CSL7020: Machine Learning-1

Programming Assignment

Name: Krishna Kumari Ravuri **Roll No:**M22AI567

Classification Task on wine dataset:

Q 1. Compare Naive Bayes classifier and Logistic regression methods using Accuracy, Confusion matrix, ROC metrics and explain which method performs better and why?

The main aim of the given question is to predict the quality of the wine good or bad based on the given set of features as input. The given dataset has certain features which are affecting the quality of the wine.

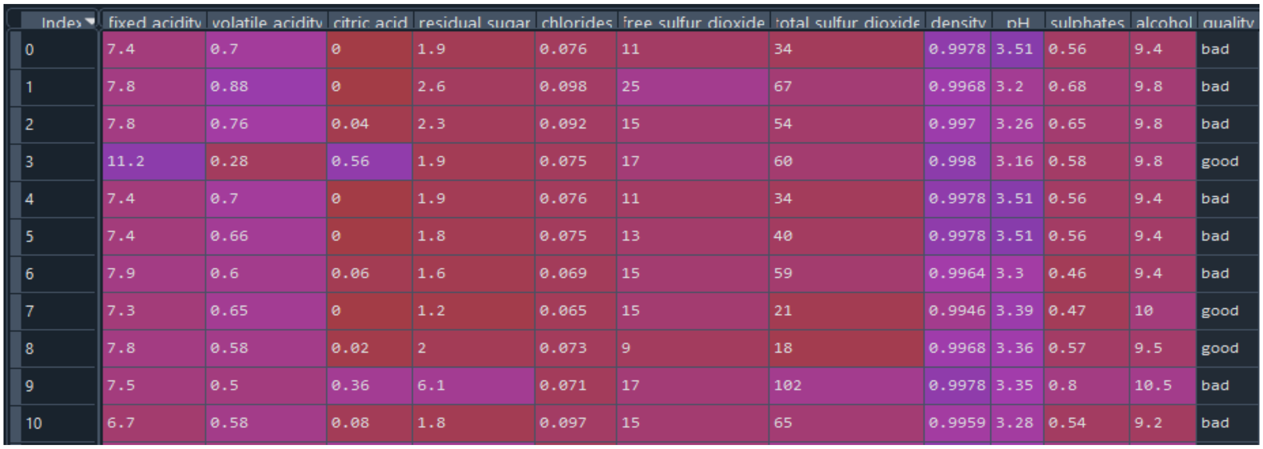
The following are the list of features which will affect the quality of the wine.

fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulphates, alcohol.

With the given dataset, we will create models using Naive Bayes classifier algorithm and Logistic regression algorithm to predict the quality of wine and also determining the best model based on the accuracy, confusion matrix and ROC Curve.

Steps for implementation of Naïve Bayes classifier algorithm and Logistic Regression algorithm.

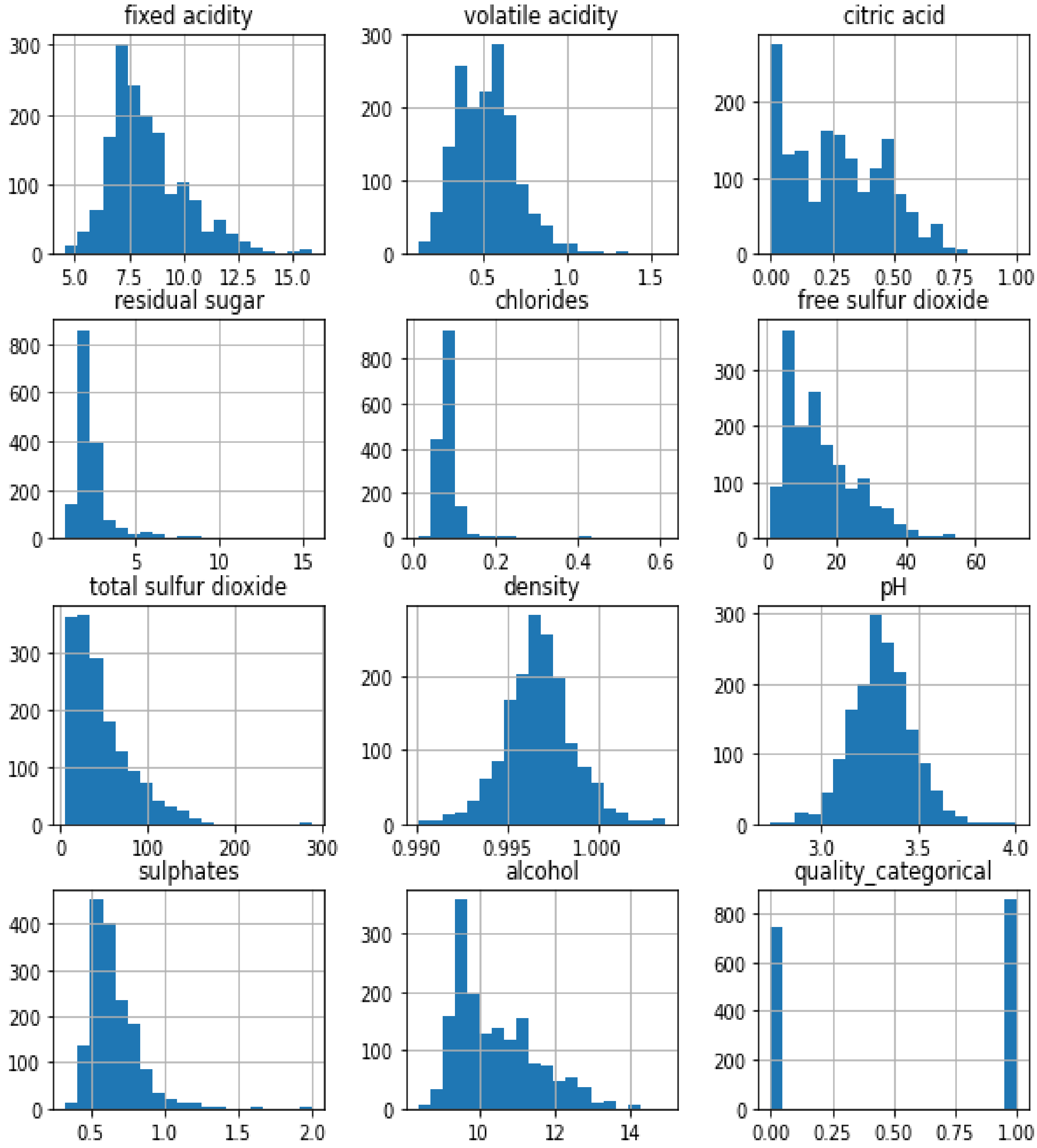
Step 1: Importing all the required libraries and loading the wine dataset.



Index	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.7	0	1.9	0.076	11	34	0.9978	3.51	0.56	9.4	bad
1	7.8	0.88	0	2.6	0.098	25	67	0.9968	3.2	0.68	9.8	bad
2	7.8	0.76	0.04	2.3	0.092	15	54	0.997	3.26	0.65	9.8	bad
3	11.2	0.28	0.56	1.9	0.075	17	60	0.998	3.16	0.58	9.8	good
4	7.4	0.7	0	1.9	0.076	11	34	0.9978	3.51	0.56	9.4	bad
5	7.4	0.66	0	1.8	0.075	13	40	0.9978	3.51	0.56	9.4	bad
6	7.9	0.6	0.06	1.6	0.069	15	59	0.9964	3.3	0.46	9.4	bad
7	7.3	0.65	0	1.2	0.065	15	21	0.9946	3.39	0.47	10	good
8	7.8	0.58	0.02	2	0.073	9	18	0.9968	3.36	0.57	9.5	good
9	7.5	0.5	0.36	6.1	0.071	17	102	0.9978	3.35	0.8	10.5	bad
10	6.7	0.58	0.08	1.8	0.097	15	65	0.9959	3.28	0.54	9.2	bad

Step 2: Analyzing the data using visual techniques. With this we can analyze patterns and make some assumptions.

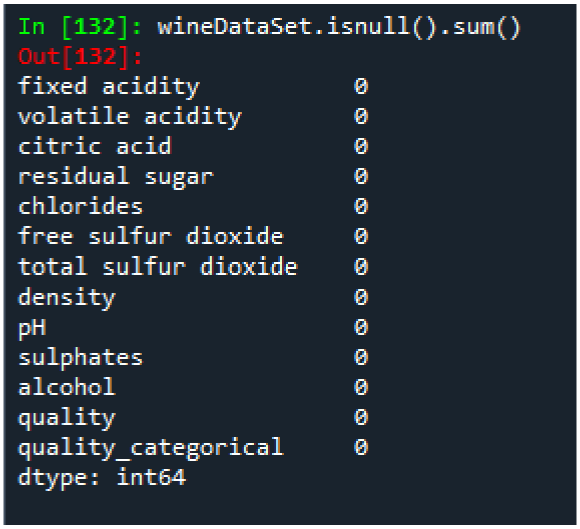
In the given dataset there is no null values. By visualizing the data in the histogram we have identify the distribution of the data with continuous values.



From these histograms we can clear see that how data is easily distributed on features.

Step 3: Data Preprocessing

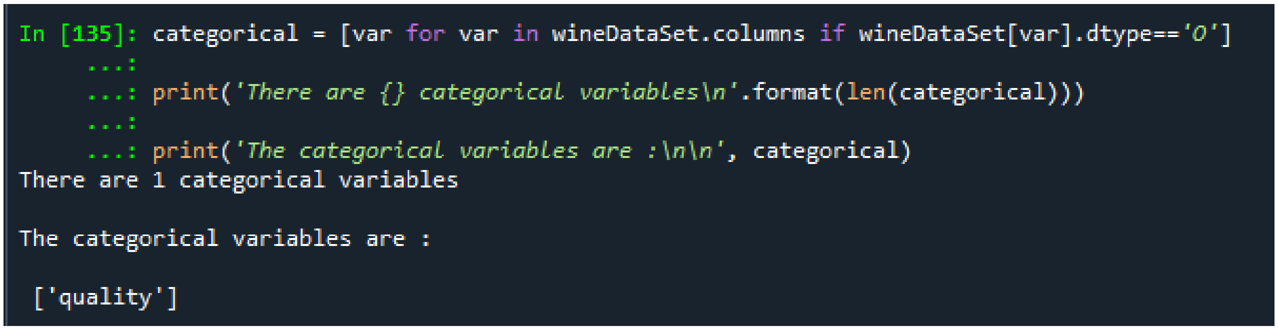
Checking for Null values in the wine dataset.



```
In [132]: wineDataSet.isnull().sum()
Out[132]:
fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar          0
chlorides               0
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
quality                 0
quality_categorical     0
dtype: int64
```

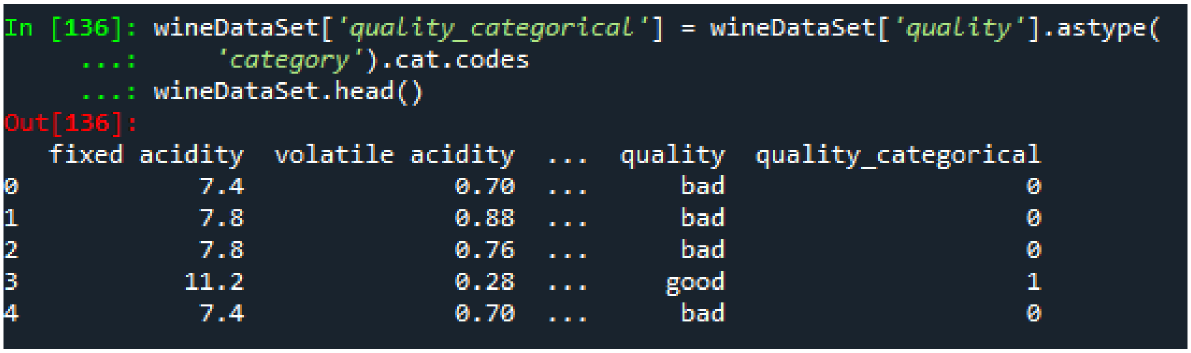
From the above image we can clearly see that null values are not present in the given wine dataset.

Identifying the type of each feature categorical/ numerical and converting categorical into numerical.

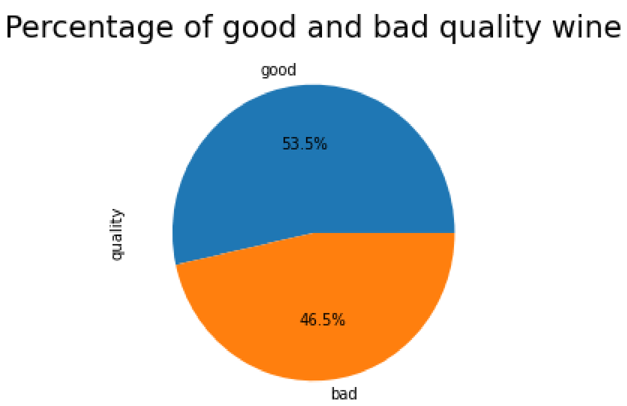


```
In [135]: categorical = [var for var in wineDataSet.columns if wineDataSet[var].dtype=='O']
     ...:
     ...: print('There are {} categorical variables\n'.format(len(categorical)))
     ...:
     ...: print('The categorical variables are :\n\n', categorical)
There are 1 categorical variables

The categorical variables are :

 ['quality']
```

From the above image we see the one categorical feature. We need convert the categorical feature into numerical. Since quality has only two different types i.e. Good and Bad. It will be converted to 0's and 1's and pushed to new column called quality_categorical.



```
In [136]: wineDataSet['quality_categorical'] = wineDataSet['quality'].astype(
     ...:     'category').cat.codes
     ...: wineDataSet.head()
Out[136]:
   fixed acidity  volatile acidity  ...  quality  quality_categorical
0            7.4              0.70  ...      bad                    0
1            7.8              0.88  ...      bad                    0
2            7.8              0.76  ...      bad                    0
3           11.2              0.28  ...     good                    1
4            7.4              0.70  ...      bad                    0
```



Step 4: Navie Bayes classifier model without using scikit learn library.

Navie Bayes algorithm is a supervised learning algorithm, which is mainly based on Bayes theorem.it predicts on the basis of the probability.

Formula for Bayes' Theorem:

$$\boxed{P(A|B) = \frac{P(B|A)\,P(A)}{P(B)}}$$

P(A|B) is the posterior probability

P(B|A) is the likelihood probability

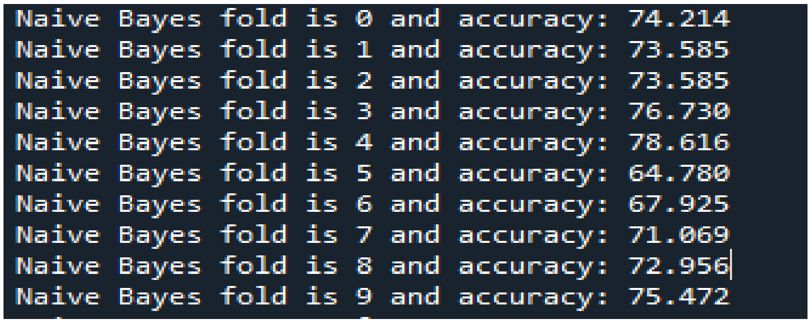
P(A) is Prior Probability

P(B) is Marginal Probability

Implemented the code the Navie byer classification for Gaussian model.

Now we will fit the model along with K-fold cross validation technique. Number of folds is 10.

The accuracy scores for each fold is



```
Naive Bayes fold is 0 and accuracy: 74.214
Naive Bayes fold is 1 and accuracy: 73.585
Naive Bayes fold is 2 and accuracy: 73.585
Naive Bayes fold is 3 and accuracy: 76.730
Naive Bayes fold is 4 and accuracy: 78.616
Naive Bayes fold is 5 and accuracy: 64.780
Naive Bayes fold is 6 and accuracy: 67.925
Naive Bayes fold is 7 and accuracy: 71.069
Naive Bayes fold is 8 and accuracy: 72.956
Naive Bayes fold is 9 and accuracy: 75.472
```

We observed that accuracy is ranging from **64.780 to 78.618** with the average accuracy **72.893.**

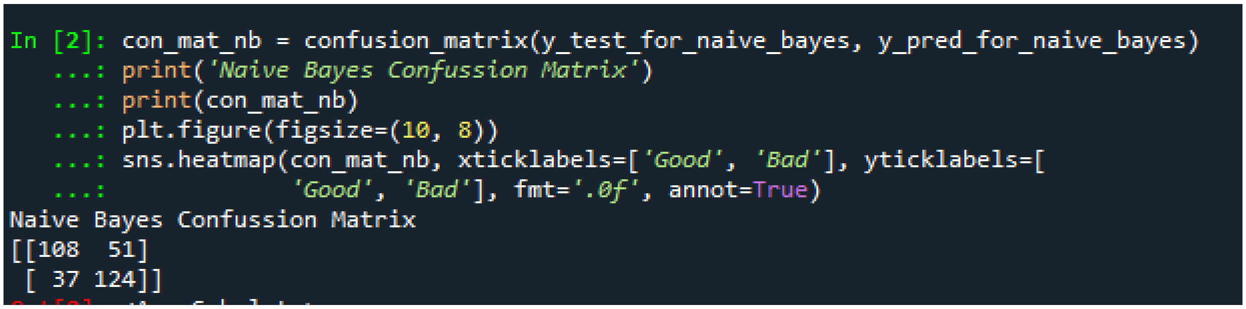
Average accuracy suggests that the model performs reasonably well. However, the accuracy scores for some folds are quite low (e.g., 64.780 and 67.925), which indicating that the model may not generalize well to those particular subsets of the data.

Confusion Matrix

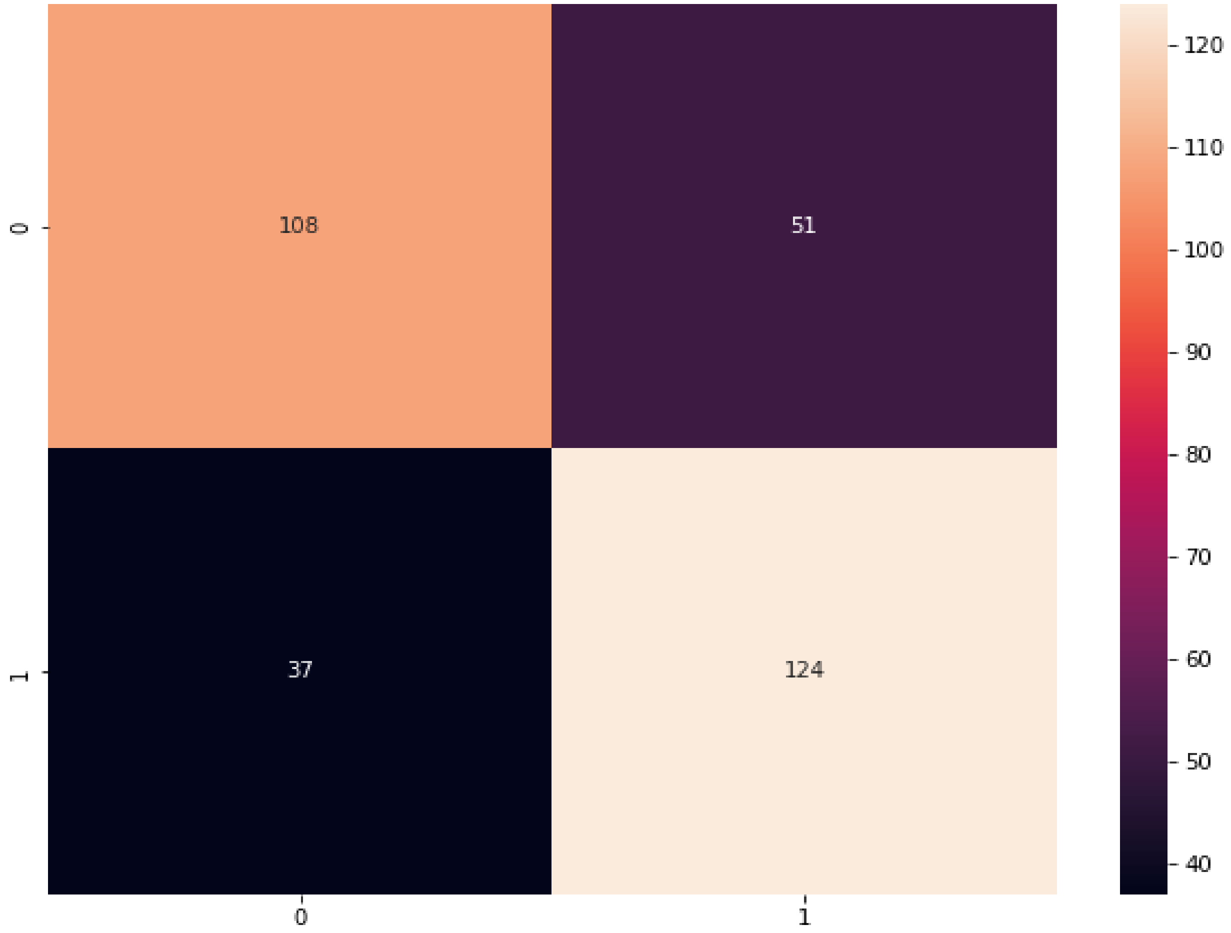
Confusion matrix help us to evaluate the classifier in a better way.

Predicated Values	Actual Values		
		NEGATIVE	POSITIVE
	NEGATIVE	TN	FP
	POSITIVE	FN	TP

High TP and TN rates and low FP and FN rates indicate a good model.



```
In [2]: con_mat_nb = confusion_matrix(y_test_for_naive_bayes, y_pred_for_naive_bayes)
   ...: print('Naive Bayes Confussion Matrix')
   ...: print(con_mat_nb)
   ...: plt.figure(figsize=(10, 8))
   ...: sns.heatmap(con_mat_nb, xticklabels=['Good', 'Bad'], yticklabels=[
   ...:                 'Good', 'Bad'], fmt='.0f', annot=True)
Naive Bayes Confussion Matrix
[[108  51]
 [ 37 124]]
```



Precision, Recall and F1 Score:

True Positives (TP): 124 number of instances that were correctly predicted as positive by the model.

False Positives (FP): 51 number of instances that were predicted as positive by the model but were actually negative.

False Negatives (FN): 37 number of instances that were actually positive but were predicted as negative by the model.

True Negatives (TN): 108 number of instances that were correctly predicted as negative by the model.

Recall:

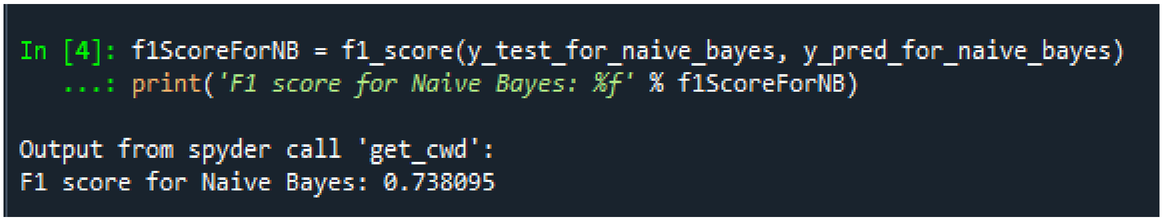
$$\text{Recall} = \frac{TP}{TP+FN} = \frac{124}{124+37} = 0.770$$

Precision:

$$\text{Precision} = \frac{TP}{TP+FP} = \frac{124}{124+51} = 0.708$$

F1 Score:

$$\text{F1 Score} = \frac{2 * \text{Precision} * \text{Recall}}{(\text{Precision} + \text{Recall})} = \frac{2 * 0.708 * 0.770}{(0.708 + 0.770)} = 0.738$$



```
In [4]: f1ScoreForNB = f1_score(y_test_for_naive_bayes, y_pred_for_naive_bayes)
   ...: print('F1 score for Naive Bayes: %f' % f1ScoreForNB)

Output from spyder call 'get_cwd':
F1 score for Naive Bayes: 0.738095
```

The mean accuracy of 72.288% indicates that the classifier correctly predicted the target variable for approximately 72.288% of the instances.

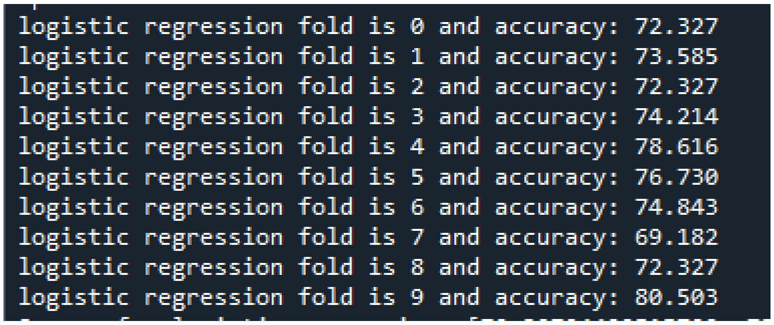
The F1 score of 0.738095 indicates a reasonably good balance between precision and recall. This means that the classifier is able to make accurate positive predictions while minimizing the number of false negatives.

The precision of 0.708 and recall of 0.770 indicate that the classifier has relatively good accuracy in predicting positive instances (precision), as well as identifying all positive instances (recall)

Step 5: Logistic regression classifier model without using scikit learn library.

Logistic regression algorithm is a supervised learning algorithm, Logistic regression uses the concept of predictive modeling as regression; therefore, it is called logistic regression

As we have completed the data preprocessing in the previous model, now we will fit the model along with K-fold cross validation technique. Number of folds is 10.

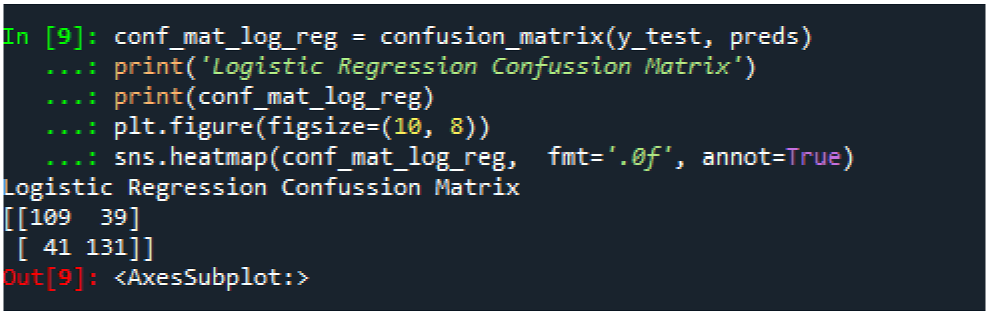


```
logistic regression fold is 0 and accuracy: 72.327
logistic regression fold is 1 and accuracy: 73.585
logistic regression fold is 2 and accuracy: 72.327
logistic regression fold is 3 and accuracy: 74.214
logistic regression fold is 4 and accuracy: 78.616
logistic regression fold is 5 and accuracy: 76.730
logistic regression fold is 6 and accuracy: 74.843
logistic regression fold is 7 and accuracy: 69.182
logistic regression fold is 8 and accuracy: 72.327
logistic regression fold is 9 and accuracy: 80.503
```

We observed that accuracy is ranging from **69.182 to 80.503** with the average accuracy **74.483.**

Average accuracy suggests that the model performs reasonably well. However, the accuracy score for one fold is quite low (e.g., 69.182), which indicating that the model may not generalize well to those particular subsets of the data.

Confusion Matrix



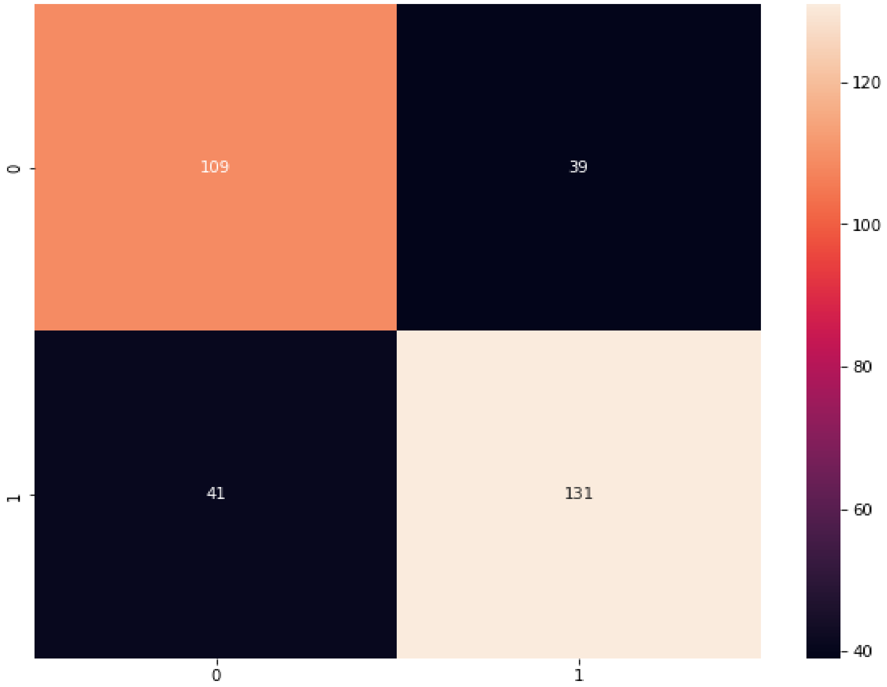
```
In [9]: conf_mat_log_reg = confusion_matrix(y_test, preds)
   ...: print('Logistic Regression Confussion Matrix')
   ...: print(conf_mat_log_reg)
   ...: plt.figure(figsize=(10, 8))
   ...: sns.heatmap(conf_mat_log_reg,  fmt='.0f', annot=True)
Logistic Regression Confussion Matrix
[[109  39]
 [ 41 131]]
Out[9]: <AxesSubplot:>
```

True Positives (TP): 109 number of instances that were correctly predicted as positive by the model.

False Positives (FP): 39 number of instances that were predicted as positive by the model but were actually negative.

False Negatives (FN): 41 number of instances that were actually positive but were predicted as negative by the model.

True Negatives (TN): 131 number of instances that were correctly predicted as negative by the model.



Precision, Recall and F1 Score:

Recall:

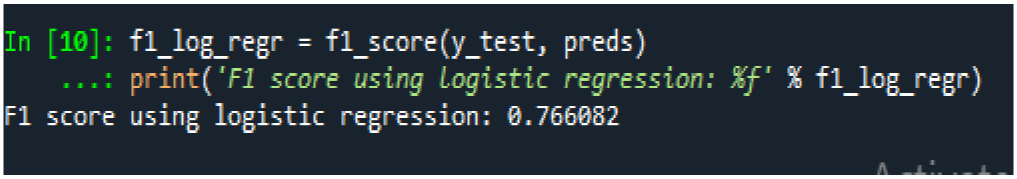
$$\text{Recall} = \frac{\text{TP}}{\text{TP+FN}} = \frac{131}{124+41} = 0.761$$

Precision:

$$\text{Precision} = \frac{\text{TP}}{\text{TP+FP}} = \frac{131}{124+39} = 0.771$$

F1 Score:

$$\text{F1 Score} = \frac{2 * \text{Precision} * \text{Recall}}{(\text{Precision} + \text{Recall})} = \frac{2 * 0.761 * 0.771}{(0.761 + 0.771)} = 0.766$$

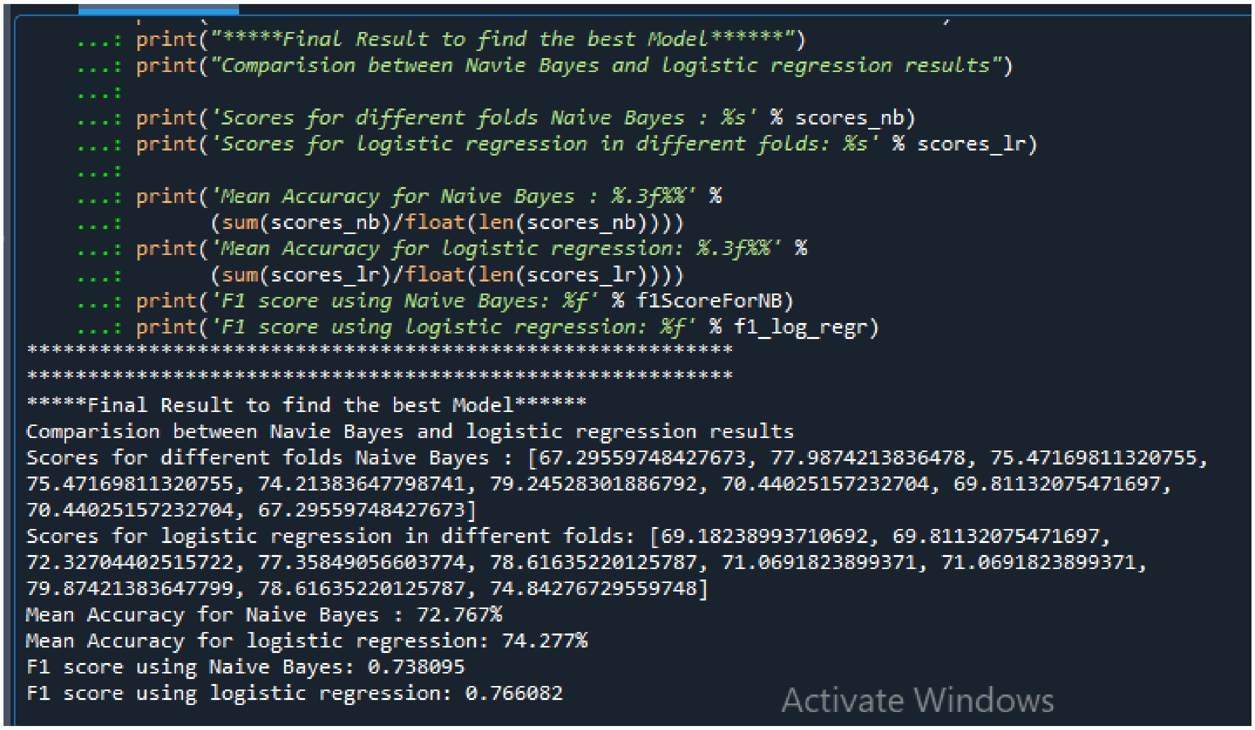


```
In [10]: f1_log_regr = f1_score(y_test, preds)
    ...: print('F1 score using logistic regression: %f' % f1_log_regr)
F1 score using logistic regression: 0.766082
```

Mean Accuracy for logistic regression is 74.403% which indicates that the classifier correctly predicted the target variable for approximately 74.403% of the instances.

F1-score of the logistic regression model is 0.766082, which indicates a reasonably good performance.

Comparison between Navie Bayes classifier and Logistic regression models:



```
    ...: print("*****Final Result to find the best Model******")
    ...: print("Comparision between Navie Bayes and logistic regression results")
    ...:
    ...: print('Scores for different folds Naive Bayes : %s' % scores_nb)
    ...: print('Scores for logistic regression in different folds: %s' % scores_lr)
    ...:
    ...: print('Mean Accuracy for Naive Bayes : %.3f%%' %
    ...:       (sum(scores_nb)/float(len(scores_nb))))
    ...: print('Mean Accuracy for logistic regression: %.3f%%' %
    ...:       (sum(scores_lr)/float(len(scores_lr))))
    ...: print('F1 score using Naive Bayes: %f' % f1ScoreForNB)
    ...: print('F1 score using logistic regression: %f' % f1_log_regr)
**********************************************************
**********************************************************
*****Final Result to find the best Model******
Comparision between Navie Bayes and logistic regression results
Scores for different folds Naive Bayes : [67.29559748427673, 77.9874213836478, 75.47169811320755,
75.47169811320755, 74.21383647798741, 79.24528301886792, 70.44025157232704, 69.81132075471697,
70.44025157232704, 67.29559748427673]
Scores for logistic regression in different folds: [69.18238993710692, 69.81132075471697,
72.32704402515722, 77.35849056603774, 78.61635220125787, 71.0691823899371, 71.0691823899371,
79.87421383647799, 78.61635220125787, 74.84276729559748]
Mean Accuracy for Naive Bayes : 72.767%
Mean Accuracy for logistic regression: 74.277%
F1 score using Naive Bayes: 0.738095
F1 score using logistic regression: 0.766082
```

we can see that the logistic regression classifier achieved a slightly higher mean accuracy (74.483%) and F1 score (0.769679) compared to the Naive Bayes classifier, which achieved a mean accuracy of 72.288% and F1 score of 0.738095.

Additionally, looking at the confusion matrices for both classifiers, we can see that the logistic regression classifier achieved a higher true positive rate (TPR) or sensitivity of 0.761194, compared to the Naive Bayes classifier with a TPR of 0.679245. The logistic regression classifier also had a slightly higher true negative rate (TNR) or specificity of 0.770492, compared to the Naive Bayes classifier with a TNR of 0.770186.

Furthermore, the logistic regression classifier had a lower false negative rate (FNR) of 0.238806 compared to the Naive Bayes classifier with a FNR of 0.320755. This means that the logistic regression classifier was better at identifying true positives than the Naive Bayes classifier.

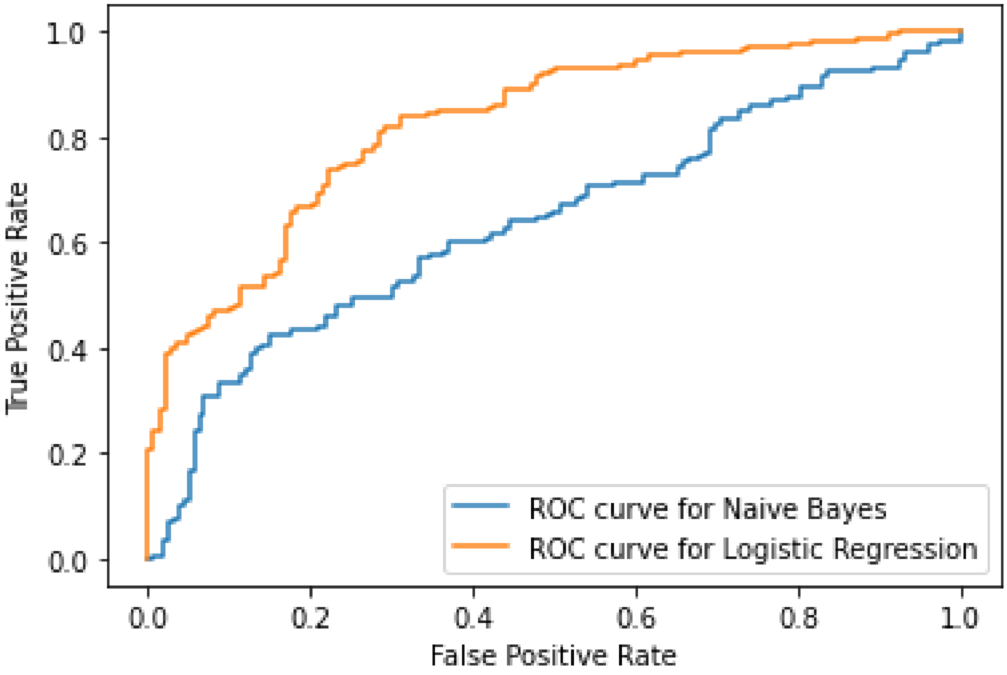
Let us look at the Receiver Operator Characteristic (ROC) curve:

(ROC) curve is an evaluation metric for binary classification problems.

The probability curve that plots the TPR against FPR at various threshold values and essentially separates the ‘signal’ from the ‘noise.

it shows the performance of a classification model at all classification thresholds. The Area Under the Curve (AUC) is the measure of the ability of a binary classifier to distinguish between classes and is used as a summary of the ROC curve.

The higher the AUC, the better the model's performance at distinguishing between the positive and negative classes.



Based the ROC for different model, these Curve didn't intersect at any point and we can clear visualize that area under the Logistic Regression ROC curve is more than the area under the Naïve Bayes ROC Curve. With this observation we can say that logistic regression model was better than Naïve Bayes model for the given wine quality dataset.

Therefore, based on the Accuracy, Confusion Matrix, Precision, Recall, F1-Score and AUC in ROC results we can conclude that the logistic regression classifier performs slightly better than the Naive Bayes classifier on this dataset and problem.

Q2 Regression Task:

(a) Implement Maximum likelihood estimator for regression task on the given salary dataset.

(b) Explain all the steps in details.

(c) Report error for both training and testing set.

Linear Regression with Maximum Likelihood Estimation:

Linear regression is a classical model for predicting a numerical quantity.

There are many techniques for solving density estimation, although a common framework used throughout the field of machine learning is maximum likelihood estimation.

Maximum likelihood estimation involves defining a likelihood function for calculating the conditional probability of observing the data sample given a probability distribution and distribution parameters.

This approach is used to search a space of possible distributions and parameters.

Steps for implementing Linear regression model with maximum likelihood estimation and predict the salary of the employees based on year of experience.

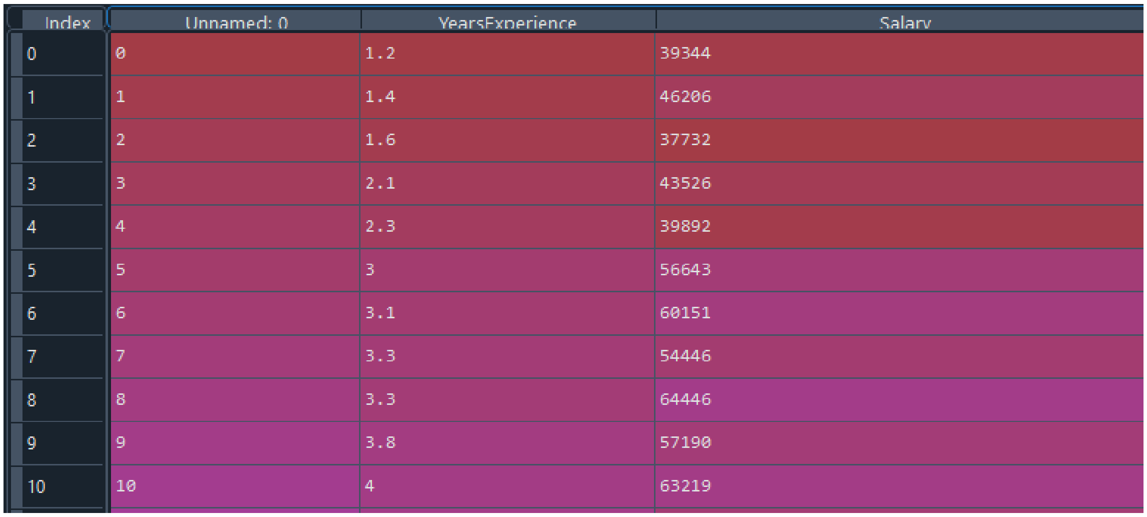
Dataset description:

The given salary dataset consists of two columns

Salary- Represent the annual salary of a person.

Years- Years of experience

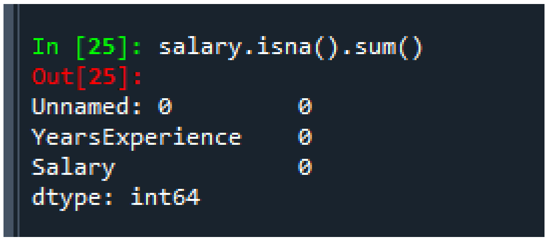
Step 1: Importing all the required libraries and loading the wine dataset.



Index	Unnamed: 0	YearsExperience	Salary
0	0	1.2	39344
1	1	1.4	46206
2	2	1.6	37732
3	3	2.1	43526
4	4	2.3	39892
5	5	3	56643
6	6	3.1	60151
7	7	3.3	54446
8	8	3.3	64446
9	9	3.8	57190
10	10	4	63219

Step 2: Data Preprocessing

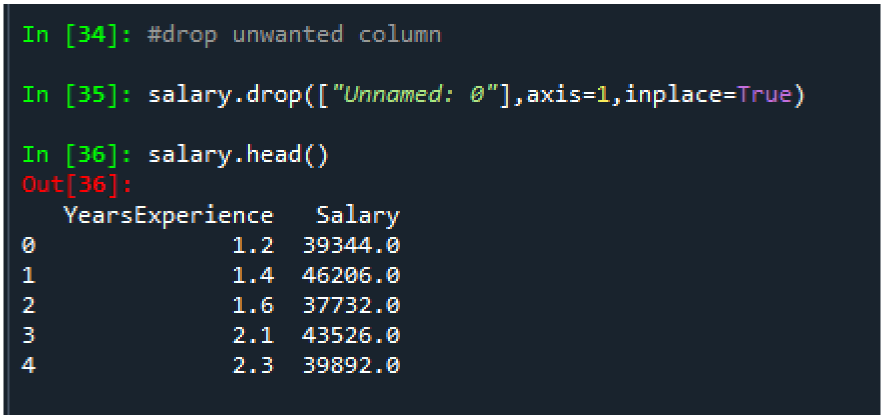
Checking for Null values in the salary dataset.



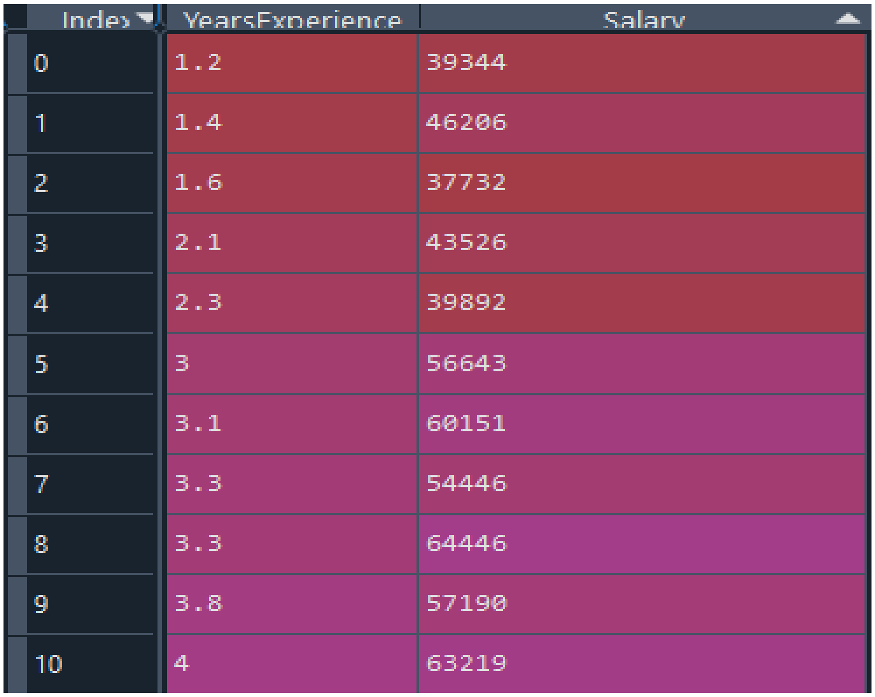
```
In [25]: salary.isna().sum()
Out[25]:
Unnamed: 0         0
YearsExperience    0
Salary             0
dtype: int64
```

We can see that there are no missing values are present in our dataset and apart from YearsExperience and Salary dataset contain one more column without any name.

Drop unnamed column



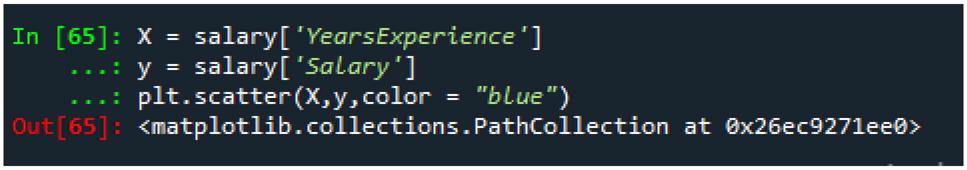
```
In [34]: #drop unwanted column

In [35]: salary.drop(["Unnamed: 0"],axis=1,inplace=True)

In [36]: salary.head()
Out[36]:
   YearsExperience   Salary
0              1.2  39344.0
1              1.4  46206.0
2              1.6  37732.0
3              2.1  43526.0
4              2.3  39892.0
```



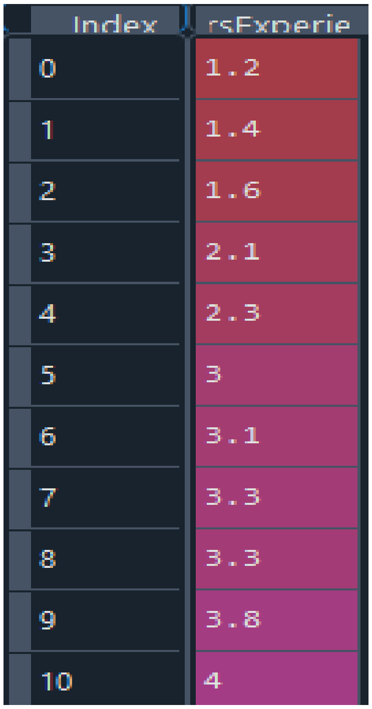
Index	YearsExperience	Salary
0	1.2	39344
1	1.4	46206
2	1.6	37732
3	2.1	43526
4	2.3	39892
5	3	56643
6	3.1	60151
7	3.3	54446
8	3.3	64446
9	3.8	57190
10	4	63219

Step 3: Model Building

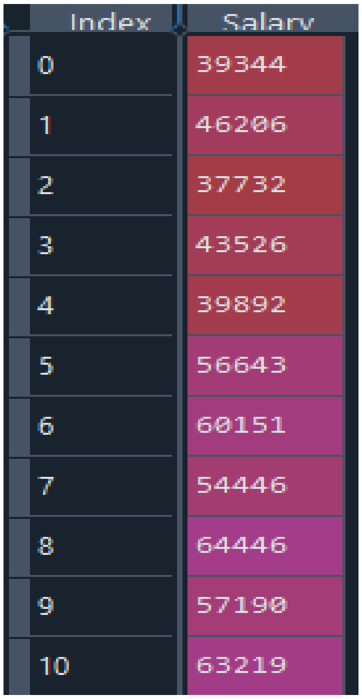
Splitting the salary dataset into independent and dependent variables



```
In [65]: X = salary['YearsExperience']
    ...: y = salary['Salary']
    ...: plt.scatter(X,y,color = "blue")
Out[65]: <matplotlib.collections.PathCollection at 0x26ec9271ee0>
```

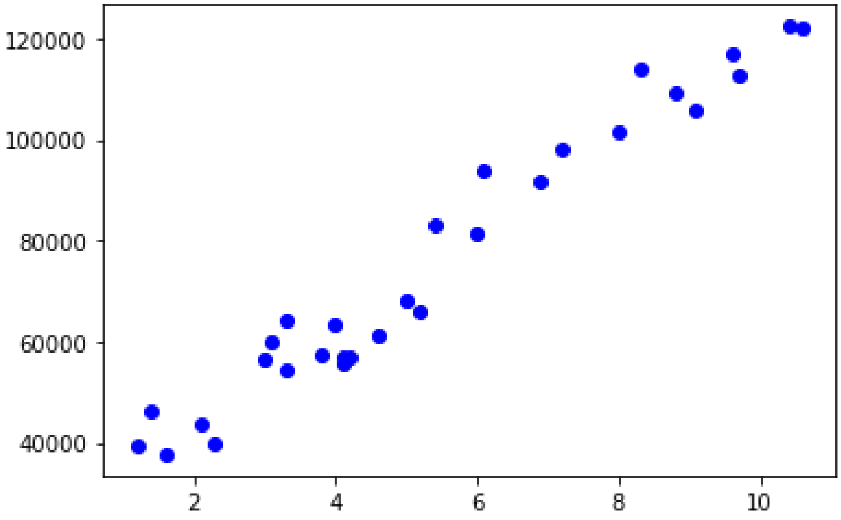


Index	rsExperie
0	1.2
1	1.4
2	1.6
3	2.1
4	2.3
5	3
6	3.1
7	3.3
8	3.3
9	3.8
10	4



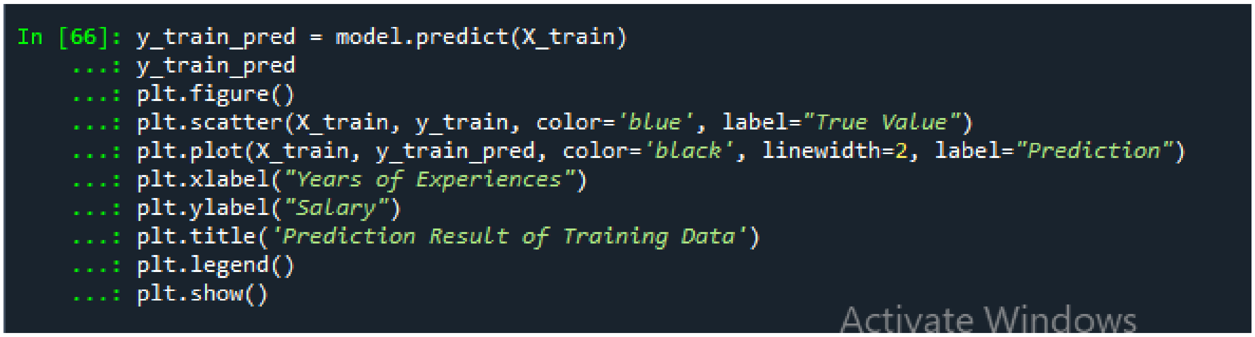
Index	Salary
0	39344
1	46206
2	37732
3	43526
4	39892
5	56643
6	60151
7	54446
8	64446
9	57190
10	63219

Plotting Scatter plot to identify the relationship between independent and dependent variables.

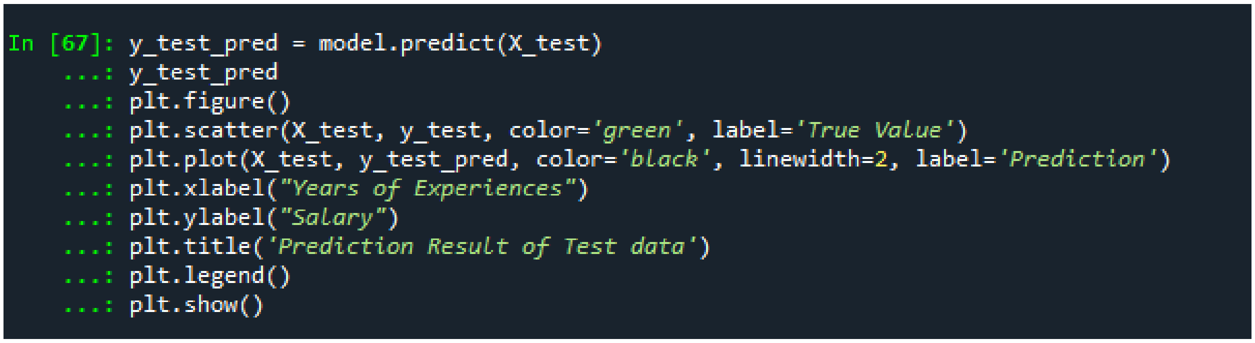


We can observe that the independent and dependent variable is linearly related to each other.

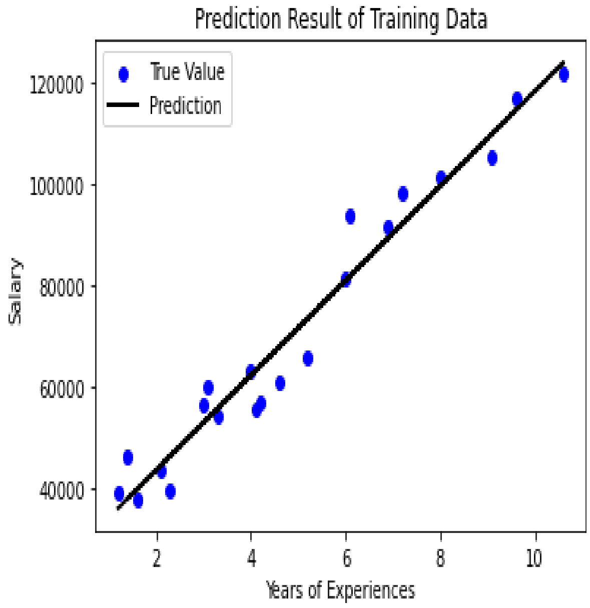
After splitting the dataset into training and testing. Training dataset is used to train the model and then used the testing dataset for prediction.

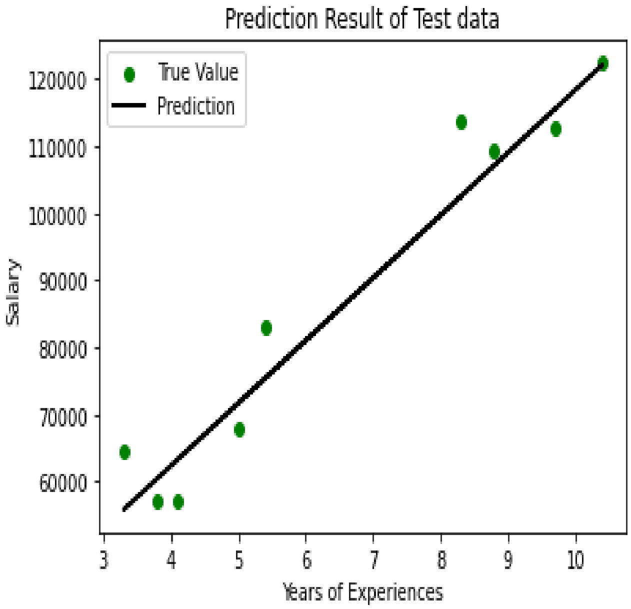


```
In [66]: y_train_pred = model.predict(X_train)
    ...: y_train_pred
    ...: plt.figure()
    ...: plt.scatter(X_train, y_train, color='blue', label="True Value")
    ...: plt.plot(X_train, y_train_pred, color='black', linewidth=2, label="Prediction")
    ...: plt.xlabel("Years of Experiences")
    ...: plt.ylabel("Salary")
    ...: plt.title('Prediction Result of Training Data')
    ...: plt.legend()
    ...: plt.show()
```



```
In [67]: y_test_pred = model.predict(X_test)
    ...: y_test_pred
    ...: plt.figure()
    ...: plt.scatter(X_test, y_test, color='green', label='True Value')
    ...: plt.plot(X_test, y_test_pred, color='black', linewidth=2, label='Prediction')
    ...: plt.xlabel("Years of Experiences")
    ...: plt.ylabel("Salary")
    ...: plt.title('Prediction Result of Test data')
    ...: plt.legend()
    ...: plt.show()
```





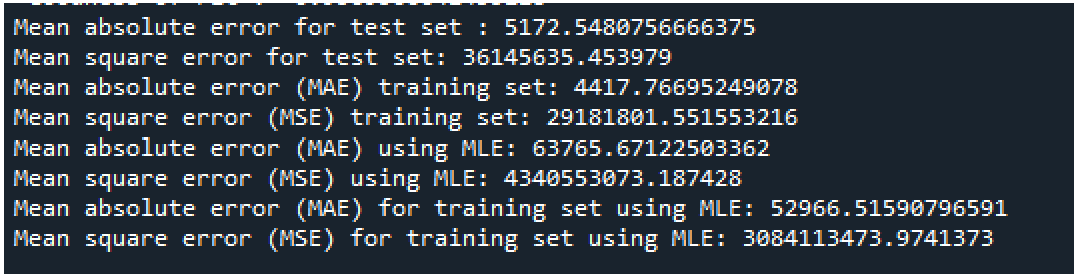
Training Data Perdition: True values are plotted as the blue dots on the chart and the predicted values are plotted as a black color straight line. The linear model fits well on the training data. This shows a linear relationship between the salary and years of experience.

Test Data Prediction: The graph shows that our linear model can fit quite well on the test set. We can observe a linear pattern of how the amount of salary is increased by the years of experience.

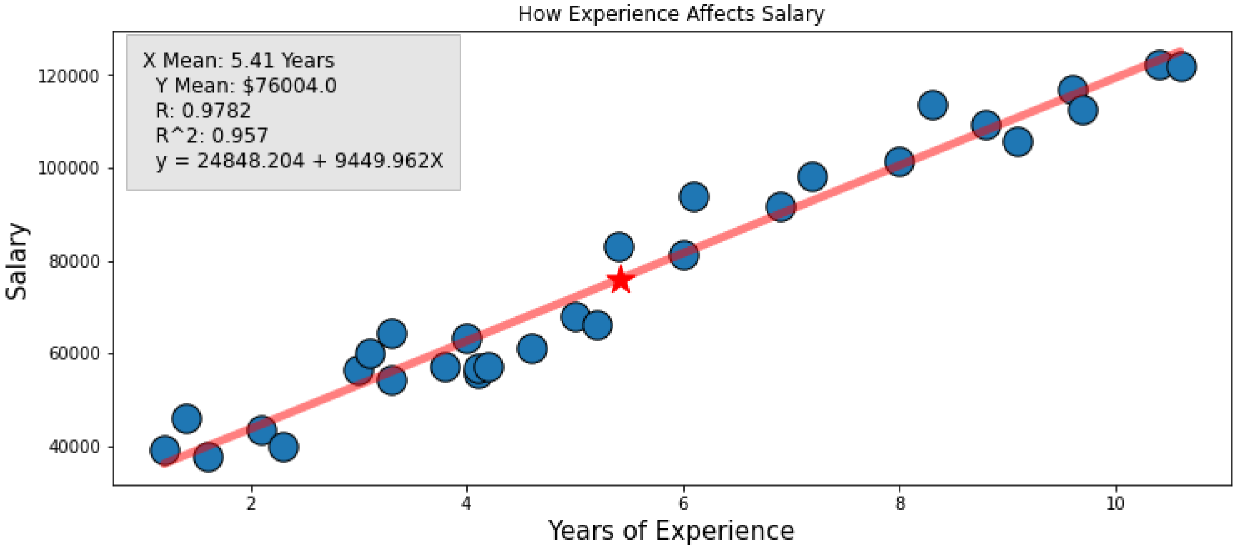
Step 4: Model Evaluation

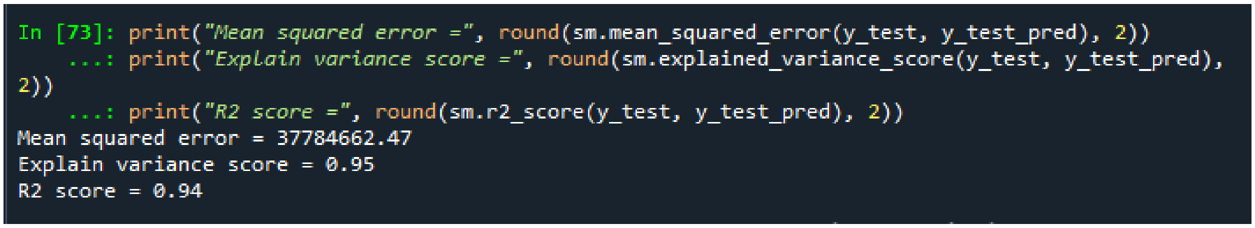
In order to evaluate the performance of a linear regression model, several metrics are used. Here are some of the most common evaluation metrics for linear regression models.

1. R Square/Adjusted R Square.
2. Mean Square Error(MSE)/Root Mean Square Error(RMSE)
3. Mean Absolute Error(MAE)



```
Mean absolute error for test set : 5172.5480756666375
Mean square error for test set: 36145635.453979
Mean absolute error (MAE) training set: 4417.76695249078
Mean square error (MSE) training set: 29181801.551553216
Mean absolute error (MAE) using MLE: 63765.67122503362
Mean square error (MSE) using MLE: 4340553073.187428
Mean absolute error (MAE) for training set using MLE: 52966.51590796591
Mean square error (MSE) for training set using MLE: 3084113473.9741373
```





```
In [73]: print("Mean squared error =", round(sm.mean_squared_error(y_test, y_test_pred), 2))
    ...: print("Explain variance score =", round(sm.explained_variance_score(y_test, y_test_pred),
2))
    ...: print("R2 score =", round(sm.r2_score(y_test, y_test_pred), 2))
Mean squared error = 37784662.47
Explain variance score = 0.95
R2 score = 0.94
```

Since the salary dataset is very small which contains only 30 records. With this we can’t completely conclude how model is performing. But based on the R2 square score is 0.94 which is good sign.

Both MSE and MAE of train set is lower value compared to the test set using MLE indicates that the model may be overfitting the training data.