ab 5:-

step-1: import pandas as pd

```
import ·csv

dfab= ·pd.red_csv(“diabetes. csv”)

dfab.

dfab. head()

dfab. shape.

df = pd-read - csv(“diabetes. csv”)

df.

d = df-columns

b  df. info()
   df.dtypes
   df.dtypes.value - counts()
```

step2:

```
X= df. drop (“Outcome”, axis=1)

y= df[[“Outcome”]]

X
y

from. sklearn. model - sle:selection import.
statified - shuffle .split.
```

Lab5. Diabetes Classification using Logistic Regression

Objectives

In this lab, you will classify using Logistic Regression model, whether a person will become diabetic or not using PIMA diabetes dataset available in UCI. You will also predict a new person who is not in the dataset will become diabetic of not based on his details.

Learning Outcomes

After completing this lab, you will be able to

- Understand data and build baseline LogisticRegression model
- Create Heatmap with confusion matrix values
- Apply scaling using StandardScaler and MinMaxScaler and rebuild LoR model
- Print classification metrics scores and plot ROC curve
- Compare the performance of LoR model with LogisticRegressionCV with L1 and L2 regularization

Business Use Case

You are a data scientist. A leading hospital in your city has approached you with the medical details of their patients related to their diabetes information. The hospital has given you a file (diabetes.csv) that contains details of 768 patients and each patient is described with these 9 features. Here, ***Outcome*** is the ***dependent variable*** and ***all other 8 variables are independent variables.***

- Pregnancies: Number of times pregnant
- Glucose: Plasma glucose concentration over 2 hours in an oral glucose tolerance test
- BloodPressure: Diastolic blood pressure (mm Hg)
- SkinThickness: Triceps skin fold thickness (mm)
- Insulin: 2-Hour serum insulin (mu U/ml)
- BMI: Body mass index (weight in kg/(height in m)2)
- DiabetesPedigreeFunction: Diabetes pedigree function (a function which scores likelihood of diabetes based on family history)
- Age: Age (years)
- Outcome: Class variable (0 if non-diabetic, 1 if diabetic)

The hospital wants you to build a model so that the model will assess when a new patient visits them he will become diabetic or not.

Step1. [Understand Data]. Using Pandas, import **"diabetes.csv"** file and print properties such as head, shape, columns, dtype, info and value_counts.

Step2. [Build Logistic Regression Model]

- Prepare X matrix (8 feature columns) and y vector (ie., Outcome column)
- Split dataset with stratified shuffle split for training and testing as X_train, X_test, y_train, y_test (use **25%** test size).
- Create LogisticRegression model, fit on training set and predict on test set

Step3. [Predict on a new sample]

- Will this person become diabetic?. His details are given below.
- new_person = [[6, 200, 90, 10, 25, 23.3, 0.672, 42]]

Step3. [Compute Classification Metrics]

- Compute and print Accuracy, Precision, Recall and AUC scores

Step4. [Understand Correlation]

- Create confusion matrix between y_test and y_pred and plot confusion matrix values in a Heatmap. Explain the meaning of the 4 numbers you get.

stratified.shuffle - split()

```
shuf = stratified shuffle split (n-split=4, test-size=0.25,
                                            random-state=0)

    shuf.get-n-splits (X,Y)

  import . warnings

  warnings, filterwarnings ("ignore")

  for train, test in shuf.split (x,Y):
   X-train, X-test = X.iloc[train], X.loc[test]
   y-train, y-test = y.loc [train], Y.iloc[test]

from sklearn. linear. model.import logistic Regression.

     logmodel = Logistic Regression()
      logmodel fit (X-train, Y-train)
        y-predic = logmodel. predict (X-test)
         y-predic
    log model. score(X-train, Y-train)
```

step 3:-

```
   new person = [[6,200,90,10,25,33.3,0.672,400]]
    Print (log model.predict (new-person))
```

Precision:-

```
   from sklearn.mattrices import precision-score
     print (precision- score, (y-test, y-predic))
```

Recall:-

```
   from sklearn.metrices.import recall score
   print (recall- score (y-test, y-predic))
```

Scanned with CamScanner

Step5. [Normalization using MinmaxScaler and rebuild LoR]

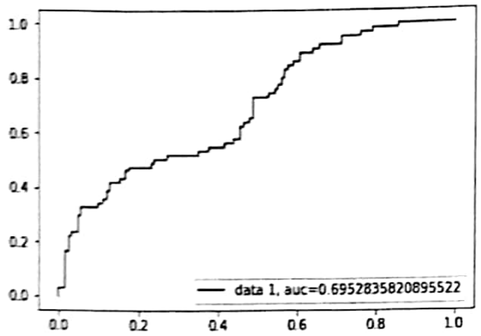
- Now, normalize your X_train and X_test values using MinmaxScaler
- Create a new LogisticRegression model, fit on normalized training set and predict on the normalized test set
- Compute and print Accuracy, Precision, Recall and AUC scores

Step6. [Normalization using StandardScaler and rebuild LoR]

- Repeat Step5 with StandardScaler
- Among the 3 models, which model gives better classification scores?

Step7. [Plot ROC curve]

- Plot ROC curve as shown below. You can use the MinmaxScaler scaled values of X_test for computing predict_proba() score.

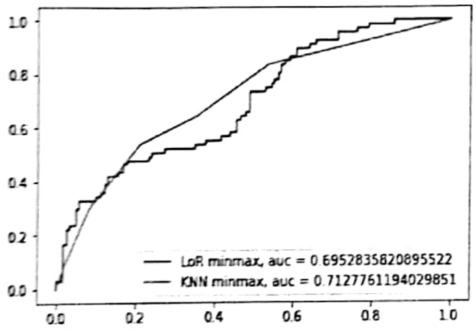


Step8. [Comparison with KNN classifier].

Create a KNN classifier with default values, fit on the scaled X using MinmaxScaler, predict and print classification metric scores.

Step9. [Update ROC curve]

- Update your ROC curve, this time, with one more curve of KNN classifier, as shown below.



Step10. [Regularization]

- In order to reduce overfitting of your data, you will use *LogisticRegressionCV* model with L1 and L2 regularization parameters. Create both models using the following statements
  - `model1 = LogisticRegressionCV(Cs=10, cv=4, penalty='l1', solver='liblinear')`
  - `model2 = LogisticRegressionCV(Cs=10, cv=4, penalty='l2')`
- Perform fit using MinmaxScaler scaled values and predict

Step11. [Update ROC curve]

- Update your ROC curve, this time, with two more curves, as shown below

Accuracy:-

```
from . sklearn .metrices . import accuracy_score
log_acscore = accuracy_score (y_test, y_predic)

log_acscore .
```

AUC Scores

```
from sklearn.metrices import roc_auc_score
print (roc_auc_score (y_test, y_predic))
```

Step:4

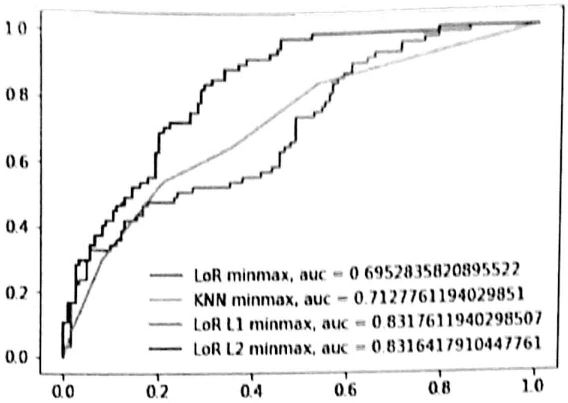
```
from sklearn.metrices import confersion_matrix
Confu_matrix = confersion_matrix (y_test, y. peredic)
Confu_matrix

Confu_accu_score = accuracy_score(y_test, y_peredic)
Confu_accu_score .

import seaborn as sns.
sns. heatmap (confusion_matrix (y_test, y_periodic)
        / len (y), cmap = 'Y1GnBU' annot = True)
```

Step:-5

```
from . sklearn. Preprocessing import . minmax Scalar.
Scales = Minmaxscaler()
X_trained_min = scaler. fit_transform(x_train)
X_test_min = Scaler.transform(x_test)
X_trained_min. shape
X_tested_min. shape
```



Out of these 4 models, which model performs the best?. How?. Why?.

```
from sklearn.Linear-model import Logistic Regression.
 logmodel 1 = logistic Regression ()
  logmodel 1 . fit (x - trained_min , y train
  y_ predict = logmodel 1 . predict(x - tested_min)
  y_ predict.
 logmodel 1.score(x-trained.min, y.train)
```

precision:

```
from sklearn.metrices import precision_score
 print (precision_score (y_test, y-predict)
```

Recall

```
from sklearn. metrices. import recall_score
 print (recall_score (y-test, y_predict)
```

Accuracy.

```
from sklearn. metrices. import acchracy- score
 min_accuscore = accuracy - score(y_test, y_predict)
 min_accscore
```

Auc Score:

```
from sklearn. metrices import roc_auc_score

log_auc_sc = roc_auc_score (y-test, y-predict)

log_auc 1 = (LOR.minman, Auc =,' log_acscore)

log_auc1.
```

step:6

```
from sklearn. preprocessing import standardscaler
scaler = . Standard scaler()
X_trained_stand = Scaler. fit_transform
                                        (X_train)
X_tested_stand = scaler - transform (X_test)
X_trained _stand.shape
X_tested_stand.shape
from sklearn. linear_model import . logistic Regression
    logmodel 2 = logistic Regression ()
log model 2.fit (Xtrained _stand , y-train)
y_predict_stand = logmodel 2. predict (X_tested
                                        _stand)
    y_predict_stand

logmodel 2. Score(x_trained. stand, y-train)
```

precision score

```
from sklearn . metrics . import . precision_Score.
point ( precision _score (y-test, y-predict stand))
```

NOTES

Recall score

```
from. Sklearnmetrices import recall_score
print (recall - score (y- test, y-predict- stand, ))
```

Accuracy:

```
from sklearn.metrices. import accuracy-store
stand_accscore = accuracy_score(y- test, y-predict-stand)
stand_accuscore.
```

AUC score

```
from sklearn. metrices import . roc_auc - score
stand _aucscore = roc_ auc - Score (y-test, y -
predict _stand)

stand_aucs = (AUC= stand_aucscore)
stand_auc3.

Print ('logistic Regression model) ', log_acscore)
Print ('Minmax Scaler:', min_accuscore
Print ('standard scaler:', stand_accscore)
```

Step #:

```
from sklearn. metrices import. roc-curve
Pred_prb3 = log model 1, . Predict - proba
(x_ tested - min)
```

```
d prb3, tprb3, threeshold = roc_curve
    (y-test, pred_prb3[:1], pos_label=1)
 import matplotlib.pyplot as plt
   plt.style.use('seaborn')
   plt.annotate(xy=[0.7,0], s=stand_auc3)
 plt.plot(fpr1, tpr1, linestyle=':', color='black',
            label='Logistic Regression')
 plt.title('Roc curve')
 plt.xlabel('False positive Rate')
 plt.ylabel('True positive Rate')
```

step: 3:

```
   from sklearn.neighbors import Kneighbours
                                 classifier.
 log model 3 = KNeighbors classifier(n-neighbors=3)
 log model 3.fit(x-trained_min, y-train)
   knn-y-pred = log model 3.predict(x-tested_min)
     knn-y-pred
```