# Bivariate Regression Model

October 29, 2020

## 1 Simple Linear Regression

Building Simple Linear Regression Model to Predict Sales(target variable) using Tv Budget(features or predictor variable)

### 1.1 Understanding the Data

Importing Data using the Pandas Libray

[1]:
```
import pandas as pd
```

[2]:
```
data = pd.read_csv("tvmarketing.csv")
```

Checking the structure of our Datasets

[3]:
```
# Display the first 5 rows
data.head()
```

[3]:

| | TV | Sales |
|---|---|---|
| 0 | 230.1 | 22.1 |
| 1 | 44.5 | 10.4 |
| 2 | 17.2 | 9.3 |
| 3 | 151.5 | 18.5 |
| 4 | 180.8 | 12.9 |

[4]:
```
# Display the last 5 rows
data.tail()
```

[4]:

| | TV | Sales |
|---|---|---|
| 195 | 38.2 | 7.6 |
| 196 | 94.2 | 9.7 |
| 197 | 177.0 | 12.8 |
| 198 | 283.6 | 25.5 |
| 199 | 232.1 | 13.4 |

[5]:
```
# Display information about the data
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
```

```
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   TV      200 non-null    float64
 1   Sales   200 non-null    float64
dtypes: float64(2)
memory usage: 3.2 KB
```

[7]:
```python
# Display the matrix dimension of the data
data.shape
```

[7]:
```
(200, 2)
```

[9]:
```python
# Display discriptitive statistics about the data
data.describe()
```

[9]:

| | TV | Sales |
|---|---|---|
| count | 200.000000 | 200.000000 |
| mean | 147.042500 | 14.022500 |
| std | 85.854236 | 5.217457 |
| min | 0.700000 | 1.600000 |
| 25% | 74.375000 | 10.375000 |
| 50% | 149.750000 | 12.900000 |
| 75% | 218.825000 | 17.400000 |
| max | 296.400000 | 27.000000 |

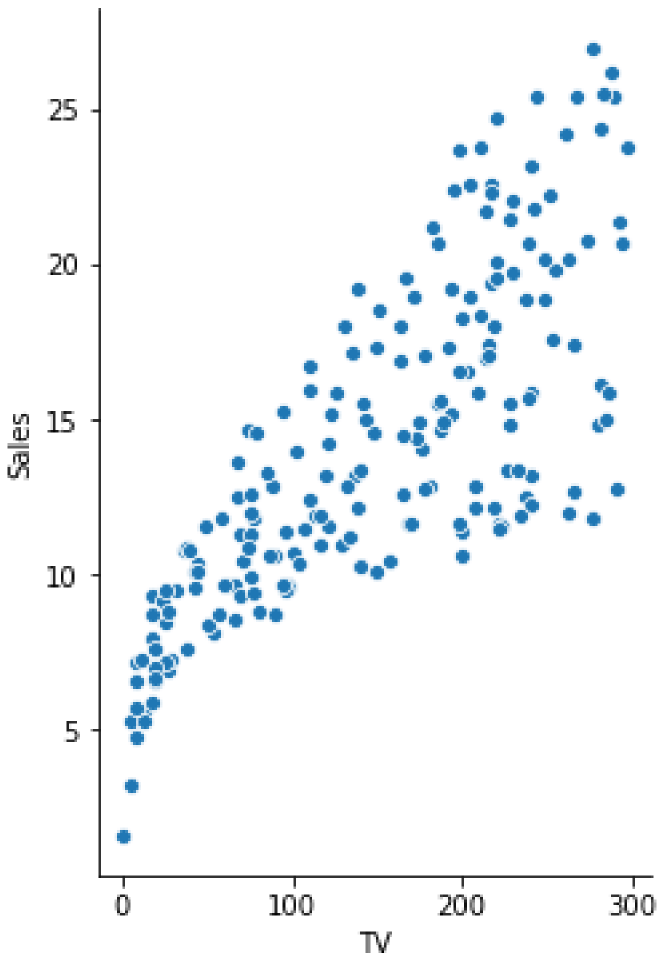
### 1.2 Visualizing Data using Seaborn Library

[10]:
```python
# Importing seaborn as sns
import seaborn as sns


# To visualize in the notebook
%matplotlib inline
```

[19]:
```python
# visualizing the correlation between features and targets using scatterplots
sns.pairplot(data, x_vars="TV", y_vars="Sales", height=5, aspect=0.7,␣
↪kind="scatter")
```

[19]:
```
<seaborn.axisgrid.PairGrid at 0x25a8f670f70>
```




### 1.3 Modelling using Basic Math

From basic primary mathematics

Equation of a line is: $y$ = c + m$x$

Let formulate it: $y = \alpha + \beta x$

where

$y$ is target or dependent or explained or response or predicted or regressand variable (Output)

$x$ is features or independent or explanatory or control or predictor or regressor variable (Input)

$\alpha$ is intercept

$\beta$ is slope

## 1.4 Steps in Model Building using Sklearn Library

[23]:
```
# Putting features variable to x
x = data["TV"]

# Putting targets variable to y
y = data["Sales"]
```

### 1.4.1 Splitting data into training set and testing set

[28]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, train_size=0.7,␣
 ↪random_state=100)
```

[29]:
```
train_test_split
```

[29]:
```
<function sklearn.model_selection._split.train_test_split(*arrays, **options)>
```

[30]:
```
# Converting into vector forms us numpy
import numpy as np

x_train = x_train[:, np.newaxis]
x_test = x_test[:, np.newaxis]
```

## 1.5 Performing Linear Regression

[31]:
```
# importing Linear regression from sklearn
from sklearn.linear_model import LinearRegression

# Representing LinearRegression as reg
reg = LinearRegression()

# fitting the model using reg.fit()
reg.fit(x_train, y_train)
```

[31]:
```
LinearRegression()
```

## 1.6 Coefficients Result

[38]:
```
# print the intercept and slope
print("Intercept:")
print(reg.intercept_)
print("Slope:")
print(reg.coef_[0])
```

```
Intercept:
6.989665857411679
```

```
Slope:
0.046497358747865765
```

$y = 6.99 + 0.047\beta$

Let use this equation to predict sales

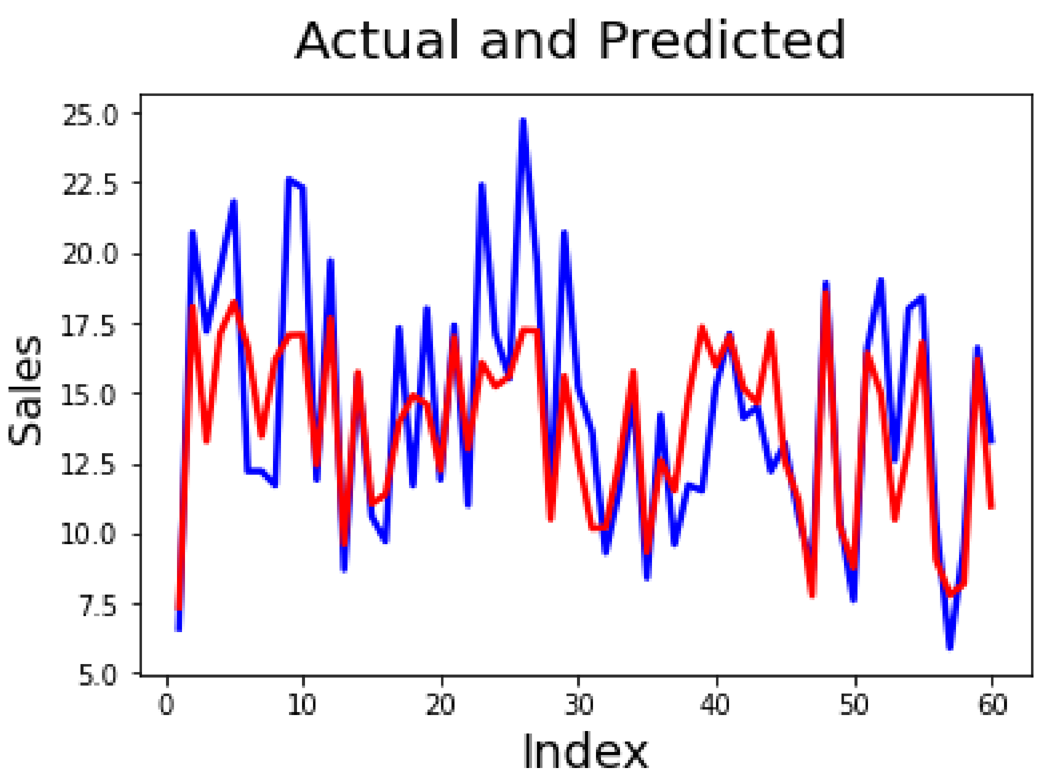
### 1.7 Predictions

[39]:
```python
# making prediction on the testing set
y_predict = reg.predict(x_test)
```

### 1.8 Plotting and computting RMSE and R-squared

[40]:
```python
# Actual vs Predicted
import matplotlib.pyplot as plt
c = [i for i in range(1,61,1)]         # generating index
fig = plt.figure()
plt.plot(c,y_test, color="blue", linewidth=2.5, linestyle="-")
plt.plot(c,y_predict, color="red",  linewidth=2.5, linestyle="-")
fig.suptitle('Actual and Predicted', fontsize=20)              # Plot heading
plt.xlabel('Index', fontsize=18)                               # X-label
plt.ylabel('Sales', fontsize=16)                               # Y-label
```

[40]:
```
Text(0, 0.5, 'Sales')
```



[41]:
```
# Error terms
c = [i for i in range(1,61,1)]
fig = plt.figure()
plt.plot(c,y_test-y_predict, color="blue", linewidth=2.5, linestyle="-")
fig.suptitle('Error Terms', fontsize=20)              # Plot heading
plt.xlabel('Index', fontsize=18)                      # X-label
plt.ylabel('ytest-ypredict', fontsize=16)             # Y-label
```

[41]: Text(0, 0.5, 'ytest-ypredict')


[42]:
```
from sklearn.metrics import mean_squared_error, r2_score
mse = mean_squared_error(y_test, y_predict)
```

[45]:
```
r_squared = r2_score(y_test, y_predict)
```

[46]:
```
print('Mean_Squared_Error :' ,mse)
print('r_square_value :',r_squared)
```

```
Mean_Squared_Error : 7.975798532854851
r_square_value : 0.5942987267783302
```

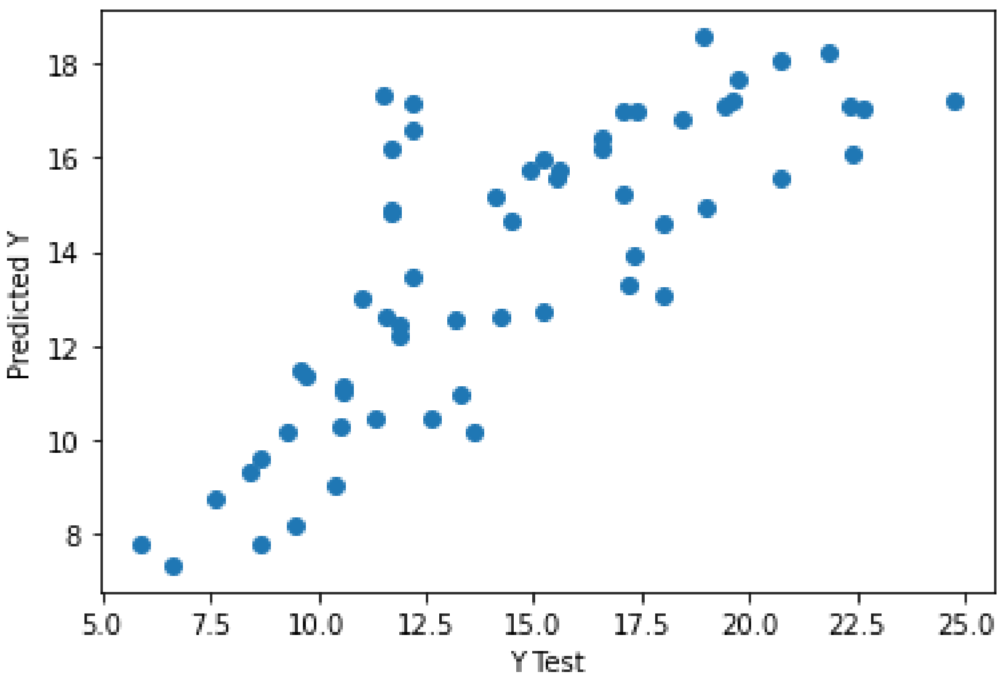
R-squared tell us the Goodness of fit of our model.

Thus almost 60% of the variation in the dataset was explained by the model

The Mean square error tells us that about 8% of value doesnt match the values of the model

[48]:
```python
import matplotlib.pyplot as plt
plt.scatter(y_test,y_predict)
plt.xlabel('Y Test')
plt.ylabel('Predicted Y')
```

[48]: Text(0, 0.5, 'Predicted Y')




[ ]: