Simple Linear Regression

In this example we will consider sales based on 'TV' marketing budget.

In this notebook, we'll build a linear regression model to predict 'Sales' using 'TV' as the predictor variable.

Understanding the Data

Let's start with the following steps:

1. Importing data using the pandas library
2. Understanding the structure of the data

```
import pandas as pd
```

```
# Reading csv file from github repo
url  = "https://raw.githubusercontent.com/devzohaib/Simple-Linear-Regression/master/tvmarketing.csv"
advertising = pd.read_csv(url)
```

```
# Display the first 5 rows
advertising.head()
```

	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

```
# Display the last 5 rows
advertising.tail()
```

	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

```
# Let's check the columns
advertising.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
```

```
# Check the shape of the DataFrame (rows, columns)
advertising.shape
```

```
(200, 2)
```

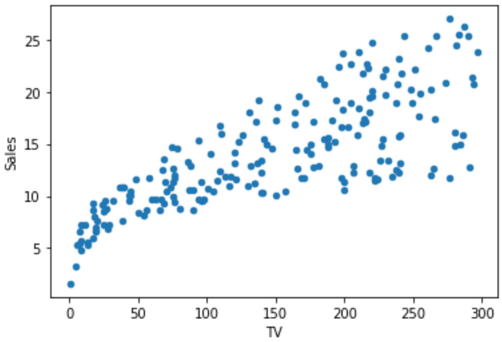
```
# Let's look at some statistical information about the dataframe.
advertising.describe()
```

	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

Visualising Data Using Plot

```
# Visualise the relationship between the features and the response using scatterplots
df = pd.DataFrame(advertising)
df.plot(x='TV',y='Sales',kind='scatter')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6d9080e2b0>
```



Perfroming Simple Linear Regression

Equation of linear regression
$y = c + m_1x_1 + m_2x_2 + ... + m_nx_n$

- y is the response
- c is the intercept
- m_1 is the coefficient for the first feature
- m_n is the coefficient for the nth feature

In our case:

$y = c + m_1 \times TV$

The m values are called the model **coefficients** or **model parameters**.

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.

```
# Putting feature variable to X
X = advertising['TV']

# Print the first 5 rows
X.head()
```

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

```
# Putting response variable to y
y = advertising['Sales']

# Print the first 5 rows
y.head()
```

```
0    22.1
1    10.4
2     9.3
3    18.5
4    12.9
Name: Sales, dtype: float64
```

Splitting Data into Training and Testing Sets

```
#random_state is the seed used by the random number generator, it can be any integer.

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=0)
```

```
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'>
```

```
train_test_split
#Press Tab+Shift to read the documentation
```

```
<function sklearn.model_selection._split.train_test_split(*arrays, test_size=None, train_size=None, random_state=None,
shuffle=True, stratify=None)>
```

```
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(140,)
(140,)
(60,)
(60,)
```

```
#It is a general convention in scikit-learn that observations are rows, while features are columns.
#This is needed only when you are using a single feature; in this case, 'TV'.

import numpy as np
#Simply put, numpy.newaxis is used to increase the dimension of the existing array by one more dimension,
X_train = X_train[:, np.newaxis]
X_test = X_test[:, np.newaxis]
```

```
<ipython-input-10-6cf08b050023>:6: FutureWarning: Support for multi-dimensional indexing (e.g. `obj[:, None]`) is deprecated and wi
  X_train = X_train[:, np.newaxis]
<ipython-input-10-6cf08b050023>:7: FutureWarning: Support for multi-dimensional indexing (e.g. `obj[:, None]`) is deprecated and wi
  X_test = X_test[:, np.newaxis]
```

```
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(140, 1)
(140,)
(60, 1)
(60,)
```

Performing Linear Regression

```
# import LinearRegression from sklearn
from sklearn.linear_model import LinearRegression

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

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

```
LinearRegression()
```

Coefficients Calculation

```
# Print the intercept and coefficients
print(lr.intercept_)
print(lr.coef_)
```

```
7.310810165411681
[0.04581434]
```

$y = 7.310 + 0.0464 \times TV$

Now, let's use this equation to predict our sales.

Predictions

```
# Making predictions on the testing set
y_pred = lr.predict(X_test)
```

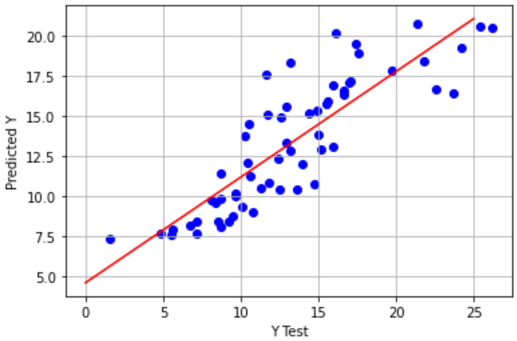
Linear Regression Plot

```
import matplotlib.pyplot as plt
lr.fit(y_test[:, np.newaxis], y_pred)

xfit = np.linspace(0, 25, 10)
yfit = lr.predict(xfit[:, np.newaxis])

plt.scatter(y_test, y_pred,c='blue')
plt.plot(xfit, yfit,c='red');
plt.xlabel('Y Test')
plt.ylabel('Predicted Y')
plt.grid()
```

```
<ipython-input-24-31109c8f4c95>:2: FutureWarning: Support for multi-dimensional indexing (e.g. `obj[:, None]`) is
  lr.fit(y_test[:, np.newaxis], y_pred)
```



Computing RMSE and R^2 Values

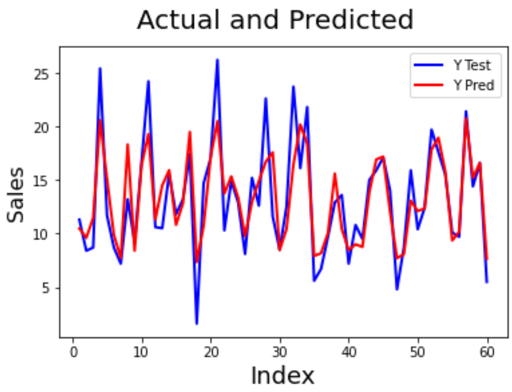
RMSE is the standard deviation of the errors which occur when a prediction is made on a dataset. This is the same as MSE (Mean Squared Error) but the root of the value is considered while determining the accuracy of the model

```
y_test.shape # cheek the shape to generate the index for plot
```

```
(60,)
```

```
# 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, linestyle="-",label="Y Test")
plt.plot(c,y_pred, color="red",  linewidth=2, linestyle="-",label="Y Pred")
fig.suptitle('Actual and Predicted', fontsize=20)              # Plot heading
plt.xlabel('Index', fontsize=18)                               # X-label
plt.ylabel('Sales', fontsize=16)                       # Y-label
plt.legend()
```

<matplotlib.legend.Legend at 0x7f6d883aca60>




```
from sklearn.metrics import mean_squared_error, r2_score
mse = mean_squared_error(y_test, y_pred)
```

```
r_squared = r2_score(y_test, y_pred)
```

```
print('Mean_Squared_Error :' ,mse)
print('r_square_value :',r_squared)
```

```
Mean_Squared_Error : 7.497479593464674
r_square_value : 0.725606346597073
```

```
# this mse =7.5 means that this model is not able to match the 7.5 percent of the values
# r2 means that your model is 72% is accurate on test data .
```

✓ 0s completed at 12:08 PM