Linear Regression

Problem Statement: Implement a regression model on train data and compute mean error in test set. Comment on R2 value in Conclusion.

Theory

Simple Linear Regression is a type of Regression algorithms that models the relationship between a dependent variable and a single independent variable. The relationship shown by a Simple Linear Regression model is linear or a sloped straight line, hence it is called Simple Linear Regression.

The key point in Simple Linear Regression is that the dependent variable must be a continuous/real value.

To implement the Simple Linear regression model in machine learning using Python, we need to follow the below steps:

Step-1: Data Pre-processing

The first step for creating the Simple Linear Regression model is data pre-processing. We have already done it earlier in this tutorial. But there will be some changes, which are given in the below steps:

First, we will import the three important libraries, which will help us for loading the dataset, plotting the graphs, and creating the Simple Linear Regression model.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as mlt

df = pd.read_csv("tvmarketing.csv")
df
```

```
        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
..     ...    ...
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

[200 rows x 2 columns]
```

For x variable, we have taken -1 value since we want to remove the last column from the dataset. For y variable, we have taken 1 value as a parameter, since we want to extract the second column and indexing starts from the zero.

```
X = df.iloc[:, :-1].values
y = df.iloc[:, 1].values

df.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
```

Next, we will split both variables into the test set and training set. We have 30 observations, so we will take 20 observations for the training set and 10 observations for the test set. We are splitting our dataset so that we can train our model using a training dataset and then test the model using a test dataset.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test= train_test_split(X, y, test_size=
1/3, random_state=0)
```

Now the second step is to fit our model to the training dataset. To do so, we will import the LinearRegression class of the linear_model library from the scikit learn. After importing the class, we are going to create an object of the class named as a regressor.

```
from sklearn.linear_model import LinearRegression
regressor= LinearRegression()
regressor.fit(X_train, y_train)
```

```
LinearRegression()
```

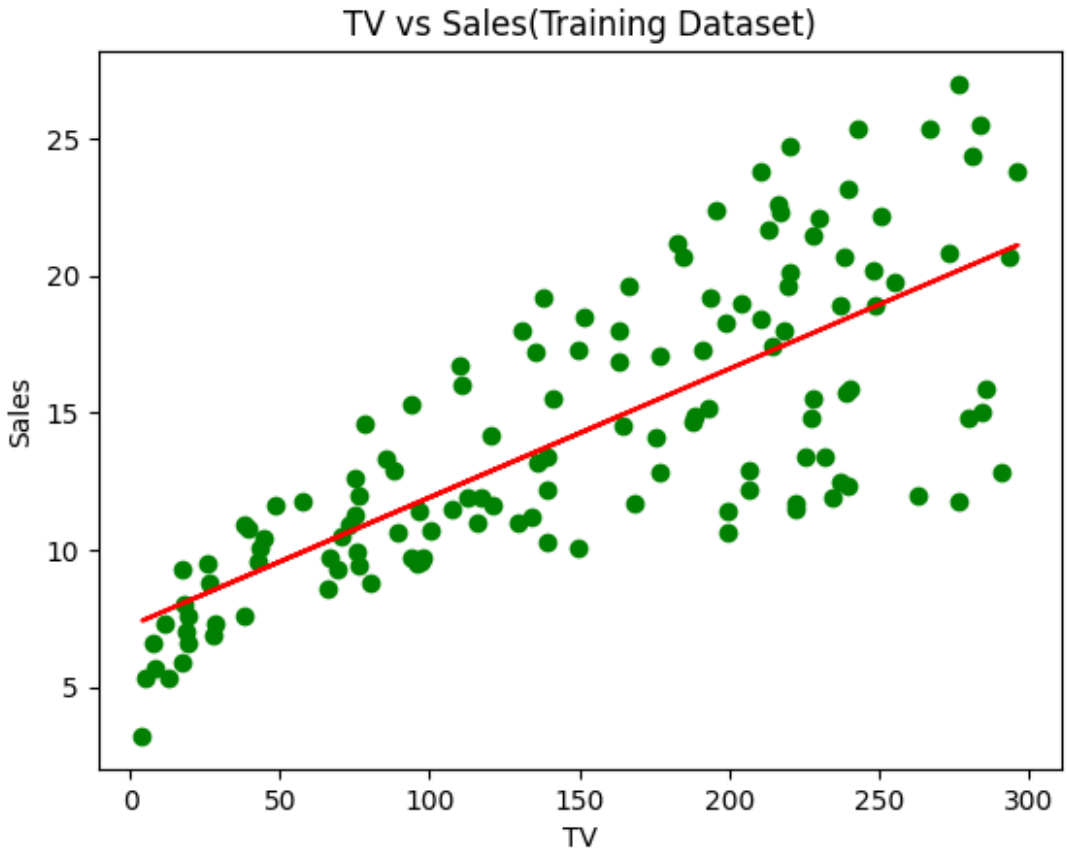
We will create a prediction vector y_pred, and x_pred, which will contain predictions of test dataset, and prediction of training set respectively.

```
#Prediction of Test and Training set result
y_pred= regressor.predict(X_test)
X_pred= regressor.predict(X_train)
```

Finally, we will represent all above things in a graph using show()

```
mlt.scatter(X_train, y_train, color="green")
mlt.plot(X_train, X_pred, color="red")
```

```
mlt.title("TV vs Sales(Training Dataset)")
mlt.xlabel("TV")
mlt.ylabel("Sales")
mlt.show()
```



In the previous step, we have visualized the performance of our model on the training set. Now, we will do the same for the Test set. The complete code will remain the same as the above code, except in this, we will use x_test, and y_test instead of x_train and y_train.

```
#visualizing the Test set results
mlt.scatter(X_test, y_test, color="blue")
mlt.plot(X_train, X_pred, color="red")
mlt.title("TV vs Sales (Test Dataset)")
mlt.xlabel("TV")
mlt.ylabel("Sales")
mlt.show()
```


```
from sklearn.metrics import mean_squared_error, r2_score

# Calculate Mean Squared Error for both training and test sets
mse_train = mean_squared_error(y_train, X_pred)
mse_test = mean_squared_error(y_test, y_pred)

print("Mean Squared Error (Training Set):", mse_train)
print("Mean Squared Error (Test Set):", mse_test)
```

```
Mean Squared Error (Training Set): 11.590065030880293
Mean Squared Error (Test Set): 8.407193534800252
```

```
# Calculate R-squared for both training and test sets
r2_train = r2_score(y_train, X_pred)
r2_test = r2_score(y_test, y_pred)

print("R-squared (Training Set):", r2_train)
print("R-squared (Test Set):", r2_test)
```

```
R-squared (Training Set): 0.575301430004961
R-squared (Test Set): 0.6799643382432321
```

Conclusion

Based on the R-squared (R^2) values obtained from your simple linear regression model, you can draw the following conclusions:

Training Set R^2: 0.575 (57.5%) The model explains approximately 57.5% of the variance in the target variable (salary) using the predictor variable (years of experience) on the training set. This indicates that the model captures a moderate proportion of the underlying patterns in the data. While it's not perfect, it still manages to explain a significant portion of the variability in salaries based on experience.

Test Set R^2: 0.680 (68.0%) The model explains approximately 68.0% of the variance in the target variable on the test set. This suggests that the model's performance is consistent when applied to new, unseen data. The higher R^2 on the test set compared to the training set might indicate that the model generalizes well to new observations.

Overall Assessment: The R^2 values in the range of 0.575 to 0.680 are reasonably good for a simple linear regression model. They suggest that the model captures a substantial part of the variability in salary based on years of experience. However, there is still some unexplained variability that the model does not account for. This could be due to other factors not included in the model, measurement error, or inherent randomness in the data.

Implications: The R^2 values provide insight into how well your model fits the data. While an R^2 of 1 would indicate a perfect fit, such values are rare in real-world data. Given the context, an R^2 of around 0.68 indicates that your model's predictions are quite aligned with the actual data trends.