################################ Q1 Start ########################################

Q1\_1:

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.impute import SimpleImputer

from category\_encoders import BinaryEncoder

df\_titanic=pd.read\_csv(r"C:\Users\Administrator.DAI-PC2\Desktop\jupiter\_demo\MACHINE LEARNING\06 Machine Learning\_Sanjay Sane\Repository\Titanic-Dataset.csv")

# eligible columns for Binary Encoding

eligible\_cols = ['Sex', 'Survived']

# Create a Binary Encoder

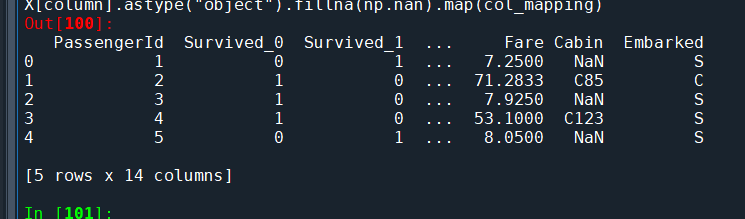
encoder = BinaryEncoder(cols=eligible\_cols)

# Fit and transform the eligible columns

titanic\_encoded = encoder.fit\_transform(df\_titanic)

# Print rows of the encoded dataset

titanic\_encoded.head()



Q1\_2:

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.impute import SimpleImputer

# titanic\_data

df\_titanic=pd.read\_csv(r"C:\Users\Administrator.DAI-PC2\Desktop\jupiter\_demo\MACHINE LEARNING\06 Machine Learning\_Sanjay Sane\Repository\Titanic-Dataset.csv")

df\_titanic['Sex\_Encoded'] = df\_titanic['Sex'].map({'male': 0, 'female': 1})

print(df\_titanic.isna().sum())

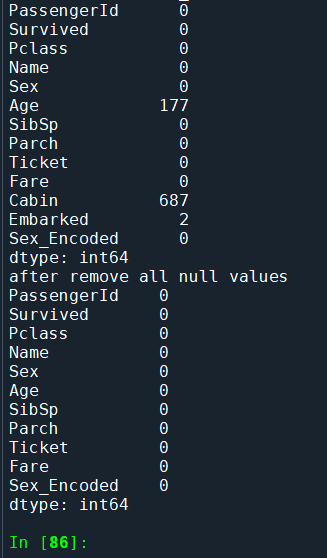
# calculate Median

print("after remove all null values ")

median\_age = df\_titanic['Age'].median()

df\_titanic['Age'] = df\_titanic['Age'].fillna(median\_age)

print(df\_titanic.isna().sum())



########################## Q1 end here ###############################

############################## Q2 Start ###########################################

Q2.

import pandas as pd

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

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix

from sklearn.impute import SimpleImputer

from sklearn.pipeline import Pipeline

# Load the Titanic dataset

df\_titanic = pd.read\_csv(r"C:\Users\Administrator.DAI-PC2\Desktop\jupiter\_demo\MACHINE LEARNING\06 Machine Learning\_Sanjay Sane\Repository\Titanic-Dataset.csv")

df\_titanic.head()

# Data Preparation

selected\_features = ['Pclass', 'Age', 'SibSp', 'Parch', 'Fare', 'Sex', 'Embarked', 'Survived']

df\_titanic = df\_titanic[selected\_features]

df\_titanic = pd.get\_dummies(df\_titanic, columns=['Sex', 'Embarked'], drop\_first=True)

imputer = SimpleImputer(strategy='median')

df\_titanic['Age'] = imputer.fit\_transform(df\_titanic[['Age']])

# Split data into features and target variable

X = df\_titanic.drop('Survived', axis=1)

y = df\_titanic['Survived']

# Split data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Train Random Forest Classifier with default parameters

rf\_model\_default = RandomForestClassifier(random\_state=42)

rf\_model\_default.fit(X\_train, y\_train)

y\_pred\_default = rf\_model\_default.predict(X\_test)

accuracy\_default = accuracy\_score(y\_test, y\_pred\_default)

print("Accuracy (Default Parameters):", accuracy\_default)

# Confusion Matrix

conf\_matrix\_default = confusion\_matrix(y\_test, y\_pred\_default)

print("\nConfusion Matrix (Default Parameters):")

print(conf\_matrix\_default)

# Tune Parameters using GridSearchCV

param\_grid = {

'n\_estimators': [100, 200, 300],

'max\_depth': [None, 10, 20],

'min\_samples\_split': [2, 5, 10],

'min\_samples\_leaf': [1, 2, 4]

}

rf\_model\_tuned = RandomForestClassifier(random\_state=42)

grid\_search = GridSearchCV(estimator=rf\_model\_tuned, param\_grid=param\_grid, cv=5, scoring='accuracy')

grid\_search.fit(X\_train, y\_train)

best\_params = grid\_search.best\_params\_

# predictions with the tuned model

rf\_model\_tuned = RandomForestClassifier(random\_state=42, \*\*best\_params)

rf\_model\_tuned.fit(X\_train, y\_train)

y\_pred\_tuned = rf\_model\_tuned.predict(X\_test)

# Model - Tuned Parameters

accuracy\_tuned = accuracy\_score(y\_test, y\_pred\_tuned)

print("\nAccuracy (Tuned Parameters):", accuracy\_tuned)

# Confusion Matrix

conf\_matrix\_tuned = confusion\_matrix(y\_test, y\_pred\_tuned)

print("\nConfusion Matrix (Tuned Parameters):")

print(conf\_matrix\_tuned)

# Save results to a single file

results\_df = pd.DataFrame({

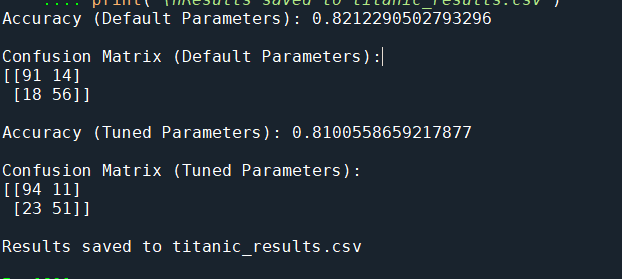
'Metric': ['Accuracy (Default)', 'Accuracy (Tuned)'],

'Value': [accuracy\_default, accuracy\_tuned]

})

results\_df.to\_csv('titanic\_results.csv', index=False)

print("\nResults saved to titanic\_results.csv")



############################ Q2 end here #######################################

################################ Q3 Start ########################################

Q3.

import seaborn as sns

from sklearn.preprocessing import StandardScaler

from sklearn.cluster import KMeans

from sklearn.metrics import silhouette\_score

from sklearn.datasets import load\_iris

import numpy as np

import pandas as pd

# Load the Iris dataset

df = load\_iris()

iris = df.data

# Scale the features

scaler = StandardScaler()

df\_scaled = scaler.fit\_transform(iris)

# Define the range of clusters to evaluate

Ks = [2, 3, 4, 5]

scores = []

# Iterate over different numbers of clusters

for k in Ks:

# Fit KMeans clustering model

clust = KMeans(n\_clusters=k, random\_state=24, init='random')

clust.fit(df\_scaled)

# Compute silhouette score

score = silhouette\_score(df\_scaled, clust.labels\_)

scores.append(score)

# Find the index of the maximum silhouette score

i\_max = np.argmax(scores)

# Print the best number of clusters and its corresponding silhouette score

print("Best number of clusters:", Ks[i\_max])

print("Best silhouette score:", scores[i\_max])

# Fit KMeans clustering model with the best number of clusters

clust = KMeans(n\_clusters=Ks[i\_max], random\_state=24)

clust.fit(df\_scaled)

# Create a DataFrame with cluster labels

clust\_data = pd.DataFrame(iris, columns=df.feature\_names)

clust\_data['Cluster'] = clust.labels\_

# Print the mean values of features for each cluster

print(clust\_data.groupby('Cluster').mean())

# Access centroids

centroids = clust.cluster\_centers\_

# Create a DataFrame with cluster labels and centroids

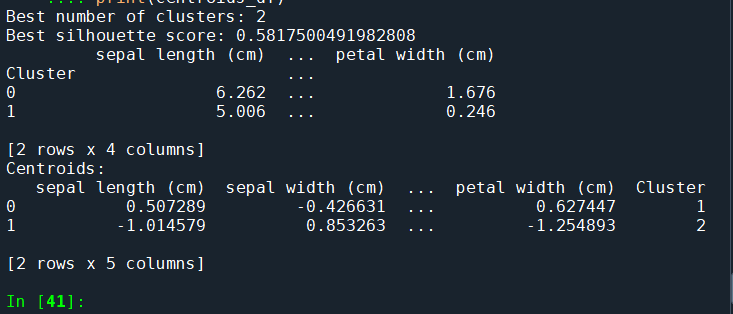
centroids\_df = pd.DataFrame(centroids, columns=df.feature\_names)

centroids\_df['Cluster'] = range(1, Ks[i\_max]+1)

# Print centroids

print("Centroids:")

print(centroids\_df)



################################ Q3 End ########################################