A beginner's guide to Linear Regression in Python with Scikit-Learn

Nagesh Singh Chauhan Follow
Feb 25 · 11 min read

source

There are two types of supervised machine learning algorithms: Regression and classification. The former predicts continuous value outputs while the latter predicts discrete outputs. For instance, predicting the price of a house in dollars is a regression problem whereas predicting whether a tumor is malignant or benign is a classification problem.

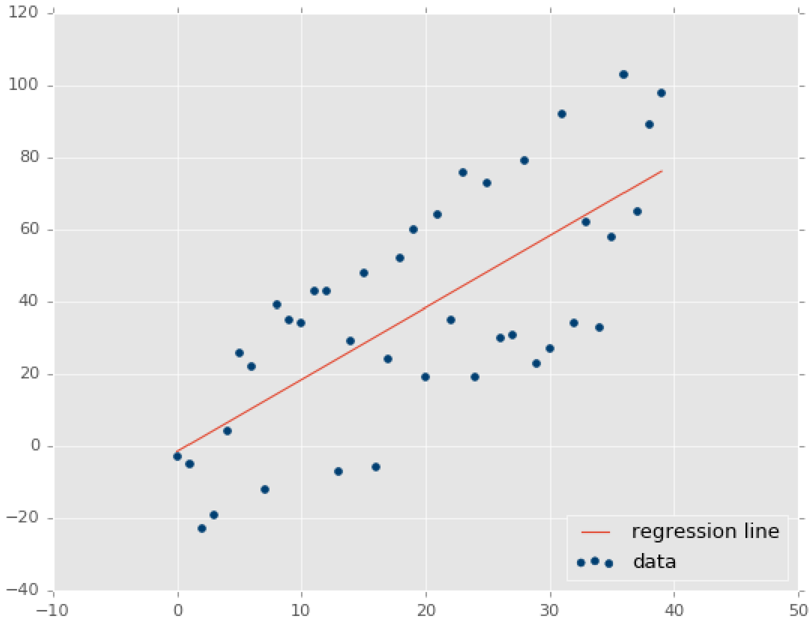
In this article, we will briefly study what linear regression is and how it can be implemented for both two variables and multiple variables using **Scikit-Learn,** which is one of the most popular machine learning libraries for Python.

Linear Regression Theory
The term "linearity" in algebra refers to a linear relationship between

two or more variables. If we draw this relationship in a two-dimensional space (between two variables), we get a straight line.

Linear regression performs the task to predict a dependent variable value (y) based on a given independent variable (x). So, this regression technique finds out a linear relationship between x (input) and y(output). Hence, the name is Linear Regression. If we plot the independent variable (x) on the x-axis and dependent variable (y) on the y-axis, linear regression gives us a straight line that best fits the data points, as shown in the figure below.

We know that the equation of a straight line is basically:



source

The equation of the above line is :

Y= mx + b

> *Where b is the intercept and m is the slope of the line. So basically, the linear regression algorithm gives us the most optimal value for the intercept and the slope (in two dimensions). The y and x variables remain the same, since they are the data features and cannot be changed. The values that we can control are the intercept(b) and slope(m). There can be multiple straight lines depending upon the values of intercept and slope. Basically what the linear regression algorithm does is it fits multiple lines on the data points and returns the line that results in the least error.*

This same concept can be extended to cases where there are more than two variables. This is called multiple linear regression. For instance, consider a scenario where you have to predict the price of the house based upon its area, number of bedrooms, the average income of the people in the area, the age of the house, and so on. In this case, the dependent variable(target variable) is dependent upon several independent variables. A regression model involving multiple variables can be represented as:

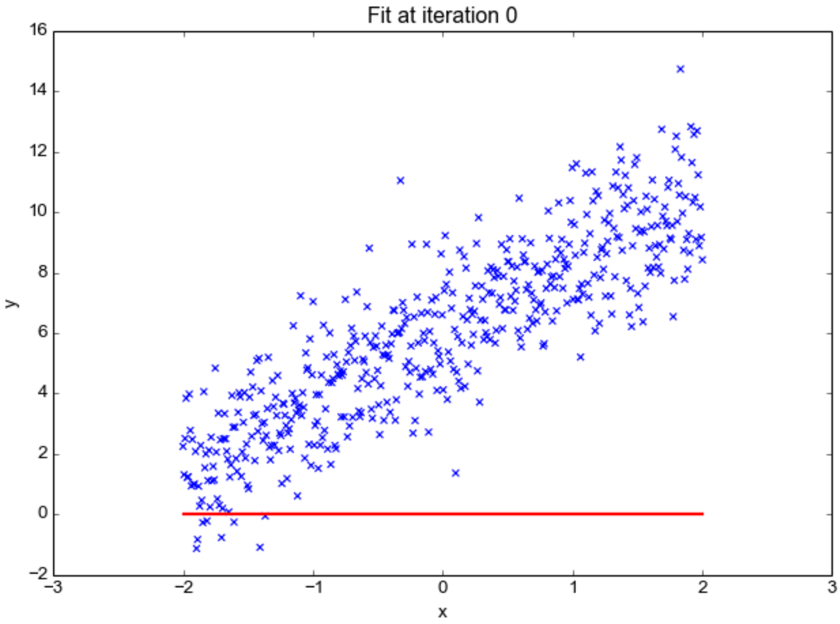
y = b0 + m1b1 + m2b2 + m3b3 + … … mnbn

This is the equation of a hyperplane. Remember, a linear regression model in two dimensions is a straight line; in three dimensions it is a plane, and in more than three dimensions, a hyperplane.

In this section, we will see how Python's Scikit-Learn library for machine learning can be used to implement regression functions. We will start with simple linear regression involving two variables and then we will move towards linear regression involving multiple variables.

. . .

Simple Linear Regression



Linear Regression

While exploring the Aerial Bombing Operations of World War Two dataset and recalling that the D-Day landings were nearly postponed due to poor weather, I downloaded these weather reports from the period to compare with missions in the bombing operations dataset.

You can download the dataset from **here**.

The dataset contains information on weather conditions recorded on each day at various weather stations around the world. Information includes precipitation, snowfall, temperatures, wind speed and whether the day included thunderstorms or other poor weather conditions.

So our task is to predict the maximum temperature taking input feature as minimum temperature.

Let's start coding :

Import all the required libraries :

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as seabornInstance
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
%matplotlib inline
```

The following command imports the CSV dataset using pandas:

```
dataset =
pd.read_csv('/Users/nageshsinghchauhan/Documents/projects/ML
/ML_BLOG_LInearRegression/Weather.csv')
```

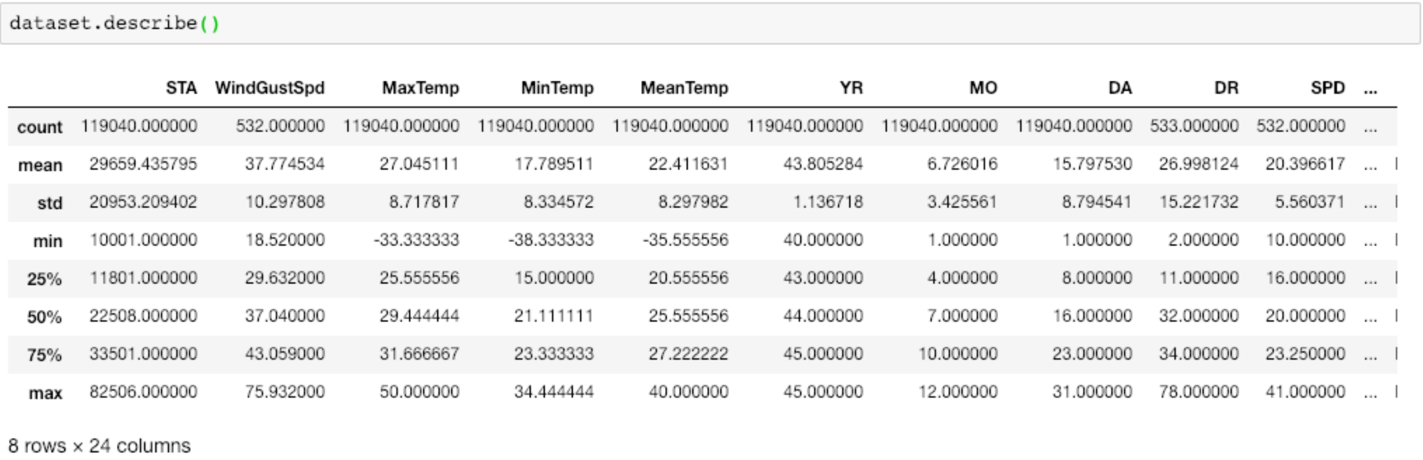
Let’s explore the data a little bit by checking the number of rows and columns in our datasets.

```
dataset.shape
```

You should receive output as (119040, 31), which means the data contains 119040 rows and 31 columns.

To see the statistical details of the dataset, we can use `describe()` :

```
dataset.describe()
```



```
dataset.describe()
```

	STA	WindGustSpd	MaxTemp	MinTemp	MeanTemp	YR	MO	DA	DR	SPD	...
count	119040.000000	532.000000	119040.000000	119040.000000	119040.000000	119040.000000	119040.000000	119040.000000	533.000000	532.000000	...
mean	29659.435795	37.774534	27.045111	17.789511	22.411631	43.805284	6.726016	15.797530	26.998124	20.396617	...
std	20953.209402	10.297808	8.717817	8.334572	8.297982	1.136718	3.425561	8.794541	15.221732	5.560371	...
min	10001.000000	18.520000	-33.333333	-38.333333	-35.555556	40.000000	1.000000	1.000000	2.000000	10.000000	...
25%	11801.000000	29.632000	25.555556	15.000000	20.555556	43.000000	4.000000	8.000000	11.000000	16.000000	...
50%	22508.000000	37.040000	29.444444	21.111111	25.555556	44.000000	7.000000	16.000000	32.000000	20.000000	...
75%	33501.000000	43.059000	31.666667	23.333333	27.222222	45.000000	10.000000	23.000000	34.000000	23.250000	...
max	82506.000000	75.932000	50.000000	34.444444	40.000000	45.000000	12.000000	31.000000	78.000000	41.000000	...

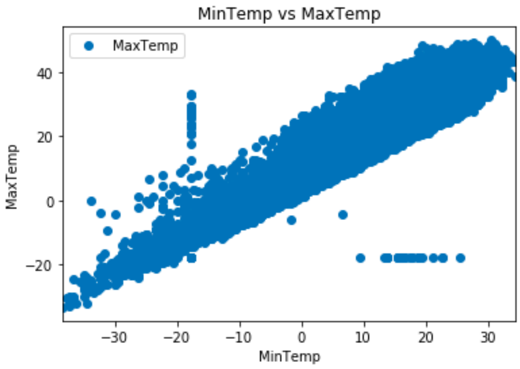
8 rows × 24 columns

statistical view of the dataset

And finally, let’s plot our data points on a 2-D graph to eyeball our dataset and see if we can manually find any relationship between the data using the below script :

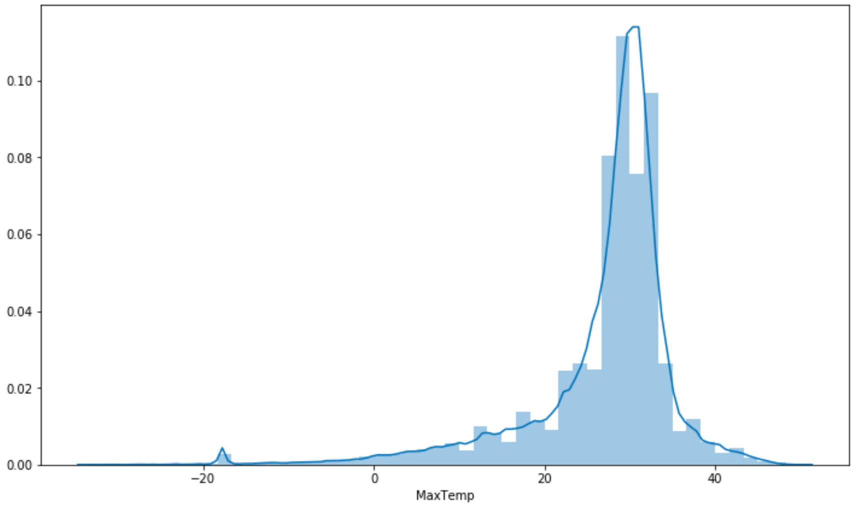
```
dataset.plot(x='MinTemp', y='MaxTemp', style='o')
plt.title('MinTemp vs MaxTemp')
plt.xlabel('MinTemp')
plt.ylabel('MaxTemp')
plt.show()
```

We have taken MinTemp and MaxTemp for doing our analysis. Below is a 2-D graph between MinTemp and MaxTemp.



Let’s check the average max temperature and once we plot it we can observe that the Average Maximum Temperature is Between Nearly 25 and 35.

```
plt.figure(figsize=(15,10))
plt.tight_layout()
seabornInstance.distplot(dataset['MaxTemp'])
```



Average maximum temperature which is in between 25 and 35.

Our next step is to divide the data into “attributes” and “labels”. Attributes are the independent variables while labels are dependent variables whose values are to be predicted. In our dataset, we only have two columns. We want to predict the MaxTemp depending upon the MinTemp recorded. Therefore our attribute set will consist of the “MinTemp” column which is stored in the X variable, and the label will be the “MaxTemp” column which is stored in y variable.

```
X = dataset['MinTemp'].values.reshape(-1,1)
y = dataset['MaxTemp'].values.reshape(-1,1)
```

Next, we split 80% of the data to the training set while 20% of the data to test set using below code.
The test_size variable is where we actually specify the proportion of the test set.

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=0)
```

After splitting the data into training and testing sets, finally, the time is to train our algorithm. For that, we need to import LinearRegression class, instantiate it, and call the `fit()` method along with our training data.

```
regressor = LinearRegression()
regressor.fit(X_train, y_train) #training the algorithm
```

As we have discussed that the linear regression model basically finds the best value for the intercept and slope, which results in a line that best fits the data. To see the value of the intercept and slop calculated by the linear regression algorithm for our dataset, execute the following code.

```
#To retrieve the intercept:
print(regressor.intercept_)


#For retrieving the slope:
print(regressor.coef_)
```

The result should be approximately 10.66185201 and 0.92033997 respectively.

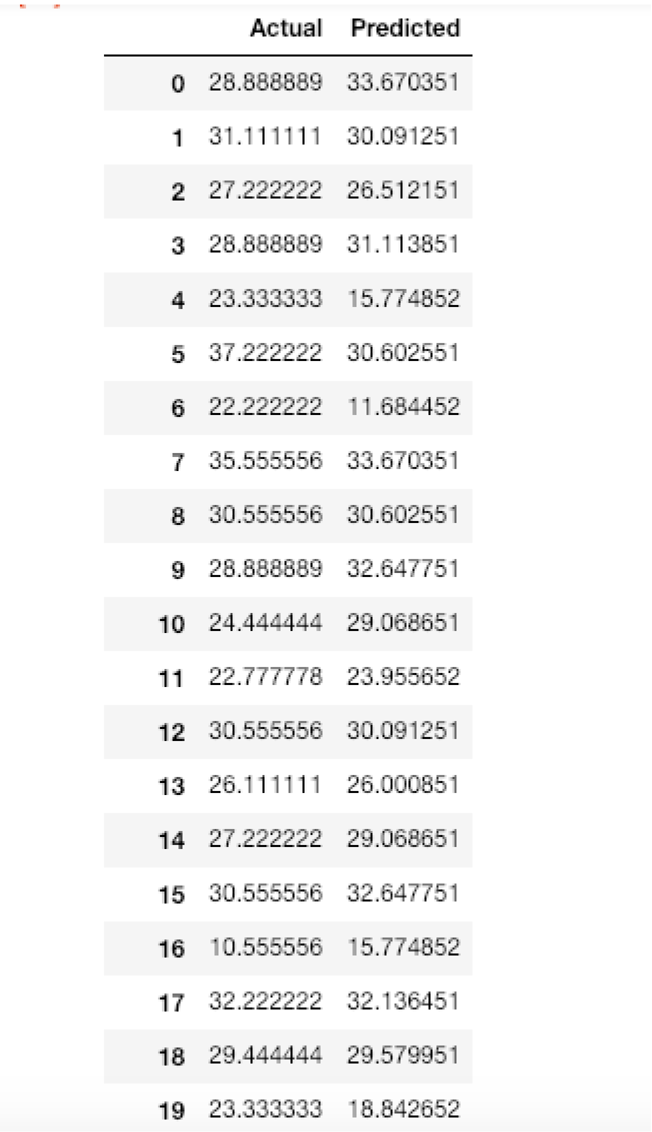
This means that for every one unit of change in Min temperature, the change in the Max temperature is about 0.92%.

Now that we have trained our algorithm, it’s time to make some predictions. To do so, we will use our test data and see how accurately our algorithm predicts the percentage score. To make predictions on the test data, execute the following script:

```
y_pred = regressor.predict(X_test)
```

Now compare the actual output values for `X_test` with the predicted values, execute the following script:

```
df = pd.DataFrame({'Actual': y_test.flatten(), 'Predicted':
y_pred.flatten()})
df
```



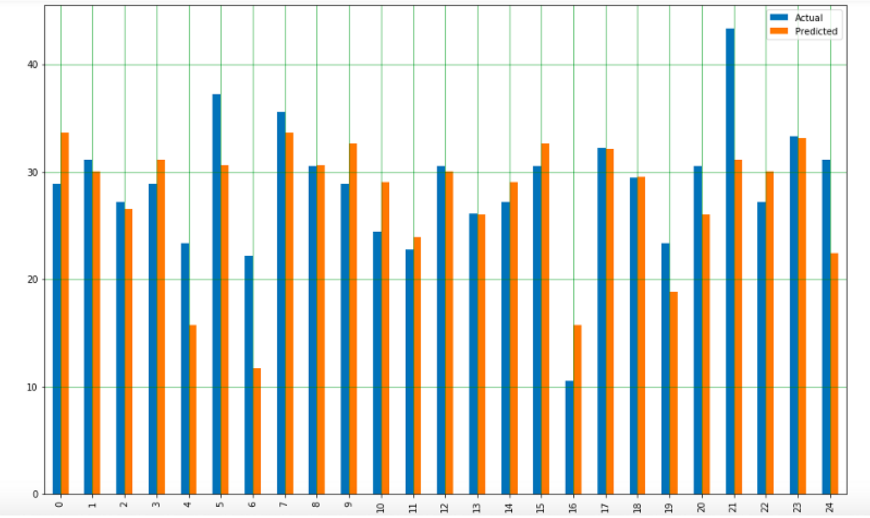
	Actual	Predicted
0	28.888889	33.670351
1	31.111111	30.091251
2	27.222222	26.512151
3	28.888889	31.113851
4	23.333333	15.774852
5	37.222222	30.602551
6	22.222222	11.684452
7	35.555556	33.670351
8	30.555556	30.602551
9	28.888889	32.647751
10	24.444444	29.068651
11	22.777778	23.955652
12	30.555556	30.091251
13	26.111111	26.000851
14	27.222222	29.068651
15	30.555556	32.647751
16	10.555556	15.774852
17	32.222222	32.136451
18	29.444444	29.579951
19	23.333333	18.842652

comparison of Actual and Predicted value

We can also visualize comparison result as a bar graph using the below script :

Note: As the number of records is huge, for representation purpose I'm taking just 25 records.

```
df1 = df.head(25)
df1.plot(kind='bar',figsize=(16,10))
plt.grid(which='major', linestyle='-', linewidth='0.5',
color='green')
plt.grid(which='minor', linestyle=':', linewidth='0.5',
color='black')
plt.show()
```

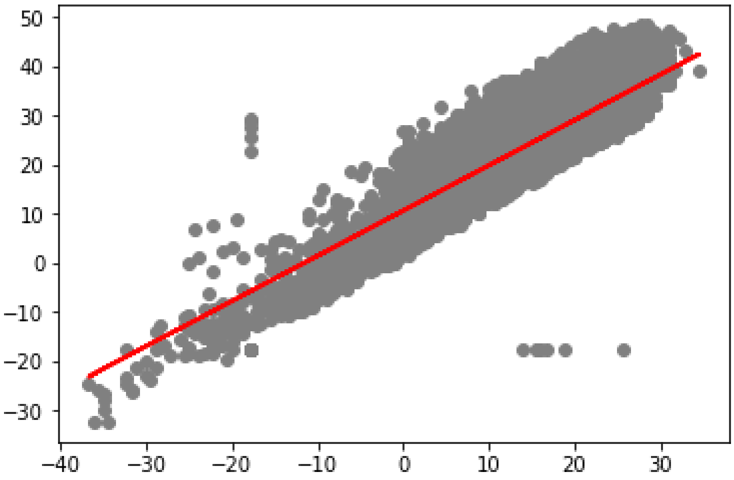


Bar graph showing the comparison of Actual and Predicted values.

Though our model is not very precise, the predicted percentages are close to the actual ones.

Let's plot our straight line with the test data :

```
plt.scatter(X_test, y_test,  color='gray')
plt.plot(X_test, y_pred, color='red', linewidth=2)
plt.show()
```



prediction vs test data

The straight line in the above graph shows our algorithm is correct.

The final step is to evaluate the performance of the algorithm. This step is particularly important to compare how well different algorithms perform on a particular dataset. For regression algorithms, three evaluation metrics are commonly used:

1. **Mean Absolute Error** (MAE) is the mean of the absolute value of the errors. It is calculated as:

$$\mathrm{MAE} = \frac{1}{n}\sum_{j=1}^{n} |y_j - y_j|$$

Mean Absolute Error

2. Mean Squared Error (MSE) is the mean of the squared errors and is calculated as:

$$MSE = \frac{1}{N}\sum_{i}^{n}(Y_i - y_i)^2$$

Mean Squared Error

3. Root Mean Squared Error (RMSE) is the square root of the mean of the squared errors:

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}$$

Root Mean Squared Error

Luckily, we don't have to perform these calculations manually. The Scikit-Learn library comes with pre-built functions that can be used to find out these values for us.

Let's find the values for these metrics using our test data.

```
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)))
```

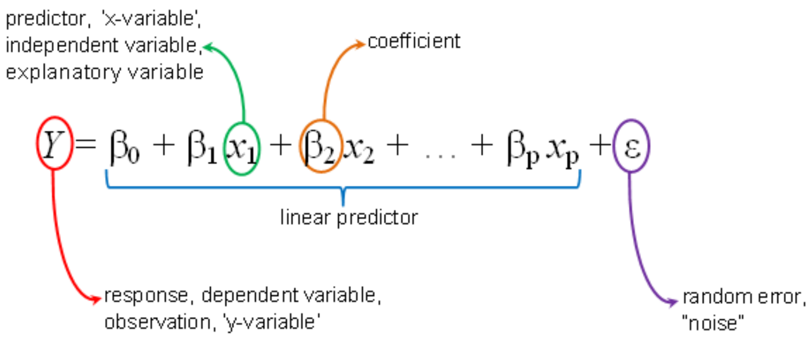
You should receive output like this (but probably slightly different):

```
('Mean Absolute Error:', 3.19932917837853)
('Mean Squared Error:', 17.631568097568447)
('Root Mean Squared Error:', 4.198996082109204)
```

You can see that the value of root mean squared error is 4.19, which is more than 10% of the mean value of the percentages of all the temperature i.e. 22.41. This means that our algorithm was not very accurate but can still make reasonably good predictions.

. . .

Multiple Linear Regression


predictor, 'x-variable',
independent variable,
explanatory variable
coefficient
$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_p x_p + \varepsilon$
linear predictor
response, dependent variable,
observation, 'y-variable'
random error,
"noise"

source

We just performed linear regression in the above section involving two variables. Almost all the real-world problems that you are going to encounter will have more than two variables. Linear regression involving multiple variables is called “multiple linear regression” or multivariate linear regression. The steps to perform multiple linear regression are almost similar to that of simple linear regression. The difference lies in the evaluation. You can use it to find out which factor has the highest impact on the predicted output and how different variables relate to each other.

In this section, I have downloaded red wine quality dataset. The dataset related to red variants of the Portuguese “Vinho Verde” wine. Due to privacy and logistic issues, only physicochemical (inputs) and sensory (the output) variables are available (e.g. there is no data about grape types, wine brand, wine selling price, etc.).

You can download the dataset from here.

We will take into account various input features like fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulphates, alcohol. Based on these features we will predict the quality of the wine.

Now, let's start our coding :

import all the required libraries :

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as seabornInstance
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
%matplotlib inline
```

The following command imports the dataset from the file you downloaded via the link above:

```
dataset =
pd.read_csv('/Users/nageshsinghchauhan/Documents/projects/ML
/ML_BLOG_LInearRegression/winequality.csv')
```

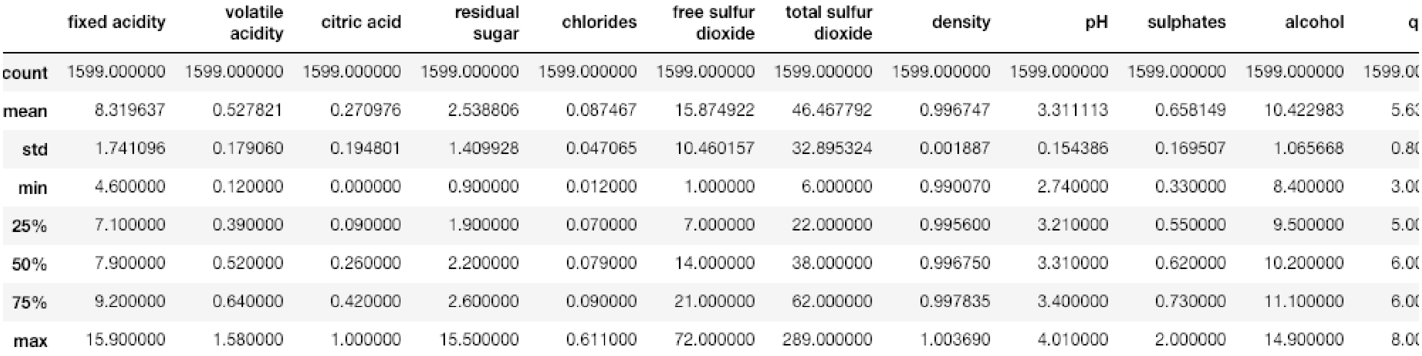
Let’s explore the data a little bit by checking the number of rows and columns in it.

```
dataset.shape
```

It will give (1599, 12) as output which means our dataset has 1599 rows and 12 columns.

To see the statistical details of the dataset, we can use `describe()` :

```
dataset.describe()
```



	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	q
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.00
mean	8.319637	0.527821	0.270976	2.538806	0.087467	15.874922	46.467792	0.996747	3.311113	0.658149	10.422983	5.63
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.895324	0.001887	0.154386	0.169507	1.065668	0.80
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.000000	0.990070	2.740000	0.330000	8.400000	3.00
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.000000	0.995600	3.210000	0.550000	9.500000	5.00
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.000000	0.996750	3.310000	0.620000	10.200000	6.00
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.000000	0.997835	3.400000	0.730000	11.100000	6.00
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.000000	1.003690	4.010000	2.000000	14.900000	8.00

Let us clean our data little bit, So first check which are the columns the contains NaN values in it :

```
dataset.isnull().any()
```

Once the above code is executed, all the columns should give False, In case for any column you find True result, then remove all the null values from that column using below code.

```
dataset = dataset.fillna(method='ffill')
```

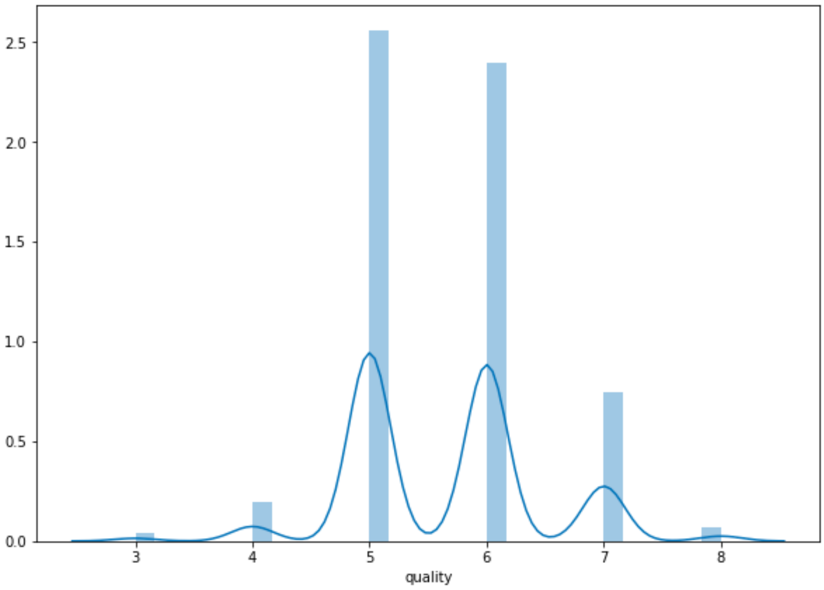
Our next step is to divide the data into “attributes” and “labels”. X variable contains all the attributes/features and y variable contains labels.

```
X = dataset[['fixed acidity', 'volatile acidity', 'citric
acid', 'residual sugar', 'chlorides', 'free sulfur dioxide',
'total sulfur dioxide', 'density', 'pH',
'sulphates','alcohol']].values

y = dataset['quality'].values
```

Let's check the average value of the “quality” column.

```
plt.figure(figsize=(15,10))
plt.tight_layout()
seabornInstance.distplot(dataset['quality'])
```



Average value of the quality of the wine.

As we can observe that most of the time the value is either 5 or 6.

Next, we split 80% of the data to training set while 20% of the data to test set using below code.

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=0)
```

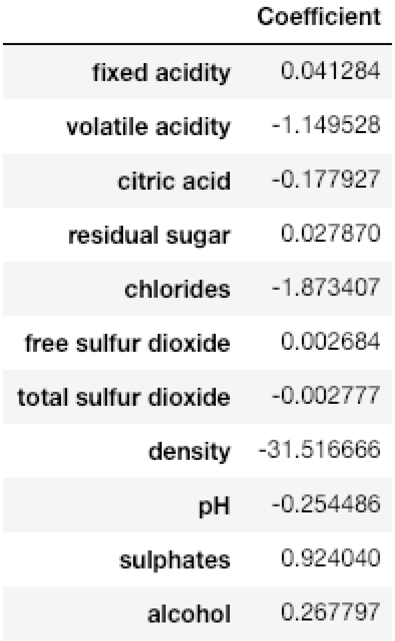
Now lets train our model.

```
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

As said earlier, in the case of multivariable linear regression, the regression model has to find the most optimal coefficients for all the attributes. To see what coefficients our regression model has chosen, execute the following script:

```
coeff_df = pd.DataFrame(regressor.coef_, X.columns, columns=
['Coefficient'])
coeff_df
```

it should give output something like :



	Coefficient
fixed acidity	0.041284
volatile acidity	-1.149528
citric acid	-0.177927
residual sugar	0.027870
chlorides	-1.873407
free sulfur dioxide	0.002684
total sulfur dioxide	-0.002777
density	-31.516666
pH	-0.254486
sulphates	0.924040
alcohol	0.267797

This means that for a unit increase in "density", there is a decrease of 31.51 units in the quality of the wine. Similarly, a unit decrease in "Chlorides" results in an increase of 1.87 units in the quality of the wine. We can see that the rest of the features have a very little effect on the quality of the wine.

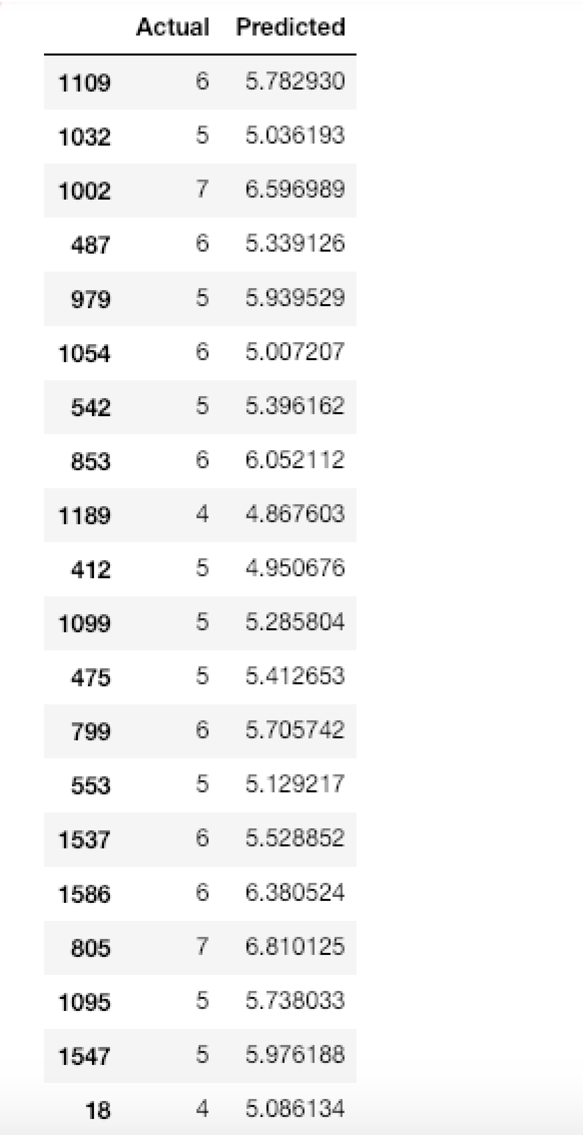
Now let's do prediction on test data.

```
y_pred = regressor.predict(X_test)
```

Check the difference between the actual value and predicted value.

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

df1 = df.head(25)
```

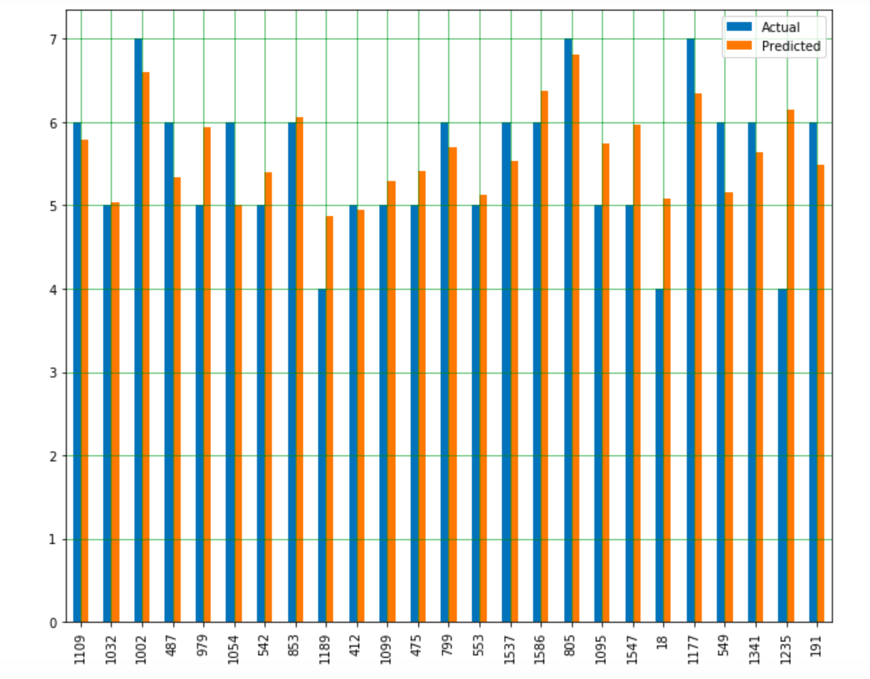


	Actual	Predicted
1109	6	5.782930
1032	5	5.036193
1002	7	6.596989
487	6	5.339126
979	5	5.939529
1054	6	5.007207
542	5	5.396162
853	6	6.052112
1189	4	4.867603
412	5	4.950676
1099	5	5.285804
475	5	5.412653
799	6	5.705742
553	5	5.129217
1537	6	5.528852
1586	6	6.380524
805	7	6.810125
1095	5	5.738033
1547	5	5.976188
18	4	5.086134

Comparison between Actual and Predicted value

Now let's plot the comparison of Actual and Predicted values

```
df1.plot(kind='bar',figsize=(10,8))
plt.grid(which='major', linestyle='-', linewidth='0.5',
color='green')
plt.grid(which='minor', linestyle=':', linewidth='0.5',
color='black')
plt.show()
```



Bar graph showing the difference between Actual and predicted value

As we can observe here that our model has returned pretty good prediction results.

The final step is to evaluate the performance of the algorithm. We'll do this by finding the values for MAE, MSE, and RMSE. Execute the following script:

```
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)))
```

The output looks like :

```
('Mean Absolute Error:', 0.46963309286611077)
('Mean Squared Error:', 0.38447119782012446)
('Root Mean Squared Error:', 0.6200574149384268)
```

You can see that the value of root mean squared error is 0.62, which is slightly greater than 10% of the mean value of the gas consumption in all states which is 5.63. This means that our algorithm was not very accurate but can still make reasonably good predictions.

There are many factors that may have contributed to this inaccuracy, for example :

Need more data: We need to have a huge amount of data to get the best possible prediction.
Bad assumptions: We made the assumption that this data has a linear relationship, but that might not be the case. Visualizing the data may help you determine that.
Poor features: The features we used may not have had a high enough correlation to the values we were trying to predict.

Conclusion

In this article, we studied the most fundamental machine learning algorithms i.e. linear regression. We implemented both simple linear regression and multiple linear regression with the help of the Scikit-Learn machine learning library.

I hope you guys have enjoyed the reading. Let me know your doubts/suggestions in the comment section.

Thanks for reading.

You can also reach me out in **LinkedIn**.

Happy Learning !!!

This article was also republished on KDnuggets.