Multiple Linear Regression

Now you know how to build a model with one X (feature variable) and Y (response variable). But what if you have three feature variables, or may be 10 or 100? Building a separate model for each of them, combining them, and then understanding them will be a very difficult and next to impossible task. By using multiple linear regression, you can build models between a response variable and many feature variables.

Let's see how to do that.

Step_1 : Importing and Understanding Data

In [1]:
```
import pandas as pd
```

In [2]:
```
# Importing advertising.csv
advertising_multi = pd.read_csv('advertising.csv')
```

In [3]:
```
# Looking at the first five rows
advertising_multi.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	9.3
3	151.5	41.3	58.5	18.5
4	180.8	10.8	58.4	12.9

In [4]:
```
# Looking at the last five rows
advertising_multi.tail()
```

Out[4]:

	TV	Radio	Newspaper	Sales
195	38.2	3.7	13.8	7.6
196	94.2	4.9	8.1	9.7
197	177.0	9.3	6.4	12.8
198	283.6	42.0	66.2	25.5
199	232.1	8.6	8.7	13.4

In [5]:
```python
# What type of values are stored in the columns?
advertising_multi.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 4 columns):
TV           200 non-null float64
Radio        200 non-null float64
Newspaper    200 non-null float64
Sales        200 non-null float64
dtypes: float64(4)
memory usage: 6.3 KB
```

In [6]:
```python
# Let's look at some statistical information about our dataframe.
advertising_multi.describe()
```

Out[6]:

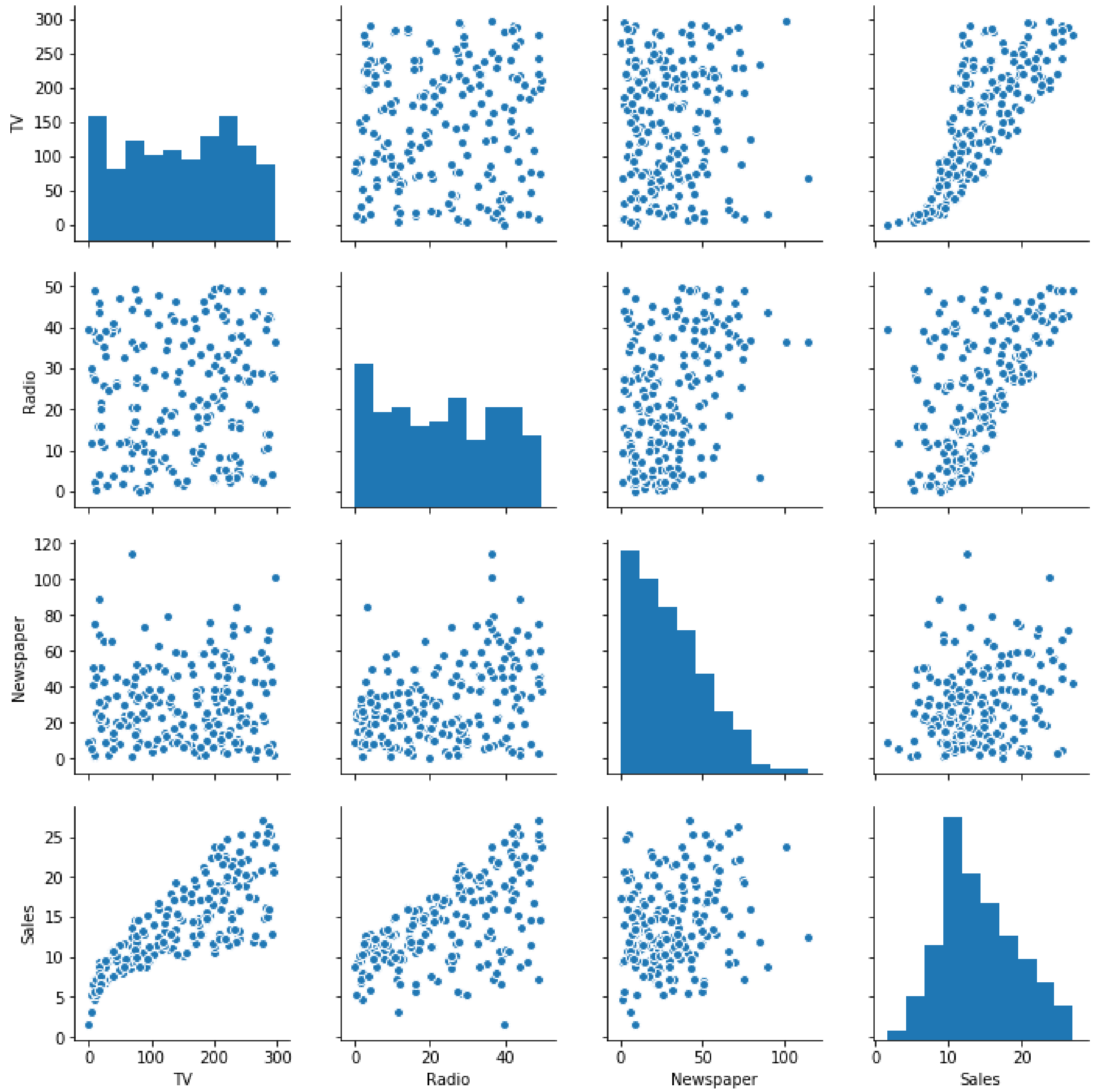
	TV	Radio	Newspaper	Sales
count	200.000000	200.000000	200.000000	200.000000
mean	147.042500	23.264000	30.554000	14.022500
std	85.854236	14.846809	21.778621	5.217457
min	0.700000	0.000000	0.300000	1.600000
25%	74.375000	9.975000	12.750000	10.375000
50%	149.750000	22.900000	25.750000	12.900000
75%	218.825000	36.525000	45.100000	17.400000
max	296.400000	49.600000	114.000000	27.000000

Step_2: Visualising Data

In [8]:
```python
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

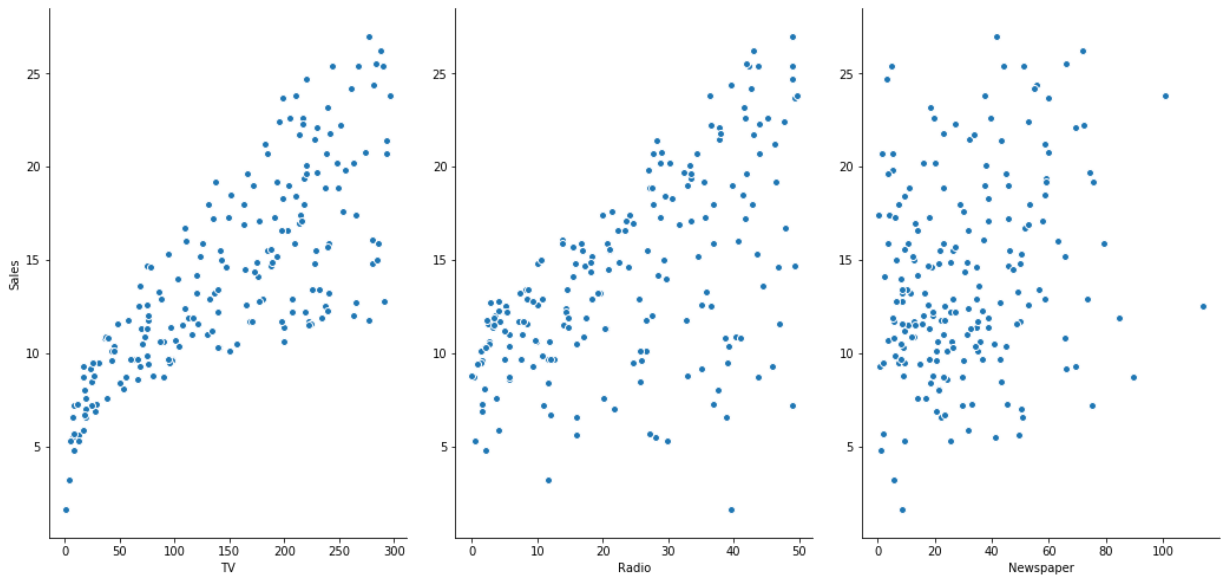
In [9]:
```
# Let's plot a pair plot of all variables in our dataframe
sns.pairplot(advertising_multi)
```

Out[9]: <seaborn.axisgrid.PairGrid at 0x1b1aac74128>



In [9]:
```python
# Visualise the relationship between the features and the response using
scatterplots
sns.pairplot(advertising_multi, x_vars=['TV','Radio','Newspaper'],
y_vars='Sales',size=7, aspect=0.7, kind='scatter')
```

Out[9]:
```
<seaborn.axisgrid.PairGrid at 0x204efb370b8>
```



Step_3: Splitting the Data for Training and Testing

In [10]:
```python
# Putting feature variable to X
X = advertising_multi[['TV','Radio','Newspaper']]

# Putting response variable to y
y = advertising_multi['Sales']
```

In [12]:
```python
#random_state is the seed used by the random number generator. It can be any
integer.
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7 ,
random_state=100)
```

Step_4 : Performing Linear Regression

In [20]:
```python
from sklearn.linear_model import LinearRegression
```

In [14]:
```python
# Representing LinearRegression as lr(Creating LinearRegression Object)
lm = LinearRegression()
```

In [15]:
```
# fit the model to the training data
lm.fit(X_train,y_train)
```

Out[15]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

Step_5 : Model Evaluation

In [14]:
```
# print the intercept
print(lm.intercept_)
```

```
2.65278966888
```

In [16]:
```
# Let's see the coefficient
coeff_df = pd.DataFrame(lm.coef_,X_test.columns,columns=['Coefficient'])
coeff_df
```

Out[16]:

	Coefficient
TV	0.045426
Radio	0.189758
Newspaper	0.004603

From the above result we may infern that if TV price increses by 1 unit it will affect sales by 0.045 units.

Step_6 : Predictions

In [17]:
```
# Making predictions using the model
y_pred = lm.predict(X_test)
```

Step_7: Calculating Error Terms

In [18]:
```
from sklearn.metrics import mean_squared_error, r2_score
mse = mean_squared_error(y_test, y_pred)
r_squared = r2_score(y_test, y_pred)
```

In [19]:
```
print('Mean_Squared_Error :' ,mse)
print('r_square_value :',r_squared)
```

```
Mean_Squared_Error : 1.8506819941636972
r_square_value : 0.9058622107532245
```

Optional Step : Checking for P-value Using STATSMODELS

In [22]:
```
import statsmodels.api as sm
X_train_sm = X_train
#Unlike SKLearn, statsmodels don't automatically fit a constant,
#so you need to use the method sm.add_constant(X) in order to add a constant.
X_train_sm = sm.add_constant(X_train_sm)
# create a fitted model in one line
lm_1 = sm.OLS(y_train,X_train_sm).fit()

# print the coefficients
lm_1.params
```

Out[22]:
```
const        2.652790
TV           0.045426
Radio        0.189758
Newspaper    0.004603
dtype: float64
```

In [22]:
```
print(lm_1.summary())
```

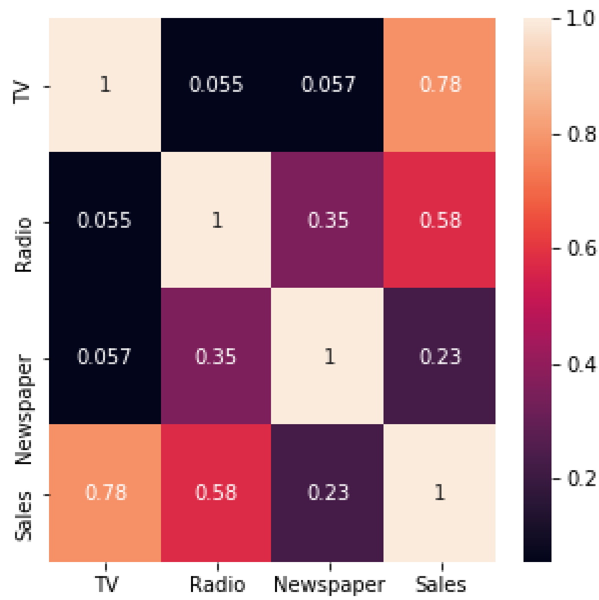
```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  Sales   R-squared:                       0.893
Model:                            OLS   Adj. R-squared:                  0.890
Method:                 Least Squares   F-statistic:                     377.6
Date:                Thu, 09 Aug 2018   Prob (F-statistic):           9.97e-66
Time:                        11:07:15   Log-Likelihood:                -280.83
No. Observations:                 140   AIC:                             569.7
Df Residuals:                     136   BIC:                             581.4
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          2.6528      0.384      6.906      0.000       1.893       3.412
TV             0.0454      0.002     27.093      0.000       0.042       0.049
Radio          0.1898      0.011     17.009      0.000       0.168       0.212
Newspaper      0.0046      0.008      0.613      0.541      -0.010       0.019
==============================================================================
Omnibus:                       40.095   Durbin-Watson:                   1.862
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               83.622
Skew:                          -1.233   Prob(JB):                     6.94e-19
Kurtosis:                       5.873   Cond. No.                         443.
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctl
y specified.
```

From the above we can see that Newspaper is insignificant.

In [23]:
```
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

In [24]:
```
plt.figure(figsize = (5,5))
sns.heatmap(advertising_multi.corr(),annot = True)
```

Out[24]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1b1acdf7e10>
```



Step_8 : Implementing the results and running the model again

From the data above, you can conclude that Newspaper is insignificant.

In [27]:
```
# Removing Newspaper from our dataset
X_train_new = X_train[['TV','Radio']]
X_test_new = X_test[['TV','Radio']]
```

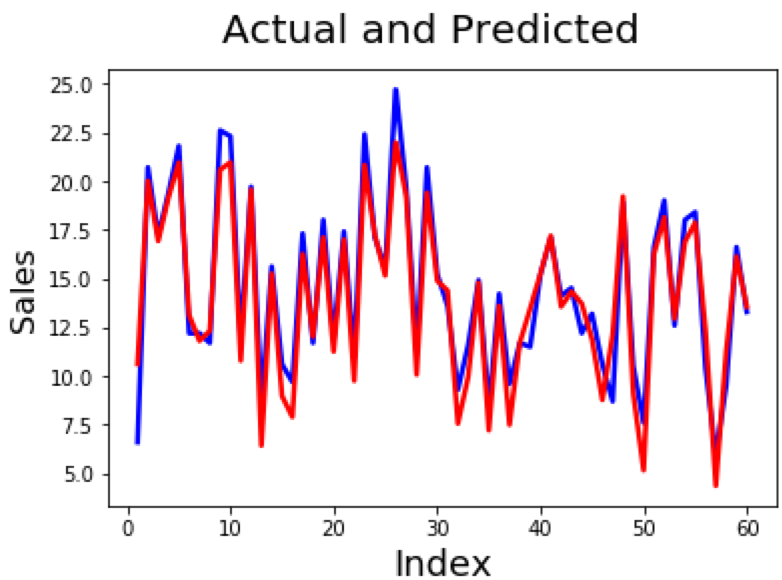
In [28]:
```
# Model building
lm.fit(X_train_new,y_train)
```

Out[28]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

In [29]:
```
# Making predictions
y_pred_new = lm.predict(X_test_new)
```

In [29]:
```python
#Actual vs Predicted
c = [i for i in range(1,61,1)]
fig = plt.figure()
plt.plot(c,y_test, color="blue", linewidth=2.5, linestyle="-")
plt.plot(c,y_pred, color="red",  linewidth=2.5, linestyle="-")
fig.suptitle('Actual and Predicted', fontsize=20)              # Plot heading
plt.xlabel('Index', fontsize=18)                               # X-label
plt.ylabel('Sales', fontsize=16)                               # Y-label
```

Out[29]: Text(0,0.5,'Sales')



In [30]:
```
# Error terms
c = [i for i in range(1,61,1)]
fig = plt.figure()
plt.plot(c,y_test-y_pred, color="blue", linewidth=2.5, linestyle="-")
fig.suptitle('Error Terms', fontsize=20)              # Plot heading
plt.xlabel('Index', fontsize=18)                      # X-label
plt.ylabel('ytest-ypred', fontsize=16)                # Y-label
```

Out[30]:
```
Text(0,0.5,'ytest-ypred')
```

In [32]:
```
from sklearn.metrics import mean_squared_error, r2_score
mse = mean_squared_error(y_test, y_pred_new)
r_squared = r2_score(y_test, y_pred_new)
```

In [32]:
```
print('Mean_Squared_Error :' ,mse)
print('r_square_value :',r_squared)
```

```
Mean_Squared_Error : 1.78474005209
r_square_value : 0.909216449172
```

In [33]:
```
X_train_final = X_train_new
#Unlike SKLearn, statsmodels don't automatically fit a constant,
#so you need to use the method sm.add_constant(X) in order to add a constant.
X_train_final = sm.add_constant(X_train_final)
# create a fitted model in one line
lm_final = sm.OLS(y_train,X_train_final).fit()

print(lm_final.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  Sales   R-squared:                       0.893
Model:                            OLS   Adj. R-squared:                  0.891
Method:                 Least Squares   F-statistic:                     568.8
Date:                Sat, 23 Feb 2019   Prob (F-statistic):           4.46e-67
Time:                        15:36:22   Log-Likelihood:                -281.03
No. Observations:                 140   AIC:                             568.1
Df Residuals:                     137   BIC:                             576.9
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          2.7190      0.368      7.392      0.000       1.992       3.446
TV             0.0455      0.002     27.368      0.000       0.042       0.049
Radio          0.1925      0.010     18.860      0.000       0.172       0.213
==============================================================================
Omnibus:                       41.530   Durbin-Watson:                   1.862
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               90.544
Skew:                          -1.255   Prob(JB):                     2.18e-20
Kurtosis:                       6.037   Cond. No.                         419.
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctl
y specified.
```

Model Refinement Using RFE

The goal of recursive feature elimination (RFE) is to select features by recursively considering smaller and smaller sets of features. First, the estimator is trained on the initial set of features and the importance of each feature is obtained either through a coef_ attribute or through a feature_importances_ attribute. Then, the less important features are pruned from the the current set of features. This procedure is recursively repeated on the pruned dataset until the desired number of features to select is reached.

In [32]:
```
from sklearn.feature_selection import RFE
```

In [33]:
```
rfe = RFE(lm, 2)
```

In [34]:
```
rfe = rfe.fit(X_train, y_train)
```

In [35]:
```
print(rfe.support_)
print(rfe.ranking_)
```

```
[ True  True False]
[1 1 2]
```

Simple Linear Regression: Newspaper(X) and Sales(y)

In [34]:
```
import pandas as pd
import numpy as np
# Importing dataset
advertising_multi = pd.read_csv('advertising.csv')

x_news = advertising_multi['Newspaper']

y_news = advertising_multi['Sales']

# Data Split
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x_news, y_news,
                                                    train_size=0.7 ,
                                                    random_state=110)

# Required only in the case of simple linear regression
X_train = X_train[:,np.newaxis]
X_test = X_test[:,np.newaxis]


# Linear regression from sklearn
from sklearn.linear_model import LinearRegression
lm = LinearRegression()

# Fitting the model
lm.fit(X_train,y_train)

# Making predictions
y_pred = lm.predict(X_test)

# Importing mean square error and r square from sklearn library.
from sklearn.metrics import mean_squared_error, r2_score

# Computing mean square error and R square value
mse = mean_squared_error(y_test, y_pred)
r_squared = r2_score(y_test, y_pred)

# Printing mean square error and R square value
print('Mean_Squared_Error :' ,mse)
print('r_square_value :',r_squared)
```

```
Mean_Squared_Error : 23.84732008485191
r_square_value : 0.08182413570736657
```

In [ ]:

```

```