Multilinear Regression ¶

In the previous session we have performed simple linear regression having one independent varibable (X) and one dependent variable (y). But, consider to the real world problems we are going to face more than two variables. Simple linear regression having multiple variables are known as Multi-Linear Regression. Step involved in the mutli-linear regression are almost similar to the step involved to simple linear regression.

The relationship between the multiple independent variables (X) and dependent variables (y) is represented by the following equation:
y = β_0 + $\beta_1 X_1$ + $\beta_2 X_2$ + ... $\beta_n X_n$ + E
Here, y is known as response/ dependent variable/ observation variable which we are going to estimate or predict
X_1 , X_2 ... X_n is the multiple independent / explanatory variables variables which we are going to make predictions.

β_1 , β_2 ... β_n is the slope coefficient for the explanatory variables.

β_0 is the y-intercert (constant term)

E is ths models' error term also known as residuals error.

Multi Linear regression model is based on the following assumptions: 1. There is a lineare relationship between the dependent variables and indepdent variables. 2. The independent variables are not highly correlated with each other. 3. y observations are selected independently and randomly from the population. 4. Residuals should be normally distributed with a mean of 0 and variance σ.

Importing the libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Importing the dataset

The below command is used by Python's Pandas library is to import CSV dataset where imput features like 'R&D Spend', 'Admisnistration', 'Marketing Spend', and 'State'. Based on these features we will predict profit.

In [2]:
```python
dataset = pd.read_csv(r'C:\Users\NITHISH\Desktop\DATA SCIENCE\UD-NITISH\all about regression\multilinear-regress
dataset.tail(10)
```

Out[2]:

	R&D Spend	Administration	Marketing Spend	State	Profit
40	28754.33	118546.05	172795.67	California	78239.91
41	27892.92	84710.77	164470.71	Florida	77798.83
42	23640.93	96189.63	148001.11	California	71498.49
43	15505.73	127382.30	35534.17	New York	69758.98
44	22177.74	154806.14	28334.72	California	65200.33
45	1000.23	124153.04	1903.93	New York	64926.08
46	1315.46	115816.21	297114.46	Florida	49490.75
47	0.00	135426.92	0.00	California	42559.73
48	542.05	51743.15	0.00	New York	35673.41
49	0.00	116983.80	45173.06	California	14681.40

Let's check number of rows and columns of our dataset where rows and columns are store as a tuple (number of rows, number of columns). In our imported dataset, there are 50 number of rows and 5 number of columns.

In [3]:
```python
dataset.shape
```

Out[3]:
```
(50, 5)
```

Pandas describe() methos is used to view to view some basic statistical details like percentile, mean, std etc.

In [4]:
```
dataset.describe()
```

Out[4]:

	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

To know whether any cell value is empty or not

In [5]:
```
dataset.isnull().any()
```

Out[5]:
```
R&D Spend          False
Administration     False
Marketing Spend    False
State              False
Profit             False
dtype: bool
```

Our next task is to seperate features and label from our dataset. Our dataset contain 5 columns i.e. R&D Spend', 'Admisnistration', 'Marketing Spend', and 'State'. Based on these features we will predict profit where Profit is the dependent variables.

y = dependent variable: Profit
X_1 = R&D Spend
X_2 = Administration X_3 = Marketing Spend
β_0 = y-intercept at time zero
β_1 = regression coefficient that measures a unit change in the dependent variable when X_1 changes. The changes in the Profit when R&D change

β_2 = regression coefficient that measures a unit change in the dependent variable when X_2 changes. The changes in the Profit when Administration change
β_3 = regression coefficient that measures a unit change in the dependent variable when X_3 changes. The changes in the Profit when Marketing Spend change

In the multi-linear regression model allows an analyst to predict an outcome based on information provided by the multiple explanatory variables.

E which is added to the last of the equations finds the difference between the actual profit and the predicted outcome, is included in the model to account for such slight variations.

In [13]:
```
features = dataset.iloc[: , :4].values
features
print(dataset['State'].unique())
```

```
['New York' 'California' 'Florida']
```

In [7]:
```
label = dataset.iloc[:, -1].values
label
```

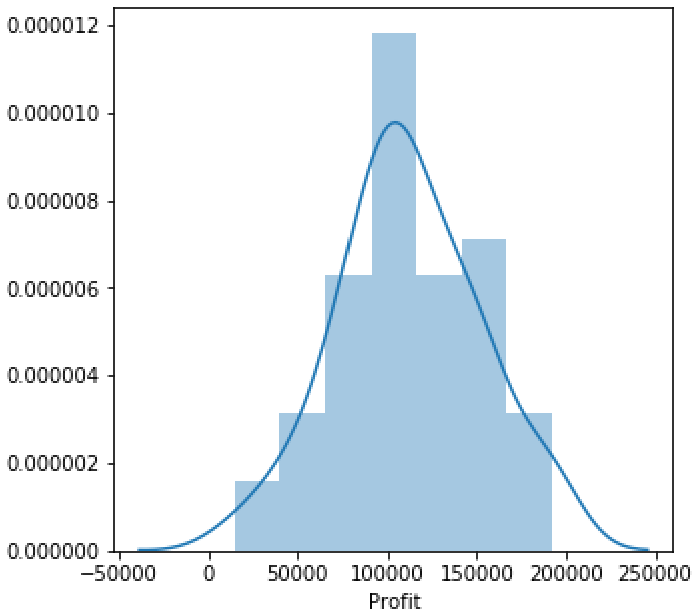
Out[7]:
```
array([192261.83, 191792.06, 191050.39, 182901.99, 166187.94, 156991.12,
       156122.51, 155752.6 , 152211.77, 149759.96, 146121.95, 144259.4 ,
       141585.52, 134307.35, 132602.65, 129917.04, 126992.93, 125370.37,
       124266.9 , 122776.86, 118474.03, 111313.02, 110352.25, 108733.99,
       108552.04, 107404.34, 105733.54, 105008.31, 103282.38, 101004.64,
        99937.59,  97483.56,  97427.84,  96778.92,  96712.8 ,  96479.51,
        90708.19,  89949.14,  81229.06,  81005.76,  78239.91,  77798.83,
        71498.49,  69758.98,  65200.33,  64926.08,  49490.75,  42559.73,
        35673.41,  14681.4 ])
```

Now, I am going to check average value of Profit column.

In [8]:
```
import seaborn as sns

plt.figure(figsize = (5,5))
plt.tight_layout()
sns.distplot(dataset['Profit'])
```

Out[8]: <matplotlib.axes._subplots.AxesSubplot at 0x2d0c4046668>



While importing our dataset, the 'State' column in the 3rd index of our dataset was presented in the form of text with the finite set of label values. So, for preprocessing sklearn provides OneHotEncoder library by which categorical variables are converted into a form that could be provided to ML algorithms to do a better job in prediction which splits the column into multiple columns and the rumber are replaced by

1s and 0s. For rows which has the first column value as 'New York', the 'New York' column will have a '1' and other two columns will have '0's. Similarly, for rows which have the first column value as 'California', the 'California' column will have a '1' and other two columns will have '0's.

first_row = " **New York** "
second_row = " **California** " third_row = " **Florida** "

first_column = " **New York** " second_column = " **California** " third_column = " **Florida** "

$$\begin{bmatrix} NewYork\,(1) & 0 & 0 \\ 0 & California\,(1) & 0 \\ 0 & 0 & Florida\,(1) \end{bmatrix}$$

In [ ]:

In [9]:

```
# Encoding categorical data
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.compose import ColumnTransformer
ct = ColumnTransformer([("State", OneHotEncoder(), [3])], remainder = 'passthrough')
encoded_features = ct.fit_transform(features)
```

In [10]:
```
encoded_features
```

Out[10]:
```
array([[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]], dtype=object)
```

skleran provides best function for partioning data into training test and testing test. We provide certain proportion of data to use as test set and we can provide the parameter random_state to ensure repeatable resuts. We split 80% of the data to the training set while 20% of data to the test using the below code. The test_size variable is where specify the propostion of the test set.

In [11]:
```
# Splitting 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(encoded_features, label, test_size = 0.2, random_state = 584
```

After spliting data into train set and test set, now our job is to train our algorithm. For that we need to import LinearRegression to minimize the residual error of squares between the observed target in the dataset and the target predicted by the linear approximation.
Now, call the fit() method along with our training data. After training our algorithm, now time to make some predictions. For this we are going to use our test data and see how correctly our algorithms predicts the percentage score.

In [12]:
```
# Fitting Multiple Linear Regression to the Training set
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

Out[12]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

In [13]:
```
X_train.shape
```

Out[13]:
```
(40, 6)
```

In [14]:
```
y_train.shape
```

Out[14]:
```
(40,)
```

In [15]:
```
X_test.shape
```

Out[15]:
```
(10, 6)
```

In [16]:
```
y_test.shape
```

Out[16]:
```
(10,)
```

The below code retrive the slope of each independent variables.

In [17]:
```
print(regressor.coef_)
```

```
[ 4.04342385e+02 -8.50379894e+02  4.46037509e+02  8.38947725e-01
 -4.30346933e-02  2.31597847e-02]
```

In [18]:
```
y_pred = regressor.predict(X_test) #y_predicted check with y_test
```

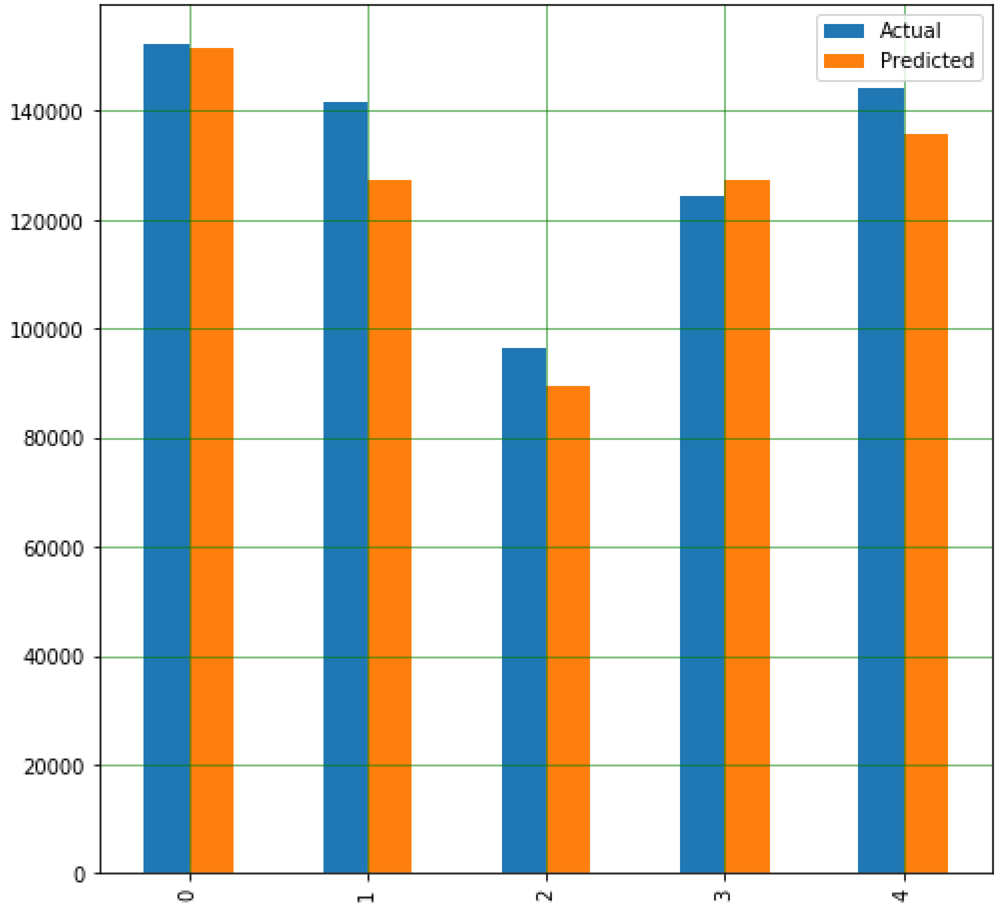
In [19]:
```
df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
df1 = df.head(5)
df1
```

Out[19]:

	Actual	Predicted
0	152211.77	151525.091979
1	141585.52	127336.793009
2	96479.51	89282.507944
3	124266.90	127172.482041
4	144259.40	135830.062704

We can use the bar graph to compare the results between the actual and predicted results. The number of record is guge, for visulizing the graph I am taking only 5 records. The below graph shows that our model has returned pretty good predictions results.

In [20]:
```
df1.plot(kind = 'bar', figsize = (8,8))
plt.grid(which='major', linestyle='-', linewidth='0.5', color='green')
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
plt.show()
```



The final step is to evaluate the performace of the algorithm. We will do this finding the values for MAE, MSE, and RMSE similarly, we have done in Simple Linear Regression.

In [21]:
```
from sklearn import metrics
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Mean Absolute Error: 8790.11684095517
Mean Squared Error: 103474053.34450155
Root Mean Squared Error: 10172.219686209179
```

Mean Absolute Error (MAE) , Mean Squared Error (MSE) and Root Mean Squared Error (RMSE) are the evaluation metrics of linear regression to compare how well algorithm perform on a particular dataset. 1. Mean Absolute Error (MAE): Mean of absolute value of the error which is calculated as: MAE = $\frac{1}{n}\sum_{i=1}^{n}|Y_i - y_i|$

```
2. Mean Squared Error: It is mean of the squared errors which is calculated as:
```

MSE = $\frac{1}{n}\sum_{i=1}^{n}(Y_i - y_i)^2$

```
3. Root Mean Squared Error (RMSE): It is squared root of the mean of the squred errors and whihc is cal
culated as:
```

RMSE = $\sqrt{\frac{1}{n}\sum_{i=1}^{n}(Y_i - y_i)^2}$

Here, the root mean square value error is 10172.219686209179 which is smaller to the mean value of Profit i.e. 112012.639200. This means that our algorithms is very accurate for good predictions

In [ ]:

In [ ]:

In [ ]: