In [1]:

```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
```

In [2]:

```python
df=pd.read_csv(r'C:\Users\sharma17\Downloads\ml\regression dataset\advertising.csv')
```

In [3]:

```python
df.head()
```

Out[3]:

	Unnamed: 0	TV	radio	newspaper	sales
0	1	230.1	37.8	69.2	22.1
1	2	44.5	39.3	45.1	10.4
2	3	17.2	45.9	69.3	9.3
3	4	151.5	41.3	58.5	18.5
4	5	180.8	10.8	58.4	12.9

In [4]:

```python
df.drop(['Unnamed: 0'],inplace=True,axis=1)
```

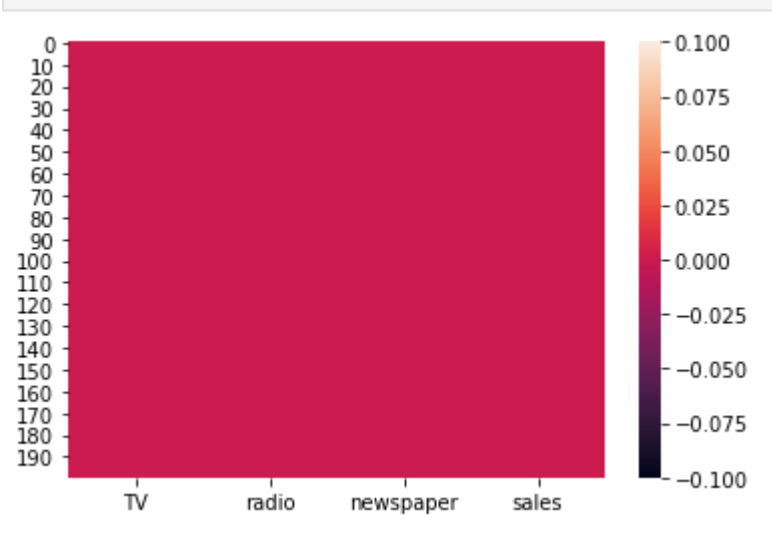
In [5]:

```python
df.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
```

In [6]:

```python
sns.heatmap(df.isnull())
plt.show()
```



In [7]:

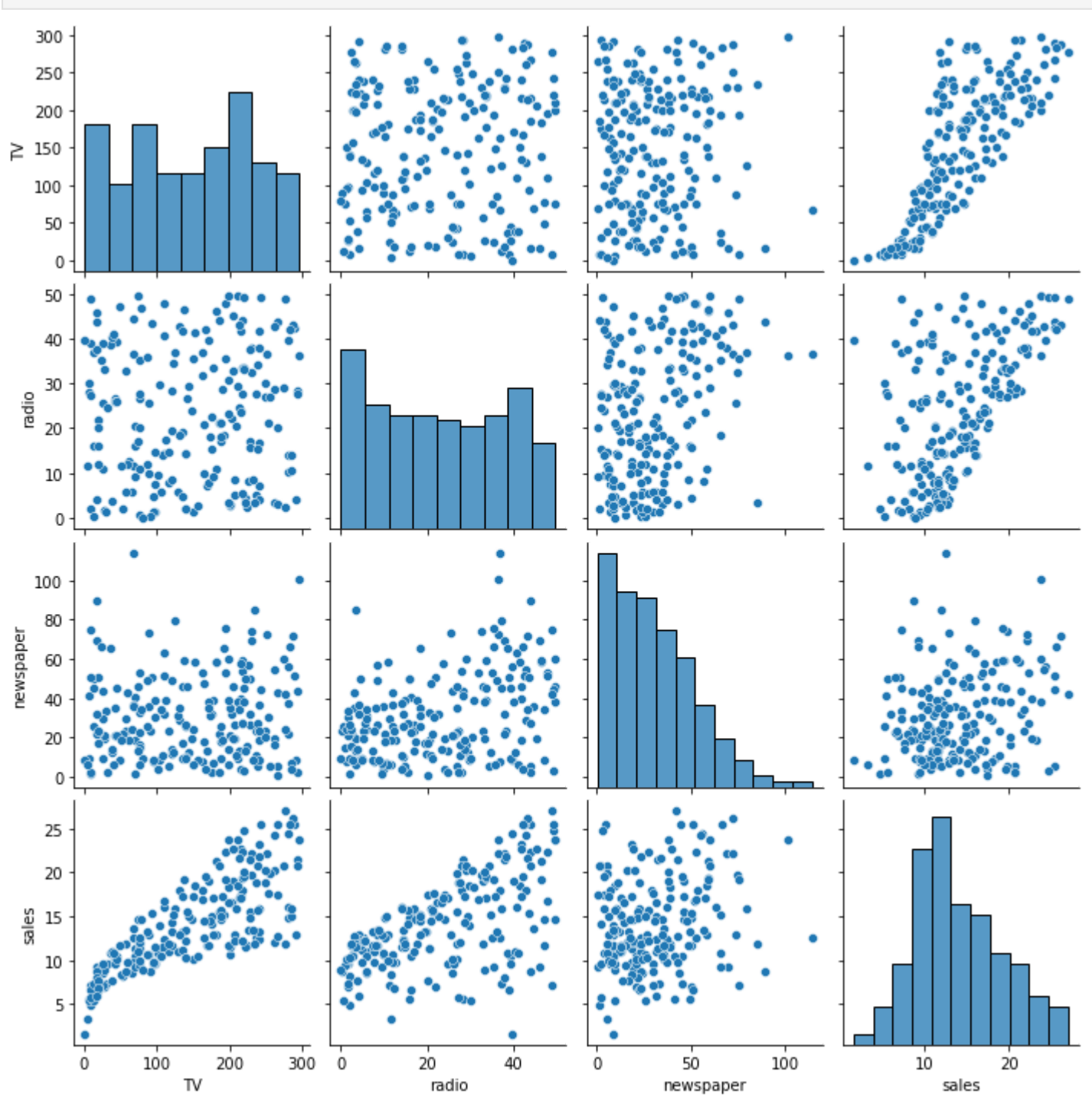
```python
df.describe()
```

Out[7]:

	TV	radio	newspaper	sales
count	200.000000	200.000000	200.000000	200.000000
mean	147.042500	23.264000	30.554000	14.022500
std	85.854236	14.846809	21.778621	5.217457
min	0.700000	0.000000	0.300000	1.600000
25%	74.375000	9.975000	12.750000	10.375000
50%	149.750000	22.900000	25.750000	12.900000
75%	218.825000	36.525000	45.100000	17.400000
max	296.400000	49.600000	114.000000	27.000000

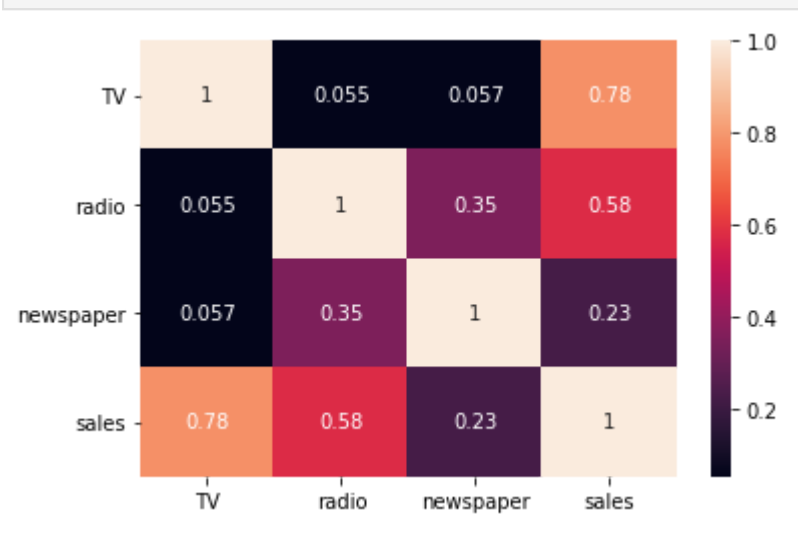
In [8]:

```python
sns.pairplot(data=df)
plt.show()
```



In [9]:

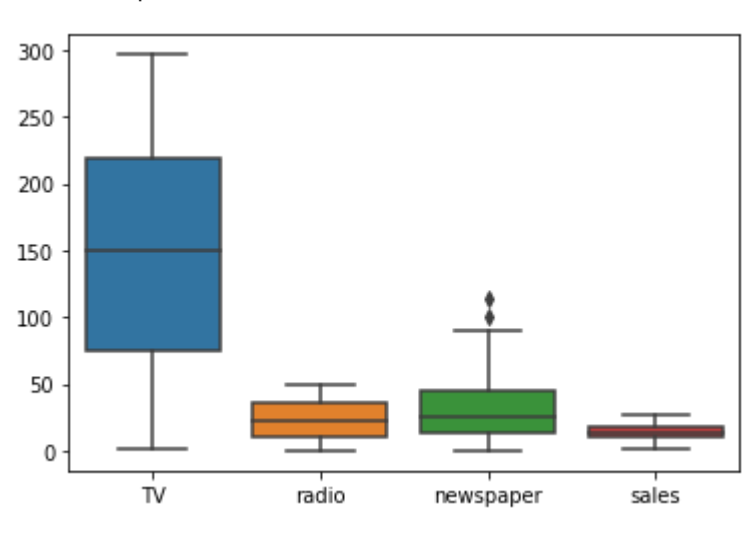
```python
sns.heatmap(df.corr(),annot=True)
plt.show()
```



In [10]:

```python
sns.boxplot(data=df)
```

Out[10]: <AxesSubplot:>



In [11]:

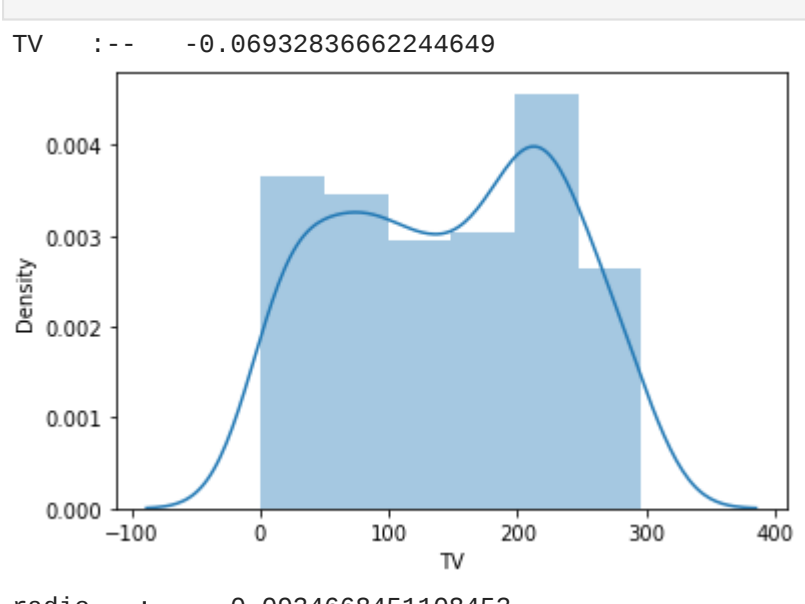
```python
from scipy.stats import skew
```

In [12]:

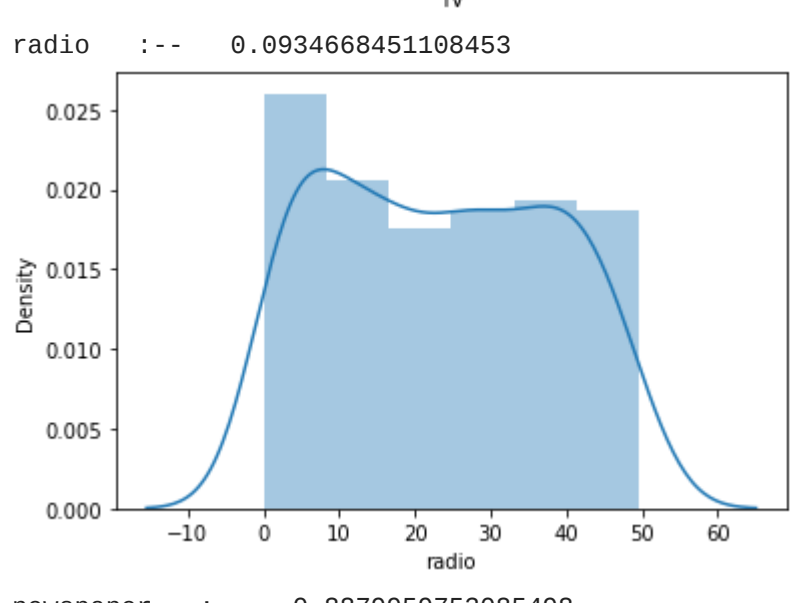
```python
for col in df:

    print (col,'  :--   ',end='' )
    print(skew(df[col]))
    sns.distplot(df[col])
    plt.show()
```

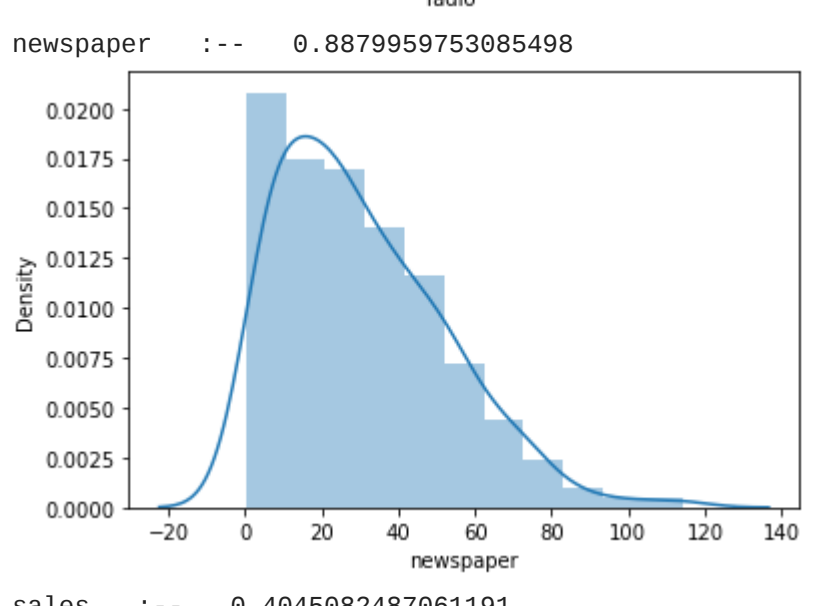
TV   :--   -0.06932836662244649



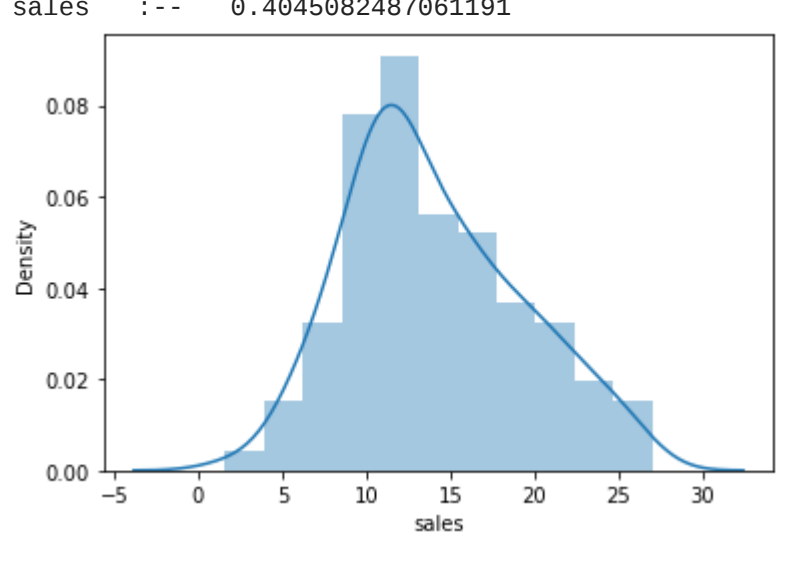
radio   :--   0.0934668451108453



newspaper   :--   0.8879959753085498



sales   :--   0.4045082487061191



In [13]:

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

In [14]:

```python
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest = train_test_split(x,y,random_state=0)
```

In [15]:

```python
from sklearn.linear_model import LinearRegression ,Lasso,Ridge
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import RandomForestRegressor ,AdaBoostRegressor,GradientBoostingRegressor
from xgboost import XGBRegressor
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import r2_score,mean_squared_error ,mean_absolute_error
```

In [16]:

```python
models=[('Logistic Regression',LinearRegression()),
        ('Decision Tree Regressor',DecisionTreeRegressor()),
        ('Support Vector Regressor ',SVR()),
        ('K Nearest Neighbors Regresor',KNeighborsRegressor()),
        ('Random Forest Regressor',RandomForestRegressor()),
        ('AdaBoost Regressor',AdaBoostRegressor()),
        ('Gradient Boosting Regressor',GradientBoostingRegressor()),
        ('Xtreame Gradient Boosting Regressor',XGBRegressor())
]
```

In [17]:

```python
accuracy=[]

for name,model in models:

    print(name,  ':--  ')
    print()
    model.fit(xtrain,ytrain)
    ypred=model.predict(xtest)
    print('       mean absolute error            :-- ',mean_absolute_error(ytest,ypred))
    print()

    print('       mean squared error             :--  ',mean_squared_error(ytest,ypred))

    print()


    print('       squar root  mean squared error :--  ',np.sqrt(mean_squared_error(ytest,ypred)))


    print()
    print('       accuracy                       :-  ',r2_score(ytest,ypred)*100)

    print()

    accuracy.append(r2_score(ytest,ypred))
```

```
Logistic Regression :--

       mean absolute error            :--  1.3000320919235449

       mean squared error             :--   4.0124975229171

       squar root  mean squared error :--   2.003121944095541

       accuracy                       :-   85.76396745320893

Decision Tree Regressor :--

       mean absolute error            :--  1.14

       mean squared error             :--   2.904

       squar root  mean squared error :--   1.7041126723312634

       accuracy                       :-   89.69683139247749

Support Vector Regressor  :--

       mean absolute error            :--  1.4488681638712604

       mean squared error             :--   4.456599302892299

       squar root  mean squared error :--   2.1110659162831227

       accuracy                       :-   84.18832850073463

K Nearest Neighbors Regresor :--

       mean absolute error            :--  1.1516000000000002

       mean squared error             :--   2.163208000000001

       squar root  mean squared error :--   1.4707848245069708

       accuracy                       :-   92.32510442247191

Random Forest Regressor :--

       mean absolute error            :--  0.7034999999999991

       mean squared error             :--   1.0132460599999975

       squar root  mean squared error :--   1.0066012418033257

       accuracy                       :-   96.40508092386783

AdaBoost Regressor :--

       mean absolute error            :--  1.0084519712694031

       mean squared error             :--   1.542052874767554

       squar root  mean squared error :--   1.2417942159502733

       accuracy                       :-   94.52891502395148

Gradient Boosting Regressor :--

       mean absolute error            :--  0.7021243131367695

       mean squared error             :--   1.0140453725653682

       squar root  mean squared error :--   1.0069981988888401

       accuracy                       :-   96.40224502437363

Xtreame Gradient Boosting Regressor :--

       mean absolute error            :--  0.6502818851470947

       mean squared error             :--   0.8375041511322796

       squar root  mean squared error :--   0.91515252888919

       accuracy                       :-   97.02859969744631
```

In [18]:

```python
print('this is the  mean accuracy all models we have used  :--  ',np.array(accuracy).mean()*100)
```

this is the mean accuracy all models we have used :--   92.04238405481652

In [19]:

```python
xtrain,xtest,ytrain,ytest = train_test_split(x,y,random_state=31)
xtreme=XGBRegressor(
    max_depth=1,
    min_child_weight= 2,
    eta=0.1,
    subsample=1,
    colsample_bytree= 1,
    objective='reg:linear',
    reg_lambda=18
)
xtreme.fit(xtrain,ytrain)
ypred=xtreme.predict(xtest)

print('training  error :--  ',xtreme.score(xtrain,ytrain)*100)
print('testing  error  :--  ',xtreme.score(xtest,ytest)*100)
print('square root of mean squared error ', np.sqrt(mean_squared_error(ytest,ypred)))
```

```
[23:16:13] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/objective/regression_obj.cu:188: reg:linear is now deprecated in favor of reg:squarederror.
training  error :--   91.59322270575599
testing  error  :--   91.16028846121134
square root of mean squared error  1.5368683108788341
```

In [20]:

```python
from sklearn.model_selection import cross_val_score
```

In [21]:

```python
cv=cross_val_score(xtreme,x,y,cv=10)
```

```
[23:16:14] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/objective/regression_obj.cu:188: reg:linear is now deprecated in favor of reg:squarederror.
[23:16:14] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/objective/regression_obj.cu:188: reg:linear is now deprecated in favor of reg:squarederror.
[23:16:14] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/objective/regression_obj.cu:188: reg:linear is now deprecated in favor of reg:squarederror.
[23:16:14] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/objective/regression_obj.cu:188: reg:linear is now deprecated in favor of reg:squarederror.
[23:16:14] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/objective/regression_obj.cu:188: reg:linear is now deprecated in favor of reg:squarederror.
[23:16:14] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/objective/regression_obj.cu:188: reg:linear is now deprecated in favor of reg:squarederror.
[23:16:14] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/objective/regression_obj.cu:188: reg:linear is now deprecated in favor of reg:squarederror.
[23:16:14] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/objective/regression_obj.cu:188: reg:linear is now deprecated in favor of reg:squarederror.
[23:16:14] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/objective/regression_obj.cu:188: reg:linear is now deprecated in favor of reg:squarederror.
[23:16:14] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/objective/regression_obj.cu:188: reg:linear is now deprecated in favor of reg:squarederror.
```

In [22]:

```python
print('this is the minimum accuracy we can get from this model  :-   ',cv.mean()*100)
```

this is the minimum accuracy we can get from this model :-   88.86340820888019

In [23]:

```python
xtreme.feature_importances_
```

Out[23]: array([0.5069558 , 0.49304426, 0.   ], dtype=float32)

In [24]:

```python
x=df.iloc[:,:2].values
y=df.iloc[:,-1].values
```

In [25]:

```python
xtrain,xtest,ytrain,ytest = train_test_split(x,y,random_state=31)
xtreme=XGBRegressor(
    max_depth=1,
    min_child_weight= 2,
    eta=0.1,
    subsample=1,
    colsample_bytree= 1,
    objective='reg:linear',
    reg_lambda=18
)
xtreme.fit(xtrain,ytrain)
ypred=xtreme.predict(xtest)

print('training  error                    :--  ',xtreme.score(xtrain,ytrain)*100)
print('testing  error                     :--  ',xtreme.score(xtest,ytest)*100)
print('square root of mean squared error  :-- ', np.sqrt(mean_squared_error(ytest,ypred)))
```

```
[23:16:14] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/objective/regression_obj.cu:188: reg:linear is now deprecated in favor of reg:squarederror.
training  error                    :--   91.59322270575599
testing  error                     :--   91.16028846121134
square root of mean squared error  :--  1.5368683108788341
```

In [26]:

```python
xtreme.feature_importances_
```

Out[26]: array([0.5069558 , 0.49304426], dtype=float32)

In [27]:

```python
pd.DataFrame(xtreme.feature_importances_,['radio','newspaper'],columns=['co-efficient'])
```

Out[27]:

	co-efficient
radio	0.506956
newspaper	0.493044

The interpretation tell as about

setting all values fixed if we increase 1 unit of radio then it will increase 0.506956 unit in sales setting all values fixed if we increase 1 unit of newspaper then it will increase 0.493044 unit in sales