In [1]:
```
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')
```

In [2]:
```
data = pd.read_csv('advertising (2).csv')
```

In [3]:
```
data.head()
```

Out[3]:

	TV	Radio	Newspaper	Sales
0	230.1	37.8	69.2	22.1
1	44.5	39.3	45.1	10.4
2	17.2	45.9	69.3	12.0
3	151.5	41.3	58.5	16.5
4	180.8	10.8	58.4	17.9

In [4]:
```
data.tail()
```

Out[4]:

	TV	Radio	Newspaper	Sales
195	38.2	3.7	13.8	7.6
196	94.2	4.9	8.1	14.0
197	177.0	9.3	6.4	14.8
198	283.6	42.0	66.2	25.5
199	232.1	8.6	8.7	18.4

In [5]:
```
data.shape
```

Out[5]:
```
(200, 4)
```

In [6]:
```
data.columns
```

Out[6]:
```
Index(['TV', 'Radio', 'Newspaper', 'Sales'], dtype='object')
```

In [7]:
```
data.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 [8]:
```
data.describe()
```

Out[8]:

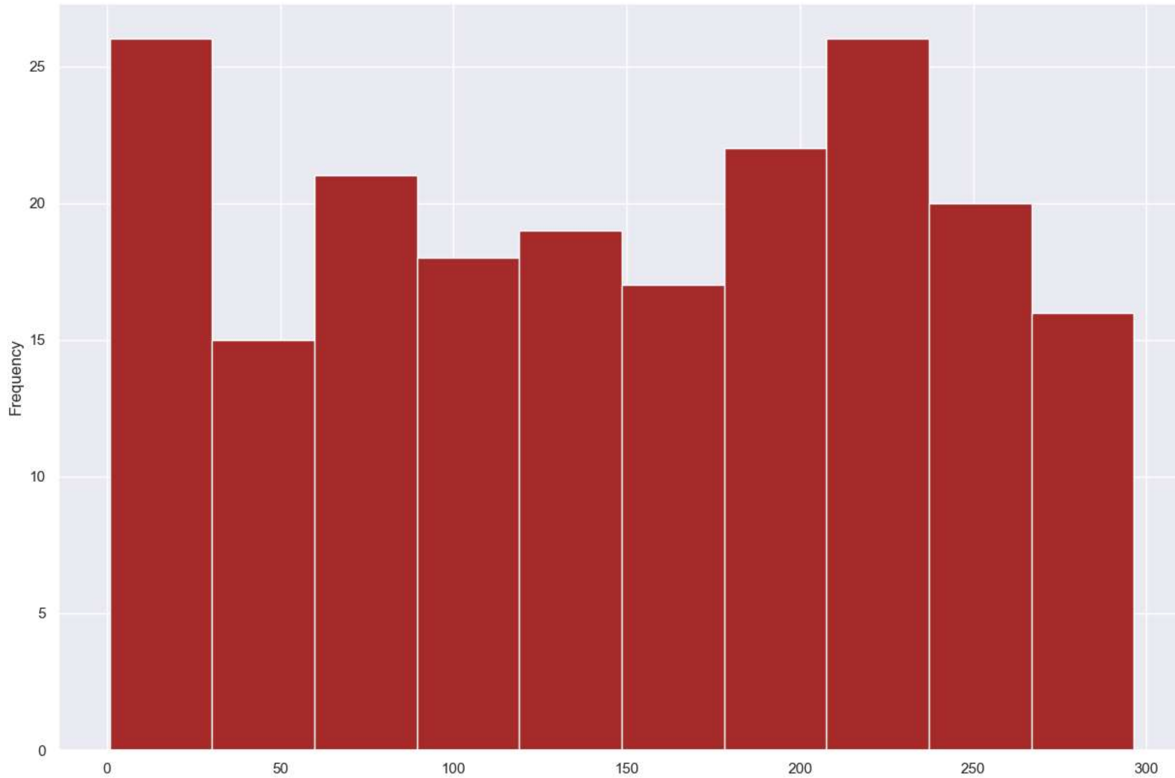
	TV	Radio	Newspaper	Sales
count	200.000000	200.000000	200.000000	200.000000
mean	147.042500	23.264000	30.554000	15.130500
std	85.854236	14.846809	21.778621	5.283892
min	0.700000	0.000000	0.300000	1.600000
25%	74.375000	9.975000	12.750000	11.000000
50%	149.750000	22.900000	25.750000	16.000000
75%	218.825000	36.525000	45.100000	19.050000
max	296.400000	49.600000	114.000000	27.000000

In [9]:
```
data.isnull().sum()/len(data)*100
```

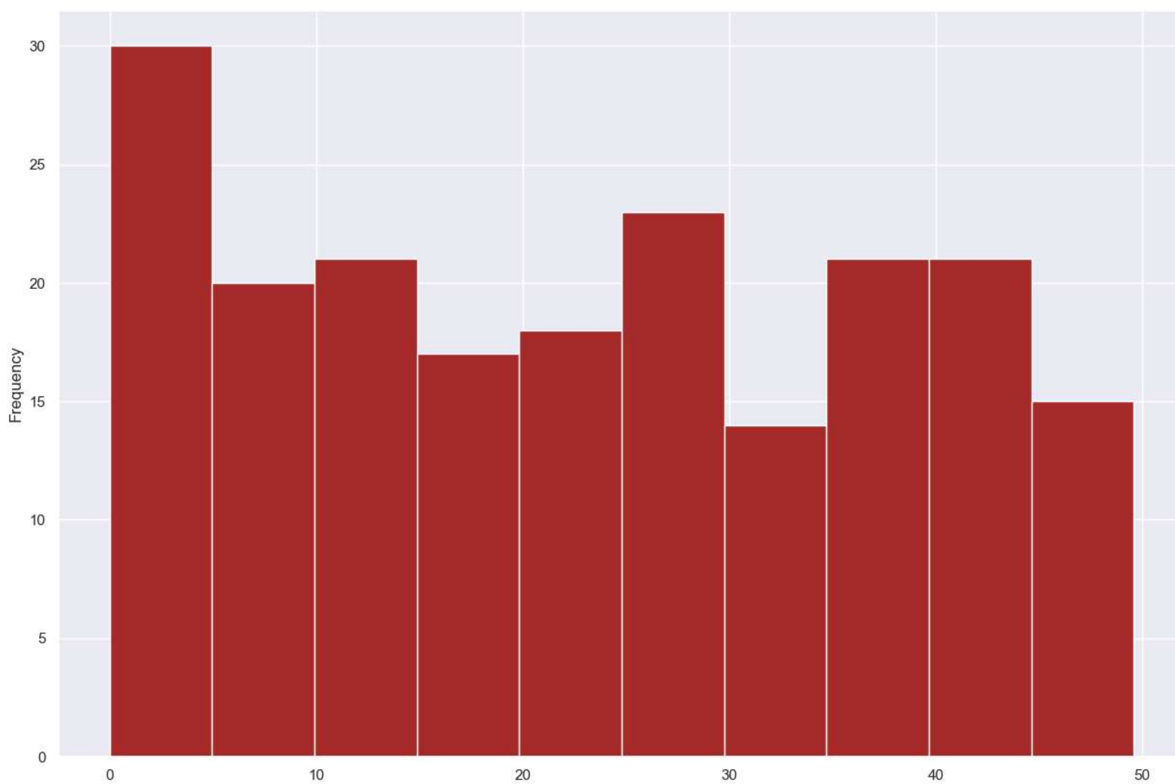
Out[9]:
```
TV           0.0
Radio        0.0
Newspaper    0.0
Sales        0.0
dtype: float64
```

In these data there is 200 rows and 4 columns.
there is no duplicate values.
All columns have float values.

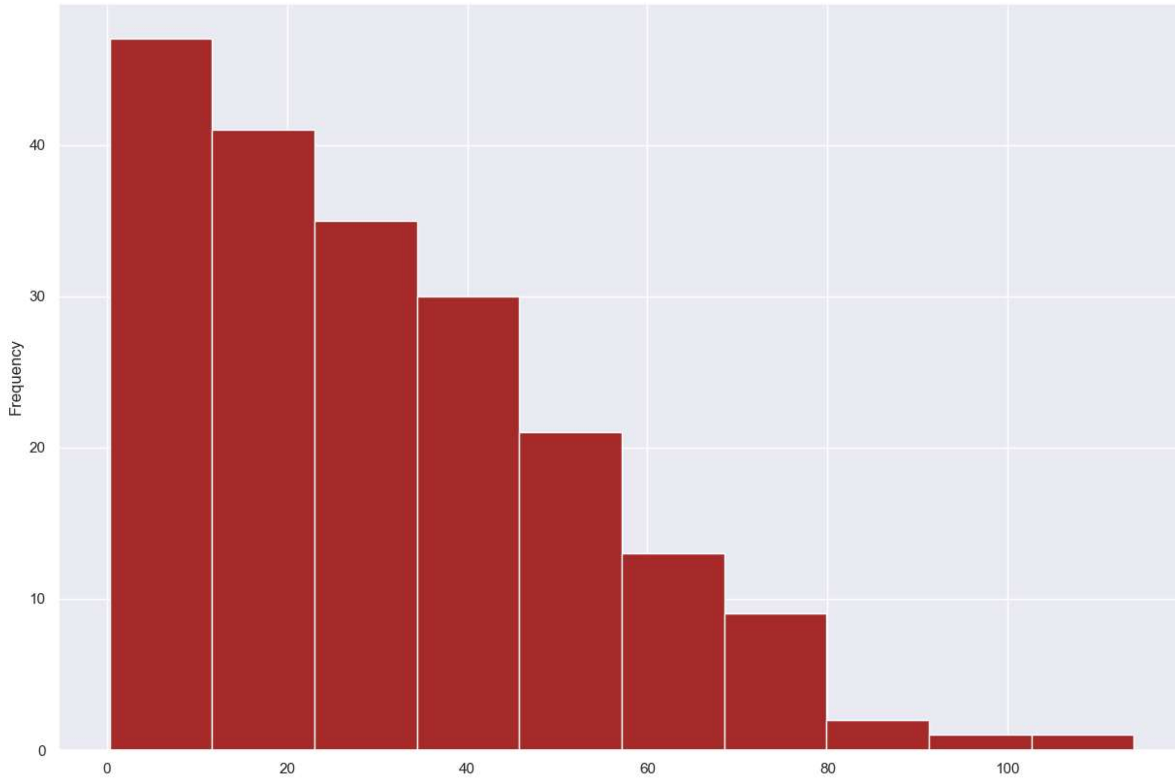
In [10]:
```
plt.figure(figsize=(15,10))
data['TV'].plot(kind='hist',color='brown')
plt.show()
```



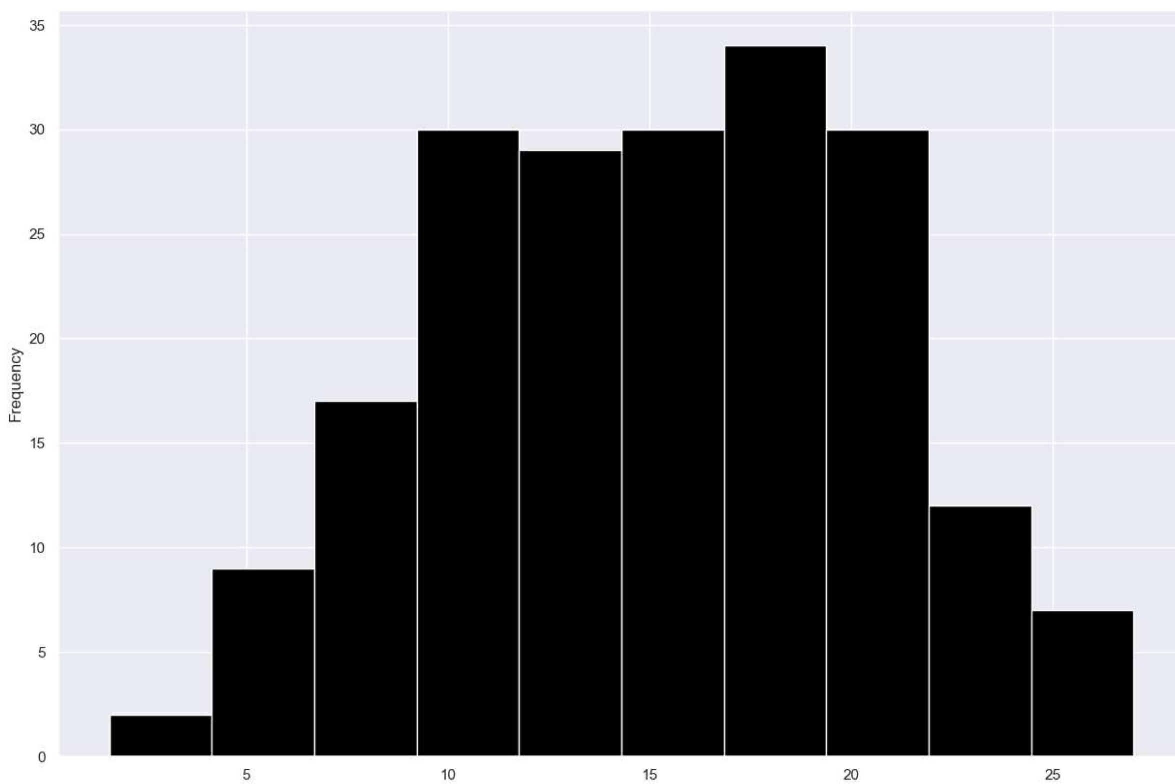
In [11]:
```
plt.figure(figsize=(15,10))
data['Radio'].plot(kind='hist',color='brown')
plt.show()
```



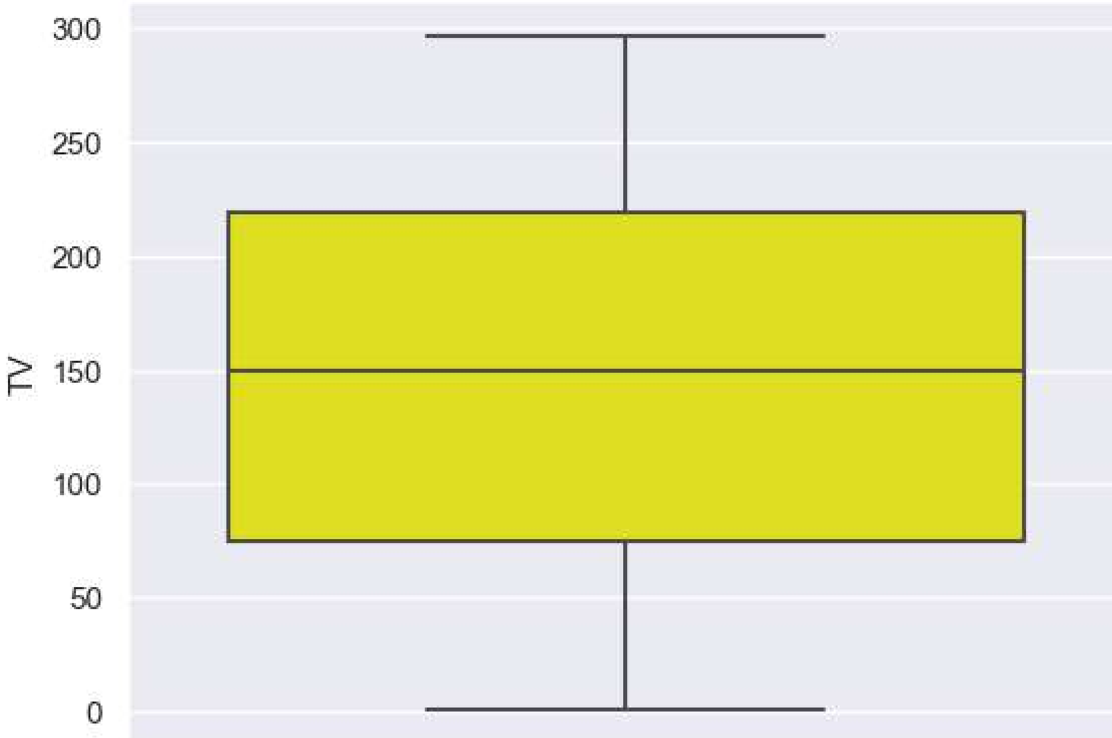
In [12]:
```
plt.figure(figsize=(15,10))
data['Newspaper'].plot(kind='hist',color='brown')
plt.show()
```



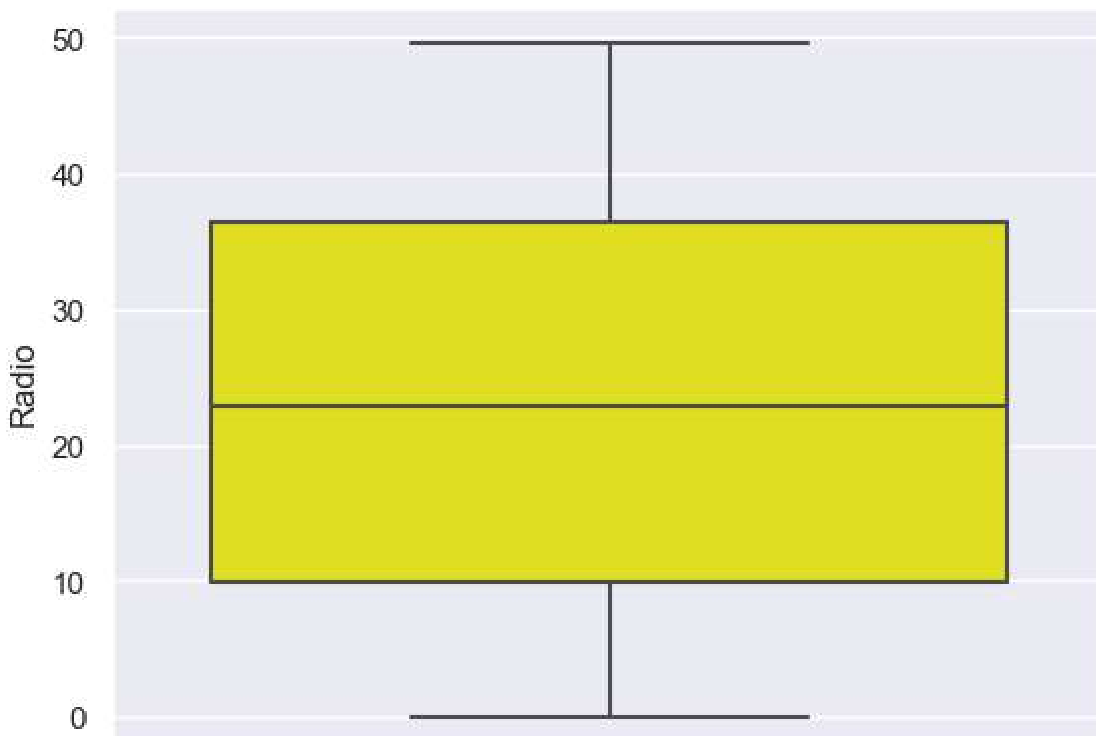
In [13]:
```
plt.figure(figsize=(15,10))
data['Sales'].plot(kind='hist',color='black')
plt.show()
```



In [14]:
```
sns.boxplot(y = 'TV', data=data,color = "yellow")
plt.show()
```

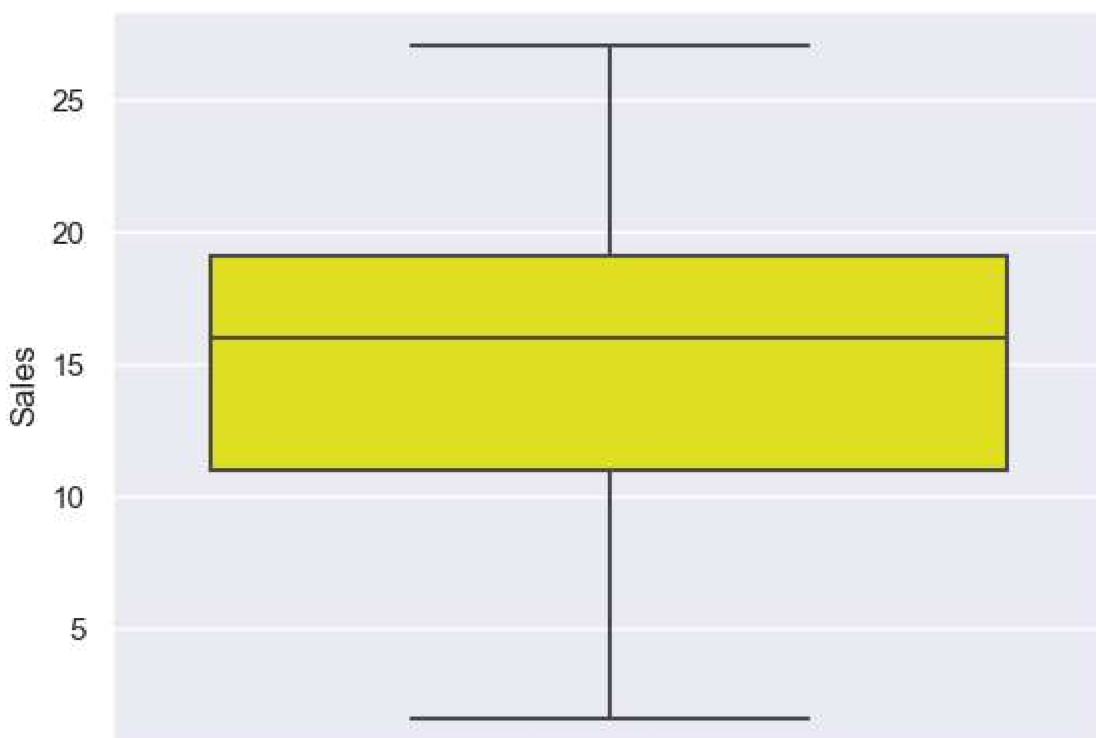


In [15]:
```
sns.boxplot(y = 'Radio', data=data,color = "yellow")
plt.show()
```



In [16]:
```
sns.boxplot(y = 'Newspaper', data=data,color = "yellow")
plt.show()
```

In [17]:
```
sns.boxplot(y = 'Sales', data=data,color = "yellow")
plt.show()
```

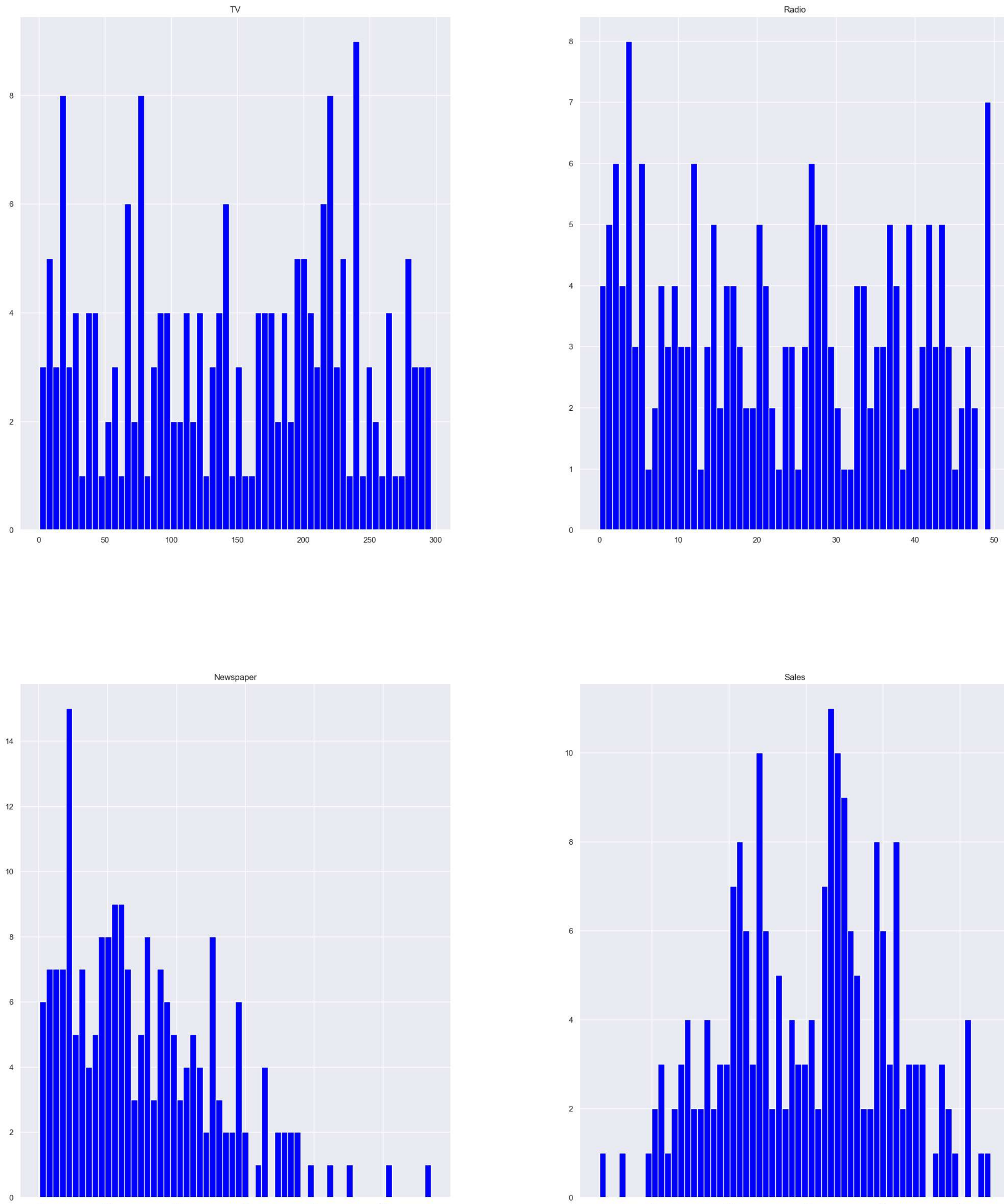


Newspapers have outliers

In [18]:
```
data['Newspaper'] = data['Newspaper'].fillna(data['Newspaper'].median())
```

plot Histogram

In [19]:
```
data.hist(bins=60,figsize=(25,30),color='blue')
plt.show()
```



In [20]:
```
for i in data.columns:
    print("*********************************************",i,
          "*********************************************")
    print()
    print(set(data[i].tolist()))
    print()
```

```
********************************************* TV ***************************
*******************

{0.7, 4.1, 5.4, 7.3, 8.7, 8.6, 7.8, 8.4, 11.7, 13.2, 13.1, 16.9, 17.2, 18.8,
19.4, 17.9, 19.6, 18.7, 23.8, 25.1, 26.8, 27.5, 28.6, 25.0, 25.6, 31.5, 36.9,
38.0, 39.5, 38.2, 43.1, 44.5, 43.0, 44.7, 48.3, 50.0, 53.5, 56.2, 57.5, 215.
4, 59.6, 62.3, 66.1, 67.8, 66.9, 69.2, 70.6, 69.0, 68.4, 73.4, 74.7, 75.3, 7
6.4, 76.3, 78.2, 75.1, 80.2, 75.5, 85.7, 87.2, 88.3, 89.7, 90.4, 93.9, 94.2,
95.7, 96.2, 97.5, 97.2, 100.4, 102.7, 104.6, 107.4, 109.8, 110.7, 112.9, 116.
0, 117.2, 120.5, 120.2, 121.0, 123.1, 125.7, 129.4, 131.1, 131.7, 134.3, 135.
2, 136.2, 137.9, 139.3, 139.2, 141.3, 142.9, 140.3, 139.5, 147.3, 149.8, 149.
7, 151.5, 156.6, 163.3, 163.5, 164.5, 165.6, 166.8, 168.4, 170.2, 171.3, 172.
5, 175.1, 175.7, 177.0, 180.8, 182.6, 184.9, 187.9, 187.8, 188.4, 191.1, 193.
2, 193.7, 195.4, 197.6, 198.9, 199.8, 199.1, 202.5, 204.1, 205.0, 206.9, 206.
8, 209.6, 210.8, 210.7, 213.4, 214.7, 213.5, 216.4, 216.8, 218.4, 217.7, 220.
3, 219.8, 222.4, 220.5, 224.0, 225.8, 218.5, 227.2, 228.3, 228.0, 230.1, 229.
5, 232.1, 234.5, 237.4, 238.2, 239.9, 240.1, 239.3, 239.8, 241.7, 243.2, 248.
8, 248.4, 250.9, 253.8, 255.4, 261.3, 262.9, 262.7, 265.6, 266.9, 265.2, 273.
7, 276.9, 276.7, 280.2, 281.4, 280.7, 283.6, 284.3, 286.0, 287.6, 289.7, 290.
7, 292.9, 293.6, 296.4}

********************************************* Radio ***********************
**********************

{0.8, 1.5, 2.1, 2.6, 3.5, 5.8, 5.1, 7.6, 1.4, 4.1, 10.8, 8.4, 12.6, 9.9, 11.
7, 15.9, 16.9, 16.7, 16.0, 19.6, 20.5, 17.4, 20.0, 23.9, 24.0, 22.3, 26.7, 2
7.7, 27.1, 29.3, 28.3, 25.7, 32.8, 32.9, 33.4, 35.1, 36.6, 37.8, 37.7, 39.3,
39.6, 41.3, 41.5, 43.8, 41.7, 45.9, 46.2, 47.7, 48.9, 49.4, 49.6, 42.7, 43.9,
44.5, 47.8, 46.4, 2.0, 11.0, 49.0, 10.0, 12.0, 14.5, 14.0, 15.5, 17.0, 18.4,
18.1, 20.6, 1.9, 20.9, 20.1, 21.0, 2.4, 2.9, 21.1, 22.5, 3.4, 23.6, 24.6, 25.
5, 25.9, 26.9, 27.5, 28.1, 28.5, 28.9, 29.6, 29.9, 29.5, 4.9, 30.6, 5.4, 31.
6, 0.0, 32.3, 33.0, 33.5, 33.2, 34.3, 34.6, 35.0, 35.4, 35.8, 35.6, 36.5, 36.
3, 36.9, 36.8, 37.6, 38.0, 38.9, 38.6, 13.9, 39.0, 39.7, 40.6, 40.3, 15.4, 2.
3, 41.1, 42.8, 42.3, 4.3, 1.3, 42.0, 43.7, 43.0, 43.5, 45.1, 7.3, 7.8, 46.8,
47.0, 0.4, 8.2, 9.3, 11.8, 14.3, 14.8, 14.7, 15.8, 3.7, 17.2, 5.7, 5.2, 19.2,
7.7, 20.3, 21.7, 21.3, 23.3, 25.8, 0.3, 26.8, 27.2, 28.8, 28.7, 30.2, 7.1, 1.
6, 8.6, 9.6, 3.1, 10.1, 10.6, 11.6, 12.1}

********************************************* Newspaper *******************
**************************

{1.0, 1.8, 3.6, 4.0, 5.0, 2.2, 7.2, 7.4, 8.5, 9.3, 11.6, 12.6, 8.4, 10.2, 15.
9, 16.6, 9.4, 18.3, 19.1, 19.5, 21.2, 22.9, 23.5, 24.2, 18.5, 26.2, 26.4, 28.
9, 21.4, 27.3, 30.0, 31.6, 32.0, 31.5, 34.6, 35.1, 35.7, 36.8, 38.6, 38.7, 4
0.8, 39.6, 41.4, 43.2, 43.3, 45.1, 46.0, 45.7, 49.6, 49.9, 51.4, 52.9, 53.4,
54.7, 55.8, 11.0, 51.2, 58.5, 58.4, 58.7, 60.0, 59.0, 63.2, 56.5, 65.9, 65.7,
65.6, 59.7, 69.3, 69.2, 71.8, 72.3, 73.4, 74.2, 75.0, 75.6, 79.2, 16.0, 84.8,
17.9, 17.0, 17.6, 89.4, 18.4, 19.4, 19.6, 100.9, 20.5, 20.6, 21.6, 2.4, 22.0,
114.0, 23.1, 23.4, 25.6, 25.9, 5.5, 26.6, 27.4, 5.9, 5.4, 6.0, 6.4, 32.5, 33.
8, 33.0, 34.5, 34.4, 35.6, 35.2, 10.9, 36.9, 11.9, 37.7, 37.9, 12.4, 12.9, 3
7.0, 38.9, 41.8, 43.1, 5.3, 43.0, 5.8, 44.3, 45.9, 45.2, 9.0, 46.2, 9.5, 47.
4, 48.7, 49.8, 49.3, 50.4, 50.6, 50.5, 0.9, 52.7, 57.6, 8.7, 8.3, 9.2, 10.7,
1.7, 12.8, 13.8, 14.2, 14.8, 66.2, 3.2, 3.7, 5.7, 18.2, 19.3, 20.7, 20.3, 22.
3, 23.2, 23.7, 24.3, 0.3, 27.2, 29.7, 30.7, 31.7, 31.3, 8.1, 2.1, 13.1, 15.6}

********************************************* Sales ***********************
*********************
```

```
{1.6, 3.2, 4.8, 5.6, 5.5, 7.2, 8.5, 9.2, 10.4, 11.3, 11.8, 12.0, 13.2, 12.6,
15.6, 16.5, 17.9, 17.4, 13.7, 19.0, 22.1, 22.4, 24.4, 18.0, 17.5, 20.5, 20.9,
21.4, 25.4, 23.2, 23.7, 24.2, 27.0, 26.2, 7.0, 8.0, 9.5, 10.5, 11.0, 11.5, 1
2.5, 14.0, 15.0, 15.5, 16.6, 16.1, 16.4, 16.0, 16.9, 17.0, 17.1, 17.6, 18.9,
18.4, 19.4, 19.6, 19.9, 20.1, 20.6, 20.0, 21.5, 22.6, 25.5, 5.9, 6.9, 8.4, 9.
4, 10.9, 11.9, 12.4, 12.9, 13.4, 15.9, 5.3, 7.3, 8.8, 8.7, 9.7, 10.7, 10.8, 1
0.3, 12.2, 12.3, 13.3, 14.7, 14.2, 14.8, 15.2, 15.3, 16.7, 16.8, 17.8, 17.3,
17.2, 17.7, 5.7, 18.3, 18.2, 6.7, 19.8, 19.7, 19.2, 20.7, 20.2, 20.8, 21.2, 2
1.7, 21.8, 22.3, 22.2, 23.8, 24.7, 6.6, 7.6, 8.1, 9.6, 10.1, 10.6, 11.6, 13.
6, 14.6}
```

Feature scaling

```
split the data into dependent and independent variables
```

In [21]:
```
x = data.iloc[:,0:-1]
y = data['Sales']
```

In [22]:
```
x.head()
```

Out[22]:

	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 [23]:
```
y.head()
```

Out[23]:
```
0    22.1
1    10.4
2    12.0
3    16.5
4    17.9
Name: Sales, dtype: float64
```

Data preprocessing

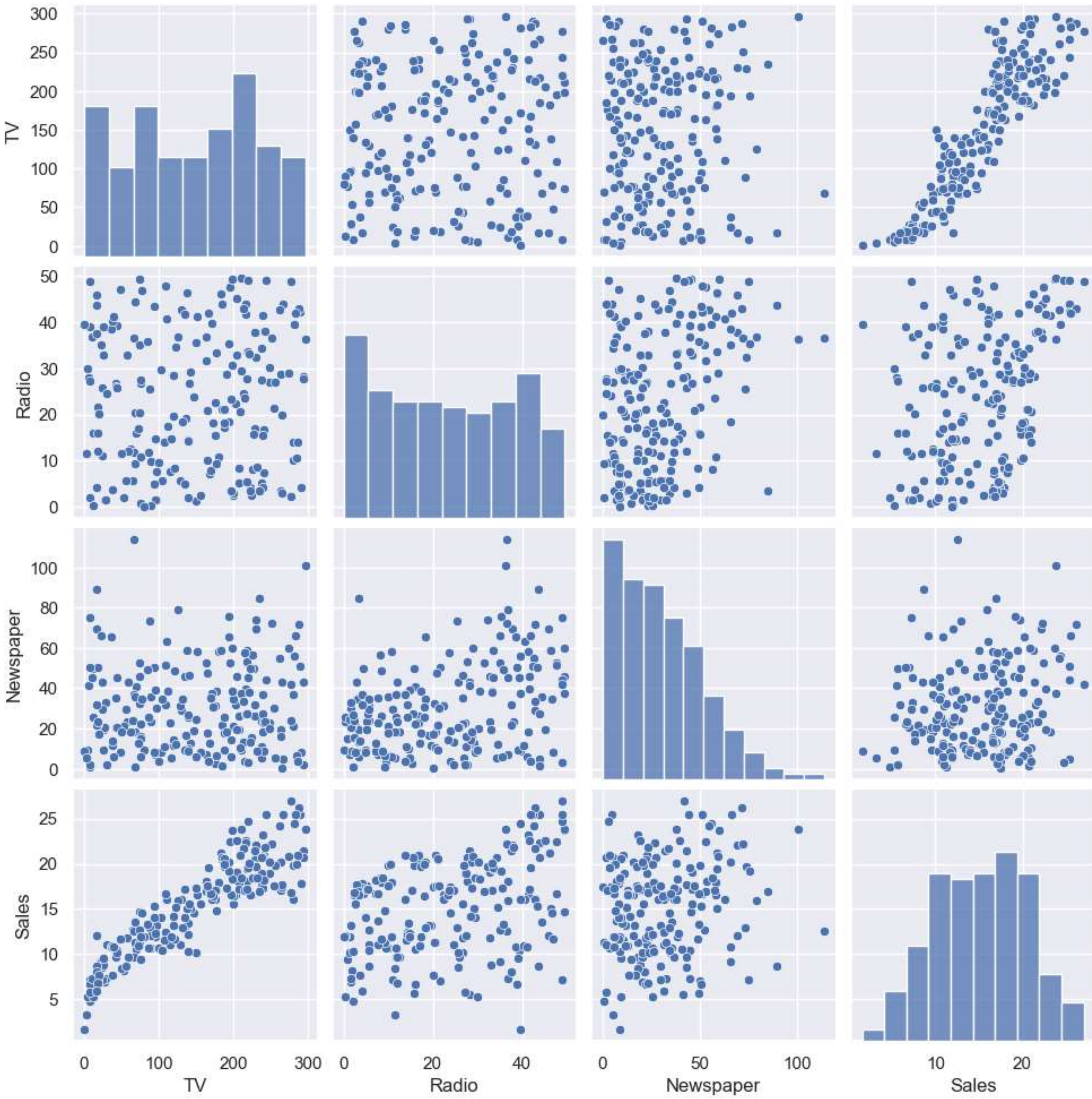
In [24]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
sc_x = sc.fit_transform(x)
pd.DataFrame(sc_x)
```

Out[24]:

	0	1	2
0	0.969852	0.981522	1.778945
1	-1.197376	1.082808	0.669579
2	-1.516155	1.528463	1.783549
3	0.052050	1.217855	1.286405
4	0.394182	-0.841614	1.281802
...	...	...	...
195	-1.270941	-1.321031	-0.771217
196	-0.617035	-1.240003	-1.033598
197	0.349810	-0.942899	-1.111852
198	1.594565	1.265121	1.640850
199	0.993206	-0.990165	-1.005979

200 rows × 3 columns

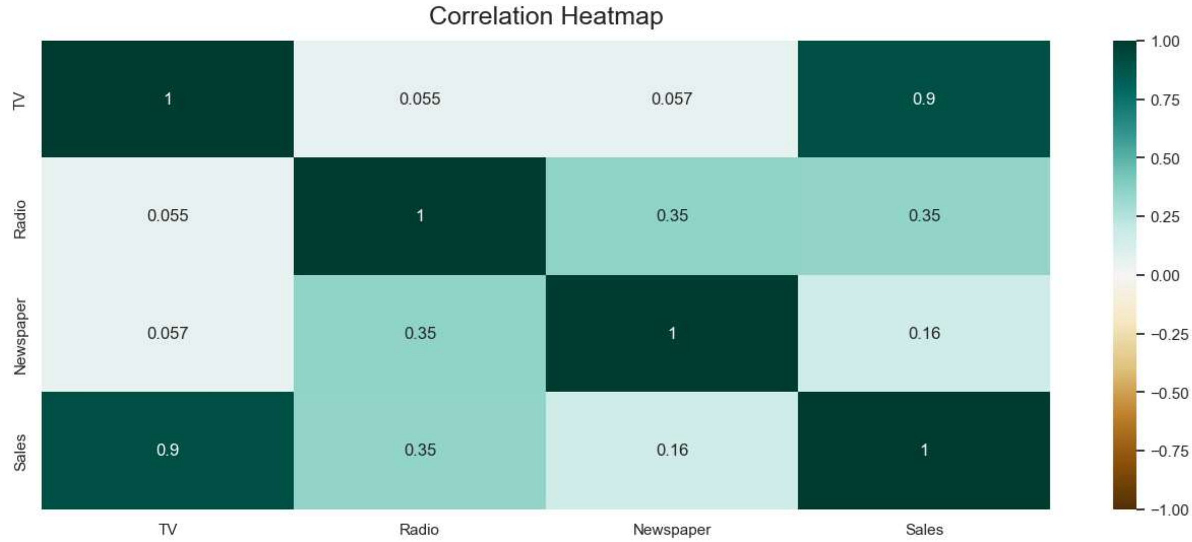
In [50]:
```
sns.pairplot(data, hue=None, palette='YlGnBu')
plt.show()
```



Finding correlation

In [25]:

```
plt.figure(figsize=(16, 6))
heatmap = sns.heatmap(data.corr(), vmin=-1, vmax=1, annot=True, cmap='BrBG')
heatmap.set_title('Correlation Heatmap', fontdict={'fontsize':18}, pad=12);
# save heatmap as .png file
# dpi - sets the resolution of the saved image in dots/inches
# bbox_inches - when set to 'tight' - does not allow the labels to be cropped
plt.savefig('heatmap.png', dpi=300, bbox_inches='tight')
```



VIF - Variance Inflation Factor - to check multicollinearity

In [26]:

```
variable = sc_x
variable.shape
```

Out[26]:

```
(200, 3)
```

In [27]:

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
variable = sc_x

vif = pd.DataFrame()

vif['Variance Inflation Factor'] = [variance_inflation_factor(variable, i ) fo

vif['Features'] = x.columns
```

In [28]:

```
vif
```

Out[28]:

	Variance Inflation Factor	Features
0	1.004611	TV
1	1.144952	Radio
2	1.145187	Newspaper

Split the data in train and test

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=0.25, rand
print(x_train.shape, x_test.shape, y_train.shape, y_test.shape)
```

```
(150, 3) (50, 3) (150,) (50,)
```

Apply Models

In [30]:
```
from sklearn.linear_model import LinearRegression
from statsmodels.regression.linear_model import OLS
import statsmodels.regression.linear_model as smf
from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge
from sklearn.linear_model import ElasticNet
from sklearn.metrics import confusion_matrix, classification_report, accuracy_
```

Linear Regression

In [31]:
```
#from sklearn.linear_model import LinearRegression
ln = LinearRegression()
ln.fit(x_train, y_train)

# Predict sales price by using ln model with test dataset
y_pred_price = ln.predict(x_test)
y_pred_price_train = ln.predict(x_train)

# Validate the actual price of the test data and predicted price
from sklearn.metrics import r2_score
print(r2_score(y_test, y_pred_price))
print(r2_score(y_train, y_pred_price_train))
```

```
0.8949009939756739
0.9037271946786809
```

OLS

In [32]:
```
#from statsmodels.regression.linear_model import OLS
#import statsmodels.regression.linear_model as smf

reg_model = smf.OLS(endog = y_train, exog=x_train).fit()
```

In [33]:
```
reg_model.summary()
```

Out[33]:

OLS Regression Results

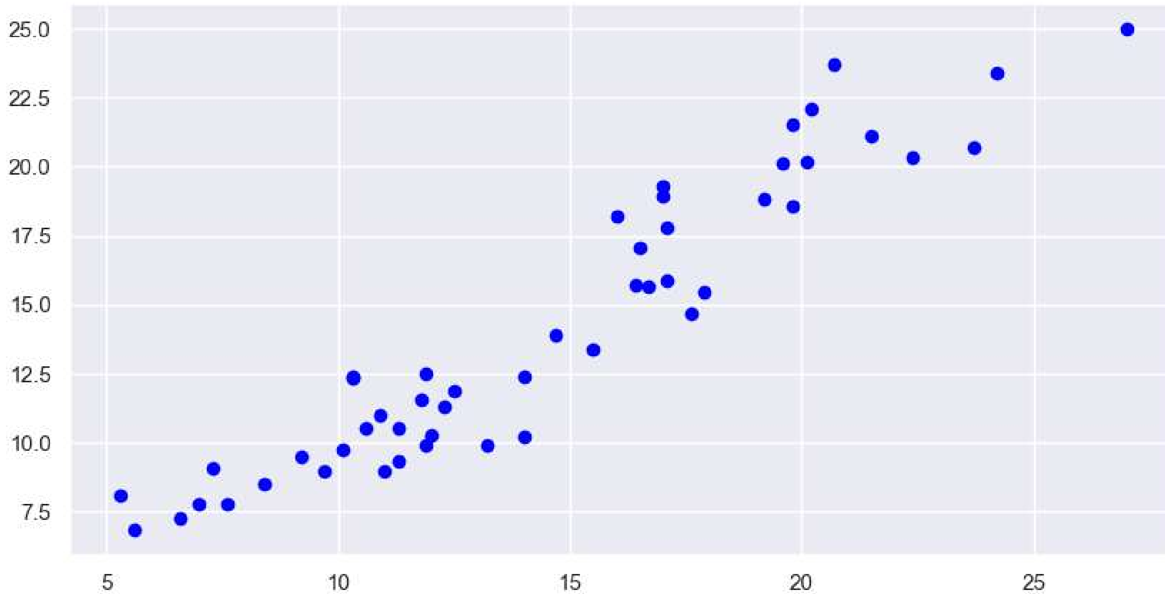
Dep. Variable:	Sales	**R-squared (uncentered):**	0.979
Model:	OLS	**Adj. R-squared (uncentered):**	0.979
Method:	Least Squares	**F-statistic:**	2279.
Date:	Mon, 09 Oct 2023	**Prob (F-statistic):**	5.55e-123
Time:	13:33:45	**Log-Likelihood:**	-341.34
No. Observations:	150	**AIC:**	688.7
Df Residuals:	147	**BIC:**	697.7
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
TV	0.0672	0.002	36.970	0.000	0.064	0.071
Radio	0.1637	0.012	13.197	0.000	0.139	0.188
Newspaper	0.0225	0.009	2.471	0.015	0.005	0.041

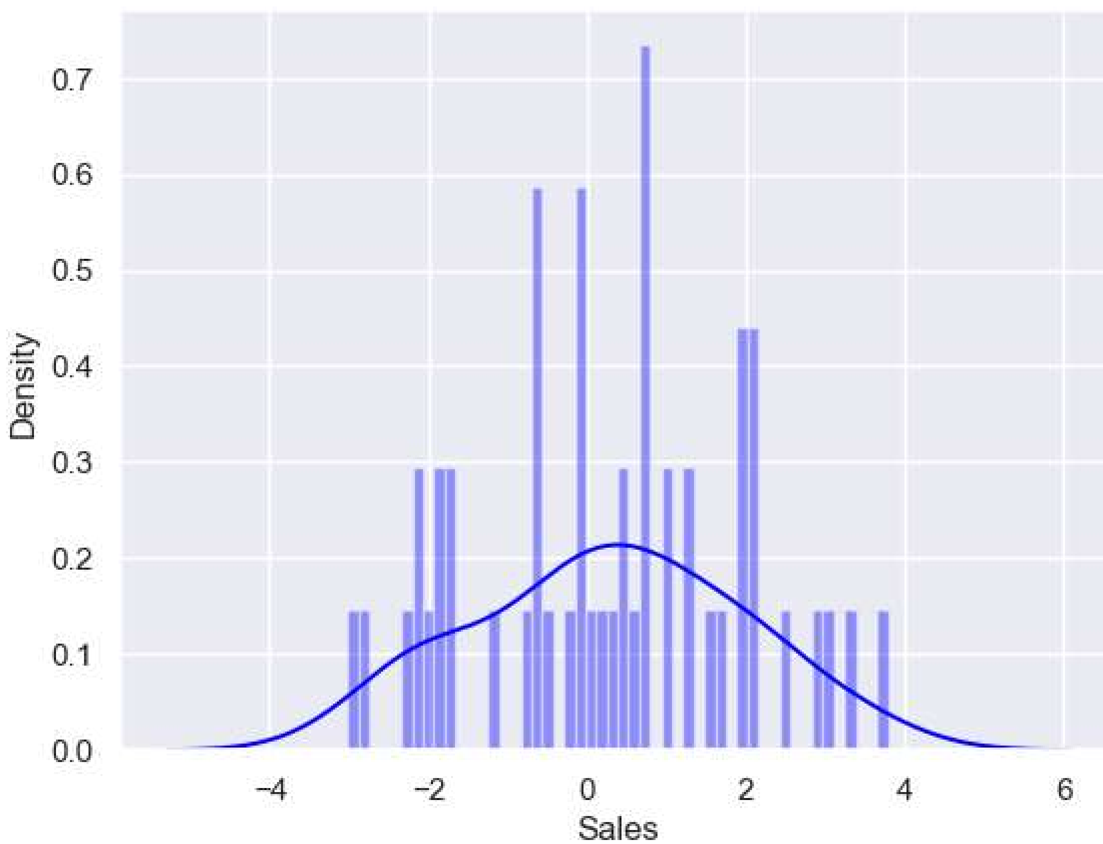
Omnibus:	0.544	**Durbin-Watson:**	2.015
Prob(Omnibus):	0.762	**Jarque-Bera (JB):**	0.257
Skew:	-0.067	**Prob(JB):**	0.879
Kurtosis:	3.153	**Cond. No.**	12.5

Notes:
[1] R^2 is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [34]:
```
# Check linearity
plt.figure(figsize=(10,5))
plt.scatter(y_test, y_pred_price,color='blue')
plt.show()
```



In [35]:
```
# Normality of Residual
sns.distplot((y_test - y_pred_price), bins=50,color='blue')
plt.show()
```



Lasso regularization

In [36]:
```
#from sklearn.linear_model import Lasso
lasso = Lasso(alpha=0.1)
lasso.fit(x_train, y_train)

print("Lasso Model :", (lasso.coef_))

y_pred_train_lasso = lasso.predict(x_train)
y_pred_test_lasso = lasso.predict(x_test)
print()
print("Training Accuracy :", r2_score(y_train, y_pred_train_lasso))
print("Test Accuracy :", r2_score(y_test, y_pred_test_lasso))
```

```
Lasso Model : [ 0.054996    0.10979344 -0.00325905]

Training Accuracy : 0.9037236817047651
Test Accuracy : 0.8951345709147266
```

Ridge Regression

In [37]:
```
#from sklearn.linear_model import Ridge
ridge = Ridge(alpha=0.3)
ridge.fit(x_train, y_train)

print("Ridge Model :", (ridge.coef_))

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

print()
print("Training Accuracy :", r2_score(y_train, y_pred_train_ridge))
print("Test Accuracy :", r2_score(y_test, y_pred_test_ridge))
```

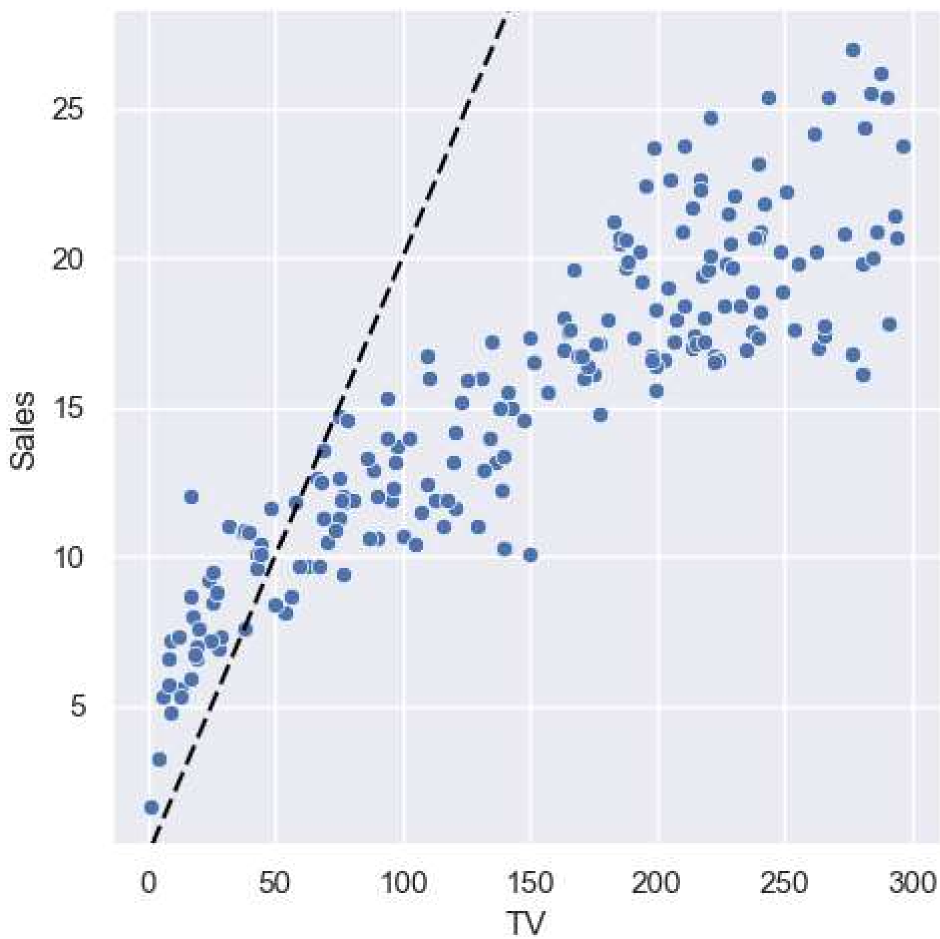
```
Ridge Model : [ 0.05501359  0.11040204 -0.00361249]

Training Accuracy : 0.9037271946696611
Test Accuracy : 0.8949012321308285
```

ElasticNet

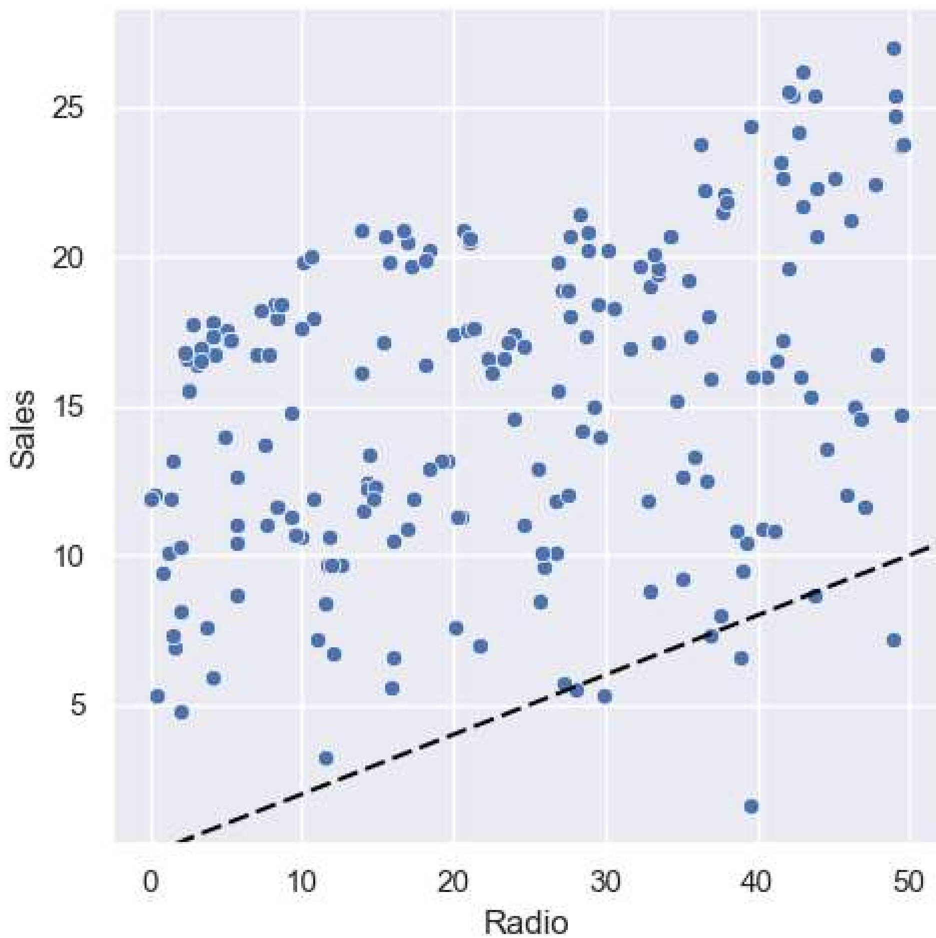
In [38]:
```
#from sklearn.linear_model import ElasticNet
elastic = ElasticNet(alpha=0.3, l1_ratio=0.1)
elastic.fit(x_train, y_train)

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

print()
print("Training Accuracy :", r2_score(y_train, y_pred_train_elastic))
print("Test Accuracy :", r2_score(y_test, y_pred_test_elastic))
```

```
Training Accuracy : 0.9037263134931904
Test Accuracy : 0.8950032174286724
```

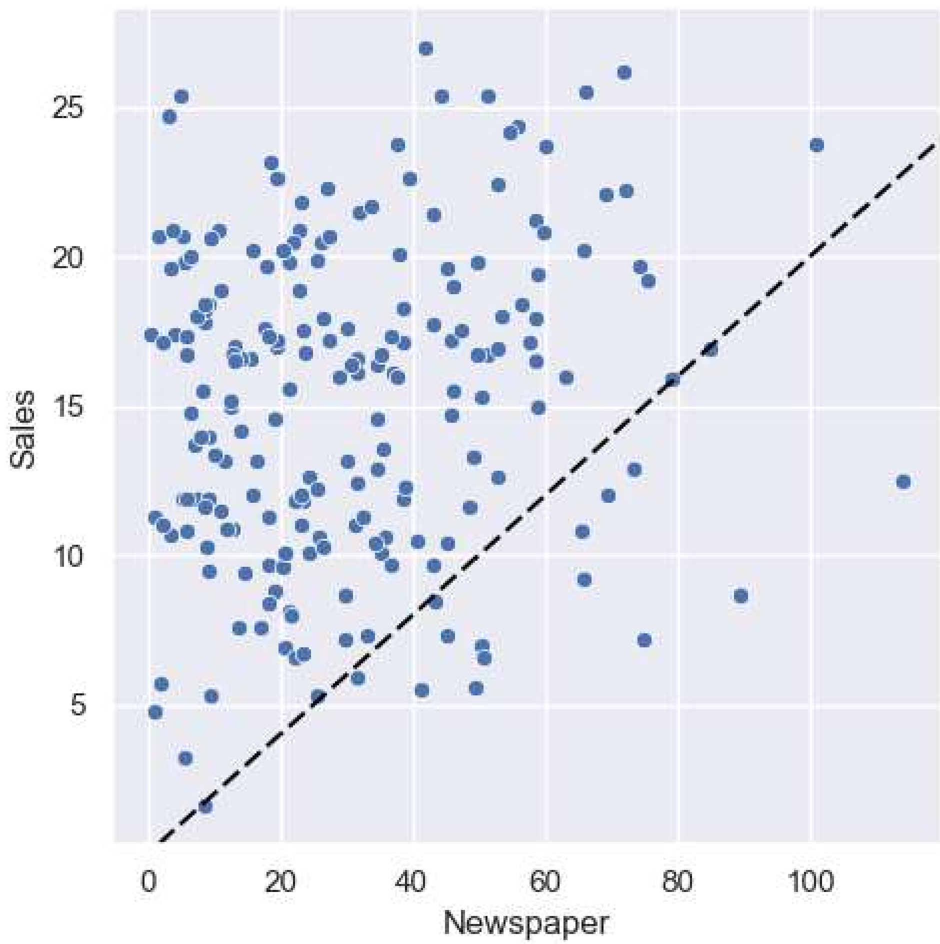
In [40]:
```
g = sns.relplot(data, x="TV", y="Sales")
g.ax.axline(xy1=(10, 2), slope=.2, color="black", dashes=(5, 2))
plt.show()
```



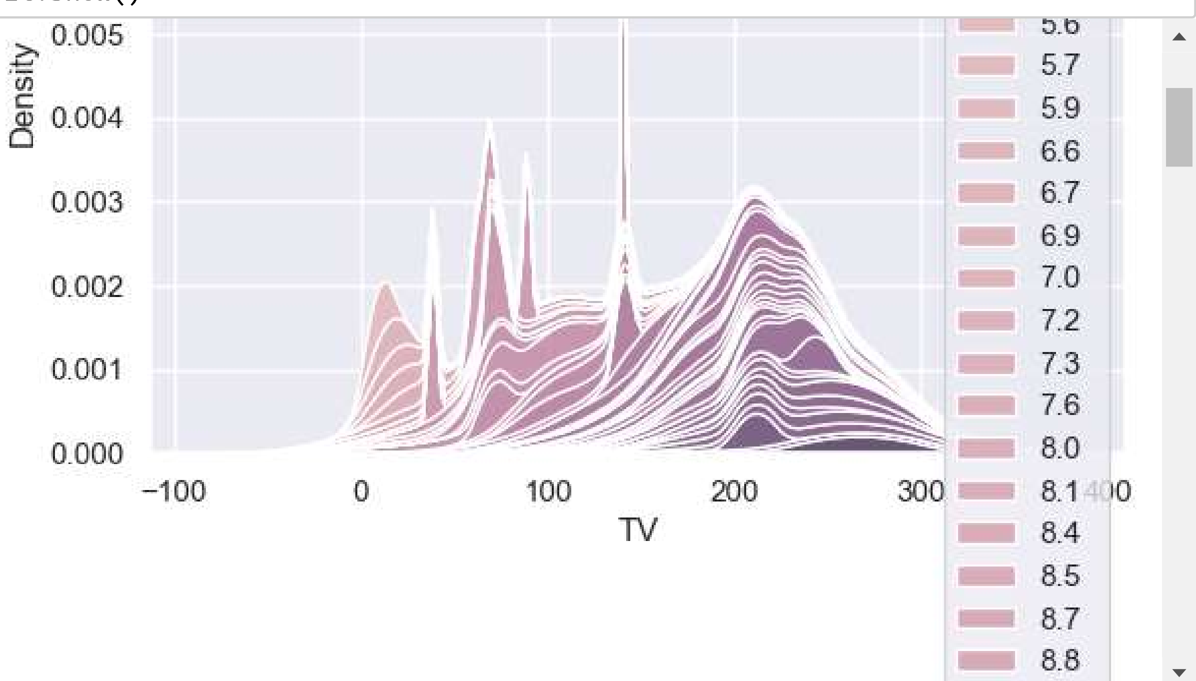
In [41]:
```
g = sns.relplot(data, x="Radio", y="Sales")
g.ax.axline(xy1=(10, 2), slope=.2, color="black", dashes=(5, 2))
plt.show()
```



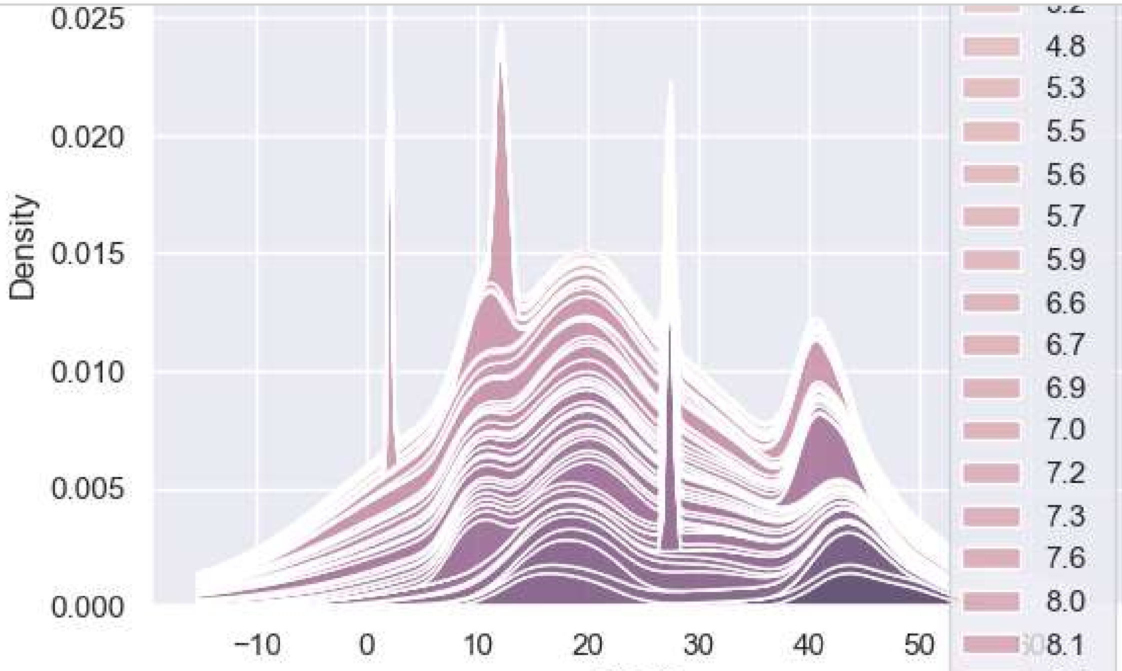
In [42]:
```
g = sns.relplot(data, x="Newspaper", y="Sales")
g.ax.axline(xy1=(10, 2), slope=.2, color="black", dashes=(5, 2))
plt.show()
```



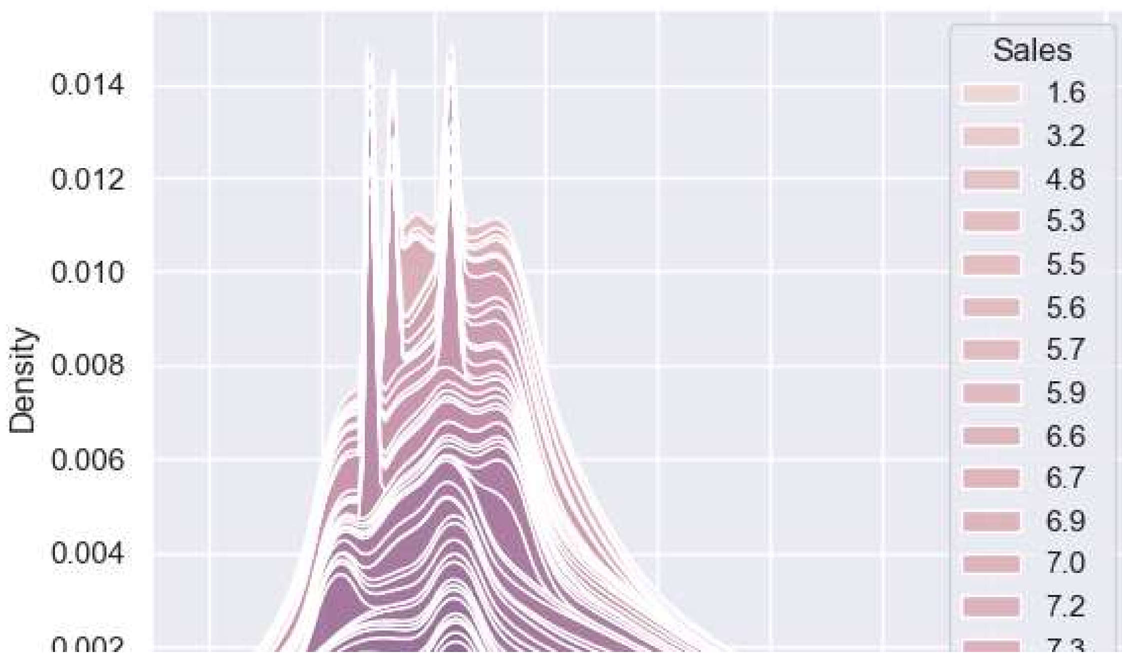
In [43]:
```
sns.kdeplot(data,x="TV", hue="Sales", multiple="stack")
plt.show()
```



In [44]:
```
sns.kdeplot(data, x="Radio", hue="Sales", multiple="stack")
plt.show()
```



In [45]:
```
sns.kdeplot(data, x="Newspaper", hue="Sales", multiple="stack")
plt.show()
```



conclusion

In [ ]:
```
TV is the strongest predictor, but both radio and TV combined are the best pre
```