Linear regresssion

In [58]:
```
import pandas as pd
import numpy as np
from sklearn import linear_model
from scipy import stats
import seaborn as sns
import matplotlib.pyplot as plt
```

Scikit-learn is probably the most useful library for machine learning in Python. The sklearn library contains a lot of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction. Please note that sklearn is used to build machine learning models.

linear_model is a module in sklearn library:The linear_model module implements generalized linear models. It includes Ridge regression, Bayesian Regression, Lasso regresssion etc.

SciPy is an open-source Python library which is used to solve scientific and mathematical problems. It is built on the NumPy extension and allows the user to manipulate and visualize data with a wide range of high-level commands. As mentioned earlier, SciPy builds on NumPy and therefore if you import SciPy, there is no need to import NumPy.

Python Seaborn library is used to ease the challenging task of data visualization and it’s based on Matplotlib. Seaborn allows the creation of statistical graphics through the following functionalities:

Seaborn supports multi-plot grids that in turn ease building complex visualizations

Availability of different color palettes to reveal various kinds of patterns

Estimates and plots linear regression automatically

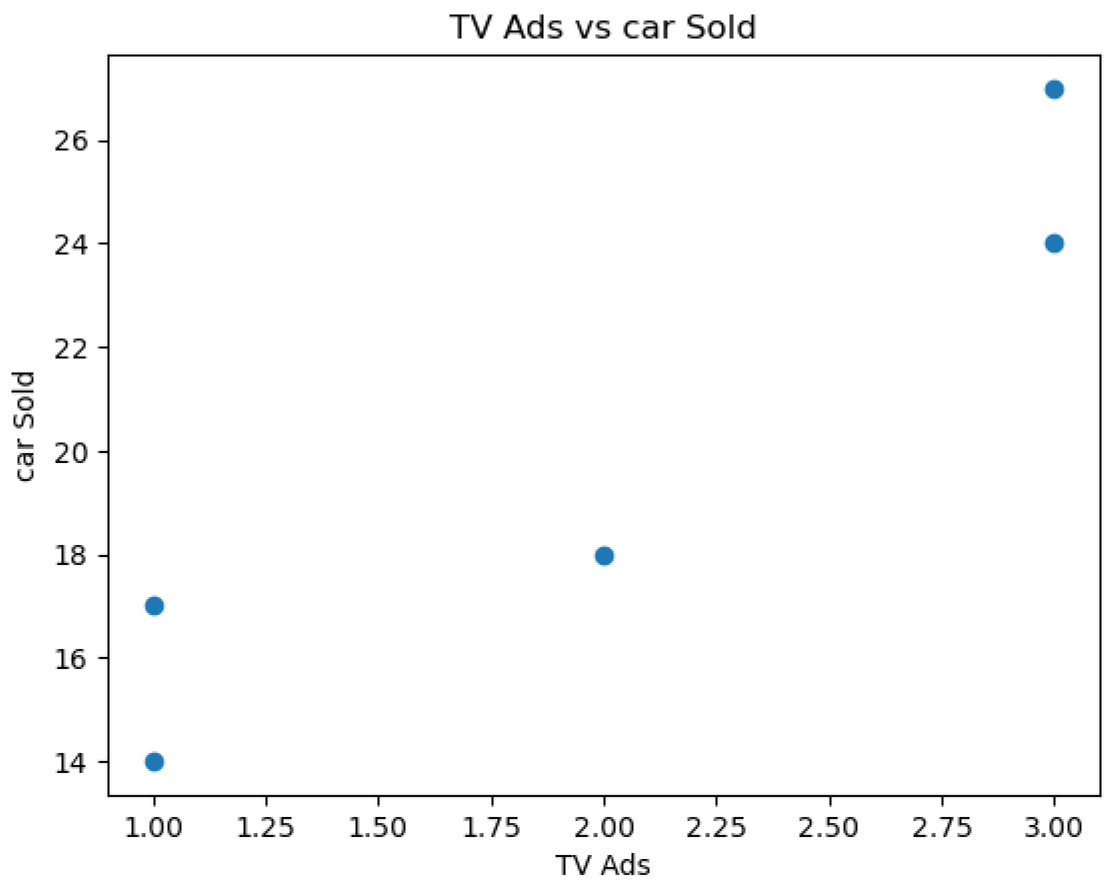
Matplotlib can be personalized but it’s difficult to figure out what settings are required to make plots more attractive. On the other hand, Seaborn comes with numerous customized themes and high-level interfaces to solve this issue.

In [59]:
```
# read the data into python notebook: the data is in excel format
data = pd.read_excel \
(r"C:\Users\lenovo\Downloads\linear_reg.xlsx")
data
```

Out[59]:

	TV Ads	car Sold
0	1	14
1	3	24
2	2	18
3	1	17
4	3	27

In [60]:
```
# plot a scatter plot between car sold and TV adds
import matplotlib.pyplot as plt
plt.scatter(data['TV Ads'], data['car Sold'])
plt.title('TV Ads vs car Sold')
plt.xlabel('TV Ads')
plt.ylabel('car Sold')
plt.show()
```



```
In [61]: x = data['TV Ads'].values
         y = data['car Sold'].values
```

```
In [62]: x.ndim
```

```
Out[62]: 1
```

reshape(-1, 1) results in an array with a single column and multiple rows (a column vector)

```
In [63]: x = data['TV Ads'].values.reshape(-1,1)
         y = data['car Sold'].values.reshape(-1,1)
```

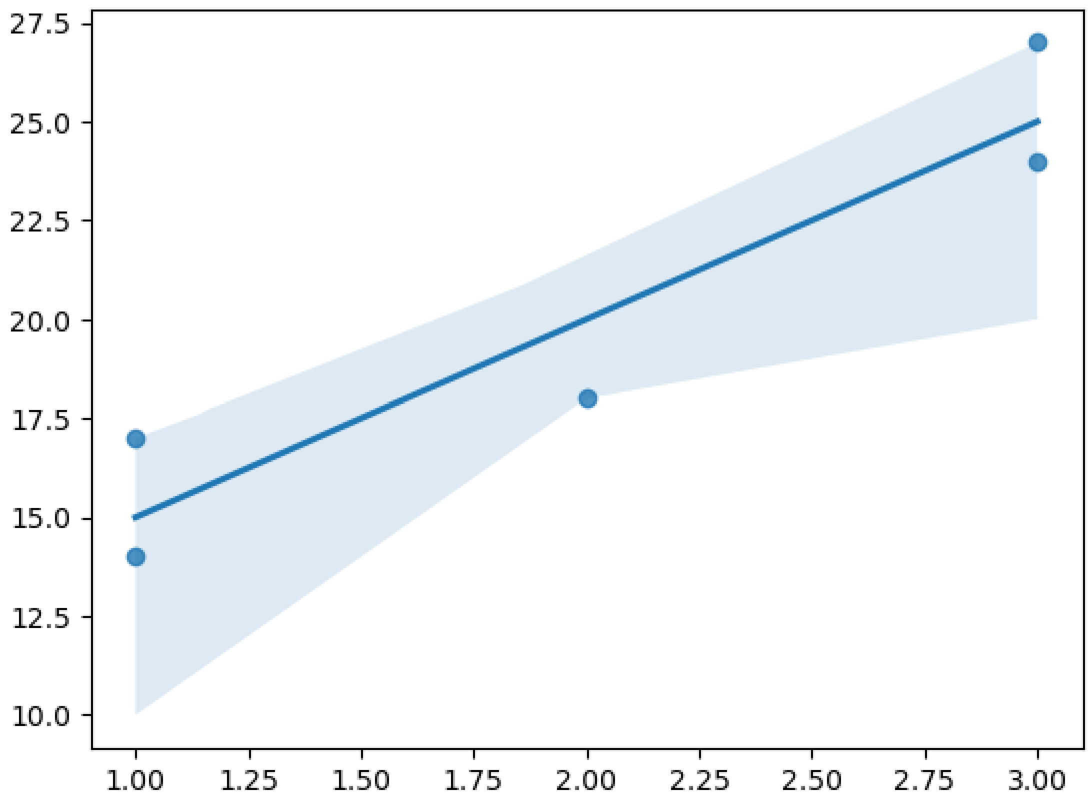
```
In [64]: x.ndim
```

```
Out[64]: 2
```

```
In [65]: x
```

```
Out[65]: array([[1],
                [3],
                [2],
                [1],
                [3]], dtype=int64)
```

```
In [66]: # regplot: Plot data and a linear regression model fit.
         sns.regplot(x=x, y=y)
         # show the plot
         plt.show()
```



model_selection module has a function train_test_split():Split arrays or matrices into random train and test subsets

```
In [67]: 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)
```

Random state ensures that the splits that you generate are reproducible. Scikit-learn uses random permutations to generate the splits. The random state that you provide is used as a seed to the random number generator. This ensures that the random numbers are generated in the same order

```
In [68]: x_train

Out[68]: array([[1],
                [3],
                [1],
                [3]], dtype=int64)

In [69]: y_train

Out[69]: array([[14],
                [24],
                [17],
                [27]], dtype=int64)

In [70]:  x_test

Out[70]: array([[2]], dtype=int64)

In [71]: y_test

Out[71]: array([[18]], dtype=int64)

In [72]: len(x_train)

         4
```

```
Out[72]:
```

```
In [73]: len(x_test)
```

```
Out[73]: 1
```

```
In [74]: # LinearRegresssion is a class that helps to implement  Ordinary least squares Linear Re
         from sklearn.linear_model import LinearRegression
         regressor = LinearRegression()
         # Fit the model to the data
         regressor.fit(x_train, y_train)
```

```
Out[74]: □ LinearRegression

         LinearRegression()
```

With Scikit-Learn it is extremely straight forward to implement linear regression models, as all you really need to do is import the LinearRegression class, instantiate it, and call the fit() method along with our training data. This is about as simple as it gets when using a machine learning library to train on your data.

```
In [75]: # find the intercept of the regresssion line
         regressor.intercept_
```

```
Out[75]: array([10.5])
```

```
In [76]: # find the slope of the regresssion line
         regressor.coef_
```

```
Out[76]: array([[5.]])
```

Making Predictions Now that we have trained our algorithm, it's time to make some predictions

```
In [77]: #Predict using the linear model
         y_predict = regressor.predict(x_test)
```

```
In [78]: y_predict
```

```
Out[78]: array([[20.5]])
```

```
In [79]: y_test
```

```
Out[79]: array([[18]], dtype=int64)
```

Evaluating the Algorithm

1. Mean Absolute Error (MAE) is the mean of the absolute value of the errors. It is calculated as:

$$\frac{1}{n}\sum_{i=1}^{n}|Actual - Predicted|$$

2. Mean Squared Error (MSE) is the mean of the squared errors and is calculated as:

$$\frac{1}{n}\sum_{i=1}^{n}|Actual - Predicted|^2$$

3. Root Mean Squared Error (RMSE) is the square root of the mean of the squared errors:

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}|Actual - Predicted|^2}$$

The final step is to evaluate the performance of algorithm. This step is particularly important to compare how well different algorithms perform on a particular dataset. For regression algorithms, three evaluation metrics are commonly used:

In [80]:
```
# metrics module is used to assess performance on different tasks

from sklearn import metrics
print('Mean Absolute Error:',\
      metrics.mean_absolute_error(y_test, y_predict))
print('Mean Squared Error:', \
      metrics.mean_squared_error \
      (y_test, y_predict))
print('Root Mean Squared Error:', \
      np.sqrt(metrics.mean_squared_error(y_test, y_predict)))
```

```
Mean Absolute Error: 2.5
Mean Squared Error: 6.25
Root Mean Squared Error: 2.5
```

R squared - coefficient of determination gives you goodness of fit, SSR/SST R-squared values range from 0 to 1 and are commonly stated as percentages from 0% to 100%

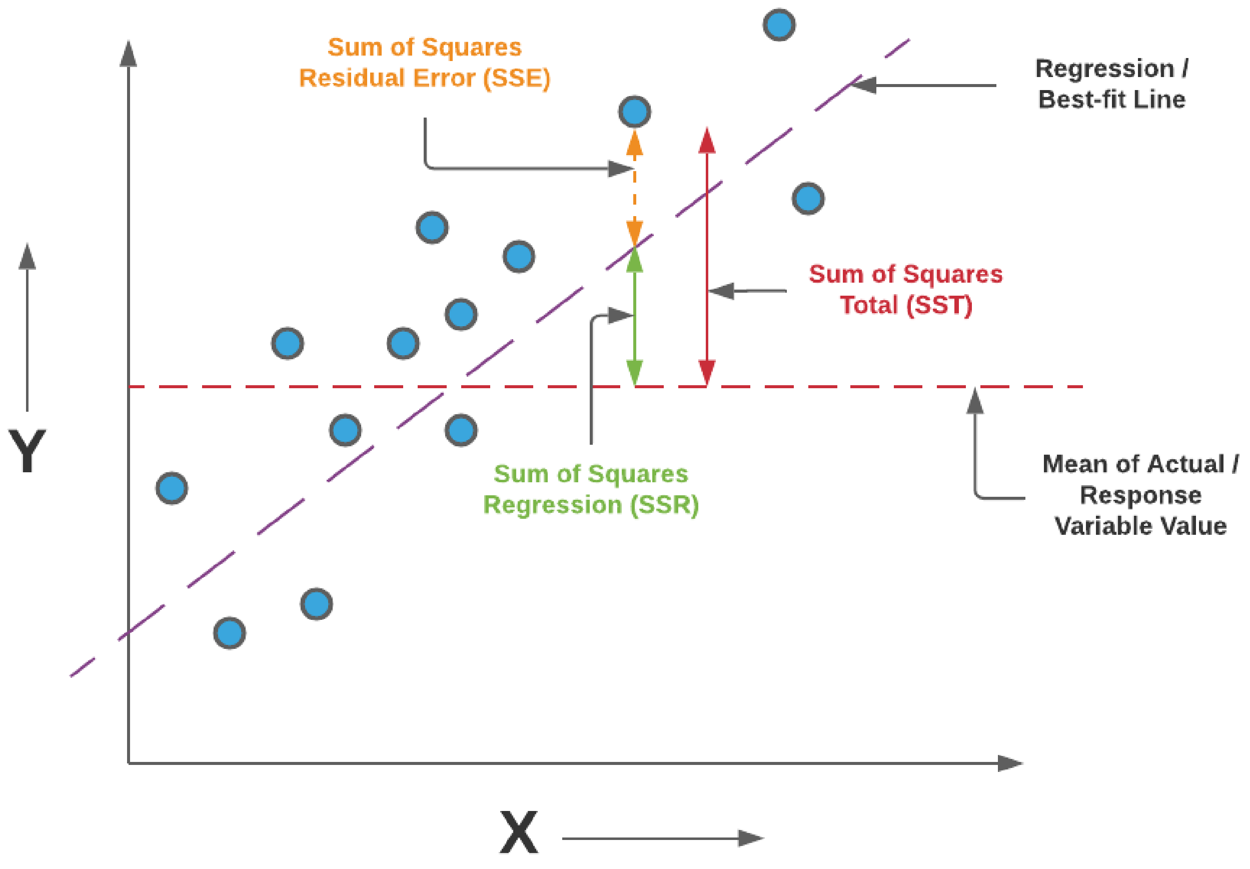
In [81]:
```
regressor.score(x_train, y_train)
```

Out[81]:
```
0.9174311926605505
```

91%of variability in Y is expressed by the independent variable X

In [82]:
```
from IPython import display
display.Image("https://vitalflux.com/wp-content/uploads/2020/09/Regression-terminologies
```

Out[82]:



In [83]:
```
from IPython import display
display.Image("https://vitalflux.com/wp-content/uploads/2019/07/R-squared-formula-functi
```

Out[83]:

$$R^2 = \frac{SSR}{SST} = \frac{\sum(\hat{y}_i - \bar{y})^2}{\sum(y_i - \bar{y})^2}$$

In [ ]:
```
```