Handling issues in linear regression

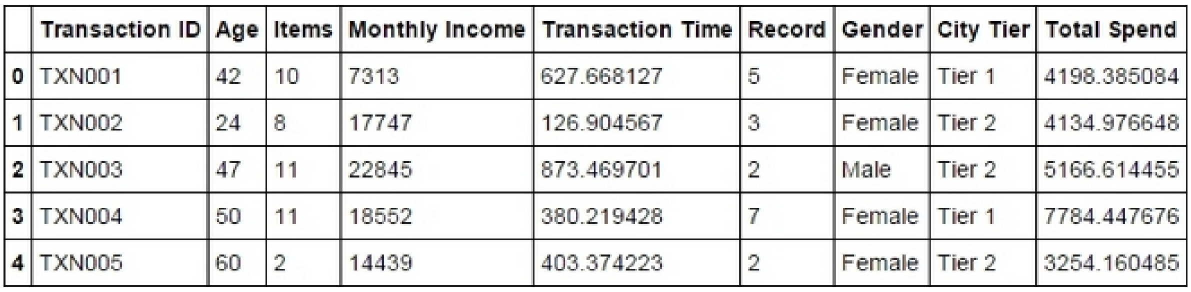
So far, we have learnt:

- How to implement a linear regression model using two methods
- How to measure the efficiency of the model using model parameters

However, there are other issues that need to be taken care of while dealing with data sources of different types. Let's go through them one by one. We will be using a different (simulated) dataset to illustrate these issues. Let's import it and have a look at it:

```
import pandas as pd
df=pd.read_csv('C:/FILE_PATH/Ecom Expense.csv')
df.head()
```

We should get the following output:



	Transaction ID	Age	Items	Monthly Income	Transaction Time	Record	Gender	City Tier	Total Spend
0	TXN001	42	10	7313	627.668127	5	Female	Tier 1	4198.385084
1	TXN002	24	8	17747	126.904567	3	Female	Tier 2	4134.976648
2	TXN003	47	11	22845	873.469701	2	Male	Tier 2	5166.614455
3	TXN004	50	11	18552	380.219428	7	Female	Tier 1	7784.447676
4	TXN005	60	2	14439	403.374223	2	Female	Tier 2	3254.160485

Ecom Expense dataset

The preceding screenshot is a simulated dataset from any-commerce website. It captures the information about several transactions done on the website. A brief description of the column names of the dataset is, as follows:

- **Transaction ID**: Transaction ID for the transaction
- **Age**: Age of the customer
- **Items**: Number of items in the shopping cart (purchased)
- **Monthly Income**: Monthly disposable income of the customer
- **Transaction Time**: Total time spent on the website during the transaction
- **Record**: How many times the customer has shopped with the website in the past
- **Gender**: Gender of the customer
- **City Tier**: Tier of the city
- **Total Spend**: Total amount spent in the transaction

The output variable is the `Total Spend` variable. The others are potential predictor variables and we suspect that the `Total Spend` is linearly related to all these predictor variables.

Handling categorical variables

Until now, we have assumed that the predictor variables can only be quantitative or numerical, but we know from real-life experiences that most of the times the dataset contains a categorical or qualitative variable and many of the times these variables will have a significant impact on the value of the output. However, the question is how to process these variables, so as to use them in the model?

We can't assign them values, such as 0, 1, 2, and so on, and then use them in the model, as it will give undue weightage to the categories because of the numbers assigned to them. Most of the time it might give a wrong result and will change, as the number assigned to a particular category changes.

In the data frame we just imported, **Gender** and **City Tier** are the categorical variables. In our, *Data Cleaning lab*, we learnt how to create dummy variables from a categorical variable. That was exactly for this purpose. Let's see how it works and why it is required.

A linear regression is of the form:

$$y\,\mathrm{mod}\,el = \alpha + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 \ldots\ldots + \beta_n X_n + d$$

Where one or more of the *Xi*'s can be categorical. Let's say X_m is that variable. For such a variable, we can define another dummy variable (if it has only two categories as in the case of Gender), such that:

$$\begin{aligned} Xg &= 1, if\ customer\ is\ male \\ &= 0, if\ customer\ is\ female \end{aligned}$$

The model then becomes:

$$y\,\mathrm{mod}\,el = \alpha + \beta 1X1 + \beta 2X2 + \beta 3X3 \ldots\ldots + \beta g \ldots\ldots + \beta nXn + d, if\ customer\ is\ male$$
$$y\,\mathrm{mod}\,el = \alpha + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 \ldots\ldots + \beta_n X_n + d, if\ customer\ is\ female$$

If there are three levels in the categorical variable, then one needs to define two variables as compared to 1 when there were two levels in the categorical variable. For example, **City Tier** variable has three levels in our dataset.

For this, we can define two variables, such that:

$$\begin{aligned} &X_{t1} = 1, if\ City\ is\ Tier\,1 \qquad && X_{t2} = 2, if\ City\ is\ Tier\,2 \\ &0, if\ City\ is\ not\ Tier\,1 && = 0, if\ City\ is\ not\ Tier\,2 \end{aligned}$$

The model then becomes:

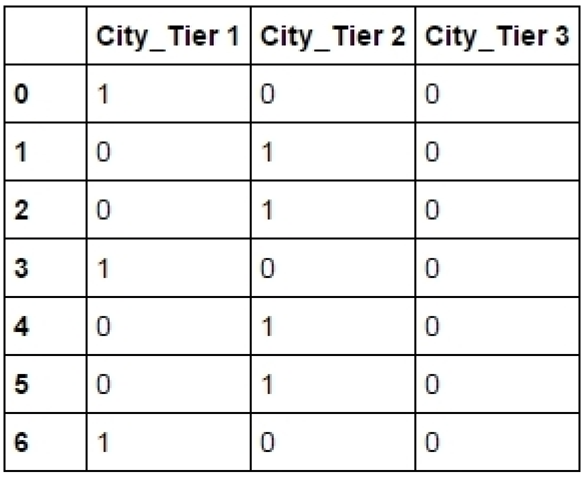
$$y\,\mathrm{mod}\,el = \alpha + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \ldots + \beta_{t1} X_{t1} \ldots\ldots + \beta_n X_n + d, if\ customer\ is\ from\ tier\,1\,city$$
$$y\,\mathrm{mod}\,el = \alpha + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \ldots + \beta_{t2} X_{t2} \ldots\ldots + \beta_n X_n + d, if\ customer\ is\ from\ tier\,2\,city$$
$$y\,\mathrm{mod}\,el = \alpha + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \ldots + \beta_n X_n + d, if\ customer\ is\ from\ tier\,3\,city$$

Note that one doesn't have to create the third variable. This is because of the nature in which these variables are defined. If a customer doesn't belong to tier 1 or tier 2 city, then he will certainly belong to a tier 3 city. Hence, no one variable is required for one of the levels. In general, for categorical variables having n levels, one should create (n-1) dummy variables.

Let's now create the dummy variables for both our categorical variables and then add them to our data frame, as shown:

```
dummy_gender=pd.get_dummies(df['Gender'],prefix='Sex')
dummy_city_tier=pd.get_dummies(df['City Tier'],prefix='City')
```

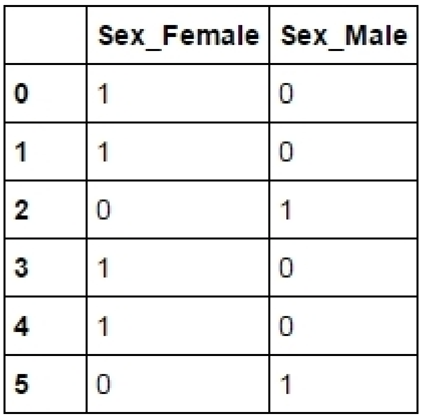
Let's see how they look and whether they satisfy the conditions we have defined earlier or not. This is how the `dummy_city_tier` looks:



	City_Tier 1	City_Tier 2	City_Tier 3
0	1	0	0
1	0	1	0
2	0	1	0
3	1	0	0
4	0	1	0
5	0	1	0
6	1	0	0

City Tier dummy variables

The `dummy_gender` looks similar to the following table:

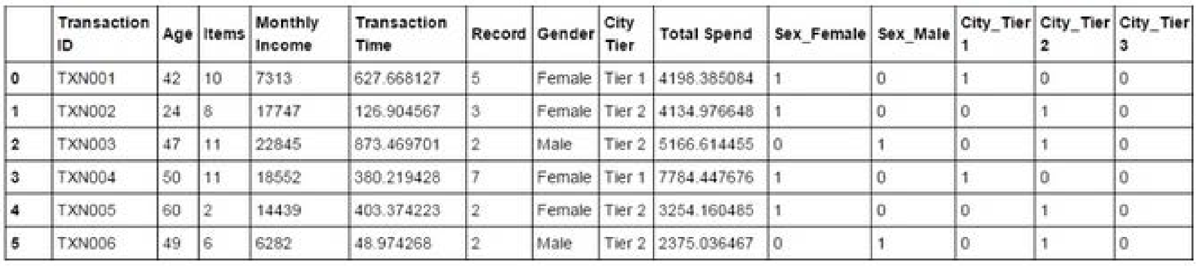


	Sex_Female	Sex_Male
0	1	0
1	1	0
2	0	1
3	1	0
4	1	0
5	0	1

Gender dummy variables

Now, we have these dummy variables created but they are not a part of the main data frame yet. Let's attach these new variables to the main data frame so that they can be used in the model:

```
column_name=df.columns.values.tolist()
df1=df[column_name].join(dummy_gender)
column_name1=df1.columns.values.tolist()
df2=df1[column_name1].join(dummy_city_tier)
df2
```



	Transaction ID	Age	Items	Monthly Income	Transaction Time	Record	Gender	City Tier	Total Spend	Sex_Female	Sex_Male	City_Tier 1	City_Tier 2	City_Tier 3
0	TXN001	42	10	7313	627.668127	5	Female	Tier 1	4198.385084	1	0	1	0	0
1	TXN002	24	8	17747	126.904567	3	Female	Tier 2	4134.976648	1	0	0	1	0
2	TXN003	47	11	22845	873.469701	2	Male	Tier 2	5166.614455	0	1	0	1	0
3	TXN004	50	11	18552	380.219428	7	Female	Tier 1	7784.447676	1	0	1	0	0
4	TXN005	60	2	14439	403.374223	2	Female	Tier 2	3254.160485	1	0	0	1	0
5	TXN006	49	6	6282	48.974268	2	Male	Tier 2	2375.036467	0	1	0	1	0

Ecom Expense dataset with dummy variables

There are five new columns in the data frame, two from the **Gender** dummy variables and three from the **City Tier** dummy variables.

If you compare it with the entire dataset, the **City_Tier_1** has value **1** if the **City_Tier** has value **Tier 1**, the **City_Tier_2** has value **1** if the **City_Tier** has value **Tier 2** and the **City_Tier_3** has value **1** if the **City_Tier** has value **Tier 3**. All the other dummy variables in that particular row will have values **0**. This is what we wanted.

Let's see how to include these dummy variables in the model and how to assess their coefficients.

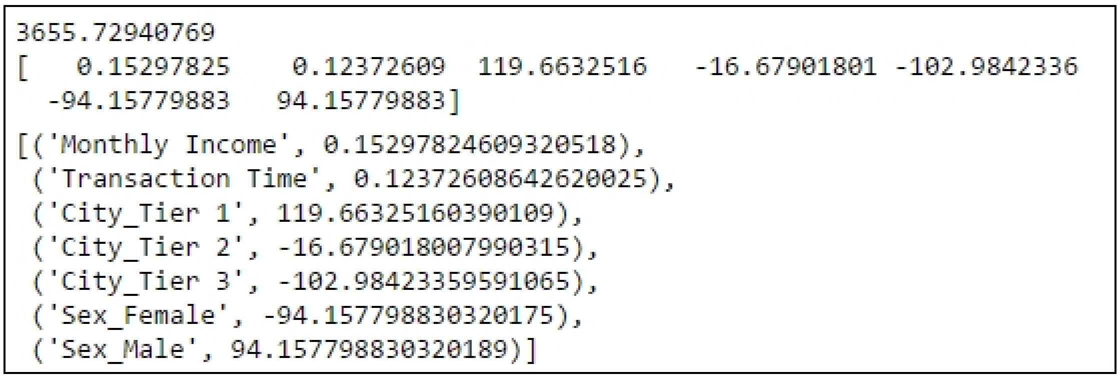
For the preceding dataset, let's assume a linear relationship between the output variable `Total Spend` and the predictor variables: `Monthly Income` and `Transaction Time`, and both set of dummy variables:

```
from sklearn.linear_model import LinearRegression
feature_cols = ['Monthly Income','Transaction Time','City_Tier
1','City_Tier 2','City_Tier 3','Sex_Female','Sex_Male']
X = df2[feature_cols]
Y = df2['Total Spend']
lm = LinearRegression()
lm.fit(X,Y)
```

The model parameters can be found out, as follows:

```
print lm.intercept_
print lm.coef_
zip(feature_cols, lm.coef_)
```

The following is the output we get:



```
3655.72940769
[   0.15297825    0.12372609  119.6632516   -16.67901801 -102.9842336
  -94.15779883   94.15779883]
[('Monthly Income', 0.15297824609320518),
 ('Transaction Time', 0.12372608642620025),
 ('City_Tier 1', 119.66325160390109),
 ('City_Tier 2', -16.679018007990315),
 ('City_Tier 3', -102.98423359591065),
 ('Sex_Female', -94.157798830320175),
 ('Sex_Male', 94.157798830320189)]
```

Coefficients of the model

The R^2 for this model can be found out by writing the following:

```
lm.score(X,Y)
```

The value comes out to be `0.19`, which might be because we haven't used the other variables and the output might be related to them as well. We need to fine-tune the model by suitably transforming some of the variables and adding them to the model. For example, if you add `Record` variable to the model, the R^2 jumps to 0.91 (try that on your own). It is a good dataset to play with.

The model can be written, as follows:

*Total_Spend=3655.72 + 0.12*Transaction Time + 0.15*Monthly Income + 119*City_Tier 1-16*City_Tier 2 - 102*City_Tier 3-94*Sex_Female+94*Sex_Male*

The RSE can be calculated, as follows:

```
import numpy as np
df2['total_spend_pred']=3720.72940769 + 0.12*df2['Transaction
Time']+0.15*df2['Monthly Income']+119*df2['City_Tier 1']-16*df2['City_Tier
2']
-102*df2['City_Tier 3']-94*df2['Sex_Female']+94*df2['Sex_Male']
df2['RSE']=(df2['Total Spend']-df2['total_spend_pred'])**2
RSEd=df2.sum()['RSE']
RSE=np.sqrt(RSEd/2354)
salesmean=np.mean(df2['Total Spend'])
error=RSE/salesmean
RSE,salesmean,error
```

The RSE comes out to be `2519` over a `Total Spend` mean of 6163, amounting to an error of around 40%, suggesting that there is a scope for improvement in the model.

However, the purpose of this section is to illustrate how the dummy variables are used in building a model and assessed in the final model.

As we can see, there are different coefficients for different dummy variables. For **City Tier**, **City_Tier_1** has 119, **City_Tier_2** has -16 and **City_Tier_3** has -102. This means that on an average, everything else being same, a customer from a **Tier 1** city will spend more than someone from **Tier 2** and **Tier 3** city. Someone from a **Tier 2** city will spend less than someone from **Tier 3**. If we take **City_Tier_1** as the baseline, the **Total Spend** is lesser by 135 units for a customer from **Tier 2** city, while it is lesser by 222 units for a customer from **Tier 3** city.

For different **Gender** and **City Tier**, the model will be reduced to following for different cases:

Gender	**City Tier**	**Model**
Male	1	*Total_Spend=3655.72 + 0.12*Transaction Time + 0.15*Monthly Income + 119*City_Tier 1 +94*Sex_Male*
Male	2	*Total_Spend=3655.72 + 0.12*Transaction Time + 0.15*Monthly Income -16*City_Tier 2 +94*Sex_Male*
Male	3	*Total_Spend=3655.72 + 0.12*Transaction Time + 0.15*Monthly Income - 102*City_Tier 3 +94*Sex_Male*
Female	1	*Total_Spend=3655.72 + 0.12*Transaction Time + 0.15*Monthly Income + 119*City_Tier 1 - 94*Sex_Female*
Female	2	*Total_Spend=3655.72 + 0.12*Transaction Time + 0.15*Monthly Income -16*City_Tier 2 - 94*Sex_Female*

One of the three dummy variables can be converted to baseline by masking it. Remember, we said earlier that only (n-1) dummy variables are needed for a categorical variable with n levels. However, here you are seeing three dummy variable for **City Tier** (three levels) and two dummy variables for *Gender* (two levels). This is just because it is easier to understand this way. There is only n-1 variables required for n-level categorical variables.

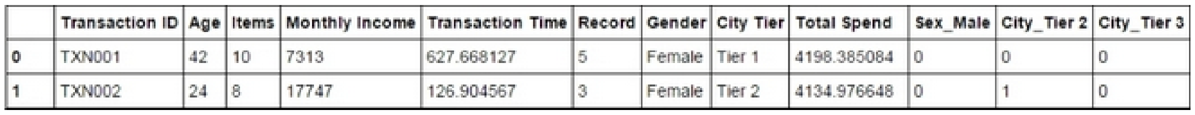
We can use (n-1) variables by masking one of the variables from the list of dummy variables going into the model. This masked variable will then become the baseline for the coefficients associated with these dummy variables.

Let's do that and see how the coefficients change:

```
dummy_gender=pd.get_dummies(df['Gender'],prefix='Sex').iloc[:, 1:]
dummy_city_tier=pd.get_dummies(df['City Tier'],prefix='City').iloc[:, 1:]
column_name=df.columns.values.tolist()
df3=df[column_name].join(dummy_gender)
column_name1=df3.columns.values.tolist()
df4=df3[column_name1].join(dummy_city_tier)
df4
```

It is the same process of converting categorical variables to dummy variables, but we are masking the first variable from the resulting list using the `iloc` method of subsetting.

The resulting data frame has one dummy variable for **Gender** and two for **City Tier** and is similar to the following screenshot:



	Transaction ID	Age	Items	Monthly Income	Transaction Time	Record	Gender	City Tier	Total Spend	Sex_Male	City_Tier 2	City_Tier 3
0	TXN001	42	10	7313	627.668127	5	Female	Tier 1	4198.385084	0	0	0
1	TXN002	24	8	17747	126.904567	3	Female	Tier 2	4134.976648	0	1	0

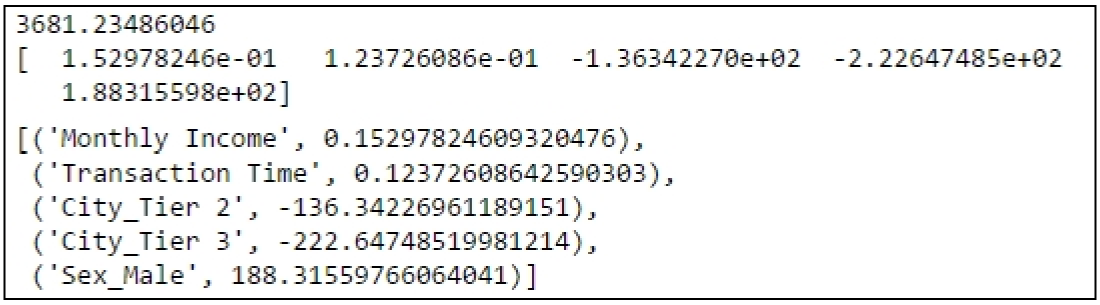
Ecom expense dataset with only (n-1) dummy variables

Let's now use these variables into the model and see how the coefficients change:

```
from sklearn.linear_model import LinearRegression
feature_cols = ['Monthly Income','Transaction Time','City_Tier
2','City_Tier 3','Sex_Male']
X = df2[feature_cols]
Y = df2['Total Spend']
lm = LinearRegression()
lm.fit(X,Y)
```

The variables and their coefficients can be obtained in the same way as earlier. They are as follows:

```
print lm.intercept_
print lm.coef_
zip(feature_cols, lm.coef_)
```



```
3681.23486046
[  1.52978246e-01   1.23726086e-01  -1.36342270e+02  -2.22647485e+02
   1.88315598e+02]
[('Monthly Income', 0.15297824609320476),
 ('Transaction Time', 0.12372608642590303),
 ('City_Tier 2', -136.34226961189151),
 ('City_Tier 3', -222.64748519981214),
 ('Sex_Male', 188.31559766064041)]
```

Coefficients of the model

As one might observe, the coefficients of **City_Tier_2** and **City_Tier_3** variables along with that of the **Sex_Male** variables have changed while that of all the others remain the same. The change in the coefficient doesn't change the model as such, but just the account for the absence of the baseline dummy variable. The new coefficient for **City_Tier_2** is -136, which can be thought of as its coefficient when the **City_Tier_1** has a coefficient of 0 (we saw earlier it has a coefficient of 119):

Variable	**Coefficient earlier**	**Coefficient later**
City_Tier_1	120	0
City_Tier_2	-16	-136 (-16-120)
City_Tier_3	-102	-222(-102-120)
Sex_Male	94	188 (94-(-94))
Sex_Female	-94	0

Transforming a variable to fit non-linear relations

Sometimes the output variable doesn't have a direct linear relationship with the predictor variable. They have a non-linear relationship. These relationships could be simple functions like quadratic, exponential, logarithm, or complex ones such as polynomials. In such cases, transforming the variable comes in very handy.

The following is a rough guideline about how to go about it:

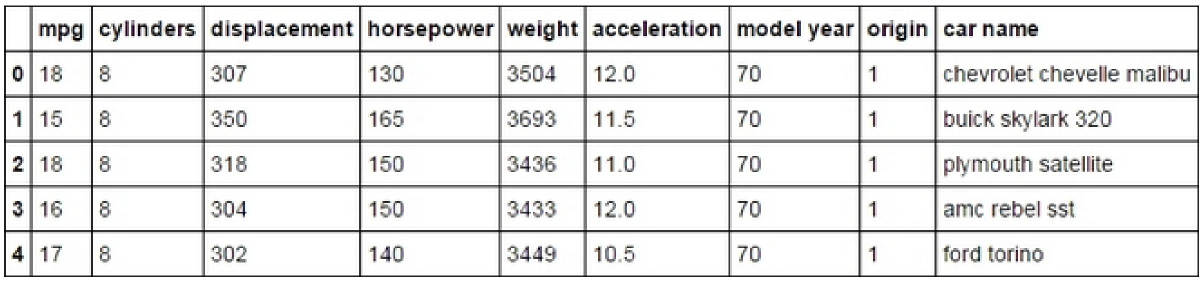
- Plot a scatter plot of the output variable with each of the predictor variables. This can be thought of as a scatter plot matrix similar to the correlation matrix.
- If the scatter plot assumes more or less a linear shape for a predictor variable then it is linearly related to the output variable.
- If the scatter plot assumes a characteristic shape of any of the non-linear shapes for a predictor variable then transform that particular variable by applying that function.

Let's illustrate this with one example. We will use the `Auto.csv` dataset for this. This dataset contains information about **miles per gallon** (**mpg**) and horsepower for a number of car models and much more. The mpg is the predictor variable and is considered to be highly dependent on the horsepower of a car model.

Let's import the dataset and have a look at it before proceeding further:

```
import pandas as pd
data = pd.read_csv('E:/Personal/Learning/Predictive Modeling Book/Book
Datasets/Linear Regression/Auto.csv')
data.head()
```

This is how the dataset looks:



	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name
0	18	8	307	130	3504	12.0	70	1	chevrolet chevelle malibu
1	15	8	350	165	3693	11.5	70	1	buick skylark 320
2	18	8	318	150	3436	11.0	70	1	plymouth satellite
3	16	8	304	150	3433	12.0	70	1	amc rebel sst
4	17	8	302	140	3449	10.5	70	1	ford torino

Auto dataset

It has 406 rows and 9 columns. Some of the variables have NA values and it makes sense to drop the NA values before using them.

Now, let's plot a scatter plot between the horsepower and the mpg variables to see whether they exhibit a linear shape or some non-linear shape:

```
import matplotlib.pyplot as plt
%matplotlib inline
data['mpg']=data['mpg'].dropna()
data['horsepower']=data['horsepower'].dropna()
```

```
plt.plot(data['horsepower'],data['mpg'],'ro')
plt.xlabel('Horsepower')
plt.ylabel('MPG (Miles Per Gallon)')
```

As can be seen in the output, the relationship doesn't seem to have a linear shape but rather assumes a non-linear shape; it is most probably an exponential or quadratic kind of relationship.

However, for the sake of comparison, let's try and fit a linear model for the relationship between mpg and horsepower first and then compare it with the scatter plot.

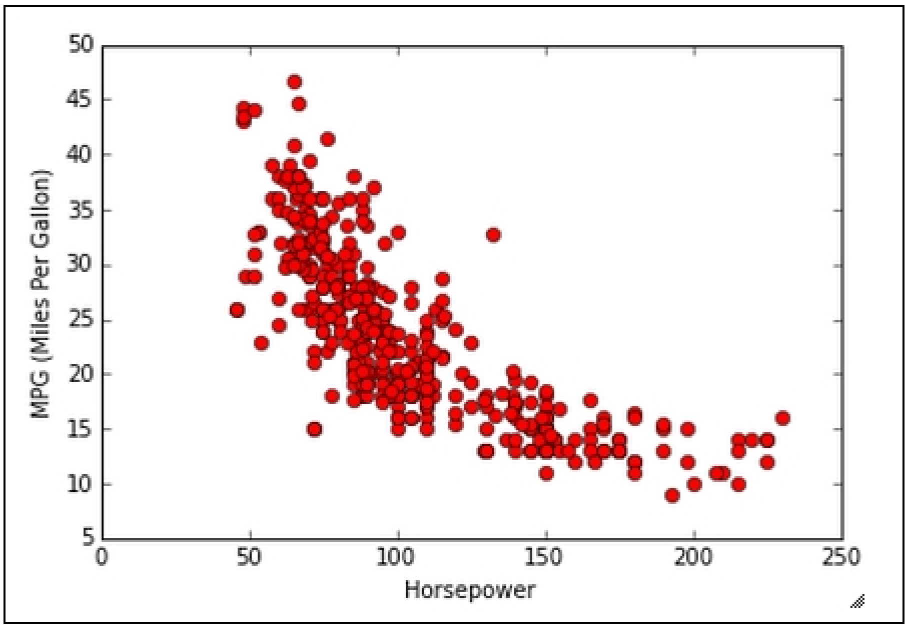
We are assuming that the model is:

$$mpg = c_0 + a_1.horsepower$$

While it looks like that the model is something similar to:

$$mpg = c_0 + a_1.horsepower^2$$

$$mpg = c_0 + a_1.horsepower + a_2.horsepower^2$$



Scatterplot of MPG vs Horsepower

The following code snippet will fit a linear model between `horsepower` and `mpg` variables. The NA values need to be dropped from the variables before they can be used in the model. Also simultaneously, let us create a model assuming a linear relationship between mpg and square of horsepower:

```
import numpy as np
from sklearn.linear_model import LinearRegression
```

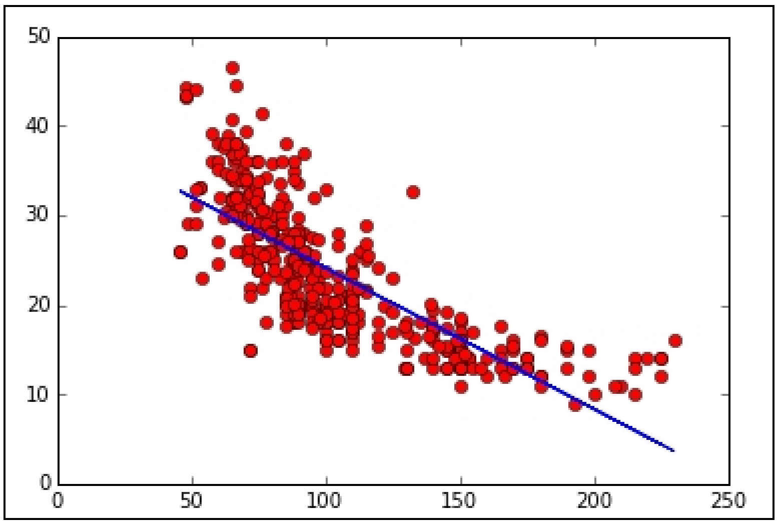
```
X=data['horsepower'].fillna(data['horsepower'].mean())
Y=data['mpg'].fillna(data['mpg'].mean())
lm=LinearRegression()
lm.fit(X[:,np.newaxis],Y)
```

The linear regression method by default requires that *X* be an array of two dimensions. Using `np.newaxis`, we are creating a new dimension for it to function properly.

The line of best fit can be plotted by the following snippet:

```
import matplotlib.pyplot as plt
%matplotlib inline
plt.plot(data['horsepower'],data['mpg'],'ro')
plt.plot(X,lm.predict(X[:,np.newaxis]),color='blue')
```

The plot looks similar to the following graph. The blue line is the line of the best fit:



The line of best fit (from the linear model) and the scatterplot

The R^2 for this model can be obtained using the following snippet:

```
lm1.score(X[:,np.newaxis],Y)
```

The value comes out to be 0.605.

Let's now calculate the RSE for this model in a different manner:

```
RSEd=(Y-lm.predict(X[:,np.newaxis]))**2
RSE=np.sqrt(np.sum(RSEd)/389)
ymean=np.mean(Y)
error=RSE/ymean
RSE,error
```

Here, we are using the predict method to calculate the predicted value from the model instead of writing them explicitly.

The value of RSE for this model comes out to be 5.14, which over a mean value of 23.51 gives an error of 21%.

If the model is of the form $mpg = c_o + a_1.horsepower^2$, then it can be fitted after transforming the horsepower variable, as shown in the following snippet:

```
import numpy as np
from sklearn.linear_model import LinearRegression
X=data['horsepower'].fillna(data['horsepower'].mean())*data['horsepower'].f
illna(data['horsepower'].mean())
Y=data['mpg'].fillna(data['mpg'].mean())
lm=LinearRegression()
lm.fit(X[:,np.newaxis],Y)
```

The R^2 value for this model comes out to be around 0.51 and there is a scope of improvement in this model. The RSE can be calculated in the same manner, as shown in the preceding section using the following code snippet:

```
type(lm.predict(X[:,np.newaxis]))
RSEd=(Y-lm.predict(X[:,np.newaxis]))**2
RSE=np.sqrt(np.sum(RSEd)/390)
ymean=np.mean(Y)
error=RSE/ymean
RSE,error,ymean
```

The value of RSE for this model comes out to be 10.51, which over a mean value of 23.51 gives an error of 45%. The RSE increased when we transformed the variable exponentially.

Thus, we need to look at some other method to fit this seemingly non-linear data. What about polynomial fits that are, as follows:

$$Model: \ mpg = c_0 + a_1.horsepower + a_2.horsepower^2$$

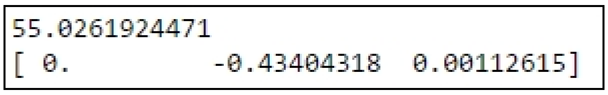
This can be fitted using the `PolynomialFeatures` method in the `scikit-learn` library. In this model, we are assuming a polynomial relationship between mpg and horsepower:

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn import linear_model
X=data['horsepower'].fillna(data['horsepower'].mean())
Y=data['mpg'].fillna(data['mpg'].mean())
poly = PolynomialFeatures(degree=2)
X_ = poly.fit_transform(X[:,np.newaxis])
clf = linear_model.LinearRegression()
clf.fit(X_, Y)
```

The `PolynomialFeatures` method used in this method automatically generates the powers (up to the specified degree) of the `X` variable using its `transform` feature. In this case, the R^2 value comes out to be 0.688. The R^2 value increased considerably when we introduced the polynomial regression.

The coefficients for this model come out to be, as follows:

```
print clf.intercept_
print clf.coef_
```



```
55.0261924471
[ 0.         -0.43404318  0.00112615]
```

Model coefficients

The model can be written as:

$$mpg = 55.02 - 0.43 * horsepower + 0.001 * horsepower^2$$

Let us increase the degree and see whether it increases the R^2 further or not. One just needs to change the degree from 2 to 5.

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn import linear_model
X=data['horsepower'].fillna(data['horsepower'].mean())
Y=data['mpg'].fillna(data['mpg'].mean())
poly = PolynomialFeatures(degree=5)
X_ = poly.fit_transform(X[:,np.newaxis])
clf = linear_model.LinearRegression()
clf.fit(X_, Y)
```

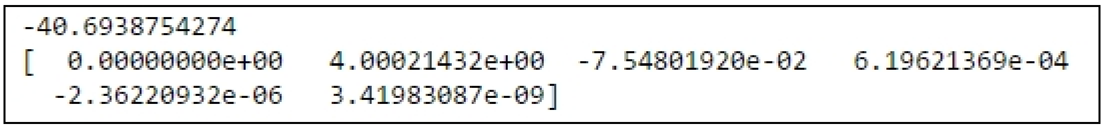
The model in this case will be:

$$mpg = c_0 + a_1.horsepower + a_2.horsepower^2 + a_3.horsepower^3 + a_4.horsepower^4 + a_5.horsepower^5$$

The R^2 for this model comes out to be 0.70. After that degree, an increase in degree doesn't improve the value of R^2.

The model coefficients can be found, as follows:

```
print clf.intercept_
print clf.coef_
```



```
-40.6938754274
[  0.00000000e+00   4.00021432e+00  -7.54801920e-02   6.19621369e-04
  -2.36220932e-06   3.41983087e-09]
```

Model coefficients

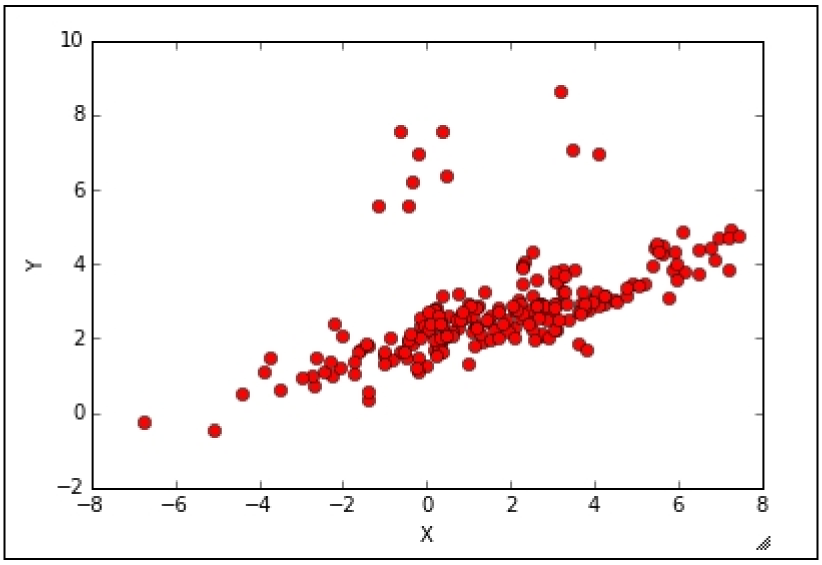
The reader can try the models of higher degrees as well and see how the coefficients change and whether it improves the model further (minimizing the error). The reader can also try to plot the results after the polynomial fit to see how it has improved the results.

Handling outliers

Outliers are the points in the dataset that are way out of the league of the other points. If a scatterplot of the concerned variable is drawn, the outliers can be easily identified, as they lay significantly away from the other data points.

The outliers need to be removed or properly treated before using the dataset for modelling. The outliers can distort the model and reduce its efficacy even if they are less in number, compared to the size of the dataset. As low as 1% outlier data is also capable enough to distort the model. It is actually not the number of outlier points but the degree to which it is different from an average point that determines the degree of distortion.

Let us look at a scatterplot of a dataset that has outliers:

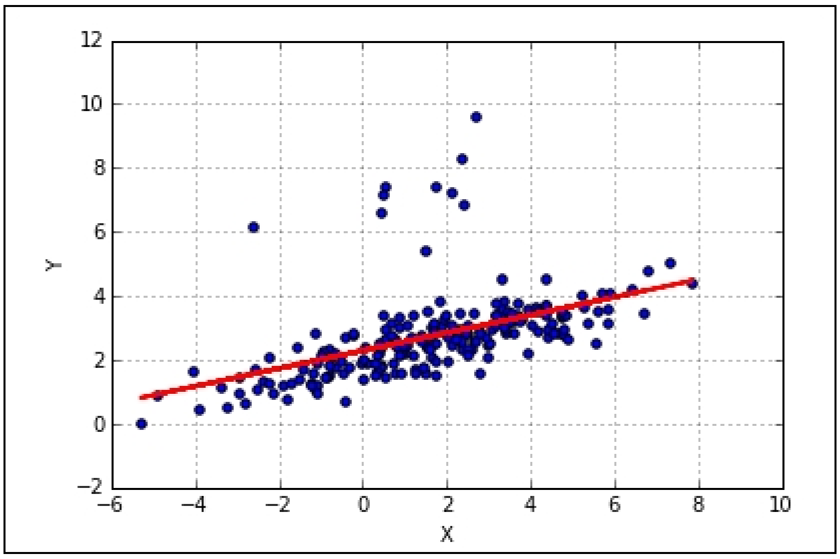


A dataset with outliers. Outliers in the encircled region

As we can see, the points in the encircled region lie away from where the majority of the points lie. These points lying in the encircled regions are outliers.

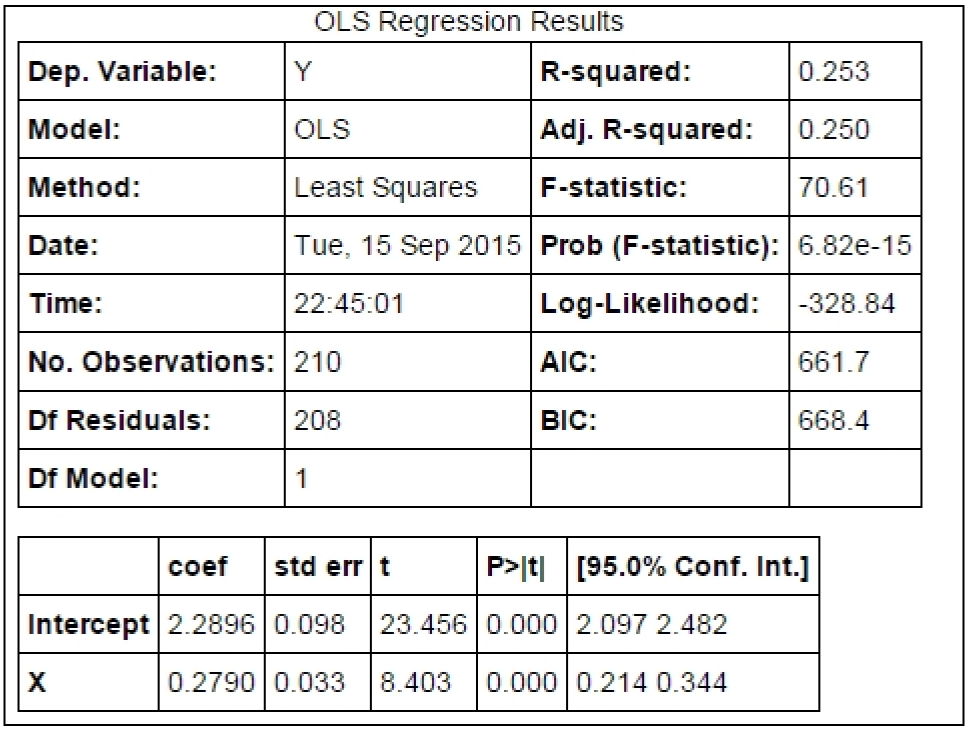
Let's now see how it affects the modelling process. To illustrate that, let's look at the result of a linear regression model built upon the dataset with outliers. We will then compare this result to the results of a linear regression model derived from the dataset from which the outliers have been removed.

The best fit line for the model developed from the dataset with outliers looks similar to the following screenshot:



Best fit line for the linear regression model developed over dataset with outliers

The model summary for this model is, as follows:



OLS Regression Results

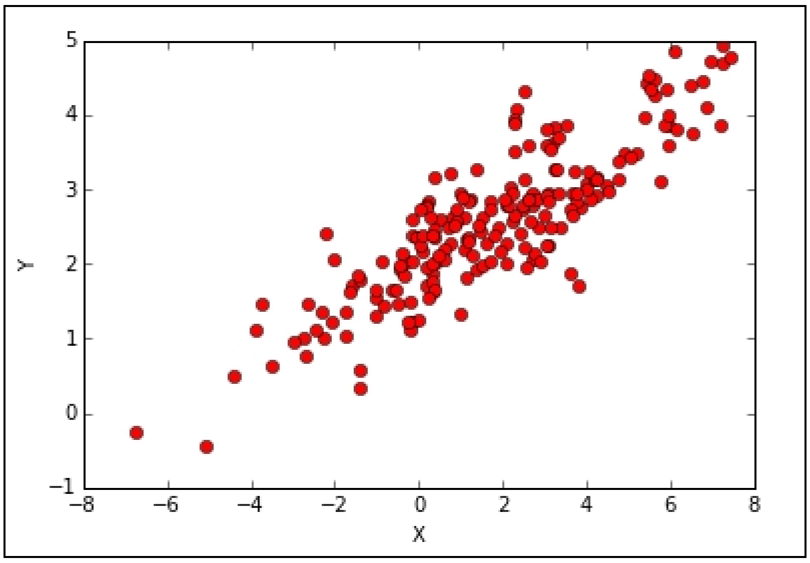
Dep. Variable:	Y	**R-squared:**	0.253
Model:	OLS	**Adj. R-squared:**	0.250
Method:	Least Squares	**F-statistic:**	70.61
Date:	Tue, 15 Sep 2015	**Prob (F-statistic):**	6.82e-15
Time:	22:45:01	**Log-Likelihood:**	-328.84
No. Observations:	210	**AIC:**	661.7
Df Residuals:	208	**BIC:**	668.4
Df Model:	1		

	coef	**std err**	**t**	**P>|t|**	**[95.0% Conf. Int.]**
Intercept	2.2896	0.098	23.456	0.000	2.097 2.482
X	0.2790	0.033	8.403	0.000	0.214 0.344

Summary of the linear regression model developed over dataset with outliers

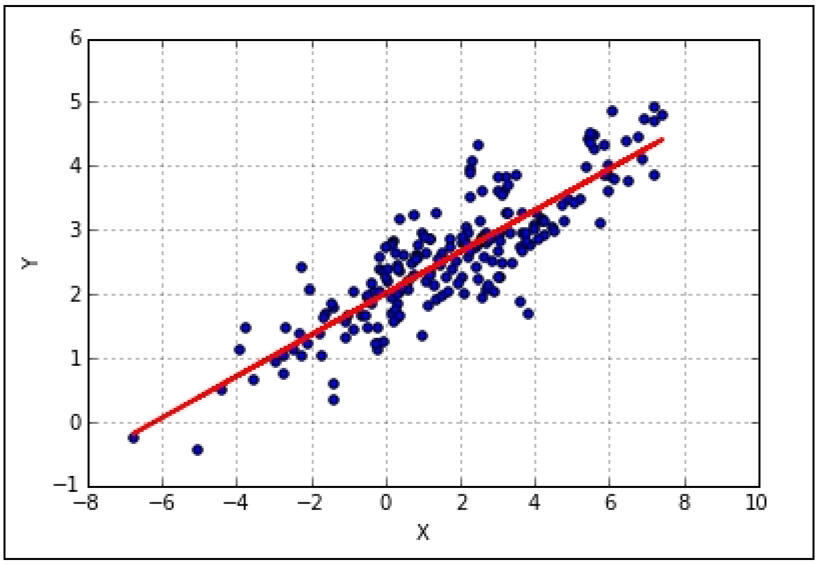
Let's now remove the outliers from the data and run the same linear regression model to check whether there is an improvement in the model or not.

The data without the outliers looks similar to the following screenshot:



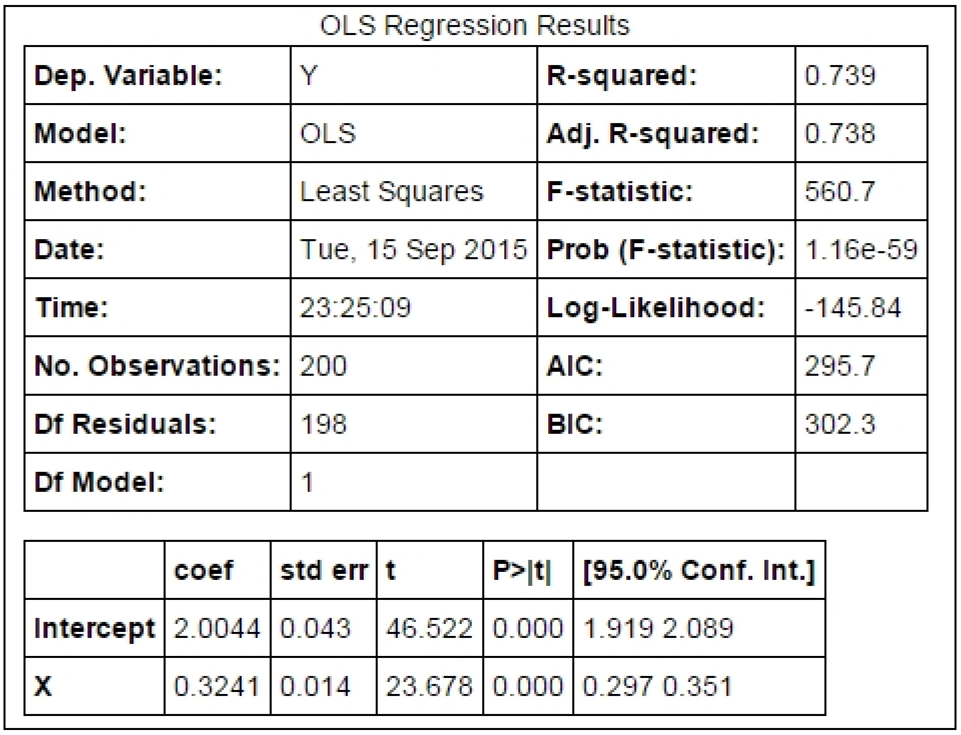
The dataset after removing outliers

The best fit line for the model developed from the dataset without outliers looks as follows:



Best fit line for the linear regression model developed over dataset without outliers

The model summary for this model is as follows:



OLS Regression Results

Dep. Variable:	Y	**R-squared:**	0.739
Model:	OLS	**Adj. R-squared:**	0.738
Method:	Least Squares	**F-statistic:**	560.7
Date:	Tue, 15 Sep 2015	**Prob (F-statistic):**	1.16e-59
Time:	23:25:09	**Log-Likelihood:**	-145.84
No. Observations:	200	**AIC:**	295.7
Df Residuals:	198	**BIC:**	302.3
Df Model:	1		

	coef	**std err**	**t**	**P>|t|**	**[95.0% Conf. Int.]**
Intercept	2.0044	0.043	46.522	0.000	1.919 2.089
X	0.3241	0.014	23.678	0.000	0.297 0.351

Summary of the linear regression model developed over the dataset without outliers

As we can see, the model has improved significantly especially in the terms of R^2 and F-statistic and the associated p-values. The model coefficients have also changed.

The following table is a comparison between the two models in terms of the parameters:

Parameter	Model w/o outliers	Model with outlier
R2	0.739	0.253
F-statistic	560.7	70.61
Coefficient	2.004	2.289
Intercept	0.3241	0.279
RSE	19.5%	41.4%

As it can be seen in the preceding comparison table, the model without outliers is better than the model with outliers in all the aspects. Thus, it is essentials to check for outliers in variables of the dataset and remove them from the dataset before using it for modelling.

The following are the ways in which one can identify outliers in the dataset:

- Plotting a scatter plot of the concerned variable.
- Boxplots are potent tools to spot outliers in a distribution. Any value *1.5*IQR* below the 1st quartile and *1.5*IQR* above the 1st quartile can be classified as an outlier. The difference in the 1st and 3rd quartile values is called the **Inter Quartile Range** (**IQR**).
- Another method is to calculate the error (the difference between the actual value and the value predicted from the model) and set a cut-off for the error. Anything outside this cut-off will be an outlier.

Other considerations and assumptions for linear regression

There are certain assumptions and considerations that need to be taken into account before finalizing on the model. Here are some of these.

Residual plots: The residual is the difference between the actual value and the predicted value of the output variable. A plot of the residuals plotted against the predictor variable should be randomly (normally with mean zero and constant variance) distributed and shouldn't have an identifiable shape. If the residual follows a characteristic curve, then it means that these errors can be predicted, which means something is wrong with the model and there is a scope for improvement. The error term for a fair estimate should be random and that's why if the residual plot shows a characteristic pattern there is a reason to improvise upon the model.

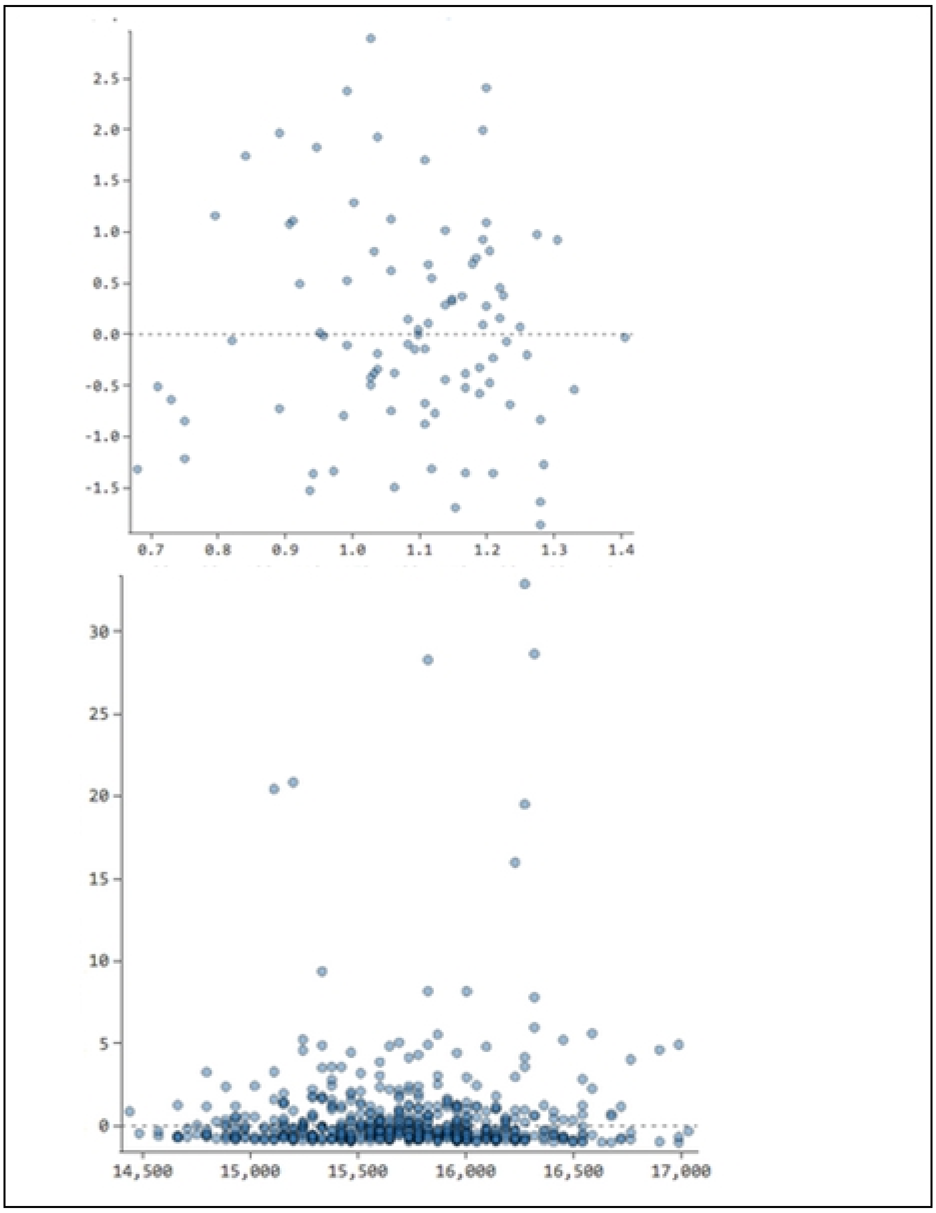
Ideally, the points in a residual plot should be:

- Symmetrically distributed or tending to cluster towards the middle of the plot
- There are no clear patterns in the plot

The non-ideal residual plots having characteristic shapes are observed because of one of the following reasons:

- Non-linear relationship
- Presence of outliers
- Very large *Y*-axis datapoint

This can be taken care of by transforming the variable or removing the outlier:



What residual plot should look like vs. what residual plot shouldn't look like

Non-constant variance of error terms: The error term associated with the model is assumed to have a constant variance and several calculations, standard errors, and confidence intervals. Hypothesis tests rely upon this assumption.

This problem is called **heteroscedasticity** and can be identified by a funnel shaped pattern in the residual plot. Transforming the output variable using a concave function, such as *sqrt(Y)* or *log(Y)* generally solves the problem.

High leverage points: In contrast to outliers, which have high values for the output variables, the high leverage points have a very high value of predictor variables. Such a value can distort the model. For a model with single predictor, it is easy to identify this issue. It can be done in the same way, as in the case of outliers.

It is difficult to do so in case of multiple regressions, where there are more than one predictor variable. In this case, a variable can have a value which brings a considerable change in the output variable compared to the same change in another variable; such points are called high leverage points. These need to be removed and sometimes, their removal increases the efficiency of the model more than the removal of the outlier.

To identify high leverage points in such cases, one calculates something called leverage statistics, which is defined as:

$$Leverage = \frac{1}{n} + \frac{(xi - xm)^2}{\sum (x'i - xm)^2}$$

Where:

- *xi*: This is the value of the i^{th} row of predictor variable *x*
- *xm*: This is the mean of the predictor variable *x*

The denominator is summed over all the variables for that particular row.

The rows with high values of leverage statistics are ruled out of the dataset before kicking off the modelling process.