# LAB2_SIMPLE_LINEAR_REGRESSION

February 9, 2024

## 1 IMPLEMENTATION OF SIMPLE LINEAR REGRESSION

### 1.1 SIMPLE LINEAR REGRESSION

Statistical method that we can use to find a relationship between two variables and make predictions. The two variables used are typically denoted as y and x. The independent variable, or the variable used to predict the dependent variable is denoted as x. The dependent variable, or the outcome/output, is denoted as y.

#### PROBLEM STATEMENT

Analysing the relationship between 'TV advertising' and 'sales' using a simple linear regression model.

[2]:
```
#Setting up the Kaggle API credentials
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
```

[3]:
```
!kaggle datasets download -d ashydv/advertising-dataset
```

```
Warning: Your Kaggle API key is readable by other users on this system! To fix
this, you can run 'chmod 600 /root/.kaggle/kaggle.json'
Downloading advertising-dataset.zip to /content
  0% 0.00/1.83k [00:00<?, ?B/s]
100% 1.83k/1.83k [00:00<00:00, 5.19MB/s]
```

[4]:
```
# Unzipping the zipped dataset file
import zipfile
zip_ref = zipfile.ZipFile('/content/advertising-dataset.zip')
zip_ref.extractall('/content')
zip_ref.close()
```

[5]:
```
# Importing the Libraries
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

[6]:
```
# Loading the dataset
data = pd.read_csv('advertising.csv')
data.head()
```

[6]:

| | 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 | 12.0 |
| 3 | 151.5 | 41.3 | 58.5 | 16.5 |
| 4 | 180.8 | 10.8 | 58.4 | 17.9 |

### Understanding the Data

[7]:
```
data.shape
```

[7]:
```
(200, 4)
```

### Data Insights

Our data has 200 rows and 4 columns.

[8]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   TV         200 non-null    float64
 1   Radio      200 non-null    float64
 2   Newspaper  200 non-null    float64
 3   Sales      200 non-null    float64
dtypes: float64(4)
memory usage: 6.4 KB
```

### Data Insights

- All columns are not having any Null Entries
- All the columns are of numerical type.

[9]:
```
data.describe()
```

[9]:

| | TV | Radio | Newspaper | Sales |
|---|---|---|---|---|
| count | 200.000000 | 200.000000 | 200.000000 | 200.000000 |
| mean | 147.042500 | 23.264000 | 30.554000 | 15.130500 |
| std | 85.854236 | 14.846809 | 21.778621 | 5.283892 |
| min | 0.700000 | 0.000000 | 0.300000 | 1.600000 |
| 25% | 74.375000 | 9.975000 | 12.750000 | 11.000000 |
| 50% | 149.750000 | 22.900000 | 25.750000 | 16.000000 |
| 75% | 218.825000 | 36.525000 | 45.100000 | 19.050000 |
| max | 296.400000 | 49.600000 | 114.000000 | 27.000000 |

**Data Insights** * Mean values * Standard Deviation * Minimum Values * Maximum Values

## 1.2 Data Cleaning

### 1.2.1 Checking for Null Values

[10]:
```
data.isnull().sum()
```

[10]:
```
TV           0
Radio        0
Newspaper    0
Sales        0
dtype: int64
```

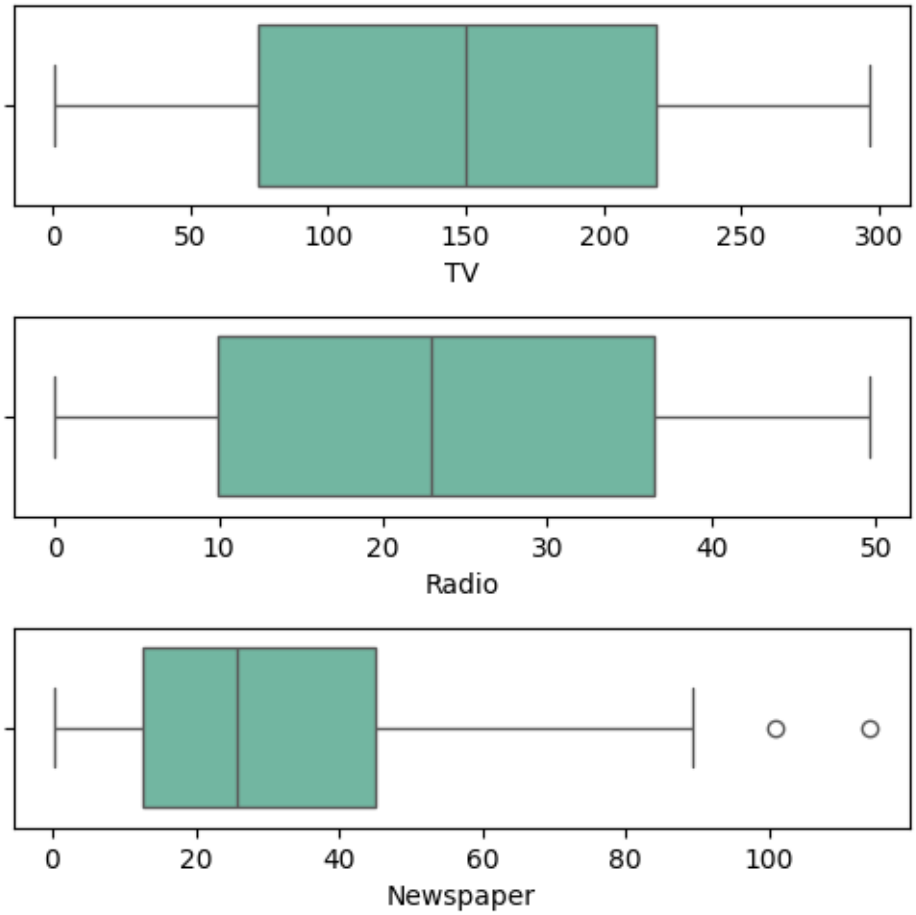
Data Insights * As stated earlier there are no null values in our dataset.

### 1.2.2 Outlier Analysis

[11]:
```
fig, axes = plt.subplots(3,figsize =(5,5))
sns.set(font_scale=.8)
sns.boxplot(data= data, x = 'TV',orient= 'v', ax= axes[0], palette='Set2')

sns.boxplot(data= data, x = 'Radio', orient= 'v', ax= axes[1], palette='Set2')

sns.boxplot(data= data, x = 'Newspaper', orient= 'v', ax=␣
 ↪axes[2],palette='Set2')

plt.tight_layout()
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:1608: UserWarning:
Vertical orientation ignored with only `x` specified.
  warnings.warn(single_var_warning.format("Vertical", "x"))
<ipython-input-11-b553dbdfc0de>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.boxplot(data= data, x = 'TV',orient= 'v', ax= axes[0], palette='Set2')
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:1608: UserWarning:
Vertical orientation ignored with only `x` specified.
  warnings.warn(single_var_warning.format("Vertical", "x"))
<ipython-input-11-b553dbdfc0de>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.boxplot(data= data, x = 'Radio', orient= 'v', ax= axes[1], palette='Set2')
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:1608: UserWarning:
```

```
Vertical orientation ignored with only `x` specified.
  warnings.warn(single_var_warning.format("Vertical", "x"))
<ipython-input-11-b553dbdfc0de>:7: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.boxplot(data= data, x = 'Newspaper', orient= 'v', ax=
axes[2],palette='Set2')
```

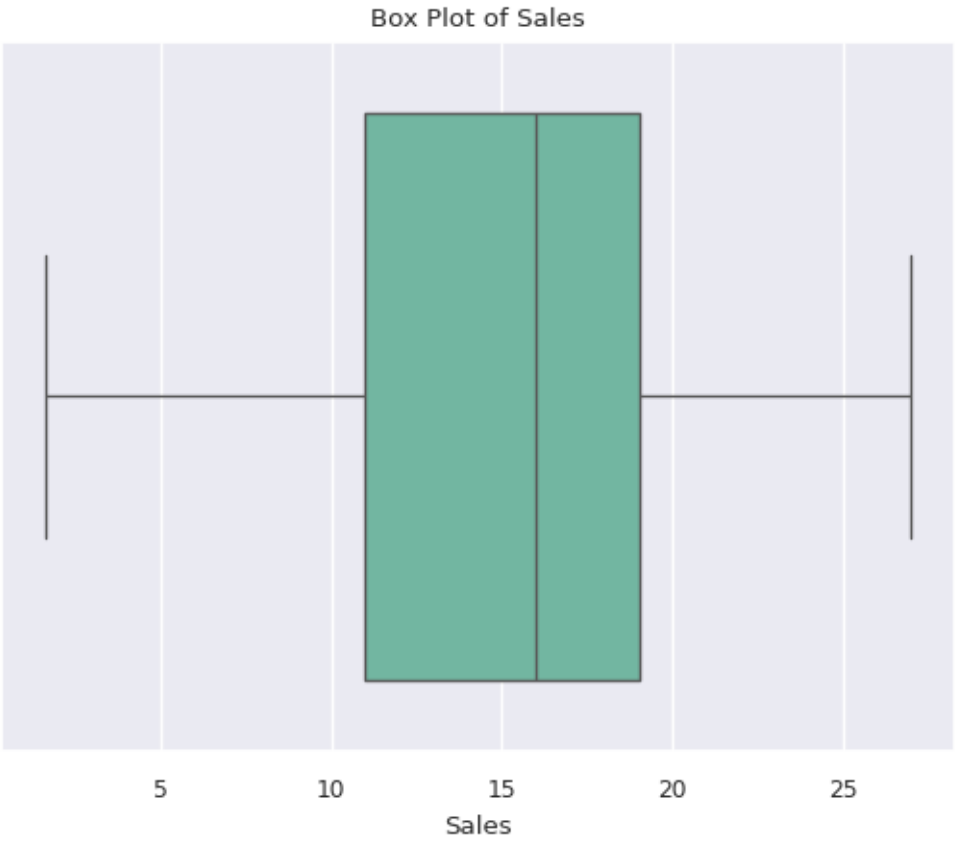



Data Insights: * TV Budget is distributed largely. * Radio Budget is also largely distributed. * Newpaper budget is not distributed much compared to TV and Radio.

[12]:
```
sns.boxplot(data= data, x = 'Sales',orient= 'v', palette='Set2', legend=False)
plt.title('Box Plot of Sales')
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:1608: UserWarning:
```

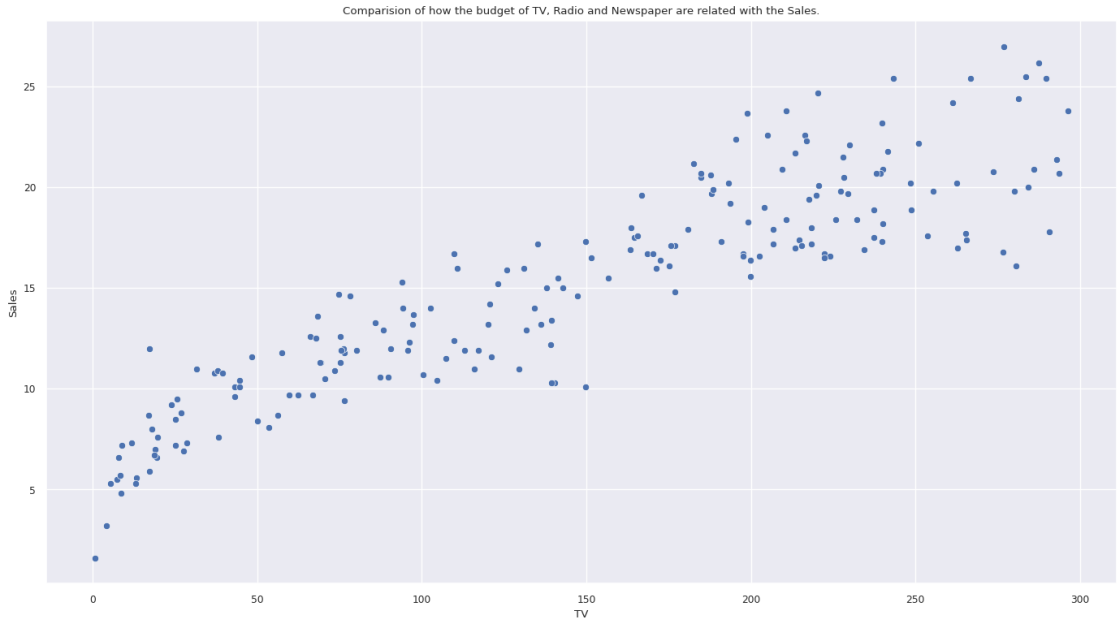
```
Vertical orientation ignored with only `x` specified.
  warnings.warn(single_var_warning.format("Vertical", "x"))
<ipython-input-12-229d3370e978>:1: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.boxplot(data= data, x = 'Sales',orient= 'v', palette='Set2', legend=False)
```




### 1.2.3 UNIVARIATE ANALYSIS USING HISTOGRAM

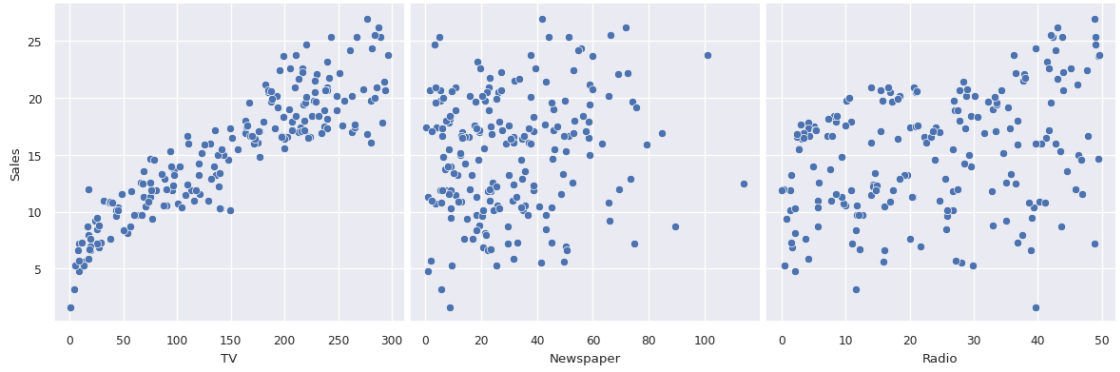
[13]:
```
plt.figure(figsize = (17,9))
plt.title("Comparision of how the budget of TV, Radio and Newspaper are related␣
  ↪with the Sales.")
sns.scatterplot(data=data, x='TV', y='Sales')
```

[13]:
```
<Axes: title={'center': 'Comparision of how the budget of TV, Radio and
Newspaper are related with the Sales.'}, xlabel='TV', ylabel='Sales'>
```




**Pairplot**

[14]:
```
sns.pairplot(data, x_vars=['TV', 'Newspaper', 'Radio'], y_vars='Sales',␣
↪height=4, aspect=1, kind='scatter')
plt.show()
```

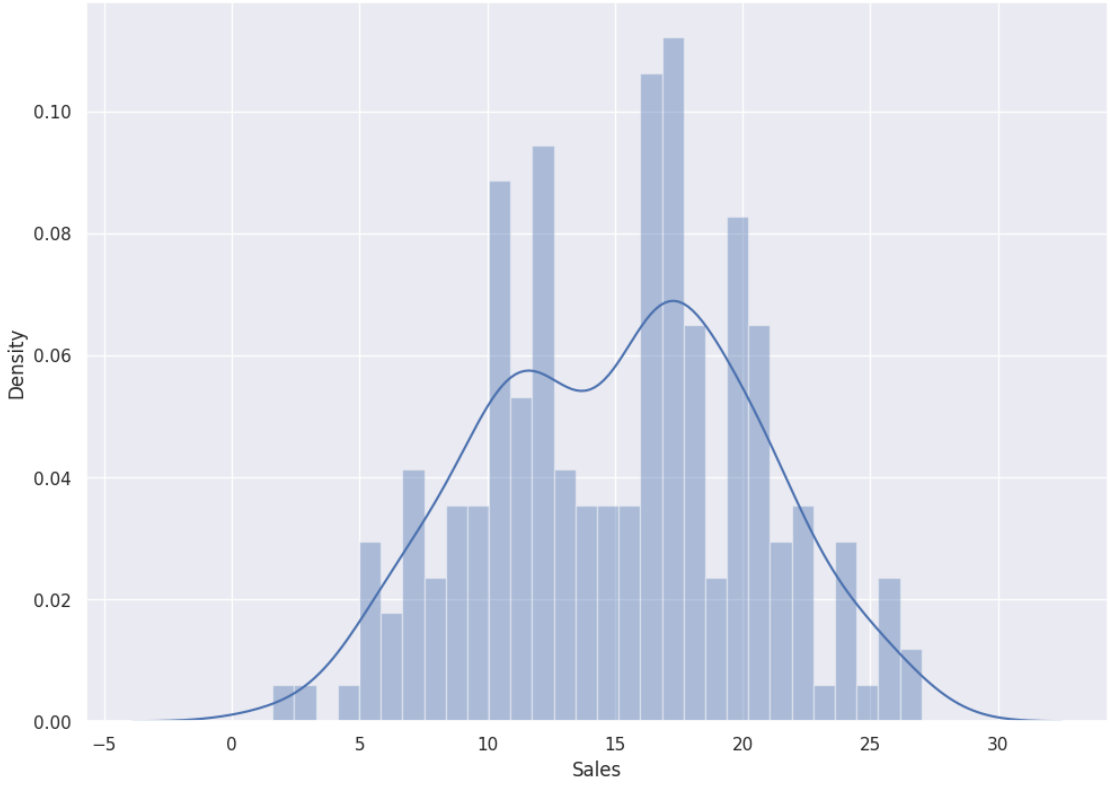



### 1.2.4 UNIVARIATE ANALYSIS USING HISTOGRAM

[61]:
```
sns.set(rc={'figure.figsize':(11.7,8.27)})
sns.distplot(data['Sales'], bins=30)
plt.show()
```

```
<ipython-input-61-92dce7ade1f5>:2: UserWarning:
```

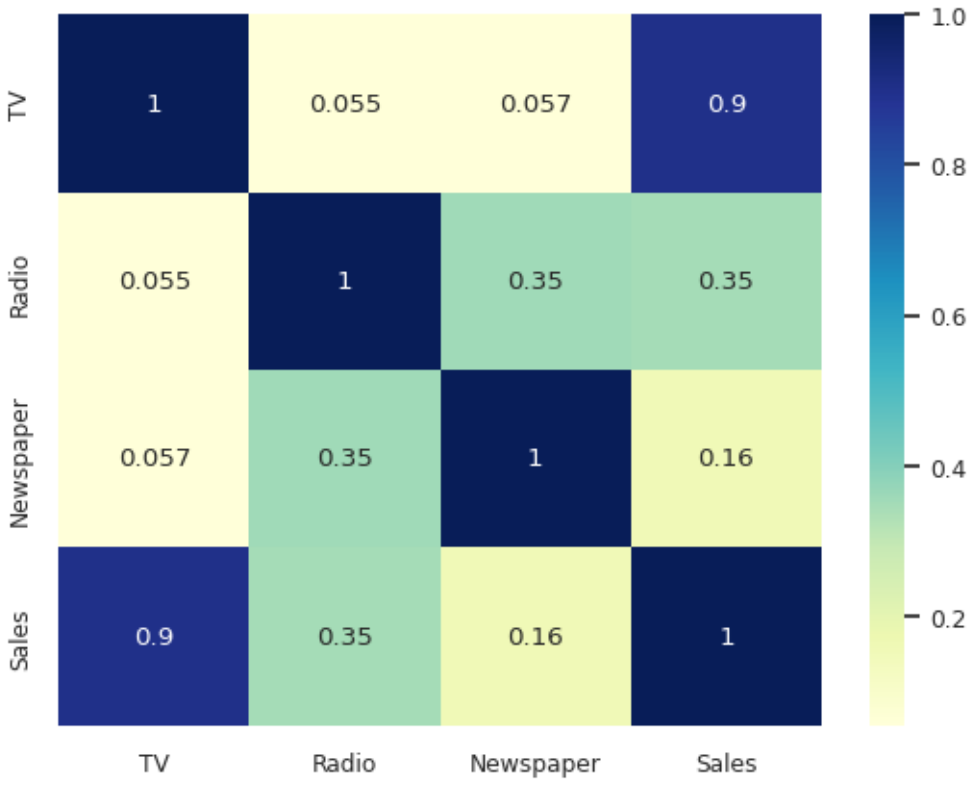
```
`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(data['Sales'], bins=30)
```




**Data Insights** * Here we can see the distribution of the sales column. The values in the sales column s normally distributed.

CHECKING CORRELATION

[15]:
```
sns.heatmap(data.corr(), cmap="YlGnBu", annot = True)
plt.show()
```




Data Insights: * TV can be considered as a feature variable as it is strongly correlated to the Sales Column compared to Radio and Newspaper.

## 1.3 MODEL BUILDING

### PERFORMING SIMPLE LINEAR REGRESSION

We have,

`y = m*x + c`

- x is the independent variable(TV)
- y is the dependent variable(Sales) Therefore we can write,

  Sales = m*TV + c
- m is the model coefficent.

### 1.3.1 Preparing X and Y

- The scikit-learn library expects X (feature variable) and y (response variable) to be NumPy arrays.
- However, X can be a dataframe as Pandas is built over NumPy.

[16]:
```
X = data['TV']
X.head()
```

[16]:
```
0    230.1
1     44.5
2     17.2
3    151.5
4    180.8
Name: TV, dtype: float64
```

[18]:
```
y = data['Sales']
y.head()
```

[18]:
```
0    22.1
1    10.4
2    12.0
3    16.5
4    17.9
Name: Sales, dtype: float64
```

### 1.3.2 Splitting Data into Training and Testing Sets

[19]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.75 ,␣
↪random_state=0000)
```

[20]:
```
print(type(X_train))
print(type(X_test))
print(type(y_train))
print(type(y_test))
```

```
<class 'pandas.core.series.Series'>
<class 'pandas.core.series.Series'>
<class 'pandas.core.series.Series'>
<class 'pandas.core.series.Series'>
```

Data Insights * The generated training and testing data is Series.

[21]:
```
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(150,)
(150,)
(50,)
(50,)
```

Data Insights: * The training data for TV and Sales has 150 observations. * The testing data for TV and sales has 50 observations.

[22]:
```
X_train
```

[22]:
```
71     109.8
124    229.5
184    253.8
97     184.9
149     44.7
       …
67     139.3
192     17.2
117     76.4
47     239.9
172     19.6
Name: TV, Length: 150, dtype: float64
```

[23]:
```
import numpy as np
X_train = X_train[:, np.newaxis]
X_test = X_test[:, np.newaxis]
```

```
<ipython-input-23-00a39a536ec8>:3: FutureWarning: Support for multi-dimensional
indexing (e.g. `obj[:, None]`) is deprecated and will be removed in a future
version.  Convert to a numpy array before indexing instead.
  X_train = X_train[:, np.newaxis]
<ipython-input-23-00a39a536ec8>:4: FutureWarning: Support for multi-dimensional
indexing (e.g. `obj[:, None]`) is deprecated and will be removed in a future
version.  Convert to a numpy array before indexing instead.
  X_test = X_test[:, np.newaxis]
```

Here we transform input data in the form of a 2D array where each row represents a sample and each column represents a feature. Because as we are dealing only with one feature, we make the input(Independent varialbe) into a 2D array which is a general convention in scikit-learn library.

[24]:
```
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(150, 1)
(150,)
(50, 1)
(50,)
```

[50]:
```
print(type(X_train))
print(type(X_test))
print(type(y_train))
print(type(y_test))
```

```
<class 'numpy.ndarray'>
<class 'numpy.ndarray'>
```

```
<class 'pandas.core.series.Series'>
<class 'pandas.core.series.Series'>
```

Data Insights: * Here the X_train has become an Array of 150 rows and 1 fearure column whereas X_test has 50 rows with 1 feature variable. * y_train and y_test is one dimentional array of 50 elements which represent the target variable Salesassociated with the X_traina and X_test.

### 1.3.3 Performing Linear Regression

[27]:
```
# importing LinearRegression from sklearn
from sklearn.linear_model import LinearRegression

# Representing LinearRegression as lr(Creating LinearRegression Object)
lr = LinearRegression()

# Fit the model using lr.fit()
lr.fit(X_train, y_train)
```

[27]:
```
LinearRegression()
```

[42]:
```
lr.score(X_train, y_train)
```

[42]:
```
0.8101117868824258
```

Calculating the Coeffiecents of the Simple Linear Regression Model

[28]:
```
print("The Intercept is", lr.intercept_)
print("The Slope of the regression equation is ",lr.coef_)
```

```
The Intercept is 7.128878060785404
The Slope of the regression equation is  [0.05452888]
```

Therefore the simple linear regression equation can be represented as,

```
y = 7.13 * (0.055*TV)
```

Predictions

[29]:
```
# Making predictions on the testing set which returns an ndarray
y_pred = lr.predict(X_test)
```

[37]:
```
y_pred.shape
```

[37]:
```
(50,)
```

[38]:
```
y_test.shape
```

[38]:
```
(50,)
```

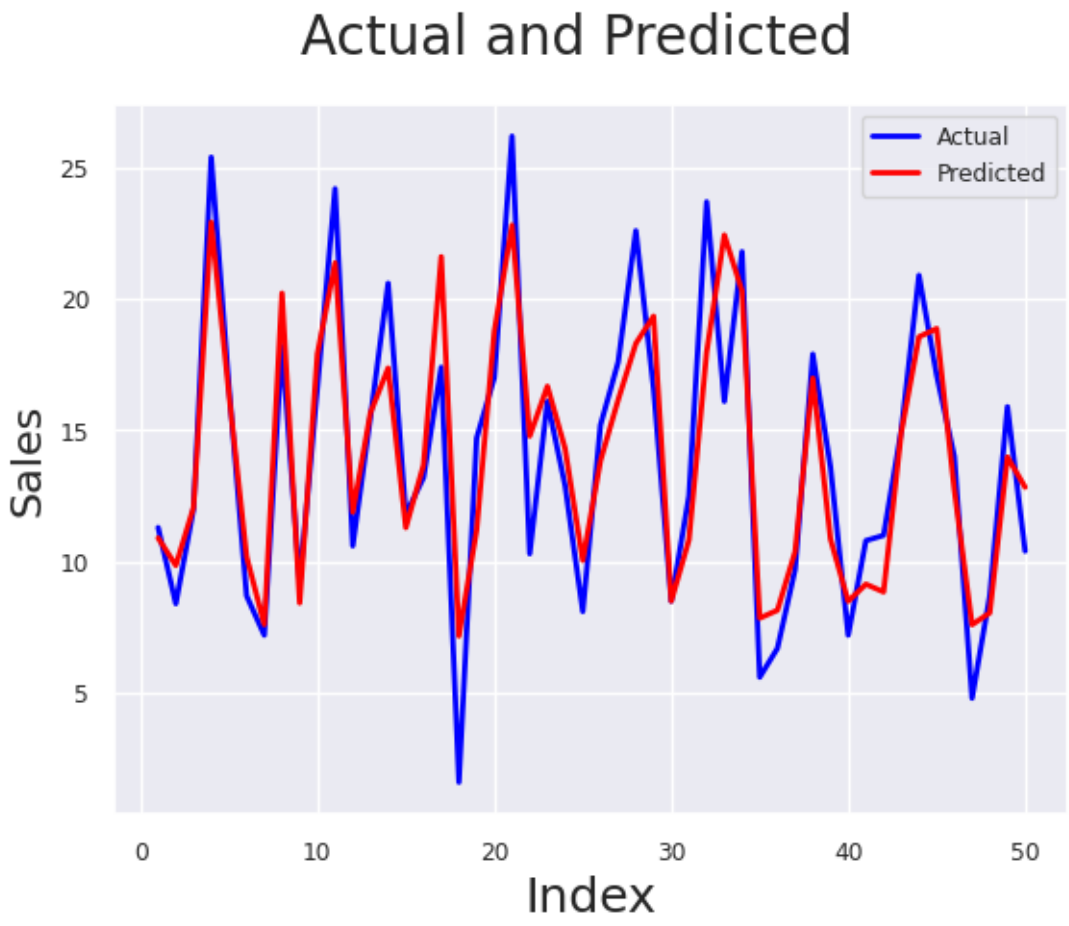
**COMPUTING RMSE and R2 SCORE**

**RMSE** * Root Mean Square Error (RMSE) is a standard way to measure the error of a model in predicting quantitative data.

- The measure of how well a regression line fits the data points.
- It is the the standard deviation of the errors which occur when a prediction is made on a dataset.

[41]:
```
# Actual vs Predicted
import matplotlib.pyplot as plt
c = [i for i in range(1,51,1)]          # generating index
fig = plt.figure()
plt.plot(c,y_test, color="blue", linewidth=2, linestyle="-", label="Actual")
plt.plot(c,y_pred, color="red",  linewidth=2, linestyle="-", label="Predicted")
fig.suptitle('Actual and Predicted', fontsize=20)               # Plot heading
plt.xlabel('Index', fontsize=18)                                # X-label
plt.ylabel('Sales', fontsize=16)
plt.legend()
```

[41]: <matplotlib.legend.Legend at 0x7db0f8ee9090>



**Data Insights**:

From the plot we can infer that there is not much deviation in the actual and predicted values.

Plotting the Error

[43]:
```
# Error terms
c = [i for i in range(1,51,1)]
fig = plt.figure()
plt.plot(c,y_test-y_pred, color="blue", linewidth=2, linestyle="-")
fig.suptitle('Error Terms', fontsize=20)              # Plot heading
plt.xlabel('Index', fontsize=18)                      # X-label
plt.ylabel('ytest-ypred', fontsize=16)
```

[43]:
```
Text(0, 0.5, 'ytest-ypred')
```


Data Insights: This plot helps us to infer that difference between actual and predicted values. In the plot most of the error values are in the range of -2 to 2 and a few are too low and too high. So

the model should work accurate as expected.

[56]:
```
from sklearn.metrics import mean_squared_error, r2_score
mse = mean_squared_error(y_test, y_pred)
print('Mean_Squared_Error :' ,mse)
```

```
Mean_Squared_Error : 6.010070421507887
```

[47]:
```
rmse = np.sqrt(mse)
print('Root_Mean_Squared_Error :' ,rmse)
```

```
Root_Mean_Squared_Error : 2.451544497150294
```

The RMSE value is obtained to be 2.45 which suggests that, on average, our model's predictions are off by approximately 2.45 units of the target variable. Hence we can say that the model performance is performing well in predicting the Sales.

[55]:
```
# Assuming you have calculated the RMSE value and determined the range of␣
↪'Sales'
RMSE = 2.451544497150294  # Example RMSE value
sales_range = max(y_test) - min(y_test)  # Example range of 'Sales' variable

# Calculate the ratio of RMSE to the range of 'Sales'
RMSE_to_sales_range_ratio = RMSE / sales_range

# Print the ratio for comparison
print("RMSE-to-Sales-Range Ratio:", RMSE_to_sales_range_ratio)
```

```
RMSE-to-Sales-Range Ratio: 0.09965628037196318
```

Since the ratio is almost 0.1, the ratio can be considered as small and the model has good predictive accuracy.

## 1.4 Inference from the R2 Score

[57]:
```
from sklearn.metrics import r2_score
r_squared = r2_score(y_test, y_pred)
print('R2 Score :',r_squared)
```

```
r_square_value : 0.8053611644334993
```

### 1.4.1 R2 Score

- Measure of how well the independent variables explain the variability of the dependent variable.
- A higher R2 score indicates a better fit of the model to the data.
- It ranges from 0 to 1.
- 0 indicates that the model does not explain any of the variability in the target variable.
- 1 indicates that the model perfectly explains all the variability in the target variable.

- In our case, an R2 score of 0.8053611644334993 suggests that approximately 80.54% of the variance in 'Sales' is explained by the independent variables in your model.

### 1.5 Plotting the Regression Line

[54]:
```
plt.plot(X_test,y_pred, color='red')
plt.scatter(data['TV'],data['Sales'])
plt.title('Tv vs Sales')
plt.xlabel("TV")
plt.ylabel('Sales ')
plt.show()
```


## CONCLUSION

In this lab, we conducted a simple linear regression analysis to model the relationship between the predictor variable (TV advertising expenditure) and the target variable (sales). Our analysis resulted in the development of a linear regression model with a Root Mean Squared Error (RMSE) of 2.45 and an R-squared (R2) score of 0.81. Based on these results, we have successfully fitted the model using the regression line, which provides valuable insights into the relationship between TV advertising expenditure and sales.

[ ]: