Multiple Linear Regression

Mohammed Meraj Mohammed Ashfaque

T2 32

TY Computer

DataSet : 50_Vehicles.csv | ML Practical 3 B

Importing the libraries

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

Importing the dataset

In [2]:
```
dataset = pd.read_csv('50_Vehicles.csv')
```

EDA Steps

In [3]:
```
dataset.head()
```

Out[3]:

	Vehicle Cost	Marketing Budget	Dealer Expenses	State	Total Sales
0	45000	25000	5000	California	74000
1	42000	27000	4800	Texas	73000
2	39000	22000	4600	Florida	71000
3	37000	24000	4400	New York	69000
4	35000	20000	4300	Florida	65000

In [4]:
```
dataset.columns
```

Out[4]:
```
Index(['Vehicle Cost', 'Marketing Budget', 'Dealer Expenses', 'State',
       'Total Sales'],
      dtype='object')
```

In [5]:
```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 51 entries, 0 to 50
Data columns (total 5 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Vehicle Cost      51 non-null     int64
 1   Marketing Budget  51 non-null     int64
 2   Dealer Expenses   51 non-null     int64
 3   State             51 non-null     object
 4   Total Sales       51 non-null     int64
dtypes: int64(4), object(1)
memory usage: 2.1+ KB
```

In [6]:
```
dataset.describe()
```

Out[6]:

	Vehicle Cost	Marketing Budget	Dealer Expenses	Total Sales
count	51.000000	51.000000	51.000000	51.000000
mean	16717.647059	22843.137255	2213.725490	39711.764706
std	12117.173035	5311.770134	1446.688578	17220.007515
min	0.000000	10500.000000	0.000000	9000.000000
25%	6750.000000	21000.000000	950.000000	26750.000000
50%	14500.000000	23500.000000	2200.000000	36500.000000
75%	25500.000000	27000.000000	3350.000000	52000.000000
max	45000.000000	32000.000000	5000.000000	74000.000000

Preprocessing Steps

1. Preparing Data as input and output

In [7]:
```python
X = dataset.iloc[:,:-1].values
y = dataset.iloc[:,-1].values
```

In [8]:
```python
print(X)
```

```
[[45000 25000 5000 'California']
 [42000 27000 4800 'Texas']
 [39000 22000 4600 'Florida']
 [37000 24000 4400 'New York']
 [35000 20000 4300 'Florida']
 [33000 21000 4100 'California']
 [34000 26000 4000 'Texas']
 [31000 25500 3900 'Florida']
 [29000 27000 3800 'California']
 [30000 23000 3700 'New York']
 [26000 23500 3600 'Florida']
 [25000 21000 3500 'Texas']
 [24000 25500 3400 'Florida']
 [23000 27000 3300 'California']
 [28000 29000 3200 'Florida']
 [27000 23000 3100 'New York']
 [18000 22500 3000 'Texas']
 [22000 26000 2900 'California']
 [21500 23500 2800 'Florida']
 [20000 27500 0 'New York']
 [17000 21000 2700 'Texas']
 [18000 27500 2600 'California']
 [16000 23000 2500 'Florida']
 [14000 21000 2400 'Florida']
 [17500 20000 2300 'New York']
 [12000 28000 2200 'Texas']
 [15000 29000 2100 'Florida']
 [14500 26000 2000 'New York']
 [13000 32000 1900 'Florida']
 [12500 28000 1800 'New York']
 [11000 22500 1700 'Florida']
 [10500 29000 1600 'New York']
 [11500 24500 1500 'California']
 [10000 20000 1400 'Florida']
 [8500 29500 1300 'Texas']
 [8000 16000 1200 'California']
 [7500 12000 1100 'New York']
 [5000 25500 1000 'Florida']
 [7000 10500 900 'Texas']
 [3500 13500 800 'New York']
 [6500 14500 700 'California']
 [5000 18000 600 'California']
 [4800 12000 500 'Florida']
 [4000 13500 450 'California']
 [2500 26000 400 'New York']
 [3800 29000 350 'Texas']
 [1800 23000 300 'New York']
 [2500 22000 200 'Florida']
 [0 27000 0 'California']
 [200 10500 0 'New York']
 [0 22000 3000 'California']]
```

2. Encoding categorical data

In [9]:
```python
dataset['State'].value_counts()
```

Out[9]:
```
State
Florida       16
California    13
New York      13
Texas          9
Name: count, dtype: int64
```

In [10]:
```python
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 [11]:
```python
print(X)
```

```
[[1.0 0.0 0.0 0.0 45000 25000 5000]
 [0.0 0.0 0.0 1.0 42000 27000 4800]
 [0.0 1.0 0.0 0.0 39000 22000 4600]
 [0.0 0.0 1.0 0.0 37000 24000 4400]
 [0.0 1.0 0.0 0.0 35000 20000 4300]
 [1.0 0.0 0.0 0.0 33000 21000 4100]
 [0.0 0.0 0.0 1.0 34000 26000 4000]
 [0.0 1.0 0.0 0.0 31000 25500 3900]
 [1.0 0.0 0.0 0.0 29000 27000 3800]
 [0.0 0.0 1.0 0.0 30000 23000 3700]
 [0.0 1.0 0.0 0.0 26000 23500 3600]
 [0.0 0.0 0.0 1.0 25000 21000 3500]
 [0.0 1.0 0.0 0.0 24000 25500 3400]
 [1.0 0.0 0.0 0.0 23000 27000 3300]
 [0.0 1.0 0.0 0.0 28000 29000 3200]
 [0.0 0.0 1.0 0.0 27000 23000 3100]
 [0.0 0.0 0.0 1.0 18000 22500 3000]
 [1.0 0.0 0.0 0.0 22000 26000 2900]
 [0.0 1.0 0.0 0.0 21500 23500 2800]
 [0.0 0.0 1.0 0.0 20000 27500 0]
 [0.0 0.0 0.0 1.0 17000 21000 2700]
 [1.0 0.0 0.0 0.0 18000 27500 2600]
 [0.0 1.0 0.0 0.0 16000 23000 2500]
 [0.0 1.0 0.0 0.0 14000 21000 2400]
 [0.0 0.0 1.0 0.0 17500 20000 2300]
 [0.0 0.0 0.0 1.0 12000 28000 2200]
 [0.0 1.0 0.0 0.0 15000 29000 2100]
 [0.0 0.0 1.0 0.0 14500 26000 2000]
 [0.0 1.0 0.0 0.0 13000 32000 1900]
 [0.0 0.0 1.0 0.0 12500 28000 1800]
 [0.0 1.0 0.0 0.0 11000 22500 1700]
 [0.0 0.0 1.0 0.0 10500 29000 1600]
 [1.0 0.0 0.0 0.0 11500 24500 1500]
 [0.0 1.0 0.0 0.0 10000 20000 1400]
 [0.0 0.0 0.0 1.0 8500 29500 1300]
 [1.0 0.0 0.0 0.0 8000 16000 1200]
 [0.0 0.0 1.0 0.0 7500 12000 1100]
 [0.0 1.0 0.0 0.0 5000 25500 1000]
 [0.0 0.0 0.0 1.0 7000 10500 900]
 [0.0 0.0 1.0 0.0 3500 13500 800]
 [1.0 0.0 0.0 0.0 6500 14500 700]
 [1.0 0.0 0.0 0.0 5000 18000 600]
 [0.0 1.0 0.0 0.0 4800 12000 500]
 [1.0 0.0 0.0 0.0 4000 13500 450]
 [0.0 0.0 1.0 0.0 2500 26000 400]
 [0.0 0.0 0.0 1.0 3800 29000 350]
 [0.0 0.0 1.0 0.0 1800 23000 300]
 [0.0 1.0 0.0 0.0 2500 22000 200]
 [1.0 0.0 0.0 0.0 0 27000 0]
 [0.0 0.0 1.0 0.0 200 10500 0]
 [1.0 0.0 0.0 0.0 0 22000 3000]]
```

3. Splitting the dataset into the Training set and Test set

In [12]:
```python
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 [13]:
```python
X_train.shape
```

Out[13]:
```
(40, 7)
```

Training the Multiple Linear Regression model on the Training set

In [14]:
```python
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train,y_train)
```

Out[14]:
```
▾ LinearRegression
LinearRegression()
```

In [15]:
```python
# check th attribute of our model
print("Coefficient of our model", regressor.coef_)
print("intercept  of our model", regressor.intercept_)
```

```
Coefficient of our model [-8.19741935e+02 -4.43415254e+01 -7.22939405e+02  1.58702287e+03
  1.39452743e+00  1.08476099e-02 -2.25992787e-01]
intercept  of our model 16747.946106526033
```

In [16]:
```
# score of our model
print("Training Accuracy of our model", regressor.score(X_train, y_train))
print("Testing Accuracy of our model", regressor.score(X_test, y_test))
```

```
Training Accuracy of our model 0.9753510142832125
Testing Accuracy of our model 0.9765828965294642
```

Predicting the Test set results

In [17]:
```
y_pred = regressor.predict(X_test)
print(y_test)
print(y_pred)
```

```
[33000 56000 57200 40000 71000 34500 19000 30500 36000 65000 30000]
[33353.54563508 52634.97977343 52402.66255826 38700.55651959
 70289.25494559 34750.1983905  23869.65641613 31892.04687682
 37461.51186343 64757.24784227 30549.44117589]
```

Making a single prediction (for example the profit of a startup with Vehicle Cost = 160000, Marketing Budget = 130000, Dealer Expenses = 300000 and State = 'California')

In [19]:
```
print(regressor.predict([[1.0,0.0,0.0,0.0,160000,130000,300000]]))
```

```
[172664.94625646]
```

Getting the final linear regression equation with the values of the coefficients

variance score: 1 means perfect prediction

In [20]:
```
print('Variance score: {}'.format(regressor.score(X_test, y_test)))
```

```
Variance score: 0.9765828965294642
```

In above example, we determine accuracy score using Explained Variance Score.

We define: **explained_variance_score = 1 – Var{y – y'}/Var{y}**

where

- y' is the estimated target output,
- y the corresponding (correct) target output
- Var is Variance, the square of the standard deviation.

***The best possible score is 1.0, lower values are worse.**

plot for residual error

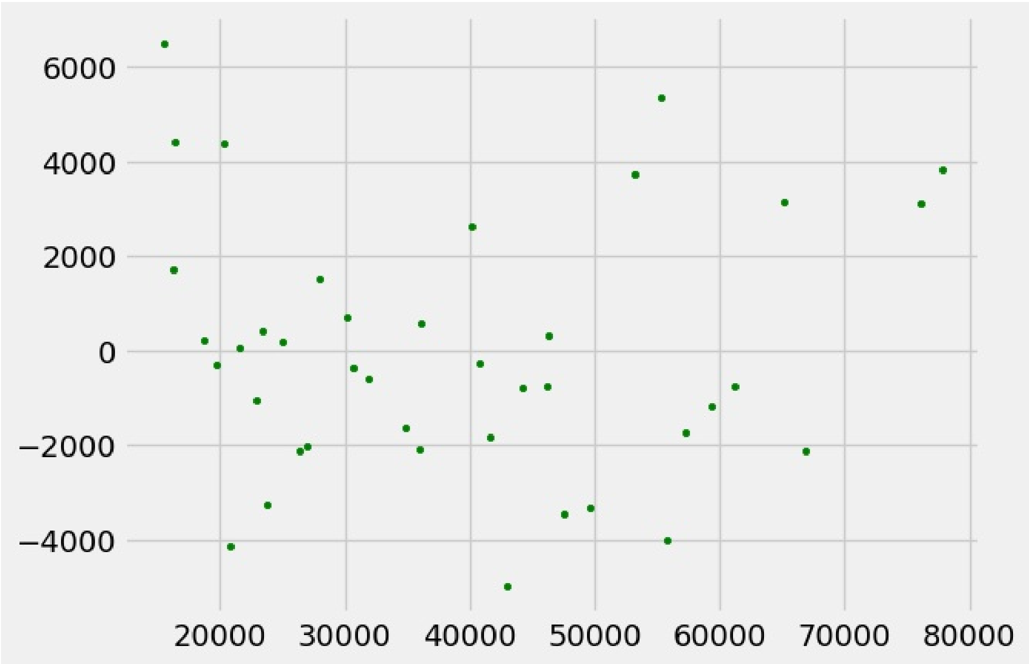
setting plot style

In [21]:
```
plt.style.use('fivethirtyeight')
```

plotting residual errors in training data

In [22]:
```
plt.scatter(regressor.predict(X_train),
            regressor.predict(X_train) - y_train,
            color = "green", s = 10, label = 'Train data')
```

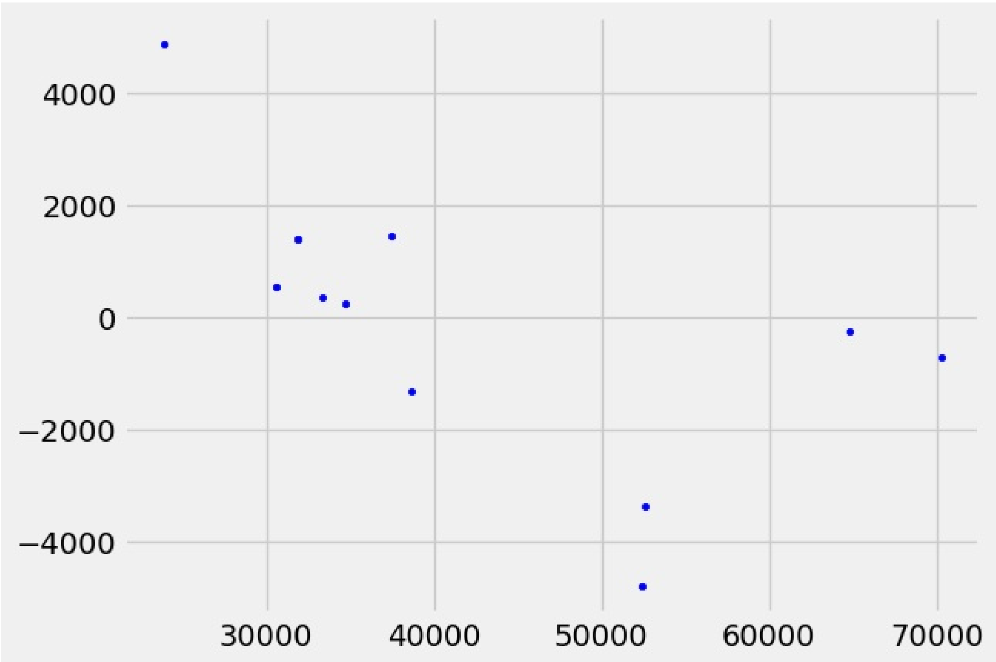
Out[22]:
```
<matplotlib.collections.PathCollection at 0x1e08401cdd0>
```



plotting residual errors in test data

In [23]:
```
plt.scatter(regressor.predict(X_test),
            regressor.predict(X_test) - y_test,
            color = "blue", s = 10, label = 'Test data')
```

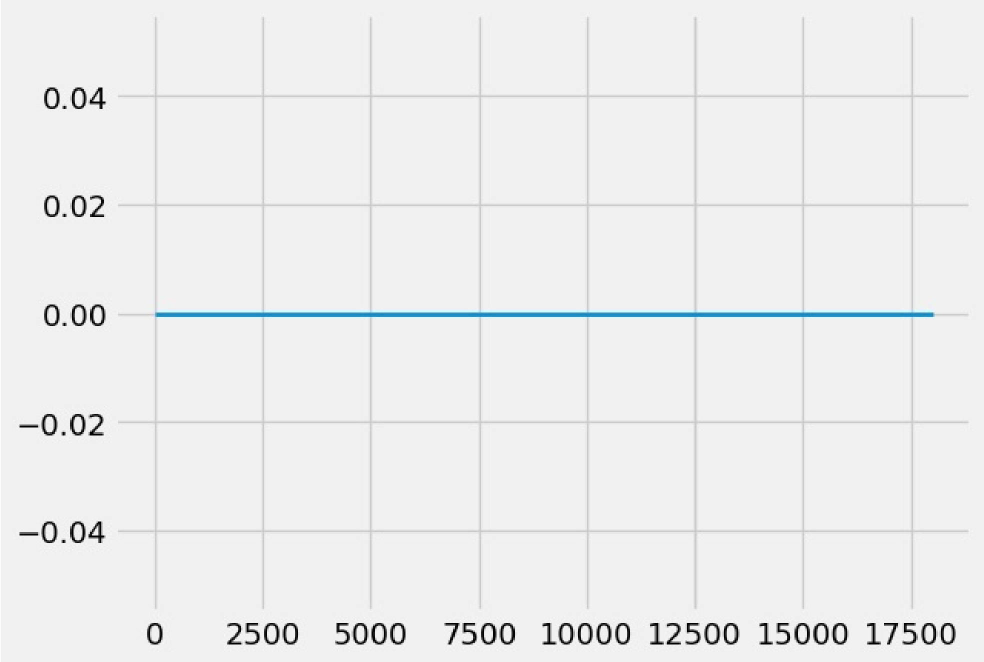
Out[23]: <matplotlib.collections.PathCollection at 0x1e084021d30>



plotting line for zero residual error

In [24]:
```
plt.hlines(y = 0, xmin = 0, xmax = 18000, linewidth = 2)
```

Out[24]: <matplotlib.collections.LineCollection at 0x1e0840c4740>

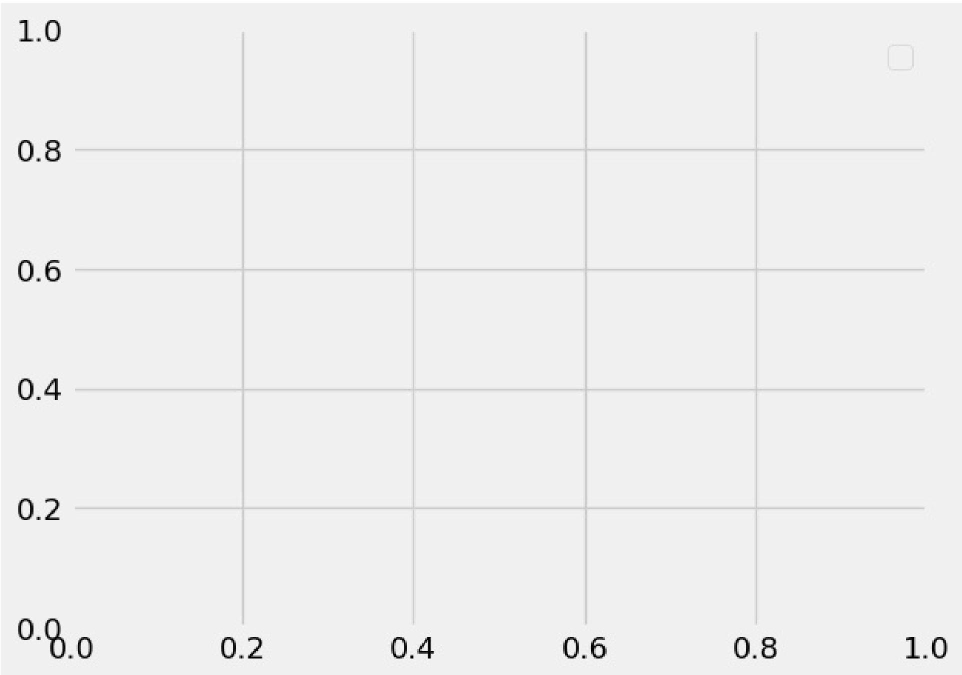


plotting legend

In [26]:
```
plt.legend(loc = 'upper right')
```

```
C:\Users\Mohammed Meraj\AppData\Local\Temp\ipykernel_9496\3738487734.py:1: UserWarning: No artists with labels f
ound to put in legend.  Note that artists whose label start with an underscore are ignored when legend() is call
ed with no argument.
  plt.legend(loc = 'upper right')
```

Out[26]: <matplotlib.legend.Legend at 0x1e086dde390>



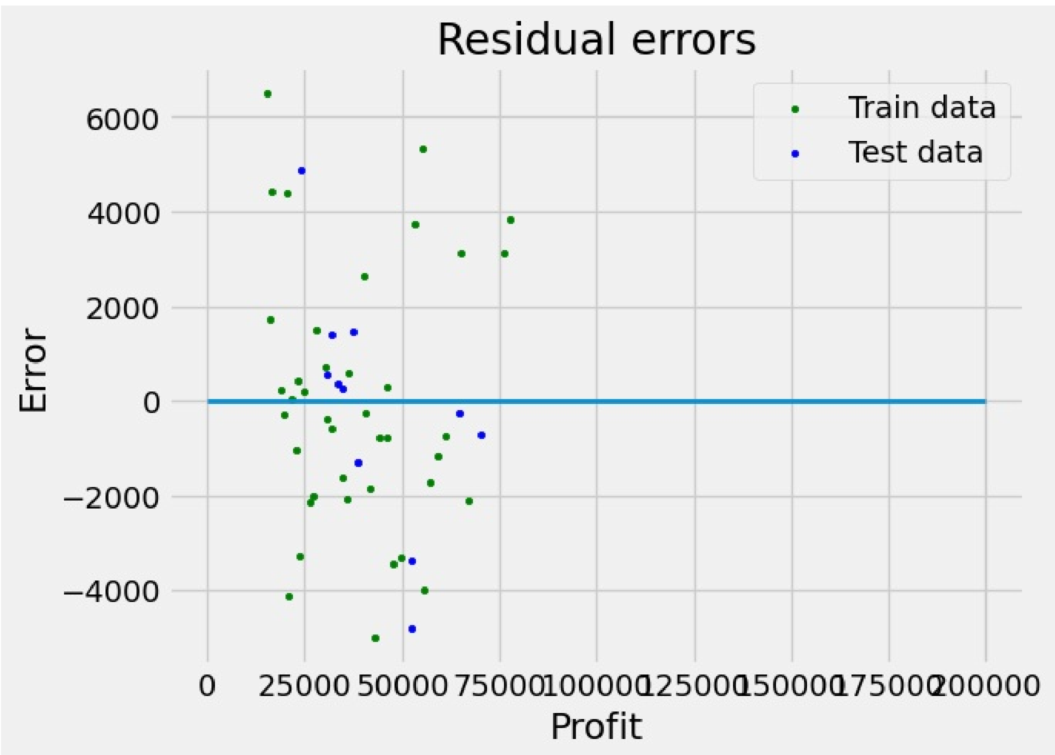
plot title

In [27]:
```
plt.title("Residual errors")
```

Out[27]: Text(0.5, 1.0, 'Residual errors')

function to show plot

```
In [28]: plt.style.use('fivethirtyeight')
plt.scatter(regressor.predict(X_train), regressor.predict(X_train) - y_train,color = "green", s = 10, label = '
plt.scatter(regressor.predict(X_test), regressor.predict(X_test) - y_test,color = "blue", s = 10, label = 'Test
plt.hlines(y = 0, xmin = 0, xmax = 200000, linewidth = 2)
plt.legend(loc = 'upper right')
plt.title("Residual errors")
plt.xlabel("Profit")
plt.ylabel("Error")
plt.show()
```



EVALUATING A MODEL USING R2 METRIC

Find the R^2

In [29]:
```
from sklearn.metrics import r2_score
print(r2_score(y_test,y_pred))
```

```
0.9765828965294642
```

In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js