**Step 1: Import libraries**

import numpy as np

import pandas as pd

import matplotlib.pylab as plt

import seaborn as sns

from sklearn.model\_selection import KFold, cross\_val\_score

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

from sklearn import metrics

from sklearn import preprocessing

from sklearn.metrics import classification\_report, confusion\_matrix , precision\_score, recall\_score, auc,roc\_curve,accuracy\_score,f1\_score

from sklearn.metrics import roc\_curve, roc\_auc\_score

from sklearn.neighbors import KNeighborsClassifier

from sklearn.model\_selection import GridSearchCV

from sklearn.naive\_bayes import GaussianNB

from sklearn.svm import SVC

from sklearn.tree import DecisionTreeClassifier

from sklearn import tree

from sklearn.ensemble import RandomForestClassifier

from tabulate import tabulate

import warnings as warn

from warnings import filterwarnings

filterwarnings("ignore")

**Step 2: Read Dataset**

**# Đọc dữ liệu từ file đầu vào**

data = pd.read\_csv("IRIS.csv")

df = pd.DataFrame(data)

**# Xem giá trị đầu ra (cột species gồm các giá trị nào và số lượng các giá trị là bao nhiêu)**

df.groupby('species').count()

**# Mã hoá đầu ra (các loại hoa) thành các giá trị số học**

df["species"].replace({"Iris-setosa":0 , "Iris-versicolor":1 , "Iris-virginica":2} , inplace = True)

df

**Step 3: Dataset Overview**

**# In ra thông tin cơ bản của dữ liệu vừa đọc**

df.describe(include = 'all')

**# Vẽ đồ thị về mối quan hệ giữa các đại lượng (ma trận tương quan)**

plt.figure (figsize = (16 , 6))

mask = np.triu(np.ones\_like(df.corr(method = "spearman") , dtype = bool))

heatmap = sns.heatmap(df.corr(method = "spearman") , mask = mask , vmin = -1 , vmax = 1 ,

annot = True , cmap="BrBG")

heatmap.set\_title("Triangle Correlation Heatmap" , fontdict = {'fontsize': 18} , pad =16 )

sns.pairplot(df , hue='species' , diag\_kind="hist" , corner=True , palette = 'hls')

**Step 4: Data science and Visualization**

**# Vẽ đồ thị boxplot để xem phân bố các giá trị của từng cột dữ liệu**

Num = ['sepal\_length' , 'sepal\_width' , 'petal\_length' , 'petal\_width']

j = 0

while j < 5:

fig = plt.figure (figsize = [20 , 4])

plt.subplot(1, 2, 1)

sns.boxplot (x = Num[j] , data = df , color='skyblue')

sns.set(font\_scale=1.25)

j += 1

plt.subplot(1, 2, 2)

sns.boxplot (x = Num[j] , data = df , color='skyblue')

sns.set(font\_scale=1.25)

j += 1

if j == 4:

break

plt.show()

**# Vẽ biểu đồ countplot để xem số lượng từng giá trị đầu ra**

sns.countplot(x = df['species'] , data = df)

**Step 5: Select Model *(K Nearest Neighbors)***

**# Xác định đầu ra và đầu vào của mô hình**

X = pd.DataFrame(df , columns = ["sepal\_length" , "sepal\_width" , "petal\_length" , "petal\_width"])

y = df["species"].values.reshape(-1,1)

print(y)

**# Chia bộ dữ liệu thành bộ trainning và bộ test**

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y , test\_size=0.5 , random\_state = 0)

**# Viết hàm tính hiệu suất của mô hình**

def Evaluate\_Performance(Model, Xtrain, Xtest, Ytrain, Ytest) :

Model.fit(Xtrain,Ytrain)

overall\_score = cross\_val\_score(Model, Xtrain,Ytrain, cv=10)

model\_score = np.average(overall\_score)

Ypredicted = Model.predict(Xtest)

avg = 'weighted'

print("\n • Training Accuracy Score : ", round(Model.score(Xtrain, Ytrain) \* 100,2))

print(f" • Cross Validation Score : {round(model\_score \* 100,2)}")

print(f" • Testing Accuracy Score :{round(accuracy\_score(Ytest, Ypredicted) \* 100,2)}")

print(f" • Precision Score is : {np.round(precision\_score(Ytest, Ypredicted , average=avg) \* 100,2)}")

print(f" • Recall Score is : {np.round(recall\_score(Ytest, Ypredicted , average=avg) \* 100,2)}")

print(f" • F1-Score Score is : {np.round(f1\_score(Ytest, Ypredicted , average=avg) \* 100,2)}")

# ****K Nearest Neighbors****

**# Khởi tạo các danh sách để lưu trữ độ chính xác trên tập huấn luyện và tập kiểm thử**

training\_acc = [] # Danh sách lưu độ chính xác trên tập huấn luyện

test\_acc = [] # Danh sách lưu độ chính xác trên tập kiểm thử

**# Xác định khoảng số lượng láng giềng (neighbors) từ 1 đến 29**

neighbors\_setting = range(1, 30)

**# Duyệt qua các giá trị số láng giềng**

for n\_neighbors in neighbors\_setting:

**# Khởi tạo mô hình KNN với số láng giềng được xác định**

KNN = KNeighborsClassifier(n\_neighbors=n\_neighbors)

**# Huấn luyện mô hình trên tập huấn luyện**

KNN.fit(X\_train, y\_train.ravel())

**# Đo độ chính xác trên tập huấn luyện và lưu vào danh sách**

training\_acc.append(KNN.score(X\_train, y\_train))

**# Đo độ chính xác trên tập kiểm thử và lưu vào danh sách**

test\_acc.append(KNN.score(X\_test, y\_test))

**# Vẽ đồ thị biểu diễn độ chính xác trên tập huấn luyện và tập kiểm thử**

plt.plot(neighbors\_setting, training\_acc, label="Độ chính xác trên tập huấn luyện")

plt.plot(neighbors\_setting, test\_acc, label="Độ chính xác trên tập kiểm thử")

plt.xlabel("Số lượng láng giềng")

plt.ylabel("Độ chính xác")

plt.grid(linestyle='-')

plt.legend()

**# Thiết lập các tham số cần tối ưu hóa cho mô hình KNN**

parameters = {"n\_neighbors": range(1, 50)}

**# Khởi tạo đối tượng GridSearchCV để tìm kiếm siêu tham số tối ưu**

grid\_kn = GridSearchCV(estimator=KNN, param\_grid=parameters, scoring="accuracy", cv=5, verbose=1, n\_jobs=-1)

**# Tiến hành tìm kiếm siêu tham số trên tập huấn luyện**

grid\_kn.fit(X\_train, y\_train.ravel())

**# Trả về giá trị của siêu tham số tối ưu được chọn bởi GridSearchCV**

grid\_kn.best\_params\_

**# Xác định số lượng láng giềng K trong mô hình K-Nearest Neighbors**

K = 3

**# Khởi tạo mô hình KNN với số láng giềng được xác định**

KNN = KNeighborsClassifier(K)

**# Huấn luyện mô hình trên tập huấn luyện**

KNN.fit(X\_train, y\_train.ravel())

**# Dự đoán nhãn cho tập kiểm thử sử dụng mô hình đã huấn luyện**

y\_pred\_KNN = KNN.predict(X\_test)

**# In thông tin về mô hình và đánh giá hiệu suất**

print("K-Nearest Neighbors : ")

Evaluate\_Performance(KNN, X\_train, X\_test, y\_train, y\_test)

**# Sử dụng phương pháp Cross-Validation để đánh giá hiệu suất của mô hình K-Nearest Neighbors**

KNN\_r = cross\_val\_score(KNN, X, y, cv=10)

**# Tính độ lệch chuẩn (standard deviation) của các điểm đánh giá Cross-Validation**

K = np.std(KNN\_r)

**# In ra giá trị độ lệch chuẩn, giúp đánh giá sự biến động của hiệu suất trên các fold khác nhau**

print(K)

**# Tính ma trận nhầm lẫn (confusion matrix) giữa các nhãn thực tế và dự đoán của mô hình K-Nearest**

Neighbors

cm = confusion\_matrix(y, KNN.predict(X))

**# Hiển thị ma trận nhầm lẫn dưới dạng hình ảnh sử dụng đồ thị**

fig, ax = plt.subplots(figsize=(8, 8))

ax.imshow(cm)

ax.grid(False)

ax.set\_xlabel('Nhãn dự đoán', fontsize=14, color='black')

ax.set\_ylabel('Nhãn thực tế', fontsize=14, color='black')

ax.xaxis.set(ticks=range(3))

ax.yaxis.set(ticks=range(3))

ax.set\_ylim(2.5, -0.5)

**# Hiển thị số lượng mẫu của từng phần tử trong ma trận nhầm lẫn**

for i in range(3):

for j in range(3):

ax.text(j, i, cm[i, j], ha='center', va='center', color='red')

**# Hiển thị đồ thị**

plt.show()