#ML 1 ( FEATURE TRANSFORMATION)  
import pandas as pd  
from sklearn.model\_selection import train\_test\_split  
from sklearn.preprocessing import StandardScaler  
from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis  
from sklearn.metrics import accuracy\_score, confusion\_matrix  
from sklearn.datasets import load\_iris  
  
data = load\_iris()  
X = pd.DataFrame(data.data, columns=data.feature\_names)  
y = data.target  
  
scaler = StandardScaler()  
X = scaler.fit\_transform(X)  
  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  
  
lda = LinearDiscriminantAnalysis()  
X\_train\_lda = lda.fit\_transform(X\_train, y\_train)  
X\_test\_lda = lda.transform(X\_test)  
  
lda.fit(X\_train\_lda, y\_train)  
y\_pred = lda.predict(X\_test\_lda)  
  
accuracy = accuracy\_score(y\_test, y\_pred)  
confm = confusion\_matrix(y\_test, y\_pred)  
print("Accuracy:", accuracy)  
print("CM :",confm)

#ML 2 (REGRESION ANALYSIS)  
import pandas as pd  
from sklearn.model\_selection import train\_test\_split  
from sklearn.linear\_model import LinearRegression, LogisticRegression  
from sklearn.metrics import accuracy\_score, r2\_score  
import warnings warnings.filterwarnings("ignore")  
  
data = pd.read\_csv('path\_to\_diabetes.csv')  
  
data.describe()  
data.skew()  
data.kurt()  
data.mode()  
  
X = data.drop(columns='Outcome', axis = 1)  
y = data['Outcome']  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  
  
linear\_reg = LinearRegression()  
linear\_reg.fit(X\_train, y\_train)  
y\_pred\_linear = linear\_reg.predict(X\_test)  
r2\_linear = r2\_score(y\_test, y\_pred\_linear)  
print(f"Linear Regression R-squared: {r2\_linear}")  
  
logistic\_reg = LogisticRegression()  
logistic\_reg.fit(X\_train, y\_train)  
y\_pred\_logistic = logistic\_reg.predict(X\_test)  
accuracy = accuracy\_score(y\_test, y\_pred\_logistic)  
print(f"Logistic Regression Accuracy: {accuracy}")

# ML3 (CLASSIFIER ANALYSIS)  
import pandas as pd  
from sklearn.model\_selection import train\_test\_split  
from sklearn.preprocessing import StandardScaler  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn.metrics import confusion\_matrix, accuracy\_score, precision\_score, recall\_score  
  
X = data[['Age', 'EstimatedSalary']]  
y = data['Purchased']  
  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  
  
scaler = StandardScaler()  
X\_train = scaler.fit\_transform(X\_train)  
X\_test = scaler.transform(X\_test)  
  
knn = KNeighborsClassifier(n\_neighbors=5)  
knn.fit(X\_train, y\_train)  
y\_pred = knn.predict(X\_test)  
  
conf\_matrix = confusion\_matrix(y\_test, y\_pred)  
accuracy = accuracy\_score(y\_test, y\_pred)  
error\_rate = 1 - accuracy  
precision = precision\_score(y\_test, y\_pred)  
recall = recall\_score(y\_test, y\_pred)  
  
print("Confusion Matrix:\n", conf\_matrix)  
print(f"Accuracy: {accuracy}")  
print(f"Error Rate: {error\_rate}")  
print(f"Precision: {precision}")  
print(f"Recall: {recall}")

#MLP4 (CLUSTERING ANALYSIS)  
import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.cluster import KMeans  
  
data = pd.read\_csv("iris.csv")  
data.head()  
  
X = data.iloc[:, [1, 2, 3, 4]]  
  
inertia = []  
for i in range(1, 11):  
 kmeans=KMeans(n\_clusters=1, max\_iter=300, random\_state=42)  
 kmeans.fit(X)  
 inertia.append(kmeans.inertia\_)  
  
# Plot the Elbow Method graph  
plt.plot(range(1, 11), inertia, marker='0')  
plt.xlabel('Number of Clusters')  
plt.ylabel('Inertia (Within-cluster sum of squares)')  
plt.title('Elbow Method')  
plt.show()

#ML 5 (ENSEMBLE LEARNING)

import pandas as pd  
from sklearn.model\_selection import train\_test\_split  
from sklearn.ensemble import RandomForestClassifier  
import category\_encoders as ce  
from sklearn.metrics import accuracy\_score, confusion\_matrix  
  
data =pd.read\_csv("car\_evaluation.csv")  
data.head()  
  
col\_names = ['buying', 'maint', 'doors', 'persons', 'lug\_boot', 'safety', 'class']  
data.columns = col\_names  
data.head()  
  
X =data.drop(['class'], axis=1)  
y= data['class']  
X\_train, X\_test,y\_train,y\_test = train\_test\_split(X,y,test\_size=0.3, random\_state=42)  
X\_train.shape, X\_test.shape  
  
encoder = ce.OrdinalEncoder(cols=['buying', 'maint', 'doors', 'persons', 'lug\_boot', 'safety', 'class'])  
X\_train = encoder.fit\_transform(X\_train)  
X\_test = encoder.transform(X\_test)  
  
rfc = RandomForestClassifier(random\_state=0)  
rfc.fit(X\_train, y\_train)  
  
y\_pred = rfc.predict(X\_test)  
accuracy = accuracy\_score(y\_test, y\_pred)  
conf\_matrix = confusion\_matrix(y\_test, y\_pred)  
print("Accuracy:", accuracy, "\n")  
print("Confusion Matrix:\n", conf\_matrix)

#ML 6 (REINFORCEMENT LEARNING)

import numpy as np  
  
maze = np.array([  
 [0, 0, 0, 0, 0],  
 [0, 1, 0, 1, 0],  
 [0, 1, 0, 1, 0],  
 [0, 1, 1, 1, 0],  
 [0, 0, 0, 0, 2]  
])  
  
learning\_rate = 0.1  
discount\_factor = 0.9  
epsilon = 0.1  
num\_episodes = 1000  
  
num\_states, num\_actions = maze.size, 4  
Q = np.zeros((num\_states, num\_actions))  
  
for \_ in range(num\_episodes):  
 state = 0  
  
 while True:  
 action = np.random.choice(num\_actions) if np.random.uniform(0, 1) < epsilon else np.argmax(Q[state, :])  
 new\_state = state + [0, 1, 2, 3][action] # Up, Down, Left, Right  
 reward = [-1, 1, 0][maze.flat[new\_state]]  
 if reward: break  
 state = new\_state  
  
current\_state = 0  
while current\_state != 16: # Goal state  
 action = np.argmax(Q[current\_state, :])  
 current\_state = current\_state + (action + 1)  
 print("Agent moved to state:", current\_state)