#Question1

import pandas as pd

import numpy as np

import seaborn as sns

# Load the Titanic dataset

df = sns.load\_dataset('titanic')

# Display the first few rows of the dataset

print(titanic.head())

# Handling missing values

df['age'].fillna(df['age'].median(), inplace=True)

df['embarked'].fillna(df['embarked'].mode()[0], inplace=True)

# Removing outliers in 'Fare'

q1, q3 = df['fare'].quantile([0.25, 0.75])

iqr = q3 - q1

lower\_bound = q1 - 1.5 \* iqr

upper\_bound = q3 + 1.5 \* iqr

df = df[(df['fare'] >= lower\_bound) & (df['fare'] <= upper\_bound)]

# Validation checks

print(df.isnull().sum())

print(df.describe())

#Question2

from sklearn.preprocessing import StandardScaler, MinMaxScaler, Binarizer

# Standardization

scaler = StandardScaler()

df['Fare\_scaled'] = scaler.fit\_transform(df[['fare']])

# Normalization

normalizer = MinMaxScaler()

df['Age\_normalized'] = normalizer.fit\_transform(df[['age']])

# Binarization (e.g., Fare > 30)

binarizer = Binarizer(threshold=30)

df['Fare\_binarized'] = binarizer.fit\_transform(df[['fare']])

#Question3

from sklearn.cluster import KMeans

import matplotlib.pyplot as plt

# Select features for clustering

X = df[['age', 'fare']].dropna()

# Perform K-Means with varying K

inertia = []

for k in range(1, 10):

    kmeans = KMeans(n\_clusters=k, random\_state=42)

    kmeans.fit(X)

    inertia.append(kmeans.inertia\_)

# Plot Elbow Curve

plt.plot(range(1, 10), inertia, marker='o')

plt.xlabel('Number of Clusters')

plt.ylabel('Inertia')

plt.title('Elbow Method for Optimal K')

plt.show()

#Question4

from sklearn.cluster import AgglomerativeClustering

from sklearn.metrics import silhouette\_score

# Partitioning (K-Means)

kmeans = KMeans(n\_clusters=3, random\_state=42).fit(X)

print("K-Means Silhouette Score:", silhouette\_score(X, kmeans.labels\_))

# Hierarchical Clustering

agglo = AgglomerativeClustering(n\_clusters=3).fit(X)

print("Hierarchical Clustering Silhouette Score:", silhouette\_score(X, agglo.labels\_))

#Question5

from sklearn.naive\_bayes import GaussianNB

from sklearn.neighbors import KNeighborsClassifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score

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

# Prepare the data

X = df[['pclass', 'age', 'fare', 'sibsp', 'parch']].fillna(0)

y = df['survived']

# Split data

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

# Naive Bayes

nb = GaussianNB()

nb.fit(X\_train, y\_train)

nb\_preds = nb.predict(X\_test)

print("Naive Bayes Accuracy:", accuracy\_score(y\_test, nb\_preds))

# K-Nearest Neighbors

knn = KNeighborsClassifier(n\_neighbors=5)

knn.fit(X\_train, y\_train)

knn\_preds = knn.predict(X\_test)

print("KNN Accuracy:", accuracy\_score(y\_test, knn\_preds))

# Decision Tree

dt = DecisionTreeClassifier(max\_depth=5, random\_state=42)

dt.fit(X\_train, y\_train)

dt\_preds = dt.predict(X\_test)

print("Decision Tree Accuracy:", accuracy\_score(y\_test, dt\_preds))

# Cross-Validation

scores = cross\_val\_score(dt, X, y, cv=10, scoring='accuracy')

print("Decision Tree Cross-Validation Accuracy:", scores.mean())

#Question6

from sklearn.ensemble import BaggingClassifier, AdaBoostClassifier

# Bagging

# Replace 'base\_estimator' with 'estimator'

bagging = BaggingClassifier(estimator=DecisionTreeClassifier(), n\_estimators=10, random\_state=42)

bagging.fit(X\_train, y\_train)

bagging\_preds = bagging.predict(X\_test)

print("Bagging Accuracy:", accuracy\_score(y\_test, bagging\_preds))

# Boosting

boosting = AdaBoostClassifier(estimator=DecisionTreeClassifier(max\_depth=3), n\_estimators=10, random\_state=42) # Replace 'base\_estimator' with 'estimator'

boosting.fit(X\_train, y\_train)

boosting\_preds = boosting.predict(X\_test)

print("Boosting Accuracy:", accuracy\_score(y\_test, boosting\_preds))