**Data Warehousing & Data Mining LAB - G2  
EXPERIMENT 10**

**-** ASHISH KUMAR  
 - 2K18/SE/041

# Aim:- Write a program in Python Language to implement Model Evaluation (Cross Validation and Generate Confusion Matrix).

**Theory: - Cross-Validation** is a validation technique designed to evaluate and assess how the results of statistical analysis (model) will generalize to an independent dataset. Cross-Validation is primarily used in scenarios where prediction is the main aim, and the user wants to estimate how well and accurately a predictive model will perform in real-world situations. Cross-Validation seeks to define a dataset by testing the model in the training phase to help minimize problems like overfitting and underfitting.

### Benefits of Cross-Validation

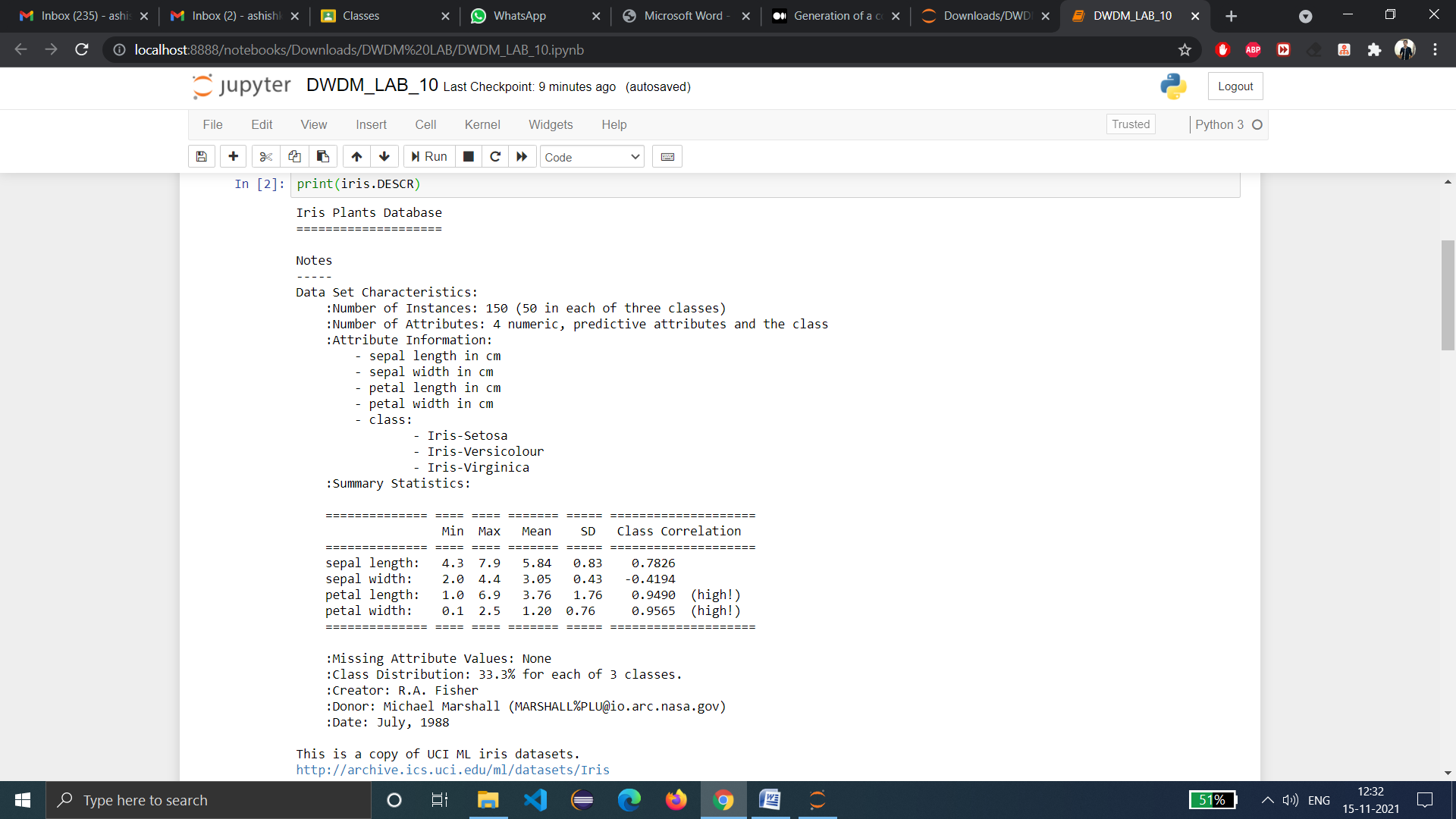
* It helps evaluate the quality of your model.
* It helps to reduce/avoid problems of overfitting and underfitting.
* It lets you select the model that will deliver the best performance on unseen data.

**K-Fold Cross Validation**: In this method, we split the data-set into k number of subsets(known as folds) then we perform training on the all the subsets but leave one(k-1) subset for the evaluation of the trained model. In this method, we iterate k times with a different subset reserved for testing purpose each time.

A **Confusion matrix** is an N x N matrix used for evaluating the performance of a classification model, where N is the number of target classes. The matrix compares the actual target values with those predicted by the machine learning model. This gives us a holistic view of how well our classification model is performing and what kinds of errors it is making.

**Note:** I have used **“iris data”** as a dataset in this experimentand used **K-Fold Cross validation** technique to generate Confusion matrix.

Picture of iris dataset:

****

**Source Code (in python):**

import itertools

import matplotlib.pyplot as plt

import numpy as np

from sklearn import svm, datasets

from sklearn.metrics import confusion\_matrix

from sklearn.model\_selection import KFold

# import IRIS dataset

iris = datasets.load\_iris()

data = iris.data

target = iris.target

class\_names = iris.target\_names

class\_names

labels, counts = np.unique(target, return\_counts=True)

print(iris.DESCR)

def evaluate\_model(data\_x, data\_y):

k\_fold = KFold(10, shuffle=True, random\_state=1)

predicted\_targets = np.array([])

actual\_targets = np.array([])

for train\_ix, test\_ix in k\_fold.split(data\_x):

train\_x, train\_y, test\_x, test\_y = data\_x[train\_ix], data\_y[train\_ix], data\_x[test\_ix],   
 data\_y[test\_ix]

# Fit the classifier

classifier = svm.SVC().fit(train\_x, train\_y)

# Predict the labels of the test set samples

predicted\_labels = classifier.predict(test\_x)

predicted\_targets = np.append(predicted\_targets, predicted\_labels)

actual\_targets = np.append(actual\_targets, test\_y)

return predicted\_targets, actual\_targets

def plot\_confusion\_matrix(predicted\_labels\_list, y\_test\_list):

cnf\_matrix = confusion\_matrix(y\_test\_list, predicted\_labels\_list)

np.set\_printoptions(precision=2)

# Plot non-normalized confusion matrix

plt.figure()

generate\_confusion\_matrix(cnf\_matrix, classes=class\_names, title='Confusion matrix, without   
 normalization')

plt.show()

# Plot normalized confusion matrix

plt.figure()

generate\_confusion\_matrix(cnf\_matrix, classes=class\_names, normalize=True,   
 title='Normalized confusion matrix')

plt.show()

def generate\_confusion\_matrix(cnf\_matrix, classes, normalize=False, title='Confusion matrix'):

if normalize:

cnf\_matrix = cnf\_matrix.astype('float') / cnf\_matrix.sum(axis=1)[:, np.newaxis]

print("Normalized confusion matrix")

else:

print('Confusion matrix, without normalization')

plt.imshow(cnf\_matrix, interpolation='nearest', cmap=plt.get\_cmap('Blues'))

plt.title(title)

plt.colorbar()

tick\_marks = np.arange(len(classes))

plt.xticks(tick\_marks, classes, rotation=45)

plt.yticks(tick\_marks, classes)

fmt = '.2f' if normalize else 'd'

thresh = cnf\_matrix.max() / 2.

for i, j in itertools.product(range(cnf\_matrix.shape[0]), range(cnf\_matrix.shape[1])):

plt.text(j, i, format(cnf\_matrix[i, j], fmt), horizontalalignment="center",

color="white" if cnf\_matrix[i, j] > thresh else "black")

plt.tight\_layout()

plt.ylabel('True label')

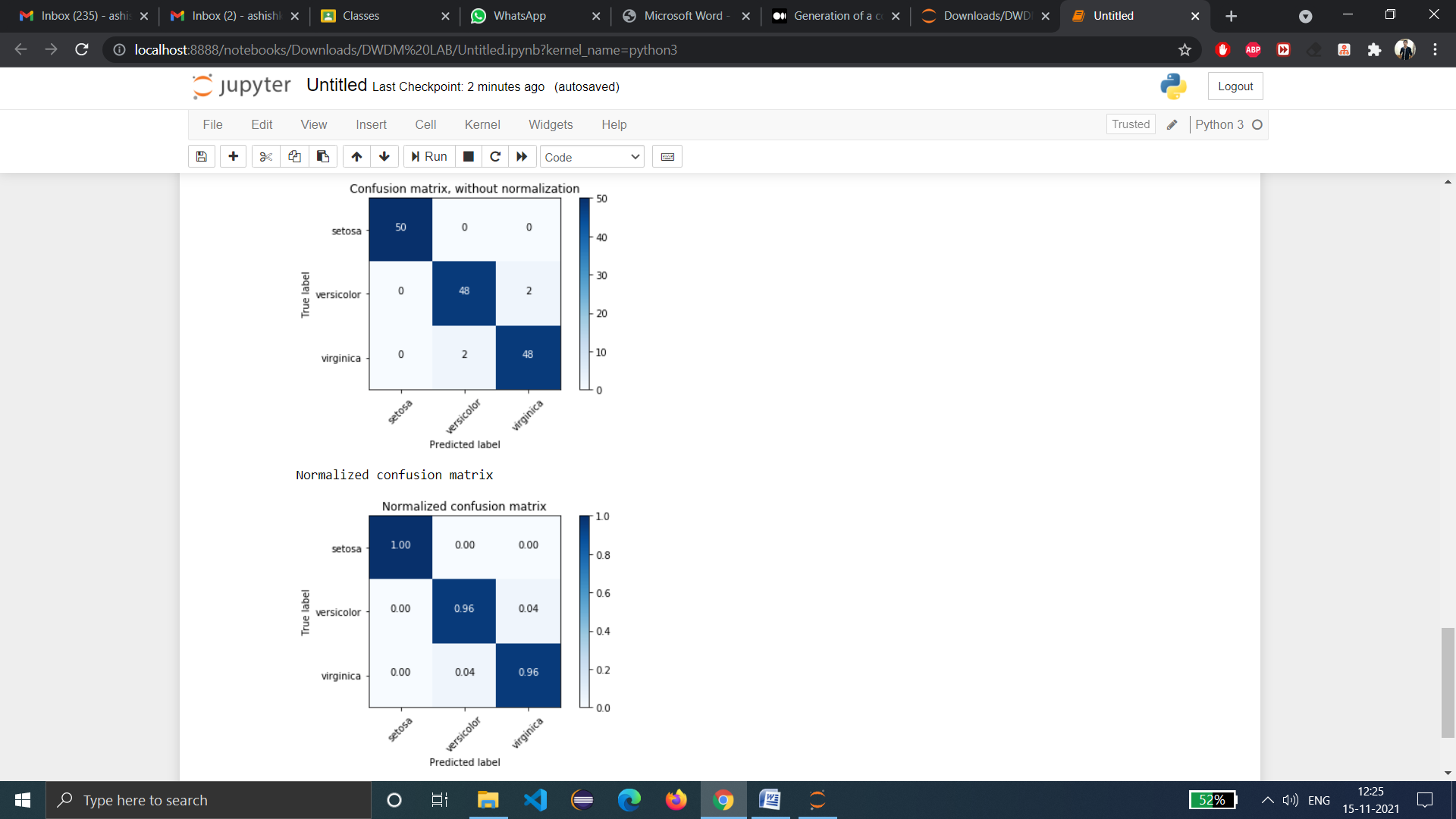
plt.xlabel('Predicted label')

return cnf\_matrix

predicted\_target, actual\_target = evaluate\_model(data, target)

plot\_confusion\_matrix(predicted\_target, actual\_target)

**OUTPUT-**

****

**Findings and Learning:**

* We have successfully implemented Cross validation and then generate confusion matrixin Python.
* We have learnt about the applications, strengths and weaknesses of Cross Validation and Confusion Matrix.