1.Linear Regression (SKLearn)

October 23, 2021

Simple Linear Regression

[166]:
```
# Import necessary package
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

0.0.1 Step 1: Load the dataset

[167]:
```
# Load the dataset into pandas dataframe
df=pd.read_csv("E:\\MY LECTURES\\DATA SCIENCE\\3.Programs\\dataset\\Advertising.
↪csv")
# Change this location based on the location of dataset in your machine
```

[168]:
```
# Display the first five records
df.head()
```

[168]:

	TV	radio	newspaper	sales
0	230.1	37.8	69.2	22.1
1	44.5	39.3	45.1	10.4
2	17.2	45.9	69.3	9.3
3	151.5	41.3	58.5	18.5
4	180.8	10.8	58.4	12.9

Advertising data comprises four features: TV, radio, newspaper, and sales. It explains the budget (in 1000$) spent on different mass media and the net outcome for every week.

sales for a product (output/dependent/target variable).

advertising budget for TV, radio, and newspaper media (input/independent/target variable).

Planning to perform regression on TV budget (X) as input and sales (Y) as output.

[169]:
```
# Dataset shape (number of rows and columns)
df.shape
```

[169]:
```
(200, 4)
```

Row <=> record, tuple, instance, sample, observation, object, case, entity Column <=> attribute, variable, field, feature, characteristic, dimension

0.0.2 Step 2: Apply EDA

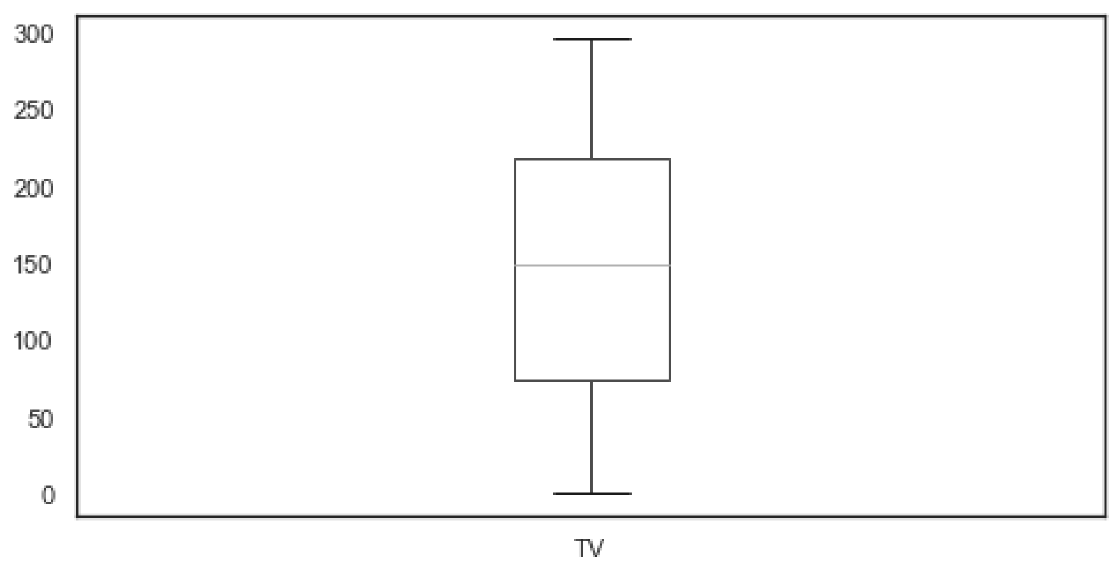
Univariate analysis

[170]:
```
# Statistics summary
df["TV"].describe()
```

[170]:
```
count    200.000000
mean     147.042500
std       85.854236
min        0.700000
25%       74.375000
50%      149.750000
75%      218.825000
max      296.400000
Name: TV, dtype: float64
```

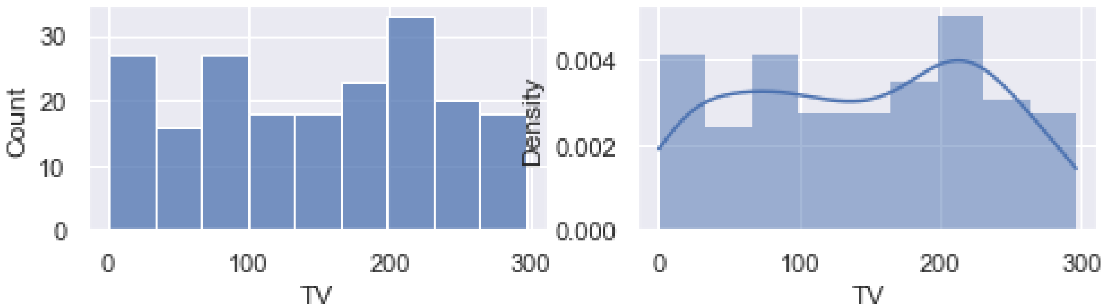
[171]:
```
# Univariate Analysis using Boxplot
sns.set_style(style='white')
df.boxplot(column =['TV'], grid = False)
```

[171]:
```
<AxesSubplot:>
```



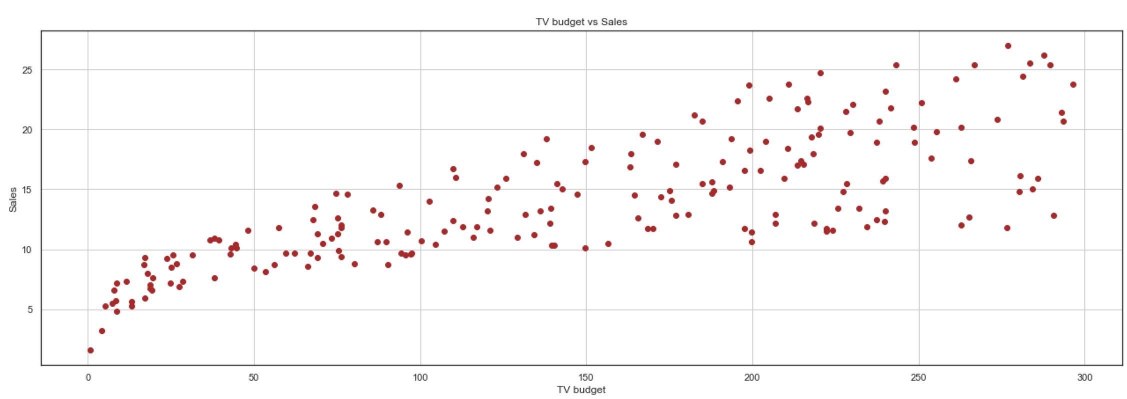
[172]:
```
# Distribution plot to find skewness
from pylab import *
```

```
sns.set(rc={"figure.figsize": (8, 4)});

subplot(2,2,1)
ax = sns.histplot(df["TV"])

subplot(2,2,2)
ax = sns.histplot(df["TV"], kde=True, stat="density", linewidth=0)

plt.show()
```



Bivariate analysis

[173]:
```
# Scatter plot
sns.set_style(style='white')
fig = plt.figure(figsize=(22,7))
plt.scatter(df["TV"],df["sales"],color="brown")
plt.grid(b=None)
plt.xlabel("TV budget")
plt.ylabel("Sales")
plt.title("TV budget vs Sales")
plt.show()
```



[174]:
```
# Correlation
r = df["TV"].corr(df["sales"])
print("Correlation value : ",round(r,2)*100,"%")
# Spearman's rho
# df["TV"].corr(df["sales"], method='spearman')
# Kendall's tau
# df["TV"].corr(df["sales"], method='kendall')
```

```
Correlation value :  78.0 %
```

0.0.3 Step 3. Pre-process and extract the features

[175]:
```
# Load TV into X and sales into Y variable
X = df.iloc[:,0].values.reshape((-1, 1))    # Budget spent on TV
Y = df.iloc[:,3].values                     # Sales
```

[176]:
```
def disp_data(feature1,feature2):
    print('Displaying only 10 records')
    print('-------------------------')
    print('TV budget','|','Sales')
    print('-------------------------')
    count = 0
    for x,y in zip(feature1,feature2):
        if count == 10:
            break
        else:
            print(x,'     ',y)
            count = count + 1
```

[177]:
```
disp_data(X,Y)
```

```
Displaying only 10 records
-------------------------
TV budget | Sales
-------------------------
[230.1]       22.1
[44.5]       10.4
[17.2]       9.3
[151.5]       18.5
[180.8]       12.9
[8.7]       7.2
[57.5]       11.8
[120.2]       13.2
[8.6]       4.8
[199.8]       10.6
```

input feature independent feature or predictor feature. Here, X1 (TV) is the input feature. output feature dependent feature or response feature or target feature. Here, Y (sales) is the output feature.

0.0.4 Step 4. Split the data for training and testing

[178]:
```
# Splitting dataset into training and testing 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)
```

[179]:
```
print("Training data")
print("==========================")
disp_data(x_train,y_train)
```

```
Training data
==========================
Displaying only 10 records
-------------------------
TV budget | Sales
-------------------------
[36.9]       10.8
[31.5]       9.5
[142.9]       15.0
[209.6]       15.9
[215.4]       17.1
[102.7]       14.0
[8.6]       4.8
[16.9]       8.7
[125.7]       15.9
[104.6]       10.4
```

[180]:
```
print("Testing data")
print("==========================")
disp_data(x_test,y_test)
```

```
Testing data
==========================
Displaying only 10 records
-------------------------
TV budget | Sales
-------------------------
[69.2]       11.3
[50.]       8.4
[90.4]       8.7
[289.7]       25.4
[170.2]       11.7
[56.2]       8.7
```

```
[8.7]       7.2
[240.1]       13.2
[23.8]       9.2
[197.6]       16.6
```

0.0.5 Step 5: Training phase (bulding the model)

[181]:
```python
# Fitting line on two dimension on the training set
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(x_train, y_train)
```

[181]:
```
LinearRegression()
```

[182]:
```python
b = model.intercept_
```

[183]:
```python
m = model.coef_
```

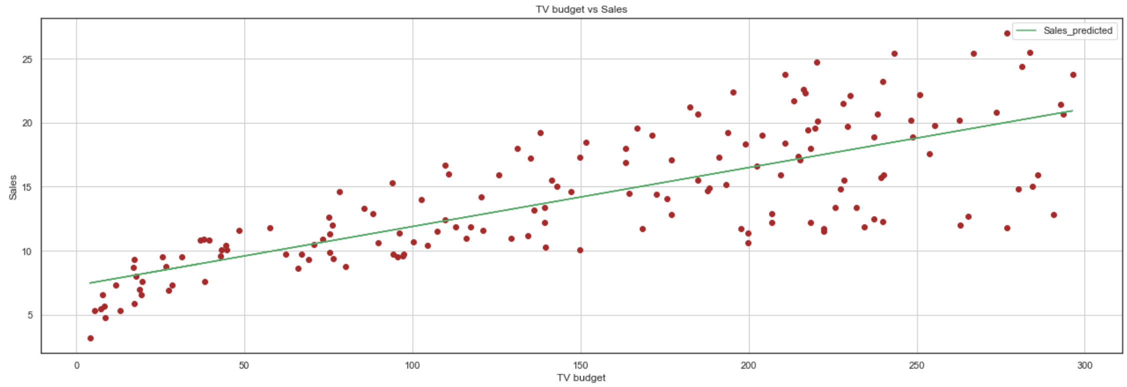
[184]:
```python
print("The linear model is ",'Y = m X + b \n')
print('Y = ',np.round(m[0],3),'X + ',np.round(b,3))
```

```
The linear model is  Y = m X + b

Y =  0.046 X +  7.292
```

[185]:
```python
# Predicting the Training set results
y_train_pred = model.predict(x_train)
```

Visualizing the model

[186]:
```python
sns.set_style(style='white')
fig = plt.figure(figsize=(22,7))
plt.scatter(x_train,y_train,color="brown")
plt.grid(b=None)
plt.plot(x_train,y_train_pred,"g",label="Sales_predicted")
plt.xlabel("TV budget")
plt.ylabel("Sales")
plt.title("TV budget vs Sales")
plt.legend()
plt.show()
```

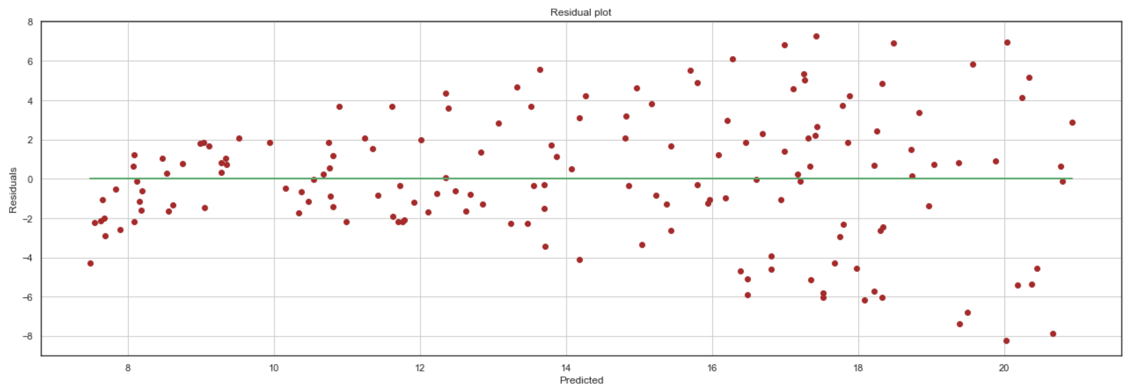


Plotting observed sale (x) and predicted sale (y) for training set

[187]:
```
# Predicting the Test set results
x = np.arange(len(y_train_pred))
fig = plt.figure(figsize=(22,7))
plt.plot(x,y_train,"g-+",label="Sales_observed")
plt.plot(x,y_train_pred,"r-*",label="Sales_predicted")
plt.grid(b=None)
plt.xlabel("TV budget")
plt.ylabel("Sales")
plt.title("TV budget vs Sales")
plt.legend()
plt.show()
```

Residual (Error) plot If the model has done good predictions, then the datapoints must be near around to horizontal line.

[188]:
```
sns.set_style(style='white')
fig = plt.figure(figsize=(22,7))
residuals = y_train-y_train_pred
zeros = y_train-y_train
plt.scatter(y_train_pred,residuals,color="brown")
plt.grid(b=None)
plt.plot(y_train_pred,zeros,"g")
plt.xlabel("Predicted")
plt.ylabel("Residuals")
plt.title("Residual plot")
plt.show()
```



0.0.6 Different error calculations to asses the model for training set

1. Sum of Squared Error (SSE)

$$SSE(m,b) = \sum_{i=1}^{n}(y_i - \hat{y})^2 = \sum_{i=1}^{n}(y_i - (m * x_i + b))^2 \quad (1)$$

[189]:
```
sum = 0
n = len(x_train)
for i in range (0,n):
  diff = y_train[i] - y_train_pred[i]
  squ_diff = diff**2
  sum = sum + squ_diff
Train_SSE = np.round(sum,2)
print("Sum of Squared Error (SSE) :",Train_SSE)
```

```
Sum of Squared Error (SSE) : 1698.76
```

2. Mean Squared Error (MSE)

$$MSE(m,b) = \frac{\sum_{i=1}^{n}(y_i - \hat{y})^2}{n} = \frac{\sum_{i=1}^{n}(y_i - (m * x_i + b))^2}{n} \tag{2}$$

[190]:
```
Train_MSE = np.round(Train_SSE/n,2)
print("Mean Squared Error (MSE) :",Train_MSE)
```

```
Mean Squared Error (MSE) : 10.62
```

3. Root Mean Squared Error (RMSE)

$$RMSE(m,b) = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y})^2}{n}} = \sqrt{\frac{\sum_{i=1}^{n}(y_i - (m * x_i + b))^2}{n}} \tag{3}$$

[191]:
```
Train_RMSE = np.round(np.sqrt(Train_MSE),2)
print("Root Mean Squared Error (RMSE) :",Train_RMSE)
```

```
Root Mean Squared Error (RMSE) : 3.26
```

4. Mean Absolute Error (MAE)

$$MAE(m,b) = \frac{\sum_{i=1}^{n}|(y_i - \hat{y})|}{n} \tag{4}$$

[192]:
```
sum = 0
n = len(x_train)
for i in range (0,n):
  diff = y_train[i] - y_train_pred[i]
  sum = sum + np.abs(diff)
Train_MAE = np.round(sum/n,2)
print("Mean Absolute Error (MAE) :",Train_MAE)
```

```
Mean Absolute Error (MAE) : 2.58
```

5. Mean Absolute Percentage Error (MAPE)

$$MAPE(m,b) = \frac{100\%}{n}\sum_{i=1}^{n}\left|\frac{(y_i - \hat{y})}{y_i}\right| = \frac{100\%}{n}\sum_{i=1}^{n}\left|\frac{(y_i - (m * x_i + b))}{y_i}\right| \tag{5}$$

[193]:
```
sum = 0
n = len(x_train)
for i in range (0,n):
  diff = (y_train[i] - y_train_pred[i])/y_train[i]
  sum = sum + np.abs(diff)
```

```
Train_MAPE = np.round(sum/n*100,2)
print("Mean Absolute Percentage Error (MAPE) :",Train_MAPE)
```

```
Mean Absolute Percentage Error (MAPE) : 19.54
```

0.0.7 Calculating R-Squred value (goodness of model) using SSE

$$R^2 = 1 - \frac{SSE}{SST} = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y})^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{6}$$

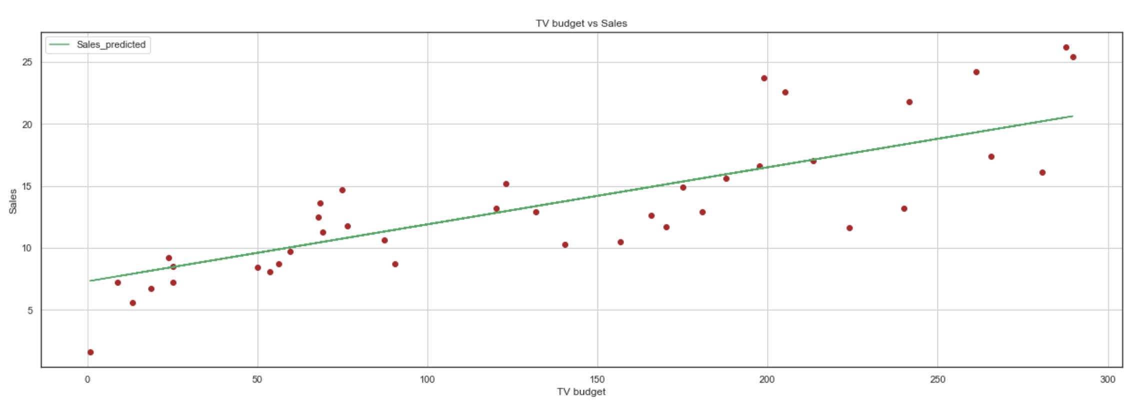
[194]:
```
from sklearn.metrics import r2_score
out = r2_score(y_train,y_train_pred)
Train_RS = np.round(out,2)*100
print("R-Squred value (goodness of model) for training set :",Train_RS,"%")
```

```
R-Squred value (goodness of model) for training set : 59.0 %
```

0.0.8 Step 6: Testing phase

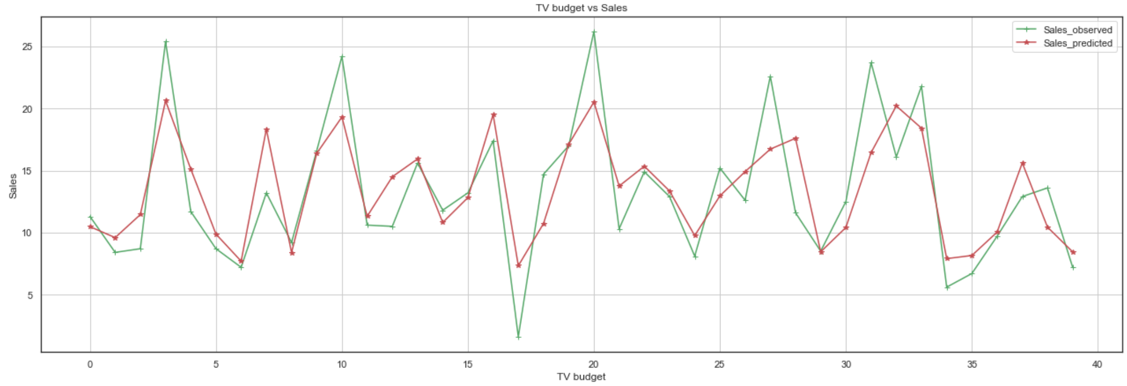
[195]:
```
# Predicting values for test input set
y_test_pred = model.predict(x_test)
```

[196]:
```
# Plotting the predicted values
sns.set_style(style='white')
fig = plt.figure(figsize=(22,7))
plt.scatter(x_test,y_test,color="brown")
plt.grid(b=None)
plt.plot(x_test,y_test_pred,"g",label="Sales_predicted")
plt.xlabel("TV budget")
plt.ylabel("Sales")
plt.title("TV budget vs Sales")
plt.legend()
plt.show()
```



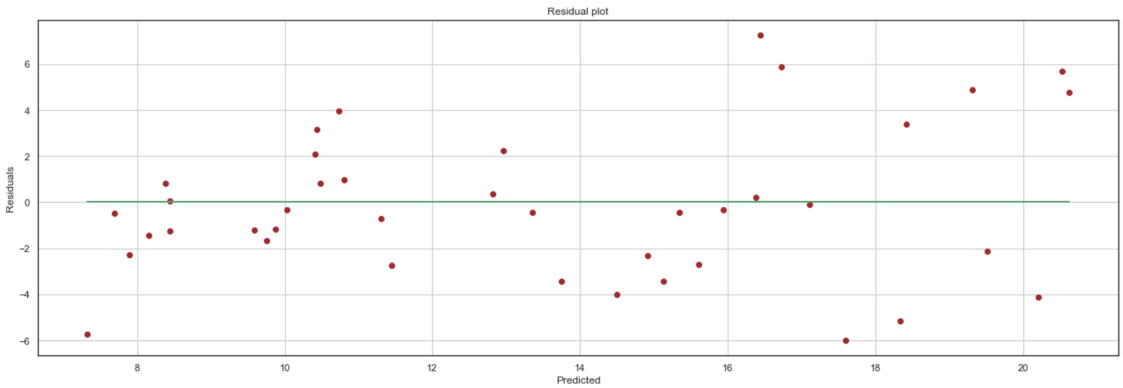
Plotting observed sale (x) and predicted sale (y) for test set

[197]:
```
x = np.arange(len(y_test_pred))
fig = plt.figure(figsize=(22,7))
plt.plot(x,y_test,"g-+",label="Sales_observed")
plt.plot(x,y_test_pred,"r-*",label="Sales_predicted")
plt.grid(b=None)
plt.xlabel("TV budget")
plt.ylabel("Sales")
plt.title("TV budget vs Sales")
plt.legend()
plt.show()
```



Residual (Error) plot If the model has done good predictions, then the datapoints must be near around to horizontal line.

[198]:
```
sns.set_style(style='white')
fig = plt.figure(figsize=(22,7))
residuals = y_test-y_test_pred
zeros = y_test-y_test
plt.scatter(y_test_pred,residuals,color="brown")
plt.grid(b=None)
plt.plot(y_test_pred,zeros,"g")
plt.xlabel("Predicted")
plt.ylabel("Residuals")
plt.title("Residual plot")
plt.show()
```



Storing the outcome in a file

[199]:
```
# Store the predicted value for sales in new column
df.rename(columns={'sales': 'observed_sales'}, inplace=True)
sales_data = df.iloc[:,0].values.reshape(-1, 1)
predicted_values = model.predict(sales_data)
df['predicted_sales'] = predicted_values
df.head()
```

[199]:

	TV	radio	newspaper	observed_sales	predicted_sales
0	230.1	37.8	69.2	22.1	17.878886
1	44.5	39.3	45.1	10.4	9.339840
2	17.2	45.9	69.3	9.3	8.083828
3	151.5	41.3	58.5	18.5	14.262674
4	180.8	10.8	58.4	12.9	15.610702

[200]:
```
# Write the above output input into new csv
# df.to_csv("1.Linear Regression SKLearn - output.csv")
```

0.0.9 Different error calculations to asses the model for the test set

1. Sum of Squared Error (SSE)

$$SSE(m,b) = \sum_{i=1}^{n}(y_i - \hat{y})^2 = \sum_{i=1}^{n}(y_i - (m * x_i + b))^2 \tag{7}$$

[201]:
```
sum = 0
n = len(x_test)
for i in range (0,n):
  diff = y_test[i] - y_test_pred[i]
  squ_diff = diff**2
  sum = sum + squ_diff
```

```python
Test_SSE = np.round(sum,2)
print("Sum of Squared Error (SSE) :",Test_SSE)
```

```
Sum of Squared Error (SSE) : 407.45
```

2. Mean Squared Error (MSE)

$$MSE(m,b) = \frac{\sum_{i=1}^{n}(y_i - \hat{y})^2}{n} = \frac{\sum_{i=1}^{n}(y_i - (m * x_i + b))^2}{n} \tag{8}$$

[202]:
```python
Test_MSE = np.round(Train_SSE/n,2)
print("Mean Squared Error (MSE) :",Test_MSE)
```

```
Mean Squared Error (MSE) : 42.47
```

3. Root Mean Squared Error (RMSE)

$$RMSE(m,b) = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y})^2}{n}} = \sqrt{\frac{\sum_{i=1}^{n}(y_i - (m * x_i + b))^2}{n}} \tag{9}$$

[203]:
```python
Test_RMSE = np.round(np.sqrt(Test_MSE),2)
print("Root Mean Squared Error (RMSE) :",Test_RMSE)
```

```
Root Mean Squared Error (RMSE) : 6.52
```

4. Mean Absolute Error (MAE)

$$MAE(m,b) = \frac{\sum_{i=1}^{n}|(y_i - \hat{y})|}{n} \tag{10}$$

[204]:
```python
sum = 0
n = len(x_test)
for i in range (0,n):
  diff = y_test[i] - y_test_pred[i]
  sum = sum + np.abs(diff)
Test_MAE = np.round(sum/n,2)
print("Mean Absolute Error (MAE) :",Test_MAE)
```

```
Mean Absolute Error (MAE) : 2.51
```

5. Mean Absolute Percentage Error (MAPE)

$$MAPE(m,b) = \frac{100\%}{n}\sum_{i=1}^{n}\left|\frac{(y_i - \hat{y})}{y_i}\right| = \frac{100\%}{n}\sum_{i=1}^{n}\left|\frac{(y_i - (m * x_i + b))}{y_i}\right| \tag{11}$$

[205]:
```
sum = 0
n = len(x_test)
for i in range (0,n):
  diff = (y_test[i] - y_test_pred[i])/y_test[i]
  sum = sum + np.abs(diff)
Test_MAPE = np.round(sum/n*100,2)
print("Mean Absolute Percentage Error (MAPE) :",Test_MAPE)
```

```
Mean Absolute Percentage Error (MAPE) : 26.4
```

0.0.10 Calculating R-Squred value (goodness of model) using SSE

$$R^2 = 1 - \frac{SSE}{SST} = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y})^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{12}$$

[206]:
```
from sklearn.metrics import r2_score
out = r2_score(y_test,y_test_pred)
Test_RS = np.round(out,2)*100
print("R-Squred value (goodness of model) for testing set :",Test_RS,"%")
```

```
R-Squred value (goodness of model) for testing set : 68.0 %
```

0.0.11 Underfitting and overfitting observation

[207]:
```
print("Error \t From training phase       From testing phase ")
print("===================================================")
print("SSE   \t\t",Train_SSE,"\t\t", Test_SSE)
print("MSE   \t\t",Train_MSE,"\t\t\t", Test_MSE)
print("RMSE   \t\t",Train_RMSE,"\t\t\t", Test_RMSE)
print("MAE   \t\t",Train_MAE,"\t\t\t", Test_MAE)
print("RS   \t\t",Train_RS,"\t\t\t", Test_RS)
```

```
Error    From training phase       From testing phase
===================================================
SSE               1698.76                  407.45
MSE               10.62                    42.47
RMSE              3.26                     6.52
MAE               2.58                     2.51
RS                59.0                     68.0
```

0.0.12 Predict the sale for amount of money spent on advertisement via TV

[208]:
```
# If 60$ is spent on TV advertisement what is the expected sales? Rember input␣
↪60$ is neither in train set nor test set
model.predict([[60]])
```

[208]:
```
array([10.05296115])
```

0.0.13 Storing and retrieving the model

Using pickle

[209]:
```
# import necessary library
import pickle
```

[210]:
```
# store the model in disks
with open('E:\\MY LECTURES\\DATA SCIENCE\\3.Programs\\code\\2.
↪REGRESSION\\pickled_linear_model','wb') as f:
    pickle.dump(model,f)
```

[211]:
```
# load the saved model
with open('E:\\MY LECTURES\\DATA SCIENCE\\3.Programs\\code\\2.
↪REGRESSION\\pickled_linear_model','rb') as f:
    model = pickle.load(f)
```

[212]:
```
# If 60$ is spent on TV advertisement what is the expected sales?
model.predict([[60]])
```

[212]:
```
array([10.05296115])
```

Using joblib

[213]:
```
# import necessary library
import joblib
```

[214]:
```
# store the model in disks
joblib.dump(model,'E:\\MY LECTURES\\DATA SCIENCE\\3.Programs\\code\\2.
↪REGRESSION\\pickled_linear_model_joblib')
```

[214]:
```
['E:\\MY LECTURES\\DATA
SCIENCE\\3.Programs\\code\\2.REGRESSION\\pickled_linear_model_joblib']
```

[215]:
```
# load the saved model
model = joblib.load('E:\\MY LECTURES\\DATA SCIENCE\\3.Programs\\code\\2.
↪REGRESSION\\pickled_linear_model_joblib')
```

[216]:
```
# If 60$ is spent on TV advertisement what is the expected sales?
model.predict([[60]])
```

[216]:
```
array([10.05296115])
```