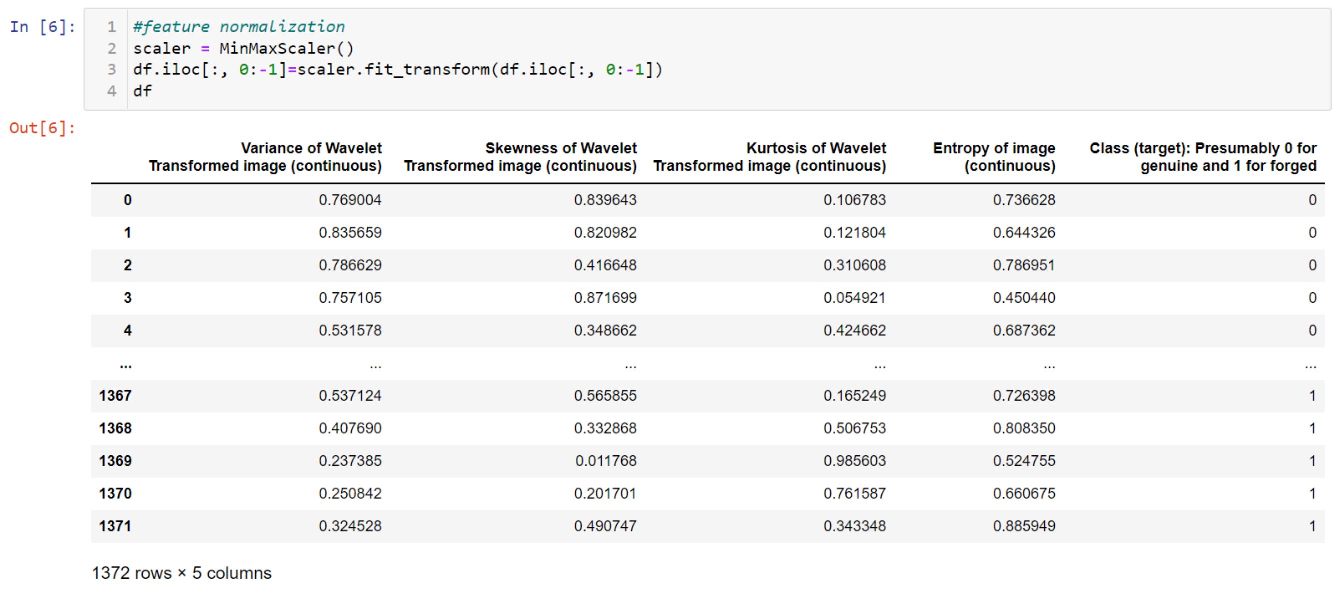
1) First load the csv file into Jupyter notebook, then convert into Dataframe.
   There are no missing data for this dataset and thus, original data will be used for this case and no data modification is to be make.
2) Refer to the screenshot below:



```
In [6]:
#feature normalization
scaler = MinMaxScaler()
df.iloc[:, 0:-1]=scaler.fit_transform(df.iloc[:, 0:-1])
df
```

Out[6]:

	Variance of Wavelet Transformed image (continuous)	Skewness of Wavelet Transformed image (continuous)	Kurtosis of Wavelet Transformed image (continuous)	Entropy of image (continuous)	Class (target): Presumably 0 for genuine and 1 for forged
0	0.769004	0.839643	0.106783	0.736628	0
1	0.835659	0.820982	0.121804	0.644326	0
2	0.786629	0.416648	0.310608	0.786951	0
3	0.757105	0.871699	0.054921	0.450440	0
4	0.531578	0.348662	0.424662	0.687362	0
...	...	...	...	...	...
1367	0.537124	0.565855	0.165249	0.726398	1
1368	0.407690	0.332868	0.506753	0.808350	1
1369	0.237385	0.011768	0.985603	0.524755	1
1370	0.250842	0.201701	0.761587	0.660675	1
1371	0.324528	0.490747	0.343348	0.885949	1

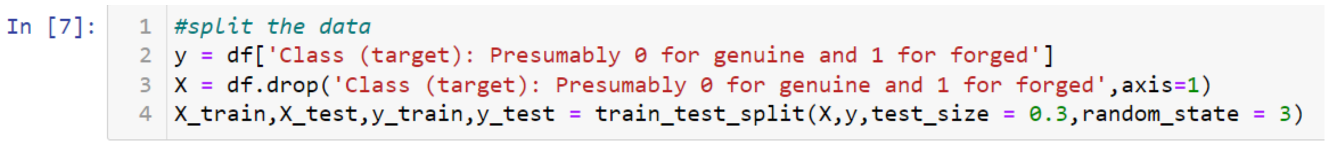
1372 rows × 5 columns

3) Logistic regression equation:

$$A=\frac{1}{1+e^{Zi}} \qquad \text{where} \quad Zi = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + b_4x_4$$

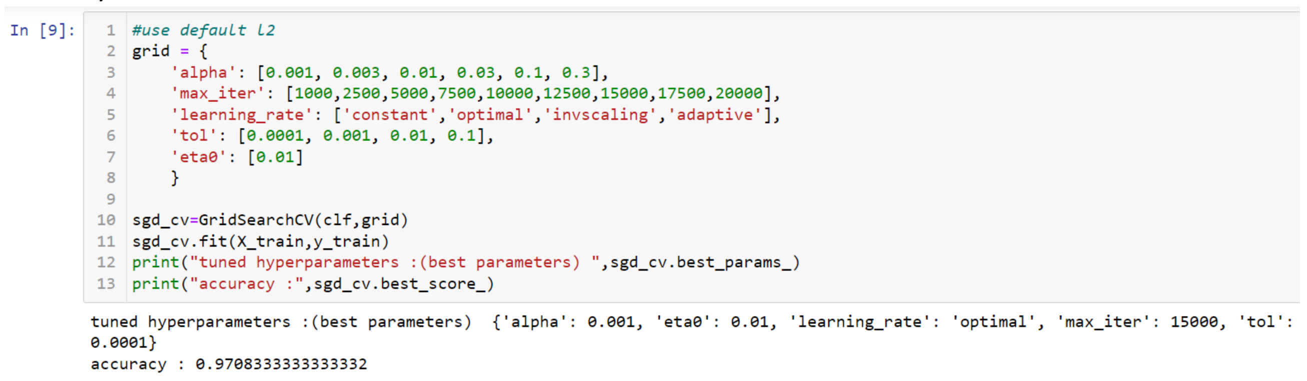
As since there are 4 numerical input variables, thus 4 parameters are needed to classify whether the bank note is genuine or forged.

4) Refer to the screenshot below:



```
In [7]:
#split the data
y = df['Class (target): Presumably 0 for genuine and 1 for forged']
X = df.drop('Class (target): Presumably 0 for genuine and 1 for forged',axis=1)
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size = 0.3,random_state = 3)
```

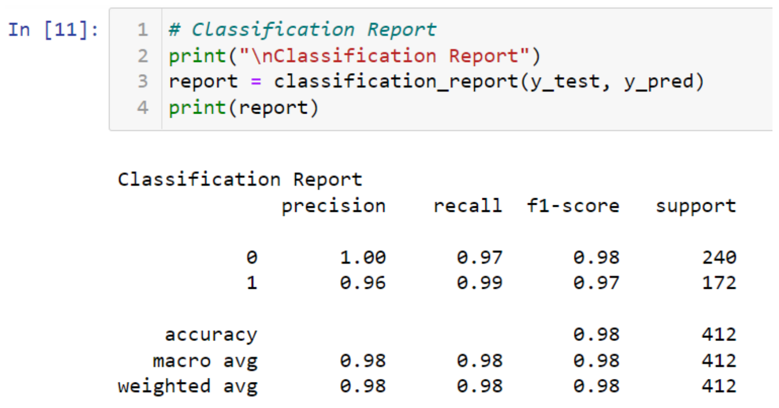
5) These are the hyperparameters which are selected through GridSearchCV using the default values and the accuracy:



```
In [9]:
#use default l2
grid = {
    'alpha': [0.001, 0.003, 0.01, 0.03, 0.1, 0.3],
    'max_iter': [1000,2500,5000,7500,10000,12500,15000,17500,20000],
    'learning_rate': ['constant','optimal','invscaling','adaptive'],
    'tol': [0.0001, 0.001, 0.01, 0.1],
    'eta0': [0.01]
    }

sgd_cv=GridSearchCV(clf,grid)
sgd_cv.fit(X_train,y_train)
print("tuned hyperparameters :(best parameters) ",sgd_cv.best_params_)
print("accuracy :",sgd_cv.best_score_)
```

```
tuned hyperparameters :(best parameters)  {'alpha': 0.001, 'eta0': 0.01, 'learning_rate': 'optimal', 'max_iter': 15000, 'tol': 
0.0001}
accuracy : 0.9708333333333332
```

6) Refer to the screenshot below:

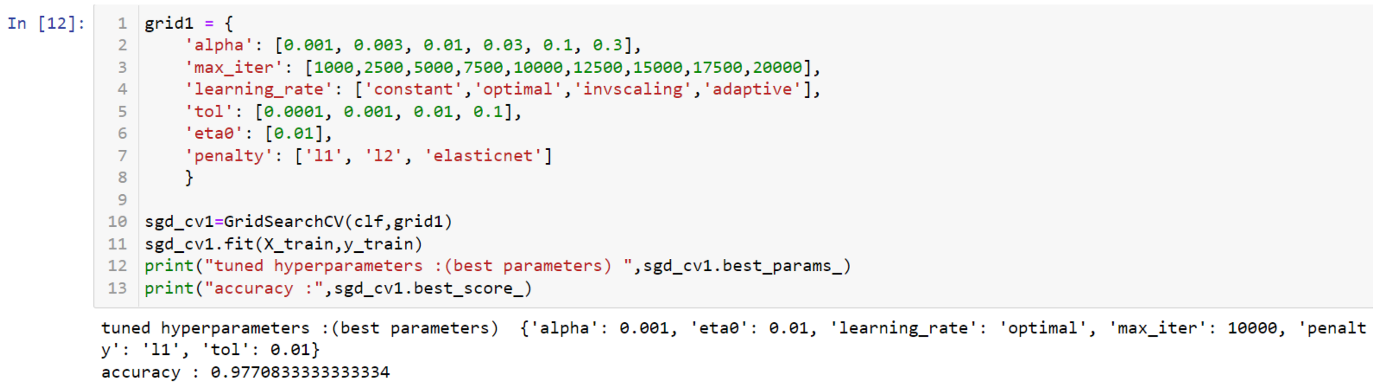


```
In [11]:
# Classification Report
print("\nClassification Report")
report = classification_report(y_test, y_pred)
print(report)
```

```
Classification Report
              precision    recall  f1-score   support

           0       1.00      0.97      0.98       240
           1       0.96      0.99      0.97       172

    accuracy                           0.98       412
   macro avg       0.98      0.98      0.98       412
weighted avg       0.98      0.98      0.98       412
```

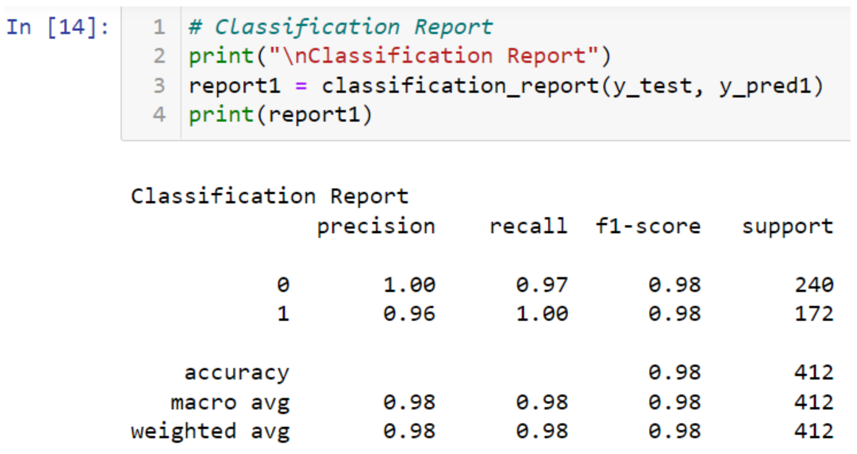
7) These are the hyperparameters which are selected through GridSearchCV using the default values. As compared to the previous question, this time round I included the list of ‘penalty’ into GridSearchCV. The differences between L1 and L2 regularization are:
   - L1 regularization penalizes the sum of absolute values of the weights, whereas L2 regularization penalizes the sum of squares of the weights.
   - The L1 regularization solution is sparse. The L2 regularization solution is non-sparse.
   - L2 regularization doesn’t perform feature selection since weights are only reduced to values near 0 instead of 0. L1 regularization has built-in feature selection.
   - L1 regularization is robust to outliers, L2 regularization is not.

   Therefore, the accuracy under l1(refer to screenshot below) is better as compared l2(refer to Qns5 screenshot) though is not very significant.
   Below are the optimal hyperparameters and the accuracy:



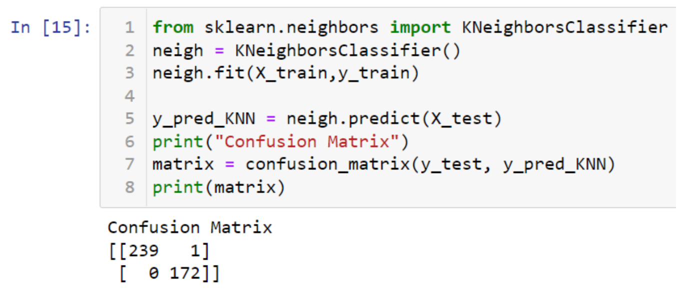
```
In [12]: grid1 = {
             'alpha': [0.001, 0.003, 0.01, 0.03, 0.1, 0.3],
             'max_iter': [1000,2500,5000,7500,10000,12500,15000,17500,20000],
             'learning_rate': ['constant','optimal','invscaling','adaptive'],
             'tol': [0.0001, 0.001, 0.01, 0.1],
             'eta0': [0.01],
             'penalty': ['l1', 'l2', 'elasticnet']
             }

         sgd_cv1=GridSearchCV(clf,grid1)
         sgd_cv1.fit(X_train,y_train)
         print("tuned hyperparameters :(best parameters) ",sgd_cv1.best_params_)
         print("accuracy :",sgd_cv1.best_score_)

tuned hyperparameters :(best parameters)  {'alpha': 0.001, 'eta0': 0.01, 'learning_rate': 'optimal', 'max_iter': 10000, 'penalt
y': 'l1', 'tol': 0.01}
accuracy : 0.9770833333333334
```

8) Refer to the screenshot below:

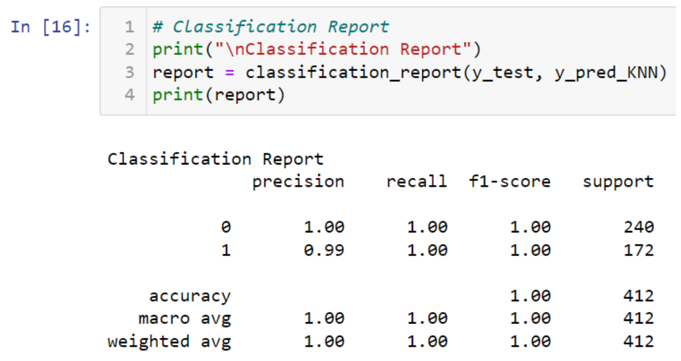


```
In [14]: # Classification Report
         print("\nClassification Report")
         report1 = classification_report(y_test, y_pred1)
         print(report1)


Classification Report
              precision    recall  f1-score   support

           0       1.00      0.97      0.98       240
           1       0.96      1.00      0.98       172

    accuracy                           0.98       412
   macro avg       0.98      0.98      0.98       412
weighted avg       0.98      0.98      0.98       412
```

9) Refer to the screenshot below:



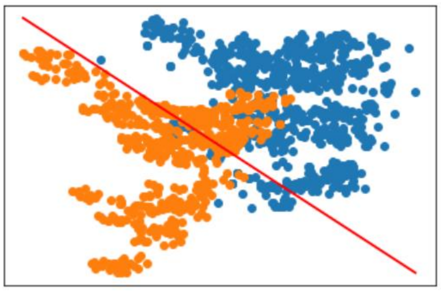
```
In [15]: from sklearn.neighbors import KNeighborsClassifier
         neigh = KNeighborsClassifier()
         neigh.fit(X_train,y_train)

         y_pred_KNN = neigh.predict(X_test)
         print("Confusion Matrix")
         matrix = confusion_matrix(y_test, y_pred_KNN)
         print(matrix)

Confusion Matrix
[[239   1]
 [  0 172]]
```



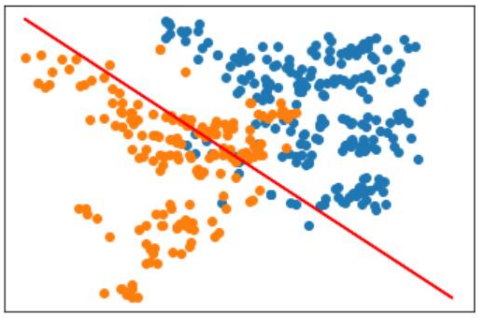
```
In [16]: # Classification Report
         print("\nClassification Report")
         report = classification_report(y_test, y_pred_KNN)
         print(report)


Classification Report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       240
           1       0.99      1.00      1.00       172

    accuracy                           1.00       412
   macro avg       1.00      1.00      1.00       412
weighted avg       1.00      1.00      1.00       412
```

As KNN is based on the shortest distances between the data points to make predictions while logistic regression is based on boundaries between two classes and models the probabilities of one class to make predictions. As the dataset is not ‘large’ enough plus the dataset given is probably already at the very ‘clean state’, thus the accuracy from KNN return as 1. Below are the graphs when the dataset is in raw state, after logistic regression and KNN:

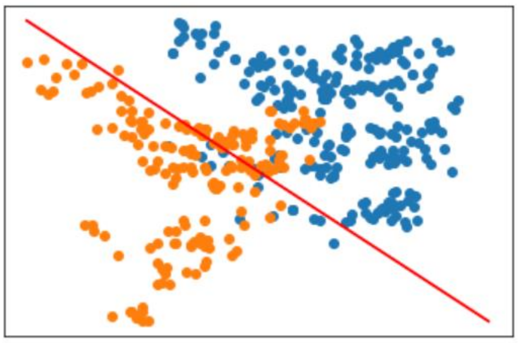
Before processing:



After logistic regression:



After KNN:



As shown above, the two classes are quite distinct from each other at the first place and compared the outcomes from logistic regression and KNN, there not much of difference between these 2 algorithms. Hence, this explains why the accuracy from these 2 models do not differ much.