

Multiple Linear Regression.

So, what is multiple linear regression?

Multiple linear regression is used to estimate the relationship between two or more independent variables and one dependent variable. You can use multiple linear regression when you want to know:

1. How strong the relationship is between two or more independent variables and one dependent variable (e.g. how rainfall, temperature, and amount of fertilizer added affect crop growth).
2. The value of the dependent variable at a certain value of the independent variables (e.g. the expected yield of a crop at certain levels of rainfall, temperature, and fertilizer addition).

Example: You are a public health researcher interested in social factors that influence heart disease. You survey 500 towns and gather data on the percentage of people in each town who smoke, the percentage of people in each town who bike to work, and the percentage of people in each town who have heart disease. Because you have two independent variables and one dependent variable, and all your variables are quantitative, you can use multiple linear regression to analyze the relationship between them.

Assumptions of multiple linear regression

1.Homogeneity of variance (homoscedasticity): the size of the error in our prediction doesn't change significantly across the values of the independent variable.

2.Independence of observations: the observations in the dataset were collected using statistically valid methods, and there are no hidden relationships among variables.

In multiple linear regression, it is possible that some of the independent variables are actually correlated with one another, so it is important to check these before developing the regression model. If two independent variables are too highly correlated ($r^2 > \sim 0.6$), then only one of them should be used in the regression model.

3. Normality: The data follows a normal distribution.

4.Linearity: the line of best fit through the data points is a straight line, rather than a curve or some sort of grouping factor.

Multiple linear regression formula The formula for a multiple linear regression is

$$\text{Equation: } Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \dots + \beta_n X_n + e$$

Y = Dependent variable / Target variable

β_0 = Intercept of the regression line

$\beta_1, \beta_2, \beta_3, \dots, \beta_n$ = Slope of the regression line which tells whether the line is increasing or decreasing

$X_1, X_2, X_3, \dots, X_n$ = Independent variable / Predictor variable

e = Error

To find the best-fit line for each independent variable, multiple linear regression calculates three things:

1.Find The regression coefficients that lead to the smallest overall model error.

2.The t-statistic of the overall model.

3.Find The associated p-value (how likely it is that the t-statistic would have occurred by chance if the null hypothesis of no relationship between the independent and dependent variables was true).

it then calculates the t-statistic and p-value for each regression coefficient in the model.

T-statistic is the ratio of the departure of the estimated value of a parameter from its hypothesized value to its standard error.

The t-value measures the size of the difference relative to the variation in your sample data.

T-statistic is used when the sample size is less than 30 units or the population standard deviation is unknown.

$$\text{T-Statistic} = \frac{\bar{X} - \mu}{s \div \sqrt{n}}$$

Formula to calculate t-statistic.

where,

X-bar is the sample mean.

μ is the population mean.

S is the sample standard deviation.

n is the number of units in the sample.

P-value is the probability that a random chance generated the data or something else that is equal or greater(in some cases less than.)

P value can be calculated from the p-value table for each associated z/t-score

Example

Every company uses multiple channels to market their product. Suppose a company has developed some product and needs to make a decision on which marketing channel to use for maximizing its ROI that it makes in marketing. It has used multiple channels to market its product in the past, whose data is available as a csv file.

Predict the sales based on the money spent on TV, Radio, and Newspaper for marketing. In this case, there are three independent variables, i.e., money spent on TV, Radio, and Newspaper for marketing, and one dependent variable, i.e., sales, that is the value to be predicted.

Problem statement: Build a Multiple Linear Regression Model to predict sales based on the money spent on TV, Radio, and Newspaper for advertising.

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

seaborn: Seaborn is a library used for making statistical graphics of the dataset. It provides a variety of visualization patterns. It uses fewer syntax and has easily interesting default themes. It is used to summarize data in visualizations and show the data's distribution.

```
In [2]: #Reading the dataset
dataset = pd.read_csv(r"C:\Users\rishu\Desktop\data\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. Finding the missing values

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

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

There are no missing values in the dataset

Check for Duplicate rows

```
In [6]: dataset.duplicated().any()
```

```
Out[6]: False
```

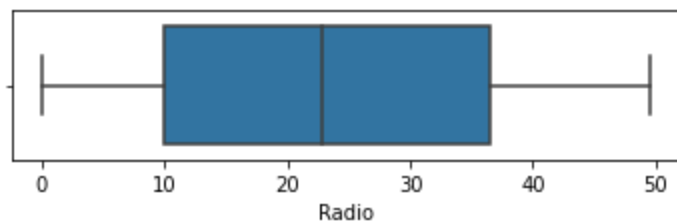
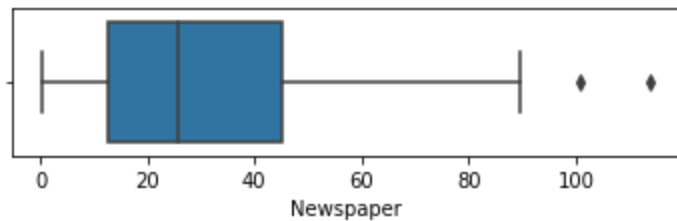
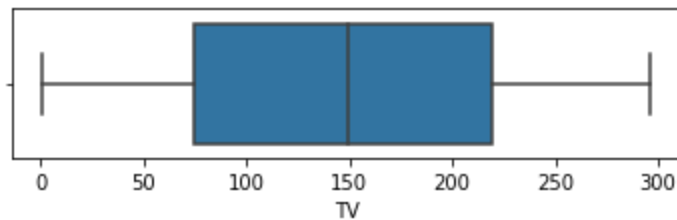
Conclusion: There are no duplicate rows present in the dataset

3. Checking for outliers

```
In [7]: 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()
```

```
C:\Users\rishu\AppData\Roaming\Python\Python39\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the o
nly valid positional argument will be `data`, and passing other arguments without an exp
licit keyword will result in an error or misinterpretation.
  warnings.warn(
C:\Users\rishu\AppData\Roaming\Python\Python39\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the o
nly valid positional argument will be `data`, and passing other arguments without an exp
licit keyword will result in an error or misinterpretation.
  warnings.warn(
C:\Users\rishu\AppData\Roaming\Python\Python39\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the o
nly valid positional argument will be `data`, and passing other arguments without an exp
licit keyword will result in an error or misinterpretation.
  warnings.warn(
```



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

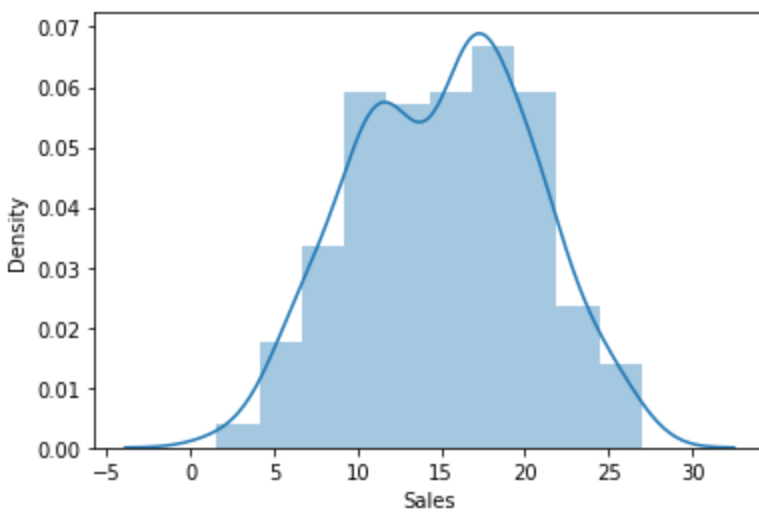
Exploratory Data Analysis

1. A. Distribution of the target variable

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

```
C:\Users\rishu\AppData\Roaming\Python\Python39\site-packages\seaborn\distributions.py:26
19: FutureWarning: `distplot` is a deprecated function and will be removed in a future v
ersion. Please adapt your code to use either `displot` (a figure-level function with sim
ilar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

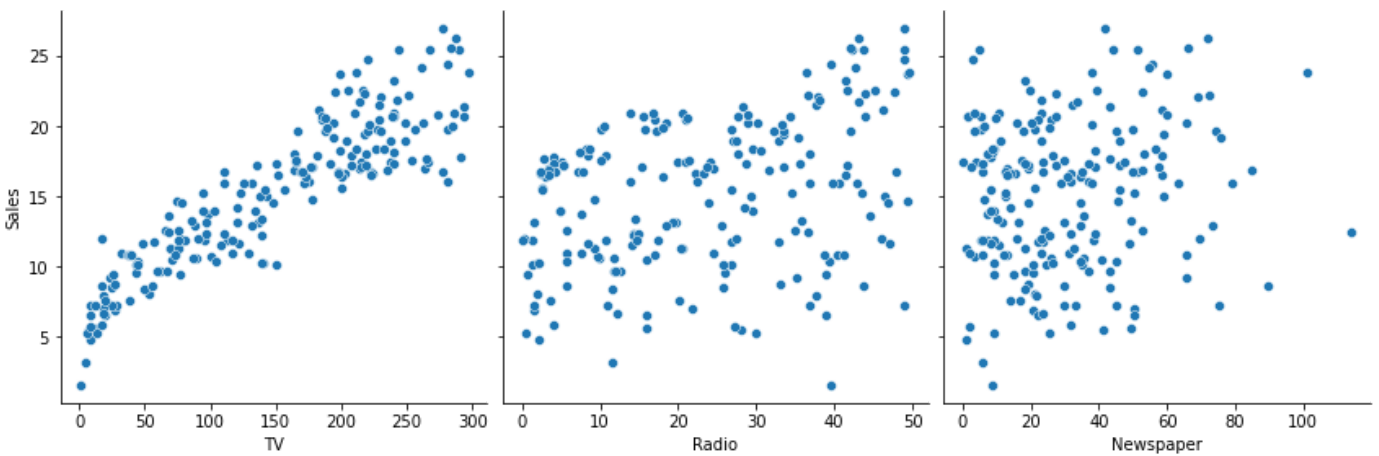
```
Out[8]: <AxesSubplot:xlabel='Sales', ylabel='Density'>
```



Conclusion: It is normally distributed

In [9]: *#2. How Sales are related with other variables*

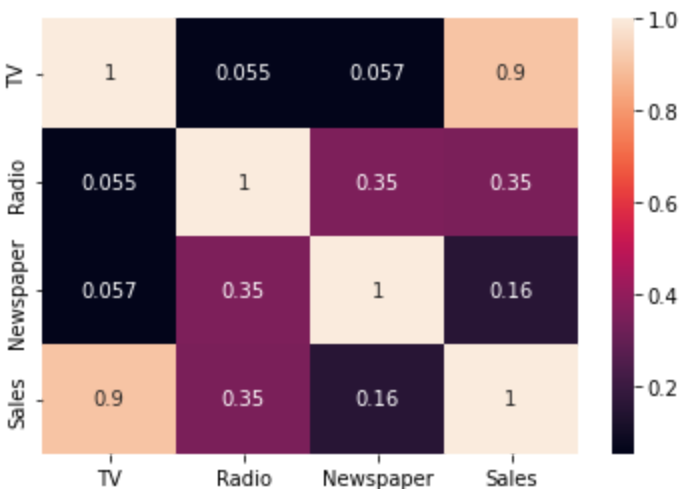
```
sns.pairplot(dataset, x_vars=['TV', 'Radio', 'Newspaper'], y_vars='Sales', height=4, asp
plt.show())
```



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

In [10]: *# to better visualize it lets make a heatmap*

```
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

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: $Y = \beta_0 + \beta_1 X + e$

Y = Dependent variable / Target variable β_0 = Intercept of the regression line β_1 = Slope of the regression line which tells whether the line is increasing or decreasing X = Independent variable / Predictor variable e = Error

```
In [11]: # Simple Linear Model
```

```
In [12]: from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
```

```
In [13]: #Setting the value for X and Y
x = dataset[['TV']]
y = dataset['Sales']
```

```
In [14]: x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.3, random_state
```

```
In [15]: slr= LinearRegression() #simple linear regression
slr.fit(x_train, y_train)
```

```
Out[15]: LinearRegression()
```

```
In [16]: #Printing the model coefficients
print('Intercept: ', slr.intercept_)
print('Coefficient:', slr.coef_)
```

```
Intercept:  6.948683200001357
Coefficient: [0.05454575]
```

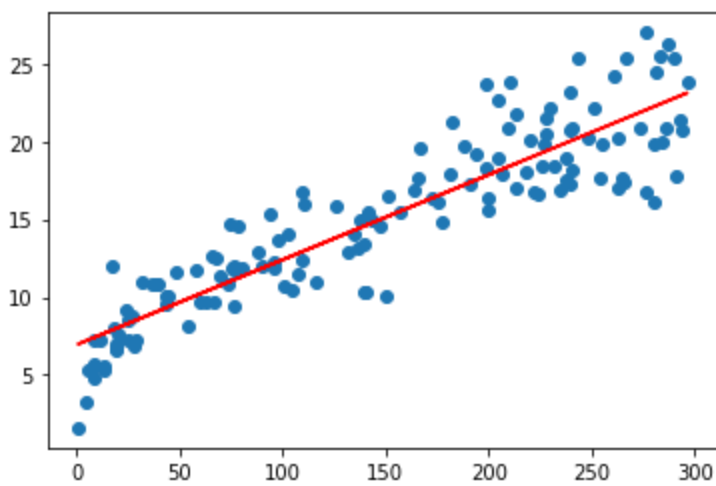
```
In [17]: print('Regression Equation: Sales = 6.948 + 0.054 * TV')

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

```
In [18]: y_train_predict = slr.predict(x_train)
```

```
In [19]: #Line of best fit
```

```
plt.scatter(x_train['TV'], y_train)
plt.plot(x_train['TV'].to_list(), y_train_predict, 'r')
plt.show()
```



```
In [20]: #Prediction of Test and Training set result
y_pred_slr= slr.predict(x_test)
x_pred_slr= slr.predict(x_train)
```

```
In [21]: print(y_pred_slr)
```

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

```
Out[22]:
```

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

C:\Users\rishu\AppData\Roaming\Python\Python39\site-packages\sklearn\base.py:450: UserWarning: X does not have valid feature names, but LinearRegression was fitted with feature names

warnings.warn(

```
Out[23]: array([10.00324536])
```

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

```
In [24]: # 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

```
In [25]: # 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: 1.6480589869746525

Mean Square Error: 4.077556371826948

Root Mean Square Error: 2.019296008966231

```
In [26]: # multiple linear regression

#Setting the parameters

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

```
In [27]: x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.3, random_state=10)
```

```
In [28]: mlr= LinearRegression()
mlr.fit(x_train, y_train)
```

```
Out[28]: LinearRegression()
```

```
In [29]: #Printing the model coefficients
print(mlr.intercept_)
# pair the feature names with the coefficients
list(zip(x, mlr.coef_))
```

4.334595861728431

```
Out[29]: [('TV', 0.053829108667250075),  
          ('Radio', 0.11001224388558056),  
          ('Newspaper', 0.006289950146130346)]
```

```
In [30]: #Predicting the Test and Train set result  
y_pred_mlr= mlr.predict(x_test)  
x_pred_mlr= mlr.predict(x_train)
```

```
In [31]: print("Prediction for test set: ",y_pred_mlr)
```

```
Prediction for test set: [ 9.35221067 20.96344625 16.48851064 20.10971005 21.67148354 1  
6.16054424  
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 [32]: #Actual value and the predicted value  
mlr_diff = pd.DataFrame({'Actual value': y_test, 'Predicted value': y_pred_mlr})  
mlr_diff
```

```
Out[32]:
```

	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
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
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

```
In [33]: #Predict for any value
mlr.predict([[56, 55, 67]])
```

C:\Users\rishu\AppData\Roaming\Python\Python39\site-packages\sklearn\base.py:450: UserWarning: X does not have valid feature names, but LinearRegression was fitted with feature names

```
warnings.warn(
Out[33]: array([13.82112602])
```

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

```
In [34]: # 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 [35]: # 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: 1.227818356658941
Mean Square Error: 2.6360765623280646
Root Mean Square Error: 1.6235998775338907

```
In [ ]:
```