Course name: **Python programming and analytics by Rahul Sir**

Topic name: **multiple Linear Regression (Manav Batch)**

Video name: **multi linear regression petroleum consumption case study**

Video length: **48 minutes 35 seconds**

Multiple linear regression (MLR), also known simply as multiple regression, is a statistical technique that uses several explanatory variables to predict the outcome of a response variable. The goal of multiple linear regression (MLR) is to model the linear relationship between the explanatory (independent) variables and response (dependent) variable.

- Multiple linear regression is the most common form of linear regression analysis.
- Multiple linear regression is used to explain the relationship between one continuous dependent variable from two or more independent variables.
- The independent variables can be continuous or categorical (dummy coded as appropriate)
- Independent variables should not be multi-collinear
- y = b0 + b1X1 + b2X2 + b3X3 + ………. + bnXn

y is the dependent variable and x1, x2, x3, xn are the independent variables; The value of b_0, also called the **intercept**, shows the point where the estimated regression line crosses the y axis. The value of b_1 determines the **slope** of the estimated regression line.

Steps 1: provide data

```
In [4]: data1 = pd.read_csv('C:/Users/Sahibjot/Desktop/linear regression/petrol consumption.csv')
```

```
In [50]: data1.head()
```

Out[50]:

	Index	Petrol tax (cents per gallon)	Average income (dollars)	Paved Highways (miles)	Population_Driver_licence(%)	Consumption of petrol (millions of gallons)
0	1	9.0	3571	1976	0.525	541
1	2	9.0	4092	1250	0.572	524
2	3	9.0	3865	1586	0.580	561
3	4	7.5	4870	2351	0.529	414
4	5	8.0	4399	431	0.544	410

```
In [51]: data1.shape
Out[51]: (48, 6)
```

```
In [52]: data1.describe()
```

Out[52]:

	Index	Petrol tax (cents per gallon)	Average income (dollars)	Paved Highways (miles)	Population_Driver_licence(%)	Consumption of petrol (millions of gallons)
count	48.00	48.000000	48.000000	48.000000	48.000000	48.000000
mean	24.50	7.668333	4241.833333	5565.416667	0.570333	576.770833
std	14.00	0.950770	573.623768	3491.507166	0.055470	111.885816
min	1.00	5.000000	3063.000000	431.000000	0.451000	344.000000
25%	12.75	7.000000	3739.000000	3110.250000	0.529750	509.500000
50%	24.50	7.500000	4298.000000	4735.500000	0.564500	568.500000
75%	36.25	8.125000	4578.750000	7156.000000	0.595250	632.750000
max	48.00	10.000000	5342.000000	17782.000000	0.724000	968.000000

Pd.read_csv() function is used to import the data. In the brackets, you need to mention the location of the data file in your computer/laptop and it must be in quotes.

To know how many rows and columns are there in the data, use data.shape

Data.describe() function is used to view some basic statistical details like percentile, mean, std etc. of a data frame or a series of numeric values.

Step 2: Set up the dependent and the independent variables

```
In [3]: X = data1[['Petrol tax (cents per gallon)', 'Average income (dollars)', 'Paved Highways (miles)',
               'Population_Driver_licence(%)']]
        y = data1['Consumption of petrol (millions of gallons)']     # double brackets: to store the data in dataframe format
```

X are the independent variables and y is the dependent variable.

Step3: split the data

You can't possibly manually split the dataset into two. And you also have to make sure you split the data in a random manner. To help us with this task, the SciKit library provides a tool, called the Model Selection library. There's a class in the library which is, aptly, named '[train_test_split](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html).' Using this we can easily split the dataset into the training and the testing datasets in various proportions.

- **test_size** — This parameter decides the size of the data that has to be split as the test dataset. This is given as a fraction. For example, if you pass 0.2 as the value, the dataset will be split 20% as the test dataset.
- **train_size** — You have to specify this parameter only if you're not specifying the test_size. This is the same as test_size, but instead you tell the class what percent of the dataset you want to split as the training set.
- **random_state** — Here you pass an integer, which will act as the seed for the random number generator during the split.

```
In [2]: from sklearn.model_selection import train_test_split
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
        # random state is used to fix the data so that it doesn't change when re-run
```

if you want to check how many rows and columns are there in train and test data for X and Y

```
In [29]: print(X_train.shape)
         print(X_test.shape)
         print(y_train.shape)
         print(y_test.shape)

(33, 4)
(15, 4)
(33,)
(15,)
```

Step 4: Create a model and fit it

```
In [1]: from sklearn.linear_model import LinearRegression
        regressor = LinearRegression()
        regressor.fit(X_train, y_train)
```

Step 5: Retrieve the intercept

```
In [36]: print(regressor.intercept_)

470.36371826645154
```

The intercept (often labeled the constant) is the expected mean value of Y when all X=0.

Step 6: Predictions

```
In [37]: y_pred = regressor.predict(X_test)

In [38]: y_pred

Out[38]: array([468.31594627, 550.39707846, 590.63932111, 572.1767937 ,
                649.89394062, 648.44378918, 515.19865018, 674.76463673,
                503.47637809, 500.07361023, 417.31504469, 587.99614777,
                624.50820356, 605.30052585, 563.47052146])
```

Step 7: Comparing the predicted value to the actual value:

In [39]:
```
df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
df
```

Out[39]:

	Actual	Predicted
29	534	468.315946
4	410	550.397078
26	577	590.639321
30	571	572.176794
32	577	649.893941
37	704	648.443789
34	487	515.198650
40	587	674.764637
7	467	503.476378
10	580	500.073610
11	471	417.315045
31	554	587.996148
33	628	624.508204
27	631	605.300526
47	524	563.470521

Step 8: Evaluate the algorithm

In [40]:
```
from sklearn import metrics
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print('R squared:', metrics.r2_score(y_test,y_pred))
```

```
Mean Absolute Error: 49.20375655663114
Mean Squared Error: 3673.2072706922636
Root Mean Squared Error: 60.606990279111066
R squared: 0.29357534437288924
```

You can see that the value of root mean squared error is 60.60, which is slightly greater than 10% of the mean value of the gas consumption in all states (57.6). This means that our algorithm was not very accurate but can still make reasonably good predictions