Sales Prediction

The aim is to build a model which predicts sales based on the money spent on different platforms such as TV, radio, and newspaper for marketing.

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

In [2]:
```
#Reading the dataset
dataset = pd.read_csv("advertising.csv")
```

In [3]:
```
dataset.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

Data Pre-Processing

In [4]:
```
dataset.shape
```

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

1. Checking for missing values

In [5]:
```
dataset.isna().sum()
```

Out[5]:
```
TV           0
Radio        0
Newspaper    0
Sales        0
dtype: int64
```

Conclusion: The dataset does not have missing values

2. Checking for duplicate rows

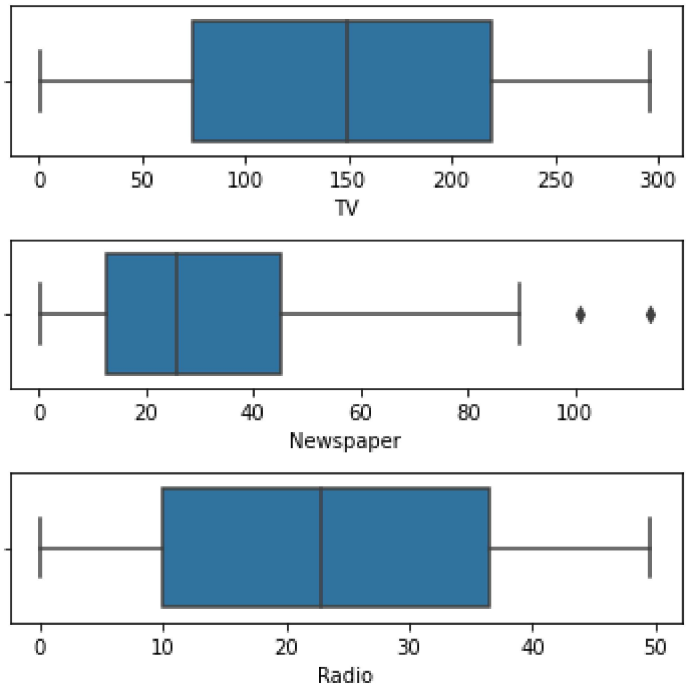
In [34]:
```
dataset.duplicated().any()
```

Out[34]:
```
False
```

Conclusion: There are no duplicate rows present in the dataset

3. Checking for outliers

In [35]:
```
fig, axs = plt.subplots(3, figsize = (5,5))
plt1 = sns.boxplot(dataset['TV'], ax = axs[0])
plt2 = sns.boxplot(dataset['Newspaper'], ax = axs[1])
plt3 = sns.boxplot(dataset['Radio'], ax = axs[2])
plt.tight_layout()
```

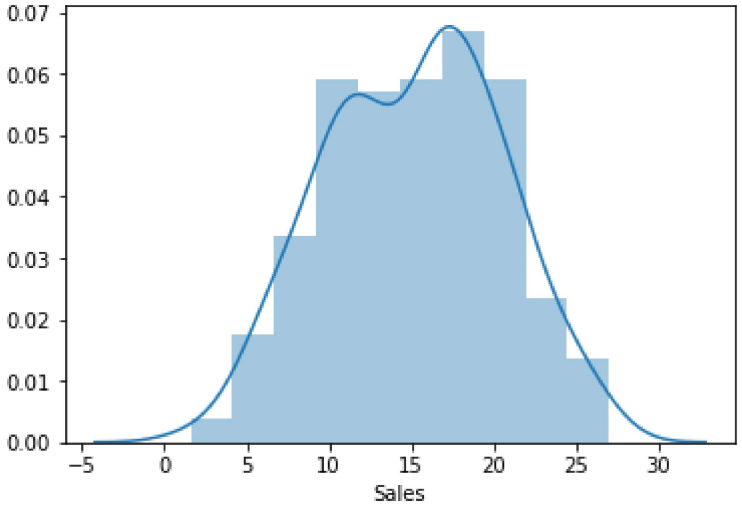


Conclusion: There are not that extreme values present in the dataset

Exploratory Data Analysis

1. Distribution of the target variable

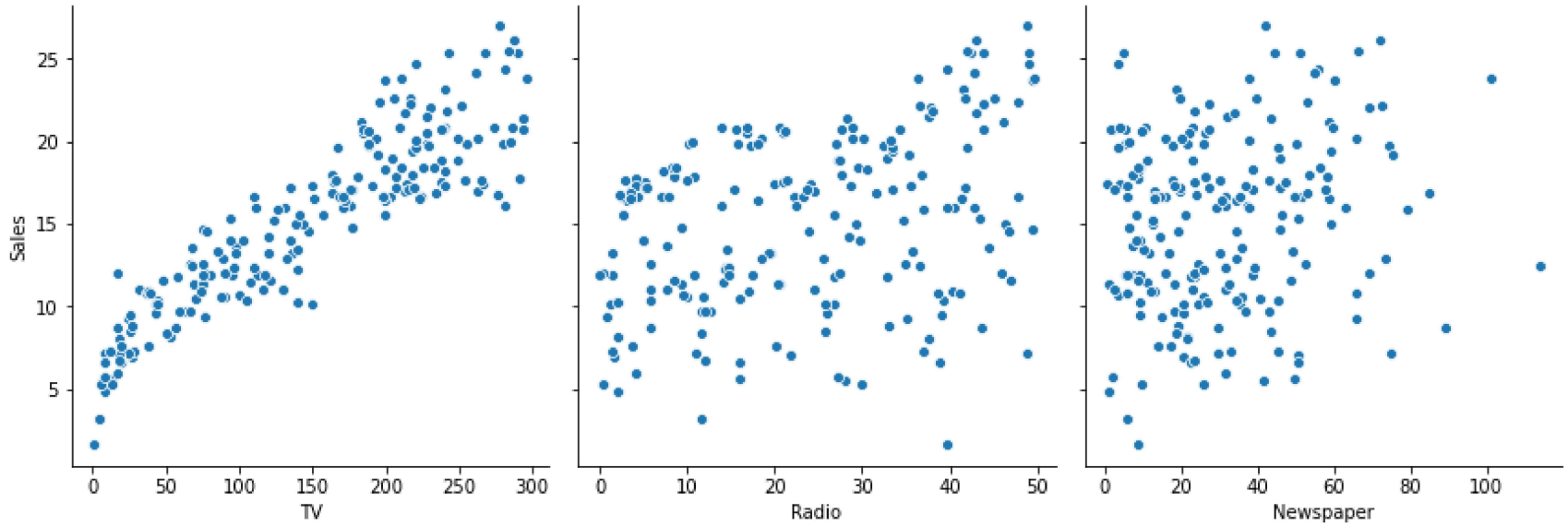
In [8]:
```
sns.distplot(dataset['Sales']);
```



Conclusion: It is normally distributed

2. How Sales are related with other variables

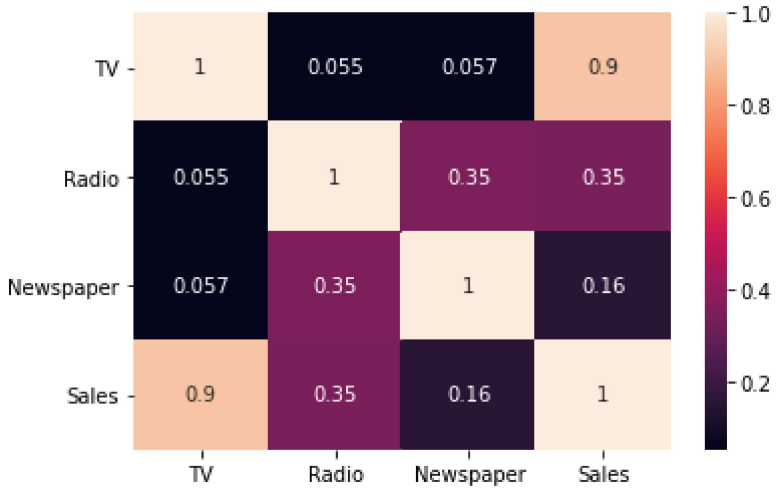
In [36]:
```
sns.pairplot(dataset, x_vars=['TV', 'Radio', 'Newspaper'], y_vars='Sales', height=4, aspect=1, kind='scatter')
plt.show()
```



Conclusion: TV is strongly, positively, linearly correlated with the target variable. Bu the Newspaper feature seems to be uncorrelated

3. Heatmap

In [10]:
```
sns.heatmap(dataset.corr(), annot = True)
plt.show()
```



Conclusion: TV seems to be most correlated with Sales as 0.9 is very close to 1

Model Building

Linear Regression is a useful tool for predicting a quantitative response.

Prediction using:

1. Simple Linear Regression
2. Multiple Linear Regression

1. Simple Linear Regression

Simple linear regression has only one x and one y variable. It is an approach for predicting a quantitative response using a single feature.

It establishes the relationship between two variables using a straight line. Linear regression attempts to draw a line that comes closest to the data by finding the slope and intercept that define the line and minimize regression errors.

Formula: Y = β0 + β1X + e

```
Y = Dependent variable / Target variable
β0 = Intercept of the regression line
β1 = Slope of the regression lime which tells whether the line is increasing or decreasing
X = Independent variable / Predictor variable
e = Error
```

Equation: Sales = β0 + β1X + TV

In [37]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
```

In [38]:
```
#Setting the value for X and Y
x = dataset[['TV']]
y = dataset['Sales']
```

In [39]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.3, random_state = 100)
```

In [40]:
```
slr= LinearRegression()
slr.fit(x_train, y_train)
```

Out[40]:
```
LinearRegression()
```

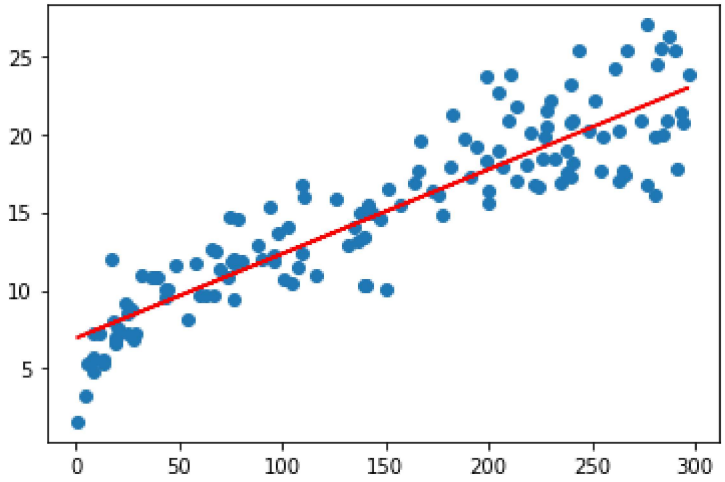
In [44]:
```
#Printing the model coefficients
print('Intercept: ', slr.intercept_)
print('Coefficient:', slr.coef_)
```

```
Intercept:  6.9486832000013585
Coefficient: [0.05454575]
```

In [16]:
```
print('Regression Equation: Sales = 6.948 + 0.054 * TV')
```

```
Regression Equation: Sales = 6.948 + 0.054 * TV
```

In [45]:
```
#Line of best fit
plt.scatter(x_train, y_train)
plt.plot(x_train, 6.948 + 0.054*x_train, 'r')
plt.show()
```



In [46]:
```
#Prediction of Test and Training set result
y_pred_slr= slr.predict(x_test)
x_pred_slr= slr.predict(x_train)
```

In [47]:
```
print("Prediction for test set: {}".format(y_pred_slr))
```

```
Prediction for test set: [ 7.37414007 19.94148154 14.32326899 18.82329361 20.13239168 18.2287449
 14.54145201 17.72692398 18.75238413 18.77420243 13.34144544 19.46693349
 10.01415451 17.1923756  11.70507285 12.08689312 15.11418241 16.23237035
 15.8669138  13.1068987  18.65965635 14.00690363 17.60692332 16.60328147
 17.03419291 18.96511257 18.93783969 11.05597839 17.03419291 13.66326538
 10.6796127  10.71234015 13.5487193  17.22510305  9.67597085 13.52144643
 12.25053038 16.13418799 19.07965865 17.48692266 18.69783838 16.53237199
 15.92145955 18.86693021 13.5050827  11.84143724  7.87050642 20.51966653
 10.79961336  9.03233096 17.99419817 16.29237067 11.04506924 14.09963141
 18.44147334  9.3759692   7.88687015  8.34505447 17.72692398 11.62325422]
```

In [48]:
```
#Actual value and the predicted value
slr_diff = pd.DataFrame({'Actual value': y_test, 'Predicted value': y_pred_slr})
slr_diff
```

Out[48]:

	Actual value	Predicted value
126	6.6	7.374140
104	20.7	19.941482
99	17.2	14.323269
92	19.4	18.823294
111	21.8	20.132392
167	17.2	18.228745
116	12.2	14.541452
96	16.7	17.726924
52	22.6	18.752384
69	22.3	18.774202
164	11.9	13.341445
124	19.7	19.466933
182	8.7	10.014155
154	20.6	17.192376
125	10.6	11.705073
196	14.0	12.086893
194	17.3	15.114182
177	16.7	16.232370
163	18.0	15.866914
31	11.9	13.106899
11	17.4	18.659656
73	11.0	14.006904
15	22.4	17.606923

	Actual value	Predicted value
41	17.1	16.603281
97	20.5	17.034193
128	24.7	18.965113
133	19.6	18.937840
82	11.3	11.055978
139	20.7	17.034193
123	15.2	13.663265
83	13.6	10.679613
65	11.3	10.712340
151	11.6	13.548719
162	19.9	17.225103
170	8.4	9.675971
77	14.2	13.521446
32	13.2	12.250530
173	16.7	16.134188
174	16.5	19.079659
85	20.2	17.486923
168	17.1	18.697838
112	17.1	16.532372
171	17.5	15.921460
181	17.2	18.866930
7	13.2	13.505083
46	10.6	11.841437
75	8.7	7.870506
28	18.9	20.519667
29	10.5	10.799613
195	7.6	9.032331

	Actual value	Predicted value
40	16.6	17.994198
153	16.0	16.292371
115	12.6	11.045069
64	16.0	14.099631
59	18.4	18.441473
1	10.4	9.375969
192	5.9	7.886870
136	9.5	8.345054
152	16.6	17.726924
161	13.3	11.623254

In [21]:
```
#Predict for any value
slr.predict([[56]])
```

Out[21]:
```
array([10.00324536])
```

Conclusion: The model predicted the Sales of 10.003 in that market

In [22]:
```
# print the R-squared value for the model
from sklearn.metrics import accuracy_score
print('R squared value of the model: {:.2f}'.format(slr.score(x,y)*100))
```

```
R squared value of the model: 81.10
```

Conclusion: 81.10% of the data fit the regression model

In [49]:
```
# 0 means the model is perfect. Therefore the value should be as close to 0 as possible
meanAbErr = metrics.mean_absolute_error(y_test, y_pred_slr)
meanSqErr = metrics.mean_squared_error(y_test, y_pred_slr)
rootMeanSqErr = np.sqrt(metrics.mean_squared_error(y_test, y_pred_slr))

print('Mean Absolute Error:', meanAbErr)
print('Mean Square Error:', meanSqErr)
print('Root Mean Square Error:', rootMeanSqErr)
```

```
Mean Absolute Error: 1.6480589869746527
Mean Square Error: 4.077556371826951
Root Mean Square Error: 2.0192960089662315
```

2. Multiple Linear Regression

Multiple linear regression has one y and two or more x variables. It is an extension of Simple Linear regression as it takes more than one predictor variable to predict the response variable.

Multiple Linear Regression is one of the important regression algorithms which models the linear relationship between a single dependent continuous variable and more than one independent variable.

Assumptions for Multiple Linear Regression:

1. A linear relationship should exist between the Target and predictor variables.
2. The regression residuals must be normally distributed.
3. MLR assumes little or no multicollinearity (correlation between the independent variable) in data.

Formula: $Y = \beta0 + \beta1X1 + \beta2X2 + \beta3X3 + ... + \beta nXn + e$

Y = Dependent variable / Target variable
β0 = Intercept of the regression line
β1, β2,..βn = Slope of the regression lime which tells whether the line is increasing or decreasing
X1, X2,..Xn = Independent variables / Predictor variables
e = Error

Equation: Sales = β0 + (β1 * TV) + (β2 * Radio) + (β3 * Newspaper)

In [50]:
```
#Setting the value for X and Y
x = dataset[['TV', 'Radio', 'Newspaper']]
y = dataset['Sales']
```

In [51]:
```
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.3, random_state=100)
```

In [52]:
```
mlr= LinearRegression()
mlr.fit(x_train, y_train)
```

Out[52]:
```
LinearRegression()
```

In [53]:
```
#Printing the model coefficients
print(mlr.intercept_)
# pair the feature names with the coefficients
list(zip(x, mlr.coef_))
```

```
4.334595861728433
```

Out[53]:
```
[('TV', 0.05382910866725006),
 ('Radio', 0.11001224388558055),
 ('Newspaper', 0.006289950146130348)]
```

In [54]:
```
#Predicting the Test and Train set result
y_pred_mlr= mlr.predict(x_test)
x_pred_mlr= mlr.predict(x_train)
```

In [55]:
```
print("Prediction for test set: {}".format(y_pred_mlr))
```

```
Prediction for test set: [ 9.35221067 20.96344625 16.48851064 20.10971005 21.67148354 16.16054424
 13.5618056  15.39338129 20.81980757 21.00537077 12.29451311 20.70848608
  8.17367308 16.82471534 10.48954832  9.99530649 16.34698901 14.5758119
 17.23065133 12.56890735 18.55715915 12.12402775 20.43312609 17.78017811
 16.73623408 21.60387629 20.13532087 10.82559967 19.12782848 14.84537816
 13.13597397  9.07757918 12.07834143 16.62824427  8.41792841 14.0456697
  9.92050209 14.26101605 16.76262961 17.17185467 18.88797595 15.50165469
 15.78688377 16.86266686 13.03405813 10.47673934 10.6141644  20.85264977
 10.1517568   6.88471443 17.88702583 18.16013938 12.55907083 16.28189561
 18.98024679 11.33714913  5.91026916 10.06159509 17.62383031 13.19628335]
```

In [56]:
```
#Actual value and the predicted value
mlr_diff = pd.DataFrame({'Actual value': y_test, 'Predicted value': y_pred_mlr})
mlr_diff
```

Out[56]:

	Actual value	Predicted value
126	6.6	9.352211
104	20.7	20.963446
99	17.2	16.488511
92	19.4	20.109710
111	21.8	21.671484
167	17.2	16.160544
116	12.2	13.561806
96	16.7	15.393381
52	22.6	20.819808
69	22.3	21.005371
164	11.9	12.294513
124	19.7	20.708486
182	8.7	8.173673
154	20.6	16.824715
125	10.6	10.489548
196	14.0	9.995306
194	17.3	16.346989
177	16.7	14.575812
163	18.0	17.230651
31	11.9	12.568907
11	17.4	18.557159
73	11.0	12.124028
15	22.4	20.433126

	Actual value	Predicted value
41	17.1	17.780178
97	20.5	16.736234
128	24.7	21.603876
133	19.6	20.135321
82	11.3	10.825600
139	20.7	19.127828
123	15.2	14.845378
83	13.6	13.135974
65	11.3	9.077579
151	11.6	12.078341
162	19.9	16.628244
170	8.4	8.417928
77	14.2	14.045670
32	13.2	9.920502
173	16.7	14.261016
174	16.5	16.762630
85	20.2	17.171855
168	17.1	18.887976
112	17.1	15.501655
171	17.5	15.786884
181	17.2	16.862667
7	13.2	13.034058
46	10.6	10.476739
75	8.7	10.614164
28	18.9	20.852650
29	10.5	10.151757
195	7.6	6.884714

	Actual value	Predicted value
40	16.6	17.887026
153	16.0	18.160139
115	12.6	12.559071
64	16.0	16.281896
59	18.4	18.980247
1	10.4	11.337149
192	5.9	5.910269
136	9.5	10.061595
152	16.6	17.623830
161	13.3	13.196283

In [57]:
```
#Predict for any value
mlr.predict([[56, 55, 67]])
```

Out[57]:
```
array([13.82112602])
```

Conclusion: The model predicted the Sales of 13.82 in that market

In [58]:
```
# print the R-squared value for the model
print('R squared value of the model: {:.2f}'.format(mlr.score(x,y)*100))
```

```
R squared value of the model: 90.11
```

Conclusion: 90.21% of the data fit the multiple regression model

In [59]:
```
# 0 means the model is perfect. Therefore the value should be as close to 0 as possible
meanAbErr = metrics.mean_absolute_error(y_test, y_pred_mlr)
meanSqErr = metrics.mean_squared_error(y_test, y_pred_mlr)
rootMeanSqErr = np.sqrt(metrics.mean_squared_error(y_test, y_pred_mlr))

print('Mean Absolute Error:', meanAbErr)
print('Mean Square Error:', meanSqErr)
print('Root Mean Square Error:', rootMeanSqErr)
```

```
Mean Absolute Error: 1.2278183566589418
Mean Square Error: 2.6360765623280664
Root Mean Square Error: 1.6235998775338911
```