, the second column corresponds to the **Florida State**, and the third column corresponds to the **New York State**.

> **Note:** We should not use all the dummy variables at the same time, so it must be 1 less than the total number of dummy variables, else it will create a dummy variable trap.

- Now, we are writing a single line of code just to avoid the dummy variable trap:

```
#avoiding the dummy variable trap:
x = x[:, 1:]
```

If we do not remove the first dummy variable, then it may introduce multicollinearity in the model.

	0	1	2	3	4
0	0	1	165349	136898	471784
1	0	0	162598	151378	443899
2	1	0	153442	101146	407935
3	0	1	144372	118672	383200
4	1	0	142107	91391.8	366168
5	0	1	131877	99814.7	362861
6	0	0	134615	147199	127717
7	1	0	130298	145530	323877
8	0	1	120543	148719	311613
9	0	0	123335	108679	304982
10	1	0	101913	110594	229161
11	0	0	100672	91790.6	249745
12	1	0	93863.8	127320	249839

As we can see in the above output image, the first column has been removed.

- Now we will split the dataset into training and test set. The code for this is given below:

```
# 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)
```

The above code will split our dataset into a training set and test set.

Output: The above code will split the dataset into training set and test set. You can check the output by clicking on the variable explorer option given in Spyder IDE. The test set and training set will look like the below image:

Test set:

y_test - NumPy array

	0
0	103282
1	144259
2	146122
3	77798.8
4	191050
5	105008
6	81229.1
7	97483.6
8	110352
9	166188

x_test - NumPy array

	0	1	2	3	4
0	1	0	66051.5	182646	118148
1	0	0	100672	91790.6	249745
2	1	0	101913	110594	229161
3	1	0	27892.9	84710.8	164471
4	1	0	153442	101146	407935
5	0	1	72107.6	127865	353184
6	0	1	20229.6	65947.9	185265
7	0	1	61136.4	152702	88218.2
8	1	0	73994.6	122783	303319
9	1	0	142107	91391.8	366168

Format Resize Background color Save and Close Close

Training set:

x_train - NumPy array

	0	1	2	3	4
0	1	0	55493.9	103057	214635
1	0	1	46014	85047.4	205518
2	1	0	75328.9	144136	134050
3	0	0	46426.1	157694	210798
4	1	0	91749.2	114176	294920
5	1	0	130298	145530	323877
6	1	0	119943	156547	256513
7	0	1	1000.23	124153	1903.93
8	0	1	542.05	51743.2	0
9	0	1	65605.5	153032	107138
10	0	1	114524	122617	261776
11	1	0	61994.5	115641	91131.2

y_train - NumPy array

	0
0	96778.9
1	96479.5
2	105734
3	96712.8
4	124267
5	155753
6	132603
7	64926.1
8	35673.4
9	101005
10	129917
11	99937.6

Format Resize Background color Save and Close Close

Note: In MLR, we will not do feature scaling as it is taken care by the library, so we don't need to do it manually.

Step: 2- Fitting our MLR model to the Training set:

Now, we have well prepared our dataset in order to provide training, which means we will fit our regression model to the training set. It will be similar to as we did in Simple Linear Regression model. The code for this will be:

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

Output:

```
Out[9]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

Now, we have successfully trained our model using the training dataset. In the next step, we will test the performance of the model using the test dataset.

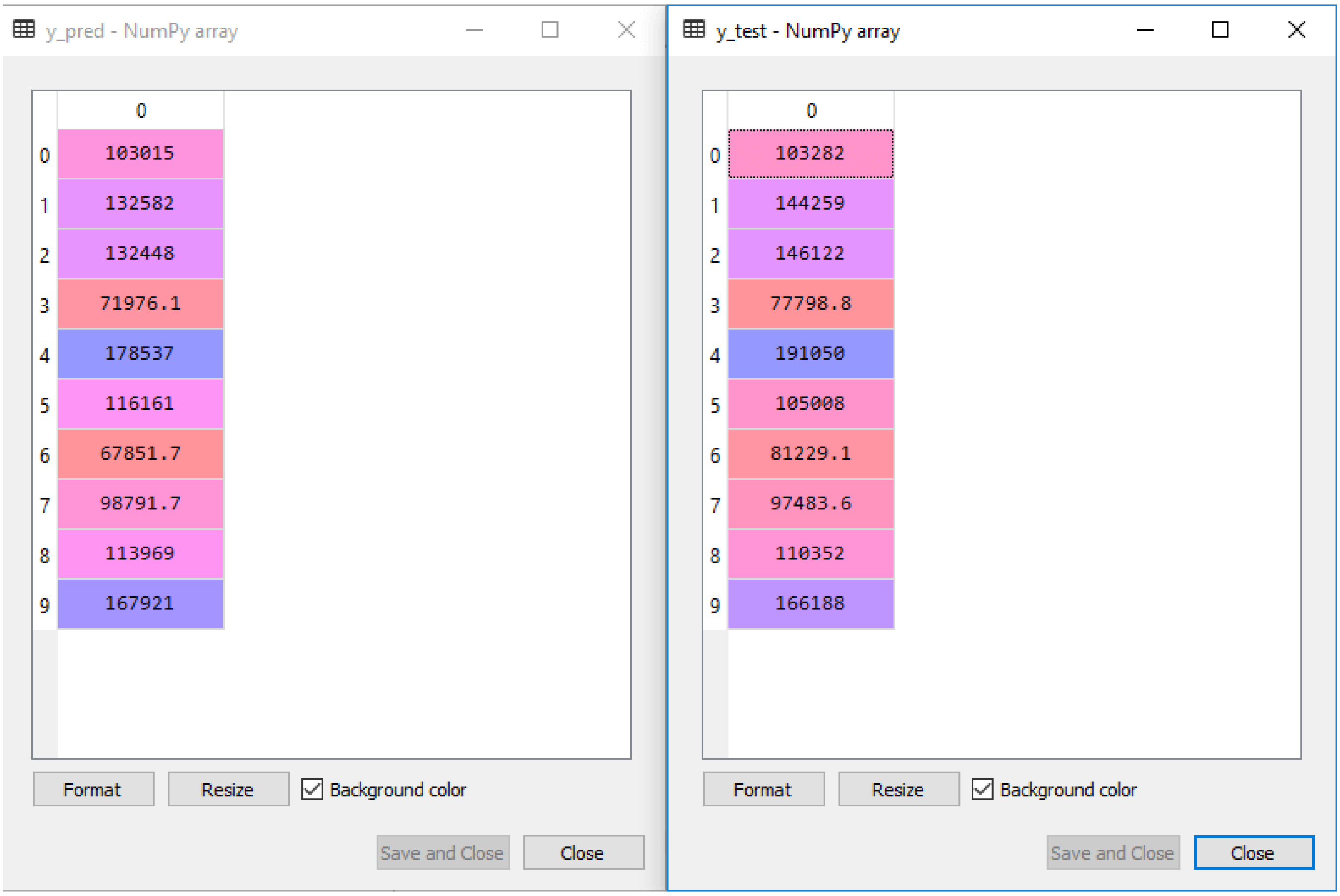
Step: 3- Prediction of Test set results:

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:

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

By executing the above lines of code, a new vector will be generated under the variable explorer option. We can test our model by comparing the predicted values and test set values.

Output:



y_pred - NumPy array

	0
0	103015
1	132582
2	132448
3	71976.1
4	178537
5	116161
6	67851.7
7	98791.7
8	113969
9	167921

Format Resize Background color
Save and Close Close

y_test - NumPy array

	0
0	103282
1	144259
2	146122
3	77798.8
4	191050
5	105008
6	81229.1
7	97483.6
8	110352
9	166188

Format Resize Background color
Save and Close Close

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$** profit and test/real value of **103282$** 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:

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

Output: The score is:

```
Train Score:  0.9501847627493607
Test Score:  0.9347068473282446
```

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

> **Note:** In the next topic, we will see how we can improve the performance of the model using the **Backward Elimination** process.

Applications of Multiple Linear Regression:

There are mainly two applications of Multiple Linear Regression:

- Effectiveness of Independent variable on prediction:
- Predicting the impact of changes:

← Prev Next →

Reinforcement Learning

R Programming

React Native tutorial

React Native

Python Design Patterns

Python Design Patterns

Python Pillow tutorial

Python Pillow

Python Turtle tutorial

Python Turtle

Keras tutorial

Keras

Preparation

Aptitude

Aptitude

Logical Reasoning

Reasoning

Verbal Ability

Verbal Ability

Interview Questions

Interview Questions

Company Interview Questions

Company Questions

Trending Technologies

Artificial Intelligence

Artificial Intelligence

AWS Tutorial

AWS

Selenium tutorial

Selenium

Cloud Computing

Cloud Computing

Hadoop tutorial

Hadoop

ReactJS Tutorial

ReactJS

Data Science Tutorial

Data Science

Angular 7 Tutorial

Angular 7

Blockchain Tutorial

Blockchain

Git Tutorial

Git

Machine Learning Tutorial

Machine Learning

DevOps Tutorial

DevOps

B.Tech / MCA

DBMS tutorial
DBMS

Data Structures tutorial
Data Structures

DAA tutorial
DAA

Operating System
Operating System

Computer Network tutorial
Computer Network

Compiler Design tutorial
Compiler Design

Computer Organization and Architecture
Computer Organization

Discrete Mathematics Tutorial
Discrete Mathematics

Ethical Hacking
Ethical Hacking

Computer Graphics Tutorial
Computer Graphics

Software Engineering
Software Engineering

html tutorial
Web Technology

Cyber Security tutorial
Cyber Security

Automata Tutorial
Automata

C Language tutorial
C Programming

C++ tutorial
C++

Java tutorial
Java

.Net Framework tutorial
.Net

Python tutorial
Python

List of Programs
Programs

Control Systems tutorial
Control System

Data Mining Tutorial
Data Mining

Data Warehouse Tutorial
Data Warehouse