# CSE 4020 - MACHINE LEARNING

## Lab 29+30

## Digital Assignment-2

Submitted by: Alokam Nikhitha(19BCE2555)

# Logistic Regression

## Question:

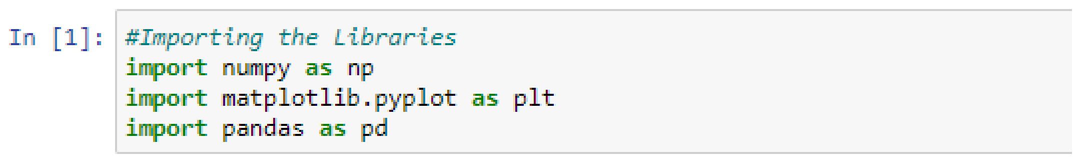
**Train a Logistic regression model to predict the charges is there or not for insurance company from given features.**

**Dataset Used:** “insurance.csv” as provided.

## Procedure:

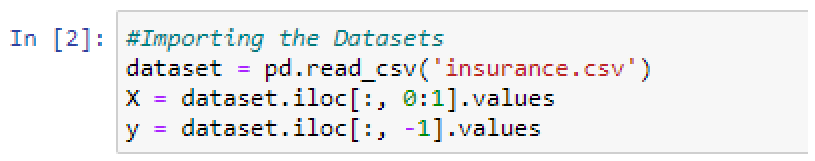
- We first import the dataset into our workspace the use of pandas.
- We then need to determine at the impartial and based attributed for use in our regression version.
- We then need to initialize our Linear regression version and logistic regression and match it to the X and y attributes.
- Next, we need to create any other variable to save the outcomes of X set as anticipated with the aid of using our regression version.
- We then can discover the scatter plot of our units and the exceptional match Regression line.
- Finally, we calculate our assessment metrics to test the accuracy of our version.

## Code Snippets and Explanation:



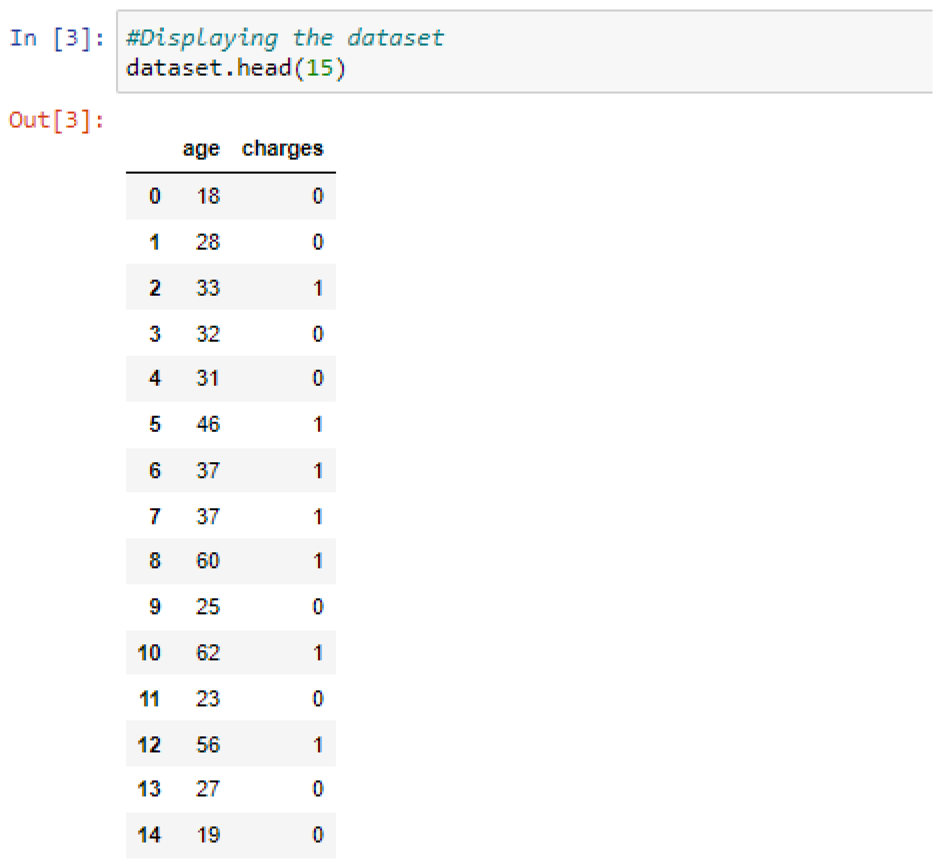
```
In [1]: #Importing the Libraries
        import numpy as np
        import matplotlib.pyplot as plt
        import pandas as pd
```

**Here we are importing the libraries.**



```
In [2]: #Importing the Datasets
        dataset = pd.read_csv('insurance.csv')
        X = dataset.iloc[:, 0:1].values
        y = dataset.iloc[:, -1].values
```

**We're importing the dataset into our workspace and naming the set of independent attributes X and the set of dependent attributes y.**



```
In [3]: #Displaying the dataset
        dataset.head(15)

Out[3]:
            age  charges
        0    18        0
        1    28        0
        2    33        1
        3    32        0
        4    31        0
        5    46        1
        6    37        1
        7    37        1
        8    60        1
        9    25        0
        10   62        1
        11   23        0
        12   56        1
        13   27        0
        14   19        0
```

**The first 15 rows of our dataset are seen here. The charges attribute is a categorical attribute, as we can see, with 'No' labelled as 0 and 'Yes' tagged as 1. With values ranging from 18 to 62, the age property is both continuous and discrete.**

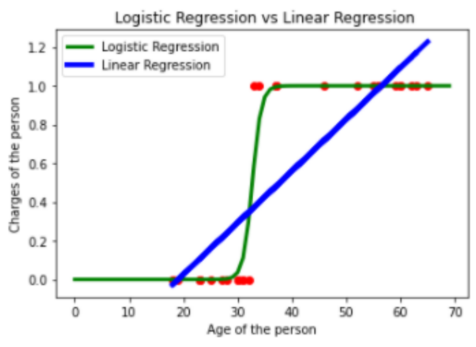
```
In [4]: # Training the Logistic Regression Model
        from sklearn.linear_model import LogisticRegression
        classifier = LogisticRegression(random_state=0)
        classifier.fit(X, y)

Out[4]: LogisticRegression(random_state=0)
```

**Here we have trained our Logistic regression model with X and y set.**

```
In [5]: # Trainig the Linear Regression model
        from sklearn.linear_model import LinearRegression
        regressor = LinearRegression()
        regressor.fit(X, y)

Out[5]: LinearRegression()
```

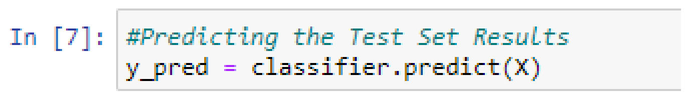
**Here we have trained our linear regression model with X and y set.**

```
In [6]: #Visualizing the Logistic curve and Linear Regressor
        Xs = [i for i in range(0, 70)]
        Ys = [classifier.predict_proba([[value]])[0][1] for value in range(0, 70)]

        plt.scatter(X, y, color='red')
        plt.plot(Xs, Ys, color='Green', linewidth=3, label='Logistic Regression')
        plt.plot(X, regressor.predict(X), color='blue', linewidth=4, label='Linear Regression')
        plt.xlabel('Age of the person')
        plt.ylabel('Charges of the person')
        plt.legend()
        plt.title(' Logistic Regression vs Linear Regression')

Out[6]: Text(0.5, 1.0, ' Logistic Regression vs Linear Regression')
```



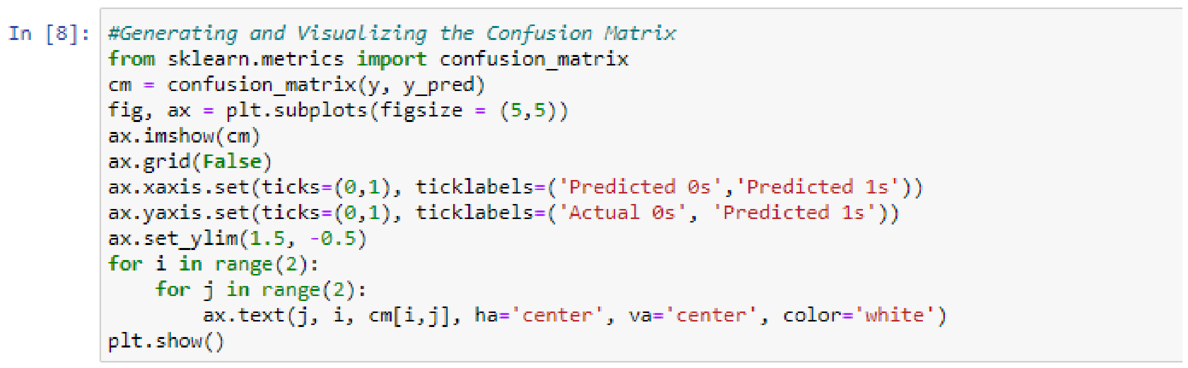
The probability line of our logistic regression is presented here. Any value more than 0.5 is defined as "charged," while any value less than 0.5 is classified as "no charge." We have also plotted the values as predicted by linear regressor.

The Logistic Regression is plotted in Green color and Linear Regression in Blue color.

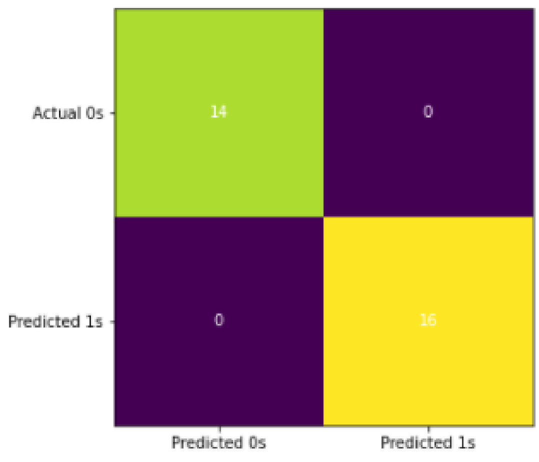


```
In [7]: #Predicting the Test Set Results
        y_pred = classifier.predict(X)
```

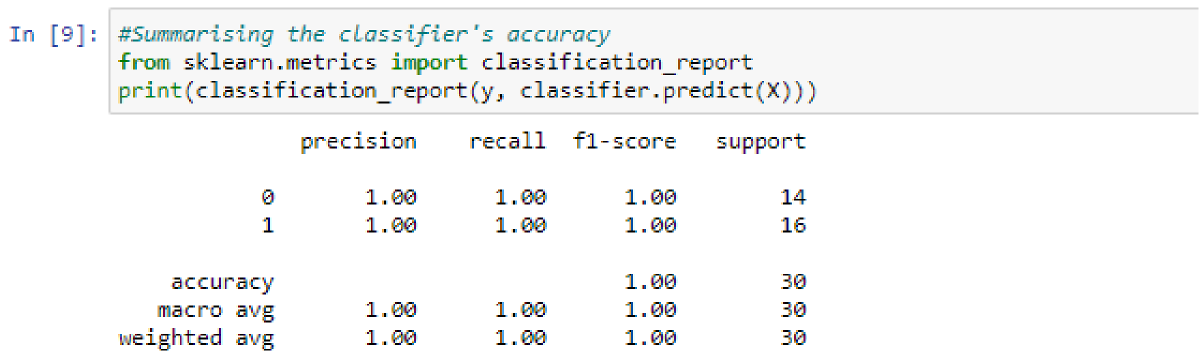
We create another vector to store the result of our X set as predicted by the trained classifier.



```
In [8]: #Generating and Visualizing the Confusion Matrix
        from sklearn.metrics import confusion_matrix
        cm = confusion_matrix(y, y_pred)
        fig, ax = plt.subplots(figsize = (5,5))
        ax.imshow(cm)
        ax.grid(False)
        ax.xaxis.set(ticks=(0,1), ticklabels=('Predicted 0s','Predicted 1s'))
        ax.yaxis.set(ticks=(0,1), ticklabels=('Actual 0s', 'Predicted 1s'))
        ax.set_ylim(1.5, -0.5)
        for i in range(2):
            for j in range(2):
                ax.text(j, i, cm[i,j], ha='center', va='center', color='white')
        plt.show()
```



Here, we have visualised our predicted results and actual results. We can clearly see that the true positives and true negatives account for complete results and thus our logistic regression classifier is 100% accurate, that is, actual set is same as the predicted set.

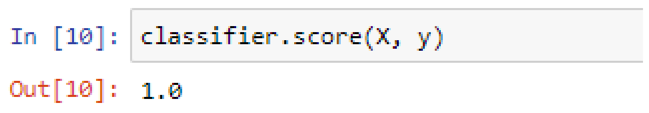


```
In [9]: #Summarising the classifier's accuracy
        from sklearn.metrics import classification_report
        print(classification_report(y, classifier.predict(X)))

                      precision    recall  f1-score   support

                   0       1.00      1.00      1.00        14
                   1       1.00      1.00      1.00        16

            accuracy                           1.00        30
           macro avg       1.00      1.00      1.00        30
        weighted avg       1.00      1.00      1.00        30
```

Here we have analysed the performance of our logistic regression classifier.

As we can see, the precision and recall for both 1 and 0 accounts for 1.00, that is all the values of 0 and 1 were identified correctly.

Also, the accuracy of out model is 1.00 (100%) as macro average and weighted average are both 1.00.



```
 In [10]: classifier.score(X, y)

Out[10]: 1.0
```

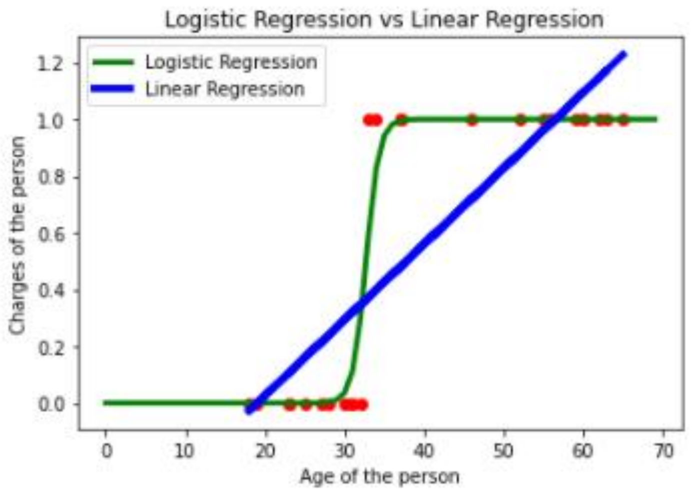
Finally, the classifier score is also 1 as all the values were correctly identified.

## Results and Conclusion:

| | |
|---|---|
| 1. Identified ‘no charge’ | = 14 |
| 2. True ‘no charge’ | = 14 |
| 3. Identified ‘charged’ | = 16 |
| 4. True ‘charged’ | = 16 |
| 5. Precision of ‘no charge’ | = 1.00 |
| 6. Precision of ‘charged’ | = 1.00 |
| 7. Recall of ‘no charge’ | = 1.00 |
| 8. Recall of ‘charged’ | = 1.00 |
| 9. Model Accuracy | = 100% |

- **Logistic Regression vs Linear Regression**

```
Out[6]: Text(0.5, 1.0, ' Logistic Regression vs Linear Regression')
```



# Multilayer Perceptron

## Question:

Use Multi-Layer Perceptron to classify the input data into ‘Iris-setosa’ or ‘Iris-versicolor’ or ‘Iris-virginica’.

## Dataset Used: “iris.csv”

Downloaded from: https://www.kaggle.com/uciml/iris

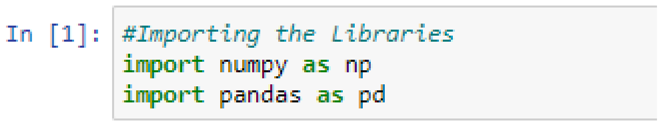
## Procedure:

- We first import the pandas and numpy.
- We then import Data set using Pandas.
- We choose the independent and dependent attributed to be used in our regression model.
- Then we encode our label class using LabelEncoder.
- Later, We split our dataset into training set and test set.
- Then we feature scale our X_train and X_test sets.
- We then have to initialize our Multi-Layer Perceptron model and fit it to the X_train and y_train attributes.
- Next, we have to create two other variables to store the results of X_train and X_test set as predicted by our classification model.
- Finally, we calculate our evaluation metrics to check the accuracy of our model.

## Procedure to adjust weights:

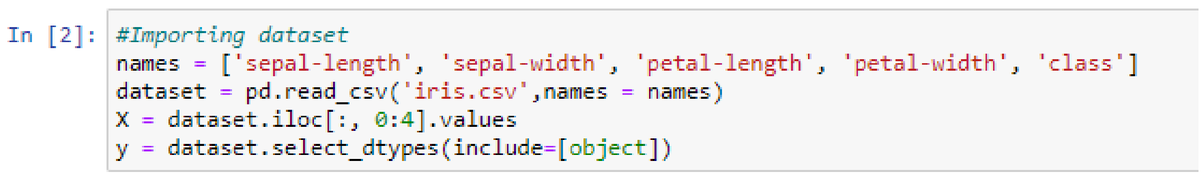
- Present inputs for the first pattern.
- Sum the weighted input to the next layer and calculate their activations.
- Present activations to the next layer, repeating the previous step until the activations of the output later are known.
- Compare output activations to the target values for the pattern and calculate deltas for the output layer.
- Propagate error backwards by using the output layer deltas to calculate the deltas for the previous layer.
- Use these deltas to calculate those of the previous layer, repeating until the first layer is reached.
- Calculate the weight changes for all weights and biases (treat biases as weights from a unit having an activation of 1).
- If training by pattern, update all the weights and biases, else repeat the cycle for all patterns, summing the changes and applying at the end of the epoch.
- Repeat the entire procedure until the total sum of squared errors is less than a specified criterion.

## Code Snippet and Explanation:



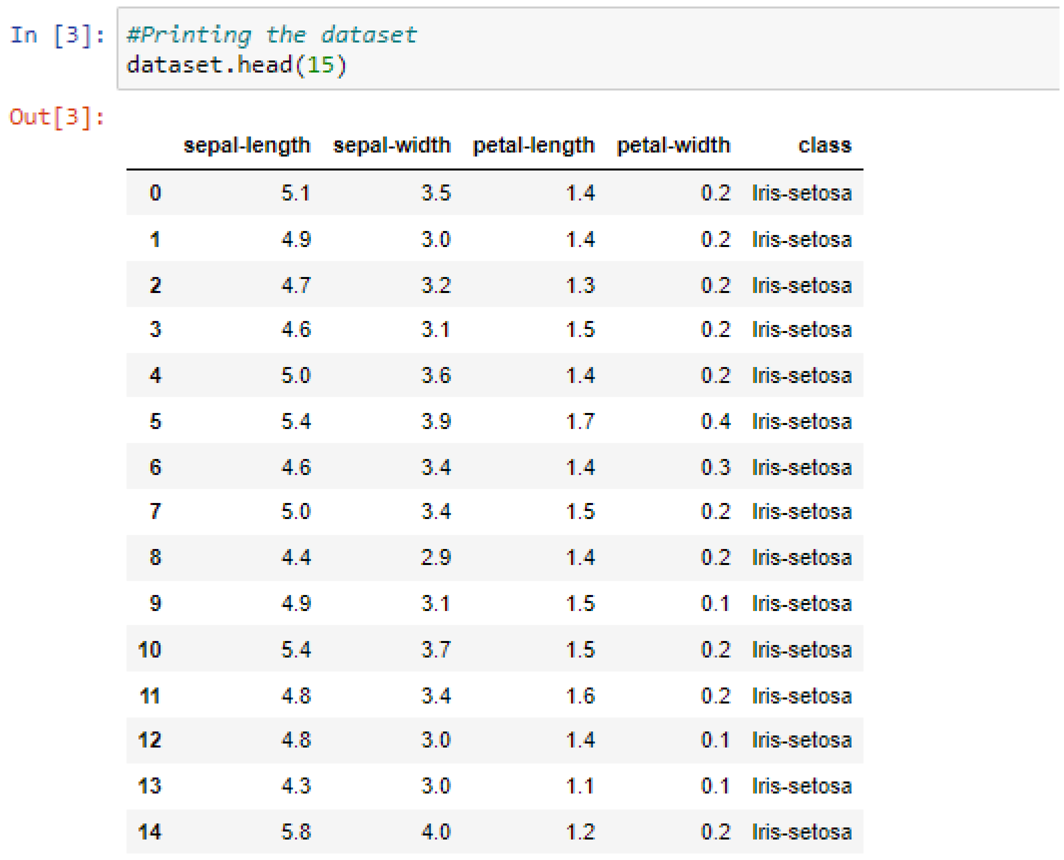
```
In [1]: #Importing the Libraries
        import numpy as np
        import pandas as pd
```

Here we are importing the necessary libraries.



```
In [2]: #Importing dataset
        names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']
        dataset = pd.read_csv('iris.csv',names = names)
        X = dataset.iloc[:, 0:4].values
        y = dataset.select_dtypes(include=[object])
```

**Here we will import the dataset into our workspace. The name list is used to assign header values to our dataset since the dataset has no attribute headers.**

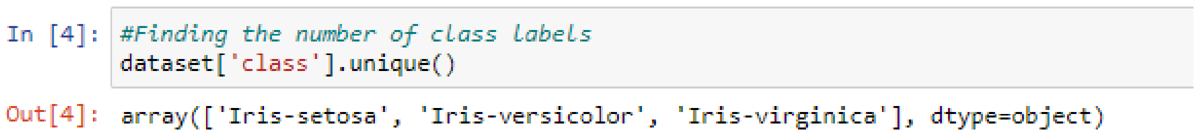


```
In [3]: #Printing the dataset
        dataset.head(15)
```

Out[3]:

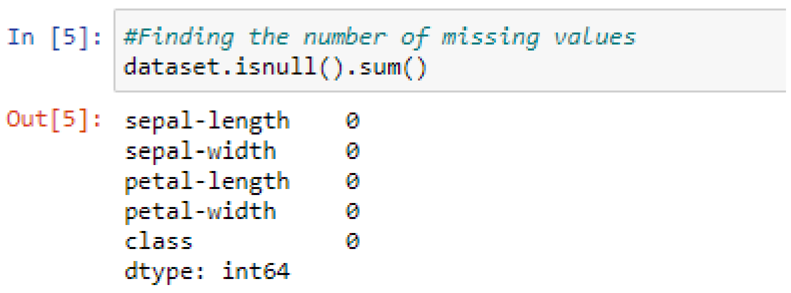
| | sepal-length | sepal-width | petal-length | petal-width | class |
|---|---|---|---|---|---|
| **0** | 5.1 | 3.5 | 1.4 | 0.2 | Iris-setosa |
| **1** | 4.9 | 3.0 | 1.4 | 0.2 | Iris-setosa |
| **2** | 4.7 | 3.2 | 1.3 | 0.2 | Iris-setosa |
| **3** | 4.6 | 3.1 | 1.5 | 0.2 | Iris-setosa |
| **4** | 5.0 | 3.6 | 1.4 | 0.2 | Iris-setosa |
| **5** | 5.4 | 3.9 | 1.7 | 0.4 | Iris-setosa |
| **6** | 4.6 | 3.4 | 1.4 | 0.3 | Iris-setosa |
| **7** | 5.0 | 3.4 | 1.5 | 0.2 | Iris-setosa |
| **8** | 4.4 | 2.9 | 1.4 | 0.2 | Iris-setosa |
| **9** | 4.9 | 3.1 | 1.5 | 0.1 | Iris-setosa |
| **10** | 5.4 | 3.7 | 1.5 | 0.2 | Iris-setosa |
| **11** | 4.8 | 3.4 | 1.6 | 0.2 | Iris-setosa |
| **12** | 4.8 | 3.0 | 1.4 | 0.1 | Iris-setosa |
| **13** | 4.3 | 3.0 | 1.1 | 0.1 | Iris-setosa |
| **14** | 5.8 | 4.0 | 1.2 | 0.2 | Iris-setosa |

**Printing First 15 Records from Dataset**



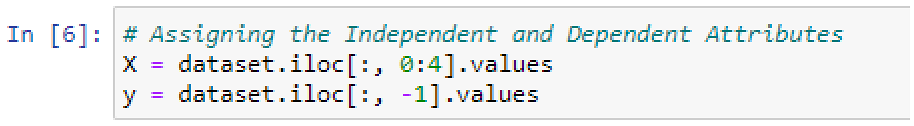
```
In [4]: #Finding the number of class labels
        dataset['class'].unique()

Out[4]: array(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'], dtype=object)
```

**Here, we are finding the number of class labels in our dataset. This suggests that there are 3 class labels, namely Iris-setosa, Iris-versicolor and Iris-virginica.**



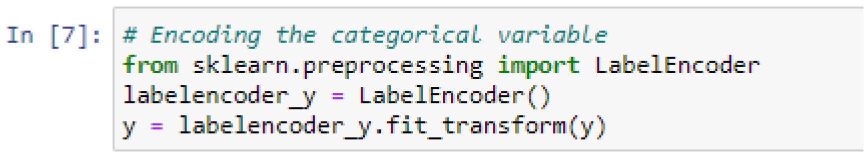
```
In [5]: #Finding the number of missing values
        dataset.isnull().sum()

Out[5]: sepal-length    0
        sepal-width     0
        petal-length    0
        petal-width     0
        class           0
        dtype: int64
```

**Here we check if our dataset contains any missing values and as we can see that there are no missing values as sum of isnull of each attribute is 0.**



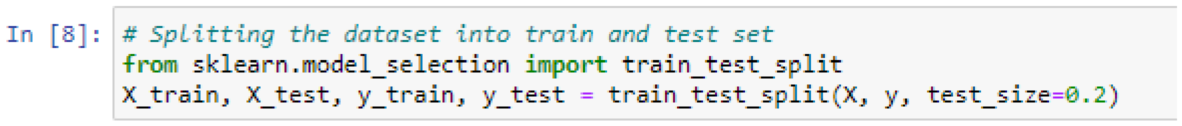
```
In [6]: # Assigning the Independent and Dependent Attributes
        X = dataset.iloc[:, 0:4].values
        y = dataset.iloc[:, -1].values
```

**Here we assigned a set of independent attributes to set X and a set of dependent attributes to set y.**



```
In [7]: # Encoding the categorical variable
        from sklearn.preprocessing import LabelEncoder
        labelencoder_y = LabelEncoder()
        y = labelencoder_y.fit_transform(y)
```

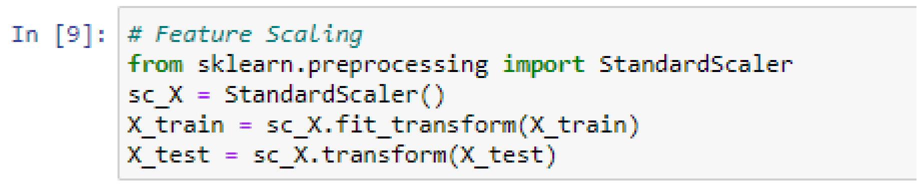
**Since our y attribute is categorical, we need to code it to numeric values, and for this we use the LabelEncoder class.**



```
In [8]: # Splitting the dataset into train and test set
        from sklearn.model_selection import train_test_split
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

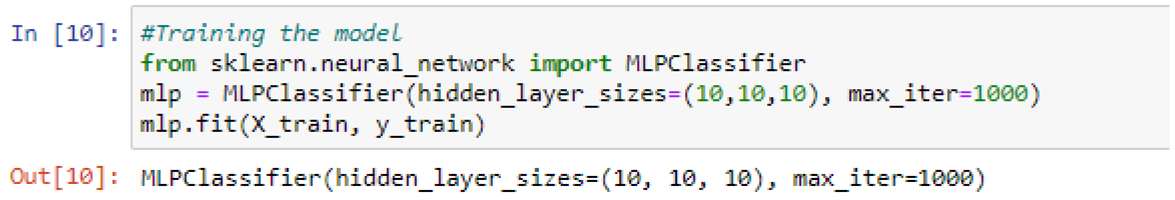
**Here, we split our dataset into a training set and a test set with 20% of the total values assigned to the test set and**

80% of the values assigned to the training set. The training set is used to train the model and the test set is used to evaluate the performance of our model.



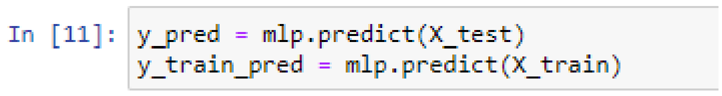
```
In [9]: # Feature Scaling
        from sklearn.preprocessing import StandardScaler
        sc_X = StandardScaler()
        X_train = sc_X.fit_transform(X_train)
        X_test = sc_X.transform(X_test)
```

Since the range of values for each attribute is different, it can affect the performance of our classifier, so we need to reduce each attribute to the same range. For this we use the mathematical normalization technique and to implement it we use the StandardScalar class.



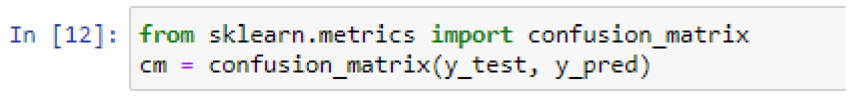
```
In [10]: #Training the model
         from sklearn.neural_network import MLPClassifier
         mlp = MLPClassifier(hidden_layer_sizes=(10,10,10), max_iter=1000)
         mlp.fit(X_train, y_train)

Out[10]: MLPClassifier(hidden_layer_sizes=(10, 10, 10), max_iter=1000)
```

Here we have trained our Muti-Layer Perceptron using training set data.

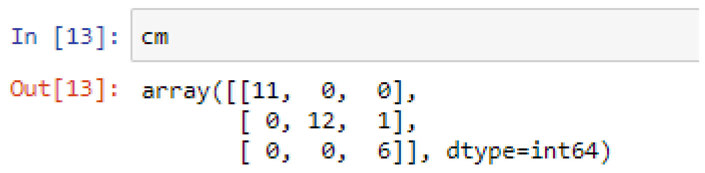


```
In [11]: y_pred = mlp.predict(X_test)
         y_train_pred = mlp.predict(X_train)
```

Here we have initialized the variable y_pred with the list of values predicted by our classifier in our test set and the variable y_train_pred with the list of values predicted by our classifier in our training set.

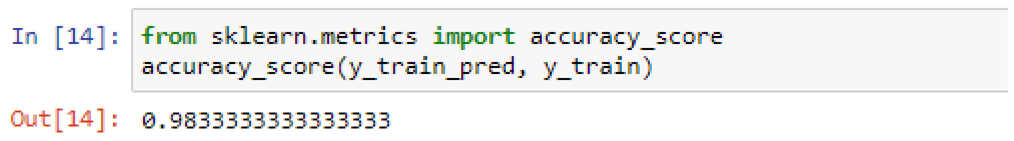


```
In [12]: from sklearn.metrics import confusion_matrix
         cm = confusion_matrix(y_test, y_pred)
```



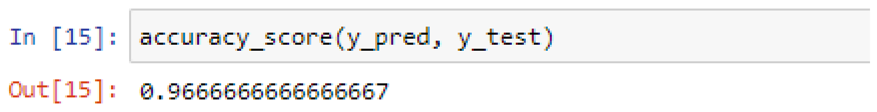
```
In [13]: cm

Out[13]: array([[11,  0,  0],
                [ 0, 12,  1],
                [ 0,  0,  6]], dtype=int64)
```

**Here we have printed our confusion matrix. As we can see, our classifier has correctly identified 30 values of each category. 11 values belonging to Iris-senosta, 12 values belonging to Iris-versicolor and 6 values belonging to Iris-virginica. And 1 from other.**



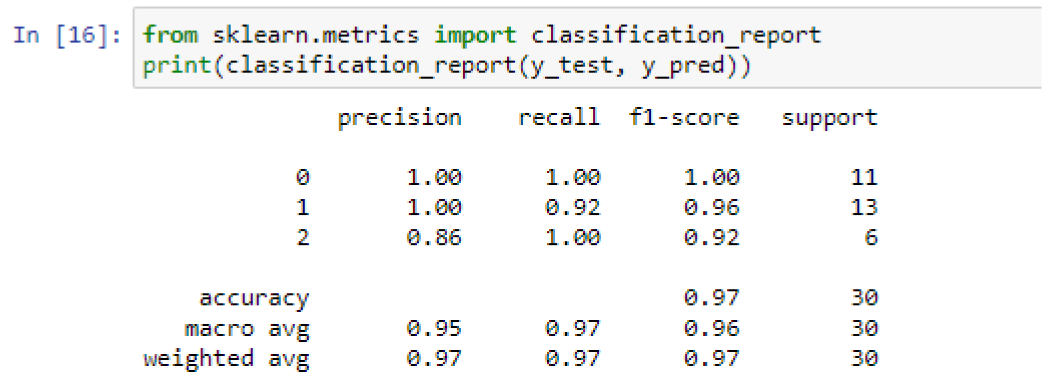
```
In [14]: from sklearn.metrics import accuracy_score
         accuracy_score(y_train_pred, y_train)

Out[14]: 0.9833333333333333
```

**Here we have printed the accuracy of our training set results and it comes out to be 98.33%.**



```
In [15]: accuracy_score(y_pred, y_test)

Out[15]: 0.9666666666666667
```

**Here we have printed the accuracy of our test set results and it comes out to be 96.67% as our classifier classified each value correctly which could be seen in the confusion matrix as well.**



```
In [16]: from sklearn.metrics import classification_report
         print(classification_report(y_test, y_pred))

              precision    recall  f1-score   support

           0       1.00      1.00      1.00        11
           1       1.00      0.92      0.96        13
           2       0.86      1.00      0.92         6

    accuracy                           0.97        30
   macro avg       0.95      0.97      0.96        30
weighted avg       0.97      0.97      0.97        30
```

Here we have printed the classification report of our classifier's performance.

## Result and Conclusion:

| | |
|---|---|
| ❖ Identified 'Iris-senosta' | = 11 |
| ❖ True 'Iris-senosta' | = 11 |
| ❖ Identified 'Iris-versicolor' | = 12 |
| ❖ True 'Iris-versicolor' | = 12 |
| ❖ Identified 'Iris-verginica' | = 6 |
| ❖ True 'Iris-verginica' | = 6 |

| | |
|---|---|
| ❖ Precision of 'Iris-senosta' | = 1.00 |
| ❖ Precision of 'Iris-versicolor' | = 1.00 |
| ❖ Precision of 'Iris-verginica' | = 1.00 |
| ❖ Recall of 'Iris-senosta' | = 1.00 |
| ❖ Recall of 'Iris-versicolor' | = 1.00 |
| ❖ Recall of 'Iris-verginica' | = 1.00 |

| | |
|---|---|
| ❖ Model Accuracy on test set | = 96.67% |
| ❖ Model Accuracy on train set | = 98.33% |