In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [19]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [8]:
```
data=pd.DataFrame(pd.read_csv("advertising.csv"))
data
```

Out[8]:

| | 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 |
| **...** | ... | ... | ... | ... |
| **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 |

200 rows × 4 columns

In [9]:
```
data.head()
```

Out[9]:

| | 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 [11]:
```
data.head(11)
```

Out[11]:

| | 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 |
| **5** | 8.7 | 48.9 | 75.0 | 7.2 |
| **6** | 57.5 | 32.8 | 23.5 | 11.8 |
| **7** | 120.2 | 19.6 | 11.6 | 13.2 |
| **8** | 8.6 | 2.1 | 1.0 | 4.8 |
| **9** | 199.8 | 2.6 | 21.2 | 15.6 |
| **10** | 66.1 | 5.8 | 24.2 | 12.6 |

In [12]:

```
data.shape
```

Out[12]:

```
(200, 4)
```

In [13]:

```
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 [14]:

```
data.describe()
```

Out[14]:

| | 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 |

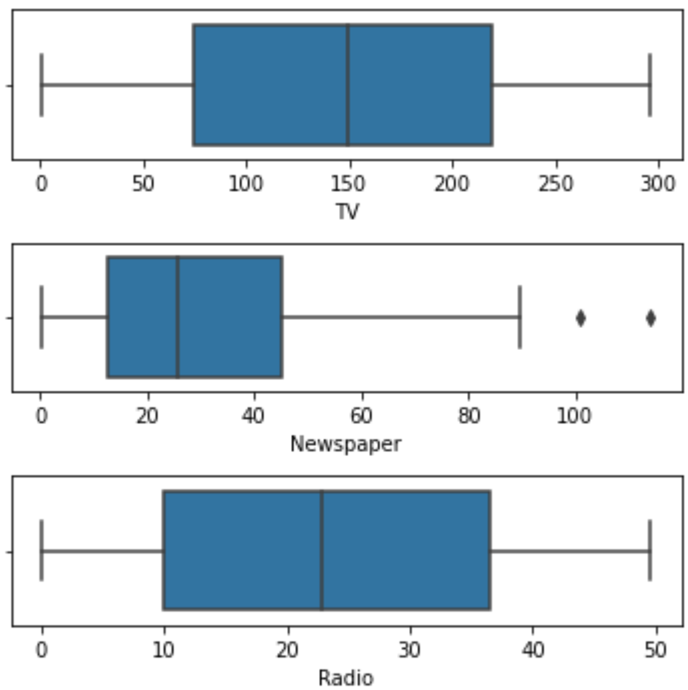
In [16]:
```
data.isnull().sum()
```

Out[16]:
```
TV           0
Radio        0
Newspaper    0
Sales        0
dtype: int64
```

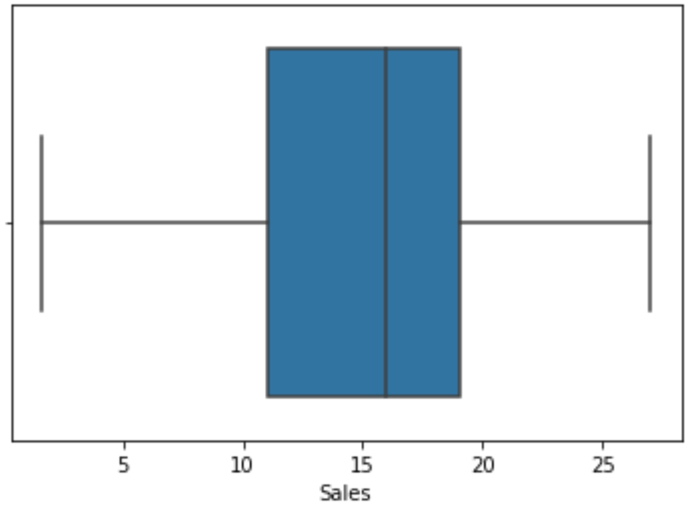
In [17]:
```
data.isnull().sum()*100/data.shape[0]
```

Out[17]:
```
TV           0.0
Radio        0.0
Newspaper    0.0
Sales        0.0
dtype: float64
```

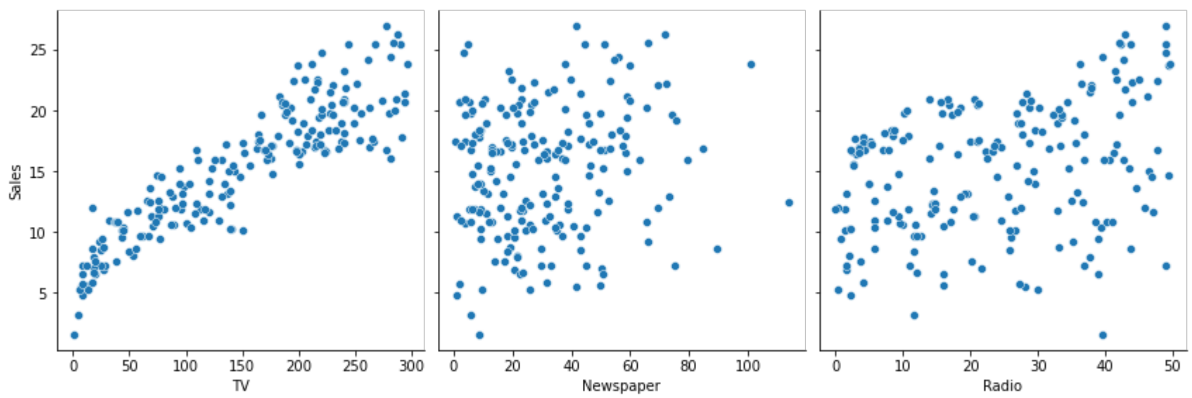
In [20]:
```
fig, axs = plt.subplots(3, figsize = (5,5))
plt1 = sns.boxplot(data['TV'], ax = axs[0])
plt2 = sns.boxplot(data['Newspaper'], ax = axs[1])
plt3 = sns.boxplot(data['Radio'], ax = axs[2])
plt.tight_layout()
```



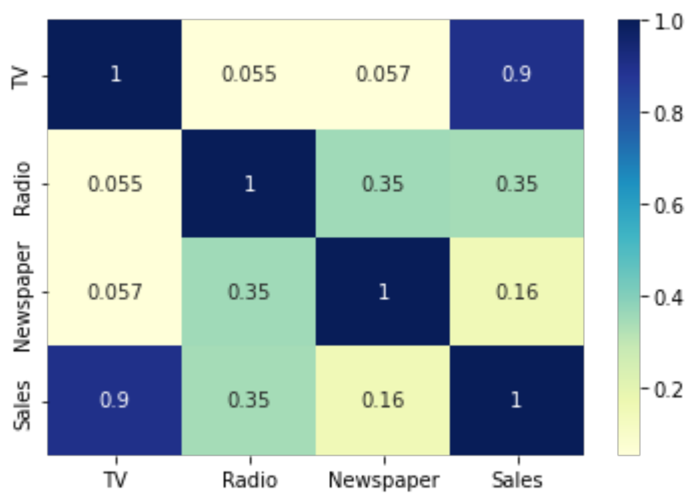

In [22]:
```
sns.boxplot(data['Sales'])
plt.show()
```




In [31]:
```
#Let's see how Sales are related with other variables using scatter plot.
sns.pairplot(data, x_vars=['TV', 'Newspaper', 'Radio'], y_vars='Sales', height=4, a
plt.show()
```




In [33]:
```
sns.heatmap(data.corr(), cmap="YlGnBu", annot = True)
plt.show()
#As is visible from the pairplot and the heatmap, the variable TV seems to be most
#So let's go ahead and perform simple linear regression using TV as our feature var
```




In [34]:
```
X = data['TV']
y = data['Sales']
```

In [35]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size = 0.7, test_si
```

In [37]:
```
X_train.head()
```

Out[37]:
```
74     213.4
3      151.5
185    205.0
26     142.9
90     134.3
Name: TV, dtype: float64
```

In [38]:
```
y_train.head()
```

Out[38]:
```
74     17.0
3      16.5
185    22.6
26     15.0
90     14.0
Name: Sales, dtype: float64
```

In [39]:
```
import statsmodels.api as sm
```

In [ ]:
```
#By default, the statsmodels library fits a line on the dataset which passes throug
#But in order to have an intercept, you need to manually use the add_constant attri
#And once you've added the constant to your X_train dataset, you can go ahead and f
#line using the OLS (Ordinary Least Squares) attribute of statsmodels as shown belo
```

In [42]:
```
# Add a constant to get an intercept
X_train_sm = sm.add_constant(X_train)

# Fit the resgression line using 'OLS'
lr = sm.OLS(y_train, X_train_sm).fit()
```

In [43]:
```
# Print the parameters, i.e. the intercept and the slope of the regression line fit
lr.params
```

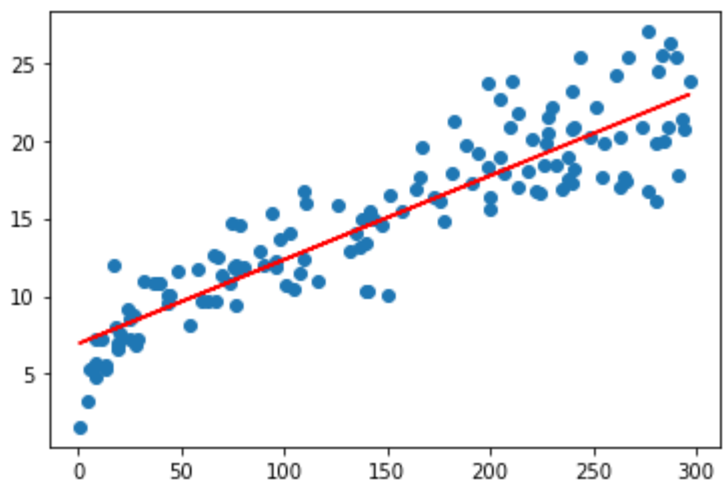
Out[43]:
```
const    6.948683
TV       0.054546
dtype: float64
```

In [ ]:
```

```

In [44]:
```
# Performing a summary operation lists out all the different parameters of the regr
print(lr.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  Sales   R-squared:                       0.816
Model:                            OLS   Adj. R-squared:                  0.814
Method:                 Least Squares   F-statistic:                     611.2
Date:                Mon, 06 Mar 2023   Prob (F-statistic):           1.52e-52
Time:                        10:50:37   Log-Likelihood:                -321.12
No. Observations:                 140   AIC:                             646.2
Df Residuals:                     138   BIC:                             652.1
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          6.9487      0.385     18.068      0.000       6.188       7.709
TV             0.0545      0.002     24.722      0.000       0.050       0.059
==============================================================================
Omnibus:                        0.027   Durbin-Watson:                   2.196
Prob(Omnibus):                  0.987   Jarque-Bera (JB):                0.150
Skew:                          -0.006   Prob(JB):                        0.928
Kurtosis:                       2.840   Cond. No.                         328.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly sp
ecified.
```

In [49]:
```
plt.scatter(X_train, y_train)
plt.plot(X_train, 6.948 + 0.054*X_train, 'r')
plt.show()
```




In [50]:
```
# Add a constant to X_test
X_test_sm = sm.add_constant(X_test)

# Predict the y values corresponding to X_test_sm
y_pred = lr.predict(X_test_sm)
```

In [51]:
```
y_pred.head()
```

Out[51]:
```
126     7.374140
104    19.941482
99     14.323269
92     18.823294
111    20.132392
dtype: float64
```

In [52]:
```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
```

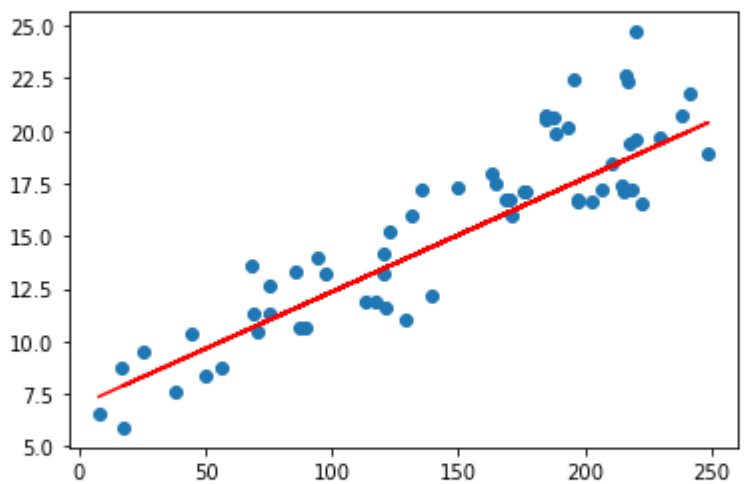
In [53]:
```
#Returns the mean squared error; we'll take a square root
np.sqrt(mean_squared_error(y_test, y_pred))
```

Out[53]:
```
2.019296008966233
```

In [54]:
```
r_squared = r2_score(y_test, y_pred)
r_squared
```

Out[54]:
```
0.7921031601245658
```

In [55]:
```
plt.scatter(X_test, y_test)
plt.plot(X_test, 6.948 + 0.054 * X_test, 'r')
plt.show()
```




In [ ]: