In [1]:

```python
import numpy as np
import pandas as pd

import os

# import model related libraries
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

# import module to calculate model perfomance metrics
from sklearn import metrics
```

In [2]:

```python
#data_path = "data/Advertising.csv" # or load the dataset directly from the link
#data_link = "http://www-bcf.usc.edu/~gareth/ISL/Advertising.csv"

#Step 1: get the data (read the csv file)
os.chdir('F:\Training Python')
data = pd.read_csv('Advertising.csv', index_col=0 , error_bad_lines = False)
```

```
C:\Users\TEMP\AppData\Local\Temp\ipykernel_7512\2235661692.py:6: FutureWarning: The err
or_bad_lines argument has been deprecated and will be removed in a future version. Use
on_bad_lines in the future.


  data = pd.read_csv('Advertising.csv', index_col=0 , error_bad_lines = False)
```

In [3]:

```python
data
```

Out[3]:

	TV	Radio	Newspaper	Sales
1	230.1	37.8	69.2	22.1
2	44.5	39.3	45.1	10.4
3	17.2	45.9	69.3	9.3
4	151.5	41.3	58.5	18.5
5	180.8	10.8	58.4	12.9
...	...	...	...	...
196	38.2	3.7	13.8	7.6
197	94.2	4.9	8.1	9.7
198	177.0	9.3	6.4	12.8
199	283.6	42.0	66.2	25.5
200	232.1	8.6	8.7	13.4

200 rows × 4 columns

What are the features?

TV: advertising dollars spent on TV for a single product in a given market (in thousands of dollars) Radio: advertising dollars spent on Radio Newspaper: advertising dollars spent on Newspaper What is the response?

Sales: sales of a single product in a given market (in thousands of widgets

In [4]:

```python
# print the shape of the DataFrame
data.shape
```

Out[4]: (200, 4)

In [5]:
```
#Step 2: select independent(X) and dependent variable (y)
# create a Python list of feature names
feature_names=['TV']

# use the list to select a subset of the original DataFrame
X = data[feature_names]

# sales
y = data.Sales
```

In [6]:
```
# Step 3: Splitting X and y into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1,test_size=0
```

In [7]:
```
# Step 4: Fit linear regression model to trainingset
# Linear Regression Model
linreg = LinearRegression()

# fit the model to the training data (learn the coefficients)
linreg.fit(X_train, y_train)
```

Out[7]:
```
LinearRegression()
```

In [8]:
```
print("Intercept=",linreg.intercept_)
print("Slope=",linreg.coef_)
```

```
Intercept= 6.799773449796854
Slope= [0.0492751]
```

Interpreting Model Coefficients How do we interpret the TV coefficient ($\beta1$)?

A "unit" increase in TV ad spending is associated with a 0.0492751 "unit" increase in Sales. Or more clearly: An additional $1,000 spent on TV ads is associated with an increase in sales of 49.2751 widgets. Note that if an increase in TV ad spending was associated with a decrease in sales, $\beta1$ would be negative.

Hypothesis Testing and p-values Generally speaking, you start with a null hypothesis and an alternative hypothesis (that is opposite the null). Then, you check whether the data supports rejecting the null hypothesis or failing to reject the null hypothesis.

(Note that "failing to reject" the null is not the same as "accepting" the null hypothesis. The alternative hypothesis may indeed be true, except that you just don't have enough data to show that.)

As it relates to model coefficients, here is the conventional hypothesis test:

null hypothesis: There is no relationship between TV ads and Sales (and thus $\beta1$ equals zero) alternative hypothesis: There is a relationship between TV ads and Sales (and thus $\beta1$ is not equal to zero) How do we test this hypothesis? Intuitively, we reject the null (and thus believe the alternative) if the 95% confidence interval does not include zero. Conversely, the p-value represents the probability that the coefficient is actually zero:#

In [9]:
```
from sklearn.feature_selection import f_regression
fregression=f_regression(X_train, y_train) #returns Fvalues of features; p values of
fregression
```

Out[9]:
```
(array([280.14341559]), array([8.01212052e-37]))
```

In [10]:
```
#Step 5: Test the model's generalization ability using testset
# make predictions on the testing set
y_pred = linreg.predict(X_test)
```

In [11]:

```
y_pred
```

Out[11]:

```
array([17.18696362, 16.77798032, 11.51540011, 20.60665526, 19.30579273,
       20.77419058, 14.84639657, 15.70871075, 10.2785952 , 17.41362906,
       14.90552669, 10.20961007, 17.37913649, 12.21017895, 17.92609005,
       12.99365298, 13.28930355, 21.12404376,  8.0612159 , 17.18203611,
       11.74699305, 10.14062494,  8.03657835, 12.09191872, 12.36293175,
       16.08320147,  8.92353007, 19.05941725, 15.01885941, 18.63072392,
       18.6208689 , 18.35478338, 14.17625527, 15.18639473, 19.03970721,
       15.91073864, 17.75855473, 13.17597083, 17.48261419,  7.76556532])
```

In [12]:

```
# Step 6: Compute the performance of the model using metrics
print("RMSE=", np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print("Rsquare=",metrics.r2_score(y_test,y_pred))
```

```
RMSE= 3.2953520791575928
Rsquare= 0.41535307148347866
```

In [13]:

```
df_predicted=pd.DataFrame()
df_predicted['Actual']=y_test
df_predicted['Predicted']=y_pred
df_predicted.head(10)
```

Out[13]:

	Actual	Predicted
59	23.8	17.186964
41	16.6	16.777980
35	9.5	11.515400
103	14.8	20.606655
185	17.6	19.305793
199	25.5	20.774191
96	16.9	14.846397
5	12.9	15.708711
30	10.5	10.278595
169	17.1	17.413629

In [ ]:

```

```