# Advertising Dataset - Linear Regression Model

## Objective of the Project

- The main objective is to build Linear Regression model to understand the impact of ad budgets on the overall sales with respect to the features.
- The advertising dataset captures the sales revenue generated with respect to advertisement costs across multiple channels like radio, tv, and newspapers.

## Brief about Linear Regression

- Linear regression is a type of supervised machine learning algorithm that computes the linear relationship between a dependent variable and one or more independent features. The goal of the algorithm is to find the best linear equation that can predict the value of the dependent variable based on the independent variables.

## Linear Regression Model assumptions :-

- Linearity: The models of Linear Regression models must be linear in the sense that the output must have a linear association with the input values, and it only suits data that has a linear relationship between the two entities.
- Homoscedasticity: Homoscedasticity means the standard deviation and the variance of the residuals (difference of (y-yˆ)2 ) must be the same for any value of x. Multiple Linear Regression assumes that the amount of error in the residuals is similar at each point of the linear model. We can check the Homoscedasticity using Scatter plots.
- Non-multicollinearity: The data should not have multicollinearity, which means the independent variables should not be highly correlated with each other. We can check the data for this using a correlation matrix.
- No Autocorrelation: When data are obtained across time, we assume that successive values of the disturbance component are momentarily independent in the conventional Linear Regression model. When this assumption is not followed, the situation is referred to be autocorrelation.
- Not applicable to Outliers: The value of the dependent variable cannot be estimated for a value of an independent variable which lies outside the range of values in the sample data.
- No Endogeneity - Endogeneity refers to situations in which a predictor (e.g., treatment variable) in a linear regression model is correlated to the error term.

## Steps followed

- Import required Libraries
- Load the dataset
- Check basic information of the dataset
- Data Visualization
- Data Preprocessing
- Find the correlation
- Split data into dependent and independent variables
- Split data into Train and test
- Build Model - Linear, Lasso, Ridge & Elastic Net method
- Predict the data
- Evaluate the model - r2 score & Performance matrix
- Compare the accuracy
- Conclusion

## Import the required Libraries

In [7]:

```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
%matplotlib inline

import warnings
warnings.filterwarnings('ignore')
```

## Load the dataset - Advertising Sales

In [8]:

```
advt=pd.read_csv(r"C:\Users\manme\Documents\Priya\Stas and ML\Dataset\Advertising.csv")
advt.head(2)
```

Out[8]:

| | TV | radio | newspaper | sales |
|---|---|---|---|---|
| **0** | 230.1 | 37.8 | 69.2 | 22.1 |
| **1** | 44.5 | 39.3 | 45.1 | 10.4 |

## Check basic information

In [9]:

```
advt.shape
```

Out[9]:

```
(200, 4)
```

In [10]:

```
advt.columns
```

Out[10]:

```
Index(['TV', 'radio', 'newspaper', 'sales'], dtype='object')
```

In [11]:

```
advt.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 [69]:

```
advt.describe().T.style.background_gradient(cmap='Blues')
```

Out[69]:

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| **TV** | 200.000000 | 147.042500 | 85.854236 | 0.700000 | 74.375000 | 149.750000 | 218.825000 | 296.400000 |
| **radio** | 200.000000 | 23.264000 | 14.846809 | 0.000000 | 9.975000 | 22.900000 | 36.525000 | 49.600000 |
| **newspaper** | 200.000000 | 30.415750 | 21.316901 | 0.300000 | 12.750000 | 25.750000 | 45.100000 | 93.625000 |
| **sales** | 200.000000 | 14.022500 | 5.217457 | 1.600000 | 10.375000 | 12.900000 | 17.400000 | 27.000000 |

## Data Pre Processing

- It is an important step and involves cleaning and transforming raw data to make it suitable for analysis.
- Checking and removing duplicate values.
- Check for missing values and treat them
- Check for outliers and treat them

## Checking duplicate values

In [18]:

```
advt.duplicated().sum()
```

Out[18]:

```
0
```

## Check for missing values

In [19]:

```
advt.isnull().sum()
```

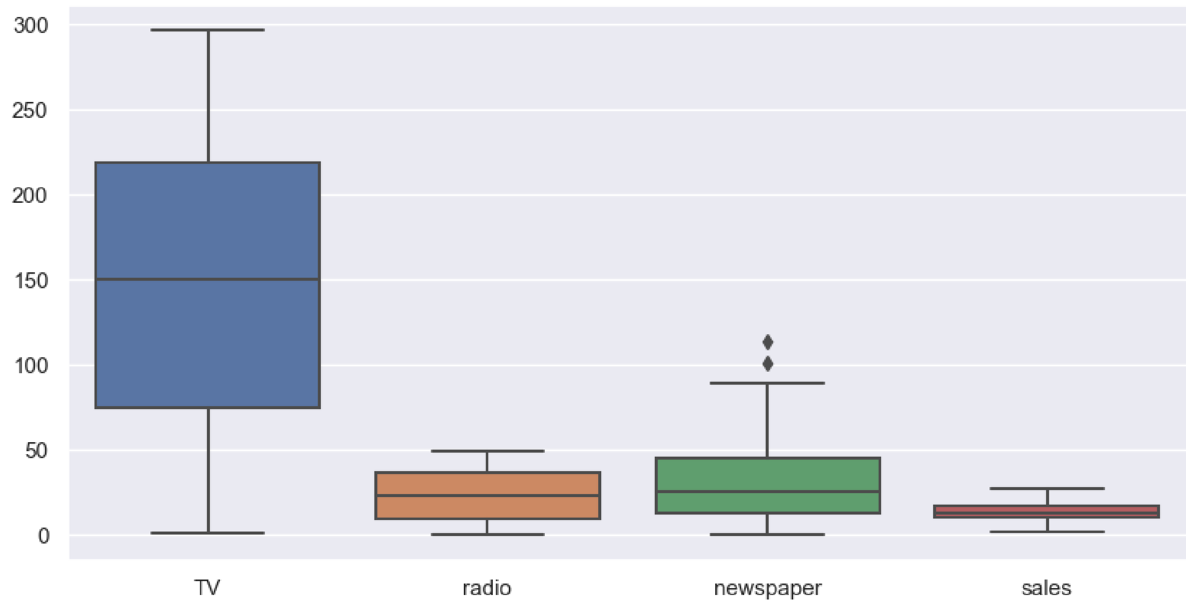
Out[19]:

```
TV           0
radio        0
newspaper    0
sales        0
dtype: int64
```

## Check for outliers

In [20]:

```
plt.figure(figsize=(10,5))
sns.boxplot(advt)
plt.show()
```




## Handling outlier in 'newspaper'

- Capping method

In [21]:

```
advt['newspaper'].describe()
```

Out[21]:

```
count    200.000000
mean      30.554000
std       21.778621
min        0.300000
25%       12.750000
50%       25.750000
75%       45.100000
max      114.000000
Name: newspaper, dtype: float64
```

In [22]:

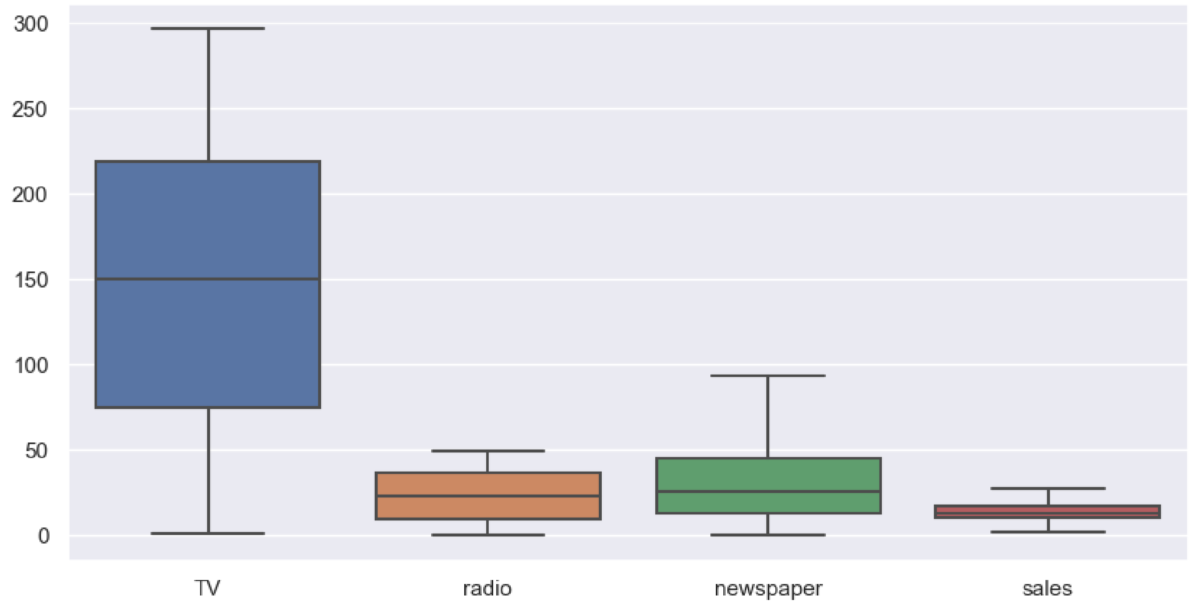
```
news_q1=advt['newspaper'].quantile(0.25)
news_q3=advt['newspaper'].quantile(0.75)
news_iqr=news_q3-news_q1
news_upper= news_q3 +1.5*news_iqr
news_lower= news_q1- 1.5 *news_iqr
```

In [23]:

```
advt['newspaper']=np.where(advt['newspaper']>news_upper, news_upper,
                                   np.where(advt['newspaper']<news_lower,news_lower,
                                          advt['newspaper']))
```

In [24]:

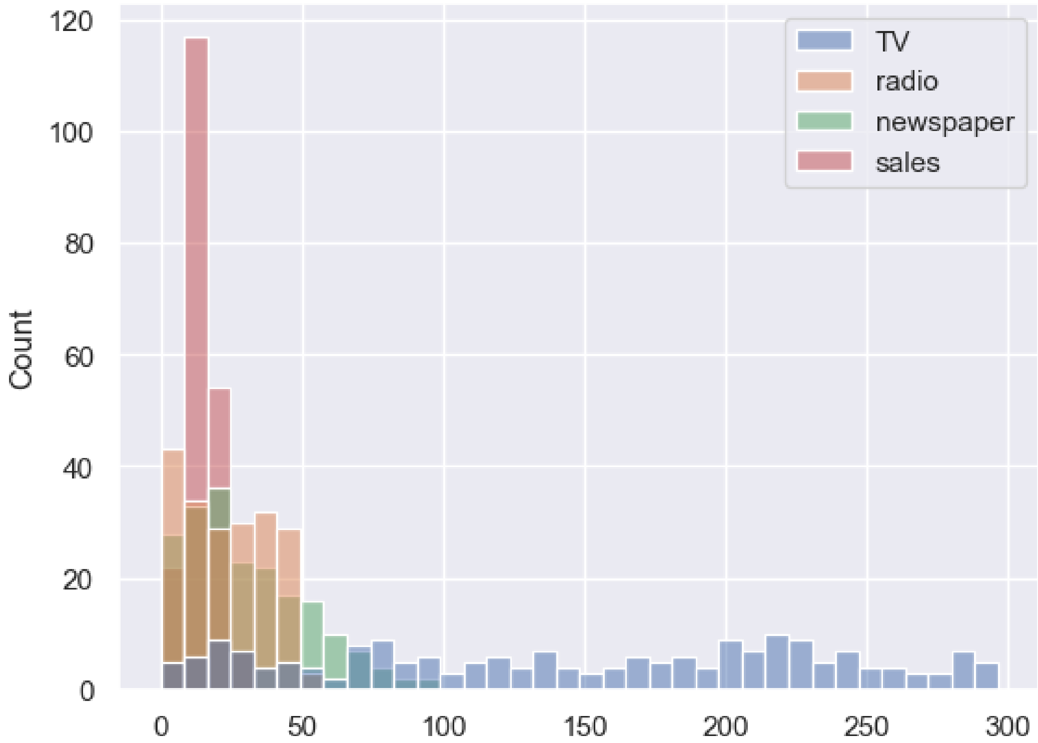
```
plt.figure(figsize=(10,5))
sns.boxplot(advt)
plt.show()
```



## Exploratory Data Analysis

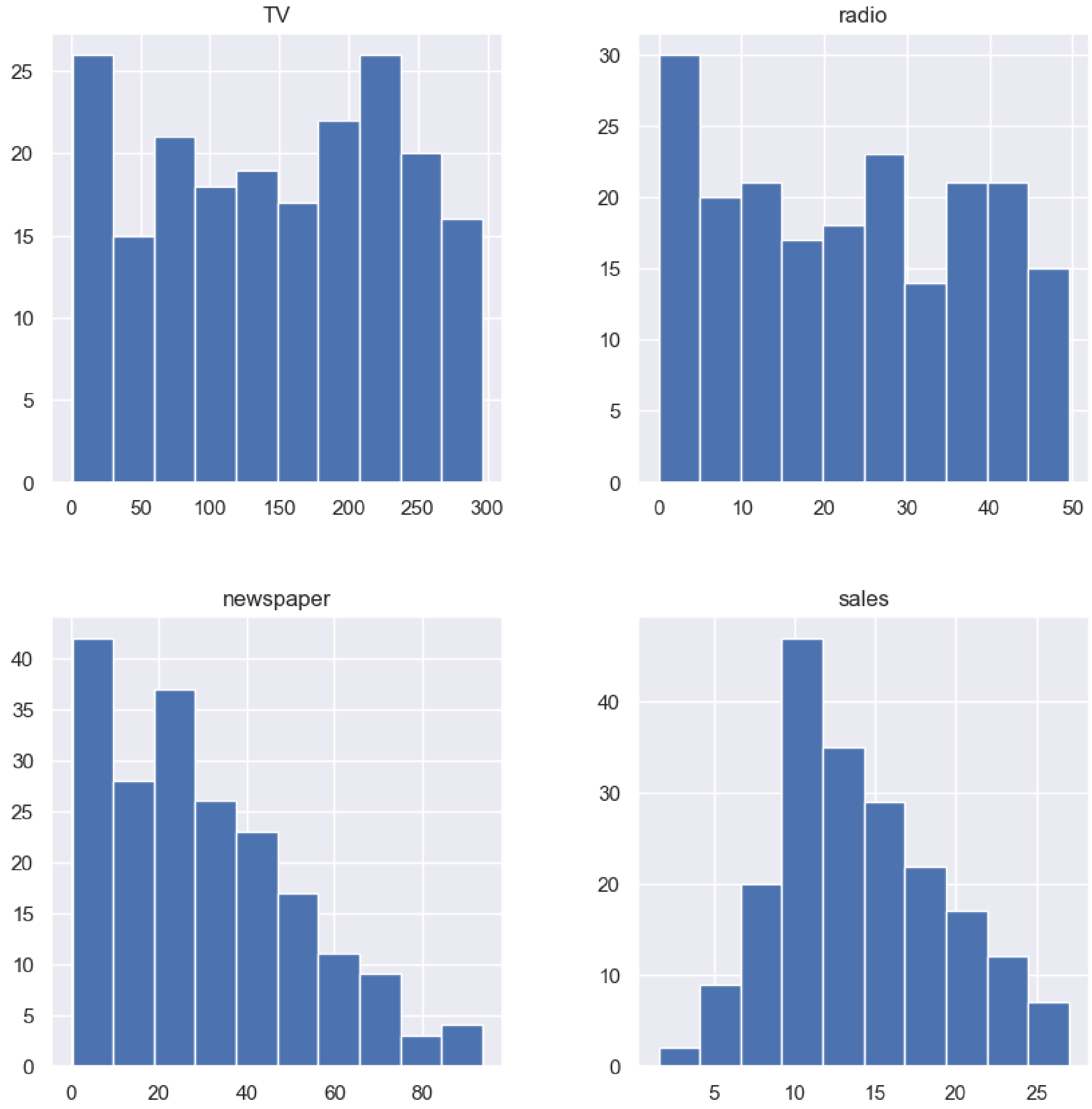
In [60]:

```
sns.histplot(advt)
plt.show()
```



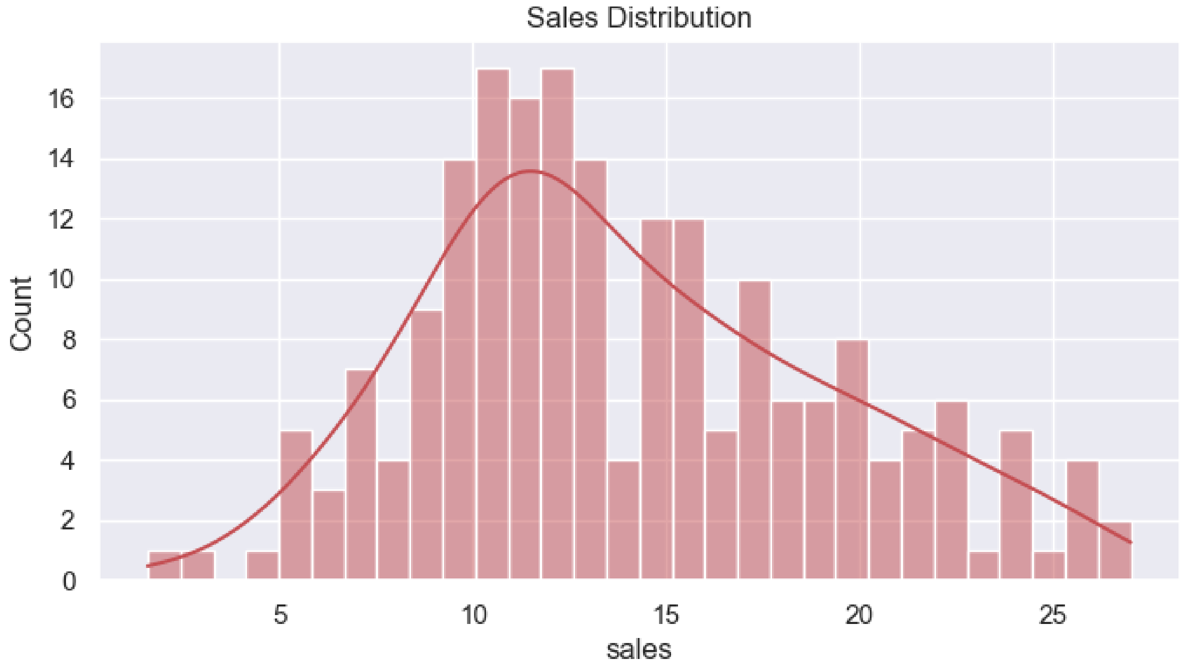
In [61]:

```
advt.hist(figsize=(10,10))
plt.show()
```

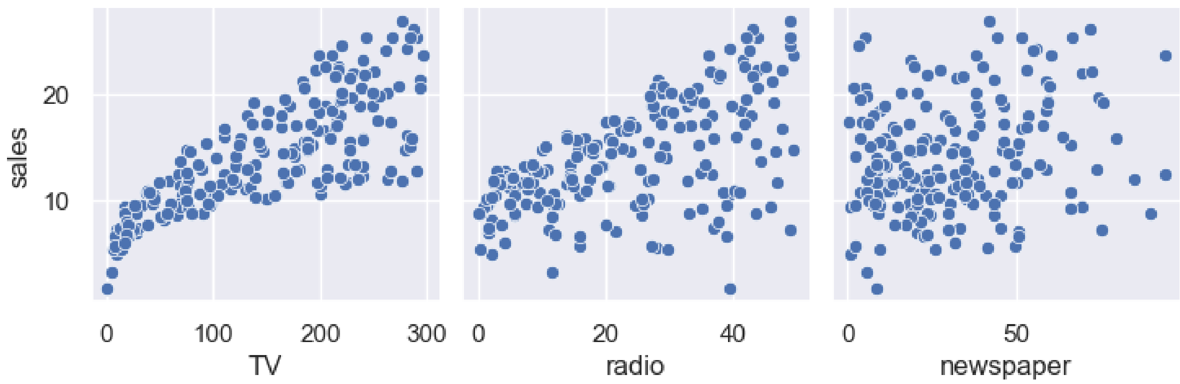


In [62]:

```
plt.figure(figsize=[8,4])
sns.histplot(advt.sales, color='r', bins=30, kde=True)
plt.title('Sales Distribution')
plt.show()
```

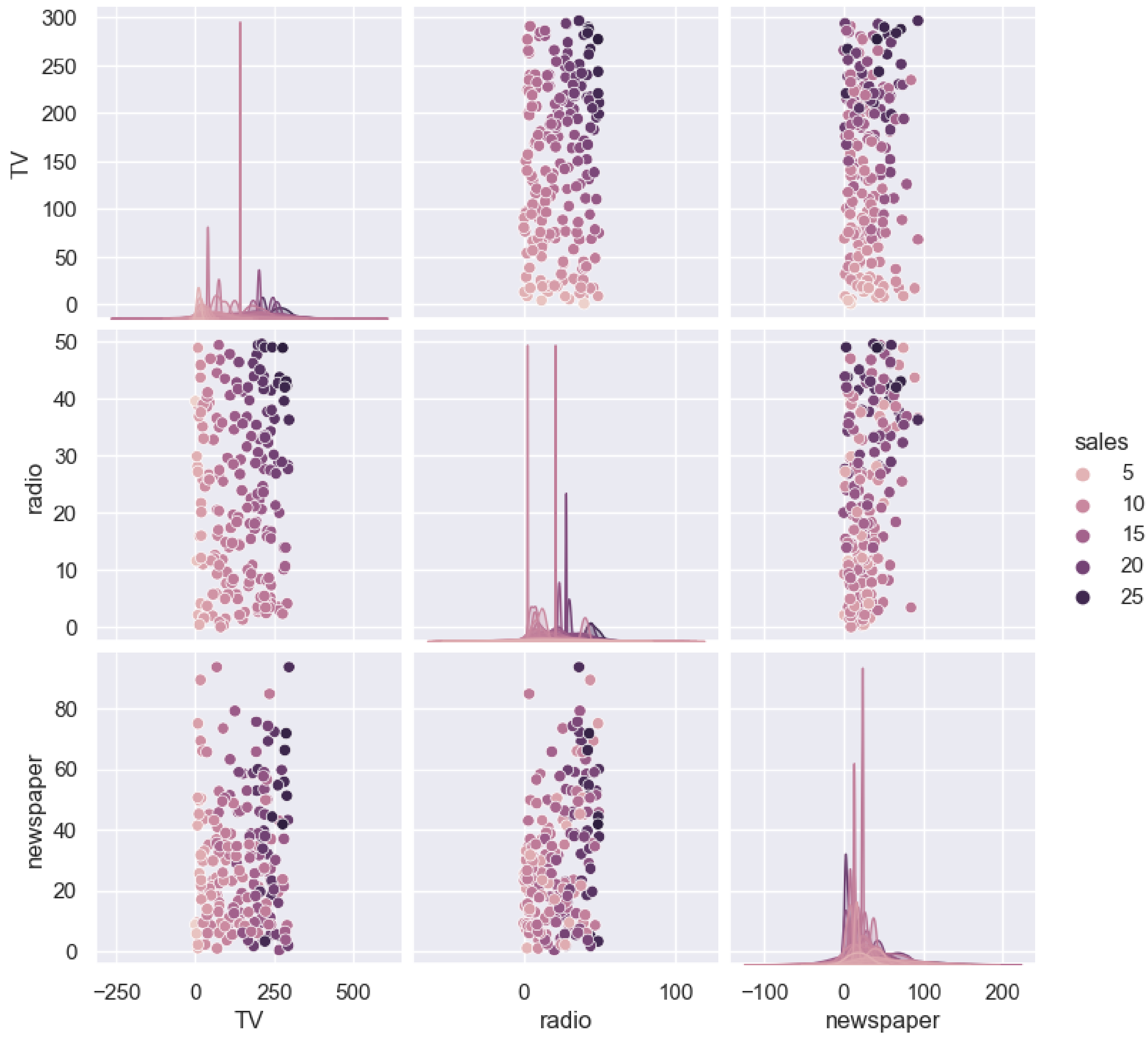



In [63]:



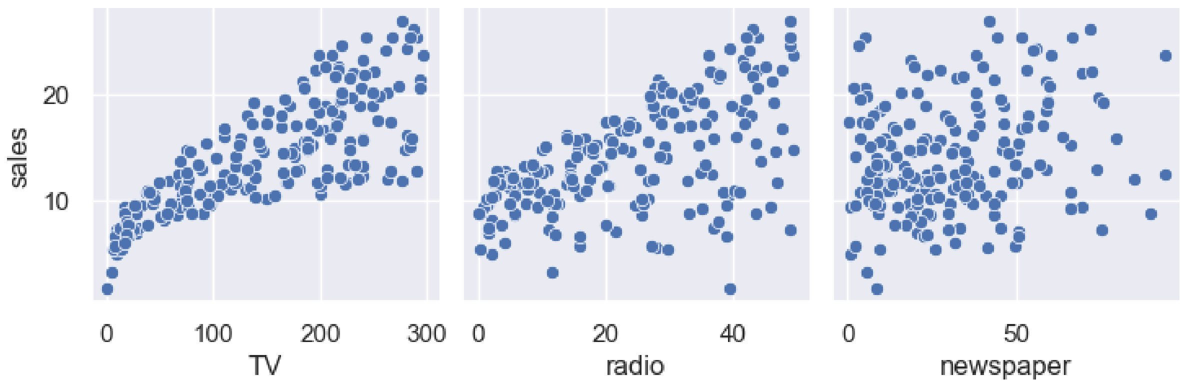
In [71]:

```
sns.pairplot(data=advt, hue='sales')
plt.show()
```




In [72]:

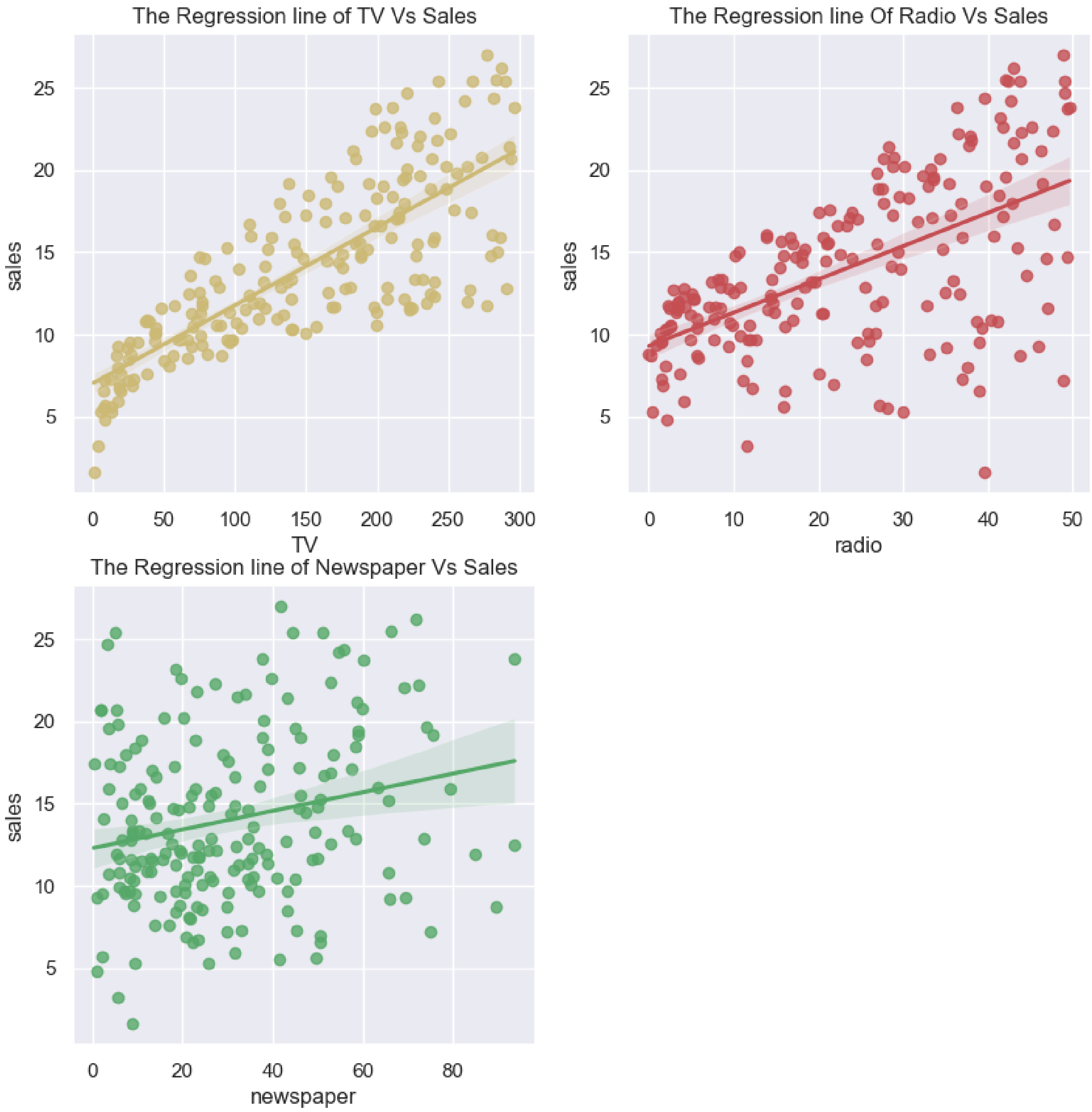
```
sns.pairplot(advt, x_vars=['TV', 'radio', 'newspaper'], y_vars='sales')
plt.show()
```



## Plotting the regression line

In [64]:

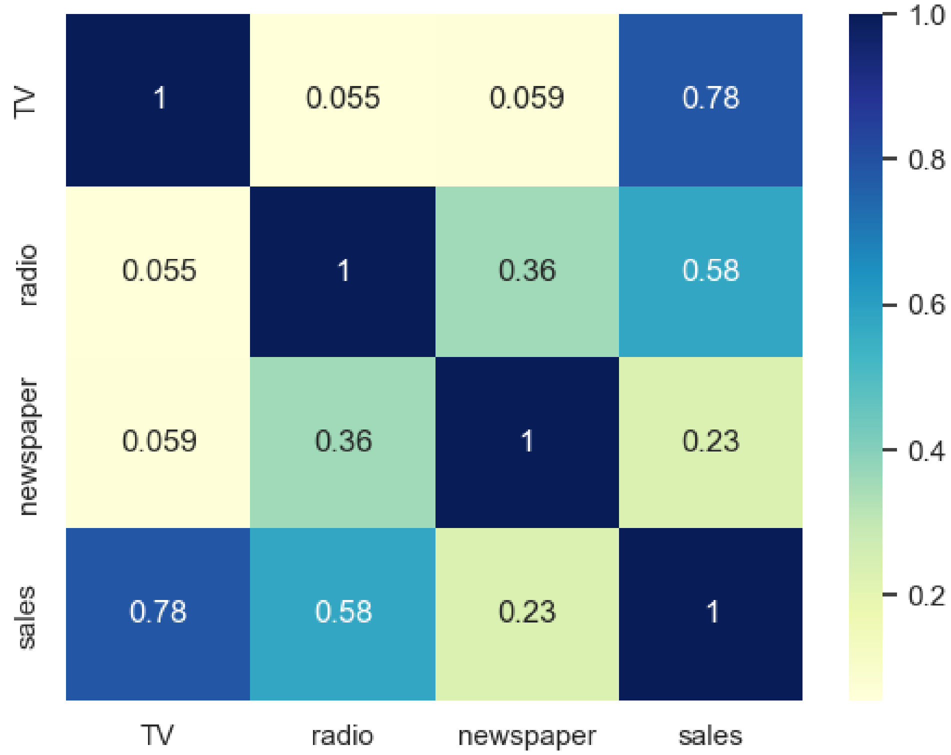
```
plt.figure(figsize =(10,10))
plt.subplot(2,2,1)
sns.regplot(data=advt,x='TV',y='sales',color='y').set_title('The Regression line of TV Vs Sales')
plt.subplot(2,2,2)
sns.regplot(data=advt,x='radio',y='sales',color='r').set_title('The Regression line of Radio Vs Sa
plt.subplot(2,2,3)
sns.regplot(data=advt,x='newspaper',y='sales',color='g').set_title('The Regression line of Newspap
plt.show()
```



### Find the Correlation by using Heatmap

In [25]:

```
sns.heatmap(advt.corr(), cmap="YlGnBu", annot = True)
plt.show()
```




### Split data into independent and Dependent variables

In [26]:

```
x = advt.drop(['sales'],axis=1)
y = advt[['sales']]
```

In [27]:

```
x.head()
```

Out[27]:

| | TV | radio | newspaper |
|---|---|---|---|
| **0** | 230.1 | 37.8 | 69.2 |
| **1** | 44.5 | 39.3 | 45.1 |
| **2** | 17.2 | 45.9 | 69.3 |
| **3** | 151.5 | 41.3 | 58.5 |
| **4** | 180.8 | 10.8 | 58.4 |

In [28]:

```
y.head()
```

Out[28]:

| | sales |
|---|---|
| 0 | 22.1 |
| 1 | 10.4 |
| 2 | 9.3 |
| 3 | 18.5 |
| 4 | 12.9 |

## Split the data into Training and Testing

In [29]:

```
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)
```

## Build Model

## Approach 1

- Linear Regression Method

In [30]:

```
from sklearn.linear_model import LinearRegression
linear_model= LinearRegression()
linear_model.fit(x_train,y_train)
```

Out[30]:

```
▾ LinearRegression
LinearRegression()
```

In [45]:

```
print("Intercept of the Linear model:  ", linear_model.intercept_)
print("Coefficients of the Linear model: ", linear_model.coef_)
```

```
Intercept of the Linear model:   [2.99690177]
Coefficients of the Linear model:  [[ 0.04458387  0.19653672 -0.00288253]]
```

## Predict

- by using Linear regression method

In [46]:

```
y_pred_sales =linear_model.predict(x_test)
y_pred_sales_train=linear_model.predict(x_train)
```

## Approach 2

- Lasso Regression

In [48]:

```
from sklearn.linear_model import Lasso
lasso = Lasso(alpha=0.1)
lasso.fit(x_train, y_train)
print("Coefficients of the Lasso Model :", (lasso.coef_))
```

```
Coefficients of the Lasso Model : [ 0.04456473  0.19590649 -0.00251167]
```

## Predict

- by using Lasso method

In [34]:

```
y_pred_train_lasso = lasso.predict(x_train)
y_pred_test_lasso = lasso.predict(x_test)
```

## Approach 3

- Ridge Regression

In [49]:

```
from sklearn.linear_model import Ridge
ridge = Ridge(alpha=0.3)
ridge.fit(x_train, y_train)
print("Coefficients of the Ridge Model :", (ridge.coef_))
```

```
Coefficients of the Ridge Model : [[ 0.04458386  0.1965348  -0.00288209]]
```

## Predict

- by using Ridge method

In [36]:

```
y_pred_train_ridge = ridge.predict(x_train)
y_pred_test_ridge = ridge.predict(x_test)
```

## Approach 4

- Elastic Net Regression

In [50]:

```
from sklearn.linear_model import ElasticNet
elastic = ElasticNet(alpha=0.3, l1_ratio=0.1)
elastic.fit(x_train, y_train)
print("Coefficients of the Elastic Net :", (elastic.coef_))
```

```
Coefficients of the Elastic Net : [ 0.04457682  0.19607405 -0.00270793]
```

## Predict

- by using Elastic Net method

In [52]:

```
y_pred_train_elastic = elastic.predict(x_train)
y_pred_test_elastic = elastic.predict(x_test)
```

## Evaluate the model

## 1. By Square Loss function

In [53]:

```
from sklearn import metrics
from sklearn.metrics import r2_score
```

In [54]:

```
print('Linear model r2 Score test accuracy:',r2_score (y_test,y_pred_sales))
print('Lasso r2 Score test accuracy :',r2_score(y_test, y_pred_test_lasso))
print('Ridge r2 Score test accuracy:', r2_score(y_test, y_pred_test_ridge))
print('Elastic Net r2 Score test accuracy :', r2_score(y_test, y_pred_test_elastic))
```

```
Linear model r2 Score test accuracy: 0.8600564143025663
Lasso r2 Score test accuracy : 0.8603039434708047
Ridge r2 Score test accuracy: 0.8600572909687636
Elastic Net r2 Score test accuracy : 0.860255622265056
```

In [55]:

```
print('Linear model r2 Score train accuracy:',r2_score(y_train,y_pred_sales_train))
print('Lasso r2 Score train accuracy:',r2_score(y_train, y_pred_train_lasso))
print('Ridge r2 Score train accuracy:',r2_score(y_train, y_pred_train_ridge))
print('Elastic Net r2 Score train accuracy :', r2_score(y_train, y_pred_train_elastic))
```

```
Linear model r2 Score train accuracy: 0.9067187984110153
Lasso r2 Score train accuracy: 0.9067148519940235
Ridge r2 Score train accuracy: 0.9067187983835776
Elastic Net r2 Score train accuracy : 0.9067170971292787
```

## 2. Performance Matrix

- Mean absolute error
- Mean absolute percent error
- Mean squared error
- Root mean squared error

In [56]:

```
from sklearn import metrics
```

In [57]:

```
print('MAE:', metrics.mean_absolute_error(y_test,y_pred_sales))
print('MAPE:', metrics.mean_absolute_error(y_test,y_pred_sales)/100)
print('MSE:', metrics.mean_squared_error(y_test,y_pred_sales))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test,y_pred_sales)))
```

```
MAE: 1.3628974741922129
MAPE: 0.013628974741922128
MSE: 4.403946798278694
RMSE: 2.098558266591303
```

## Comparison

In [59]:

```
dict1={"Test r2_score":[0.8611251045785454,0.8615629087416048,0.8611262752219822,0.861402707158276
      "Train r2_score":[0.9055124964622553,0.9055104089549475,0.905512496425828,0.9055112371840714
pd.DataFrame(dict1,index=["Linear","Lasso","Ridge","Elastic Net"])
```

Out[59]:

| | Test r2_score | Train r2_score |
|---|---|---|
| **Linear** | 0.861125 | 0.905512 |
| **Lasso** | 0.861563 | 0.905510 |
| **Ridge** | 0.861126 | 0.905512 |
| **Elastic Net** | 0.861403 | 0.905511 |

## Conclusion

- r2 score of all 4 methods are almost similar
- Both test and train data accuracy is more than the commonly taken threshold value of 75%

In [ ]:

```

```

In [ ]:

```

```