#min max

from sklearn.preprocessing import MinMaxScaler

import numpy as np

data = np.array([[1, 2, 3],[4, 5, 6],[7, 8, 9]])

scaler = MinMaxScaler()

# Fit scaler to data and transform it

scaled\_data = scaler.fit\_transform(data)

print("Original Data:")

print(data)

print("\nScaled Data:")

print(scaled\_data)

import numpy as np

def min\_max\_scaling(data):

min\_vals = np.min(data, axis=0)

max\_vals = np.max(data, axis=0)

scaled\_data = (data - min\_vals) / (max\_vals - min\_vals)

return scaled\_data

data = np.array([[1, 2, 3],[4, 5, 6],[7, 8, 9]])

scaled\_data = min\_max\_scaling(data)

print("Original Data:")

print(data)

print("\nScaled Data:")

print(scaled\_data)

======================================================

#zscore

import numpy as np

def z\_score\_normalization(data):

mean = np.mean(data, axis=0)

std\_dev = np.std(data, axis=0)

standardized\_data = (data - mean) / std\_dev

return standardized\_data

data = np.array([[1, 2, 3],[4, 5, 6],[7, 8, 9]])

standardized\_data = z\_score\_normalization(data)

print("Original Data:")

print(data)

print("\nStandardized Data:")

print(standardized\_data)

from sklearn.preprocessing import StandardScaler

data = np.array([[1, 2, 3],[4, 5, 6],[7, 8, 9]])

scaler = StandardScaler()

standardized\_data = scaler.fit\_transform(data)

print("Original Data:")

print(data)

print("\nStandardized Data:")

print(standardized\_data)

===============================

#find null values

import pandas as pd

data = pd.DataFrame({

'A': [1, 2, None, 4, 5],

'B': [None, 10, 20, None, 50]

})

mean\_imputed = data.fillna(data.mean())

median\_imputed = data.fillna(data.median())

mode\_imputed = data.fillna(data.mode().iloc[0])

print("Mean Imputation:")

print(mean\_imputed)

print("\nMedian Imputation:")

print(median\_imputed)

print("\nMode Imputation:")

print(mode\_imputed)

forward\_fill = data.ffill()

backward\_fill = data.bfill()

linear\_interpolation = data.interpolate()

print("Forward Fill:")

print(forward\_fill)

print("\nBackward Fill:")

print(backward\_fill)

print("\nLinear Interpolation:")

print(linear\_interpolation)

=================================

#Remove Records containing null values(Imputation)

import pandas as pd

data = pd.DataFrame({

'A': [1, 2, None, 4, 5],

'B': [None, 10, 20, None, 50]

})

cleaned\_data = data.dropna()

print("Original Data:")

print(data)

print("\nCleaned Data (Rows with Null Values Removed):")

print(cleaned\_data)

cleaned\_data\_all = data.dropna(how='all')

print("Cleaned Data (Rows with All Null Values Removed):")

print(cleaned\_data\_all)

=========================================

#Re-sampling

import pandas as pd

data = pd.DataFrame({

'feature': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],

'label': [0, 1, 0, 1, 0, 1, 0, 1, 0, 1]

})

majority\_class = data[data['label'] == 0]

minority\_class = data[data['label'] == 1]

from sklearn.utils import resample

upsampled\_minority = resample(minority\_class, replace=True, n\_samples=len(majority\_class))

upsampled\_data = pd.concat([majority\_class, upsampled\_minority])

print("Upsampled Data:")

print(upsampled\_data)

downsampled\_majority = resample(majority\_class, replace=False, n\_samples=len(minority\_class))

downsampled\_data = pd.concat([downsampled\_majority, minority\_class])

print("\nDownsampled Data:")

print(downsampled\_data)

===================================

#correlation

import pandas as pd

data = pd.DataFrame({

'A': [1, 2, 3, 4, 5],

'B': [2, 4, 6, 8, 10],

'C': [5, 4, 3, 2, 1]

})

pearson\_corr = data.corr(method='pearson')

print("Pearson Correlation:")

print(pearson\_corr)

spearman\_corr = data.corr(method='spearman')

print("\nSpearman Correlation:")

print(spearman\_corr)

kendall\_corr = data.corr(method='kendall')

print("\nKendall's Tau:")

print(kendall\_corr)

=================================

#Sillhoute score

from sklearn.datasets import make\_blobs

from sklearn.cluster import KMeans

from sklearn.metrics import silhouette\_score

X, \_ = make\_blobs(n\_samples=100, n\_features=2, centers=4, random\_state=42)

kmeans = KMeans(n\_clusters=4, random\_state=42)

labels = kmeans.fit\_predict(X)

silhouette\_avg = silhouette\_score(X, labels)

print("Silhouette Score:", silhouette\_avg)

===================================================

#Linear Regression

import numpy as np

from sklearn.linear\_model import LinearRegression

X = np.array([[1], [2], [3], [4], [5]]) # Independent variable

y = np.array([2, 3, 4, 5, 6])

model = LinearRegression()

model.fit(X, y)

print("Intercept:", model.intercept\_)

print("Coefficient:", model.coef\_[0])

new\_X = np.array([[6], [7], [8]]) # New data

predicted\_y = model.predict(new\_X)

print("Predicted values:", predicted\_y)

===================================================

#SVM

#i-Classification:

from sklearn import datasets

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

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score

iris = datasets.load\_iris()

X = iris.data

y = iris.target

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

svm\_classifier = SVC(kernel='linear')

svm\_classifier.fit(X\_train, y\_train)

y\_pred = svm\_classifier.predict(X\_test)

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

print("Accuracy:", accuracy)

#ii-Regression:

from sklearn import datasets

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

from sklearn.svm import SVR

from sklearn.metrics import mean\_squared\_error

diabetes = datasets.load\_diabetes()

X = diabetes.data

y = diabetes.target

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

svm\_regressor = SVR(kernel='linear')

svm\_regressor.fit(X\_train, y\_train)

y\_pred = svm\_regressor.predict(X\_test)

mse = mean\_squared\_error(y\_test, y\_pred)

print("Mean Squared Error:", mse)

===================================================

#decision tree

#i-Classification:

from sklearn import datasets

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

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score

iris = datasets.load\_iris()

X = iris.data

y = iris.target

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

dt\_classifier = DecisionTreeClassifier()

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

y\_pred = dt\_classifier.predict(X\_test)

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

print("Accuracy:", accuracy)

#ii-Regression:

from sklearn import datasets

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

from sklearn.tree import DecisionTreeRegressor

from sklearn.metrics import mean\_squared\_error

diabetes = datasets.load\_diabetes()

X = diabetes.data

y = diabetes.target

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

dt\_regressor = DecisionTreeRegressor()

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

y\_pred = dt\_regressor.predict(X\_test)

mse = mean\_squared\_error(y\_test, y\_pred)

print("Mean Squared Error:", mse)

=============================================

#kmeans

from sklearn.datasets import make\_blobs

from sklearn.cluster import KMeans

import matplotlib.pyplot as plt

X, \_ = make\_blobs(n\_samples=300, centers=4, cluster\_std=0.60, random\_state=0)

kmeans = KMeans(n\_clusters=4)

kmeans.fit(X)

labels = kmeans.predict(X)

centroids = kmeans.cluster\_centers\_

plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', alpha=0.5)

plt.scatter(centroids[:, 0], centroids[:, 1], marker='\*', s=300, c='r', label='Centroids')

plt.xlabel('Feature 1')

plt.ylabel('Feature 2')

plt.title('K-means Clustering')

plt.legend()

plt.show()

===========================================

#KNN

from sklearn.datasets import load\_iris

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

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy\_score

# Load the Iris dataset

iris = load\_iris()

X = iris.data

y = iris.target

# Split the 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 the KNeighborsClassifier model

knn\_classifier = KNeighborsClassifier(n\_neighbors=3)

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

# Make predictions on the testing set

y\_pred = knn\_classifier.predict(X\_test)

# Calculate accuracy

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

print("Accuracy:", accuracy)

from sklearn.datasets import fetch\_california\_housing

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

from sklearn.neighbors import KNeighborsRegressor

from sklearn.metrics import mean\_squared\_error

# Load the California housing dataset

housing = fetch\_california\_housing()

X = housing.data

y = housing.target

# Split the 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 the KNeighborsRegressor model

knn\_regressor = KNeighborsRegressor(n\_neighbors=3)

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

# Make predictions on the testing set

y\_pred = knn\_regressor.predict(X\_test)

# Calculate Mean Squared Error

mse = mean\_squared\_error(y\_test, y\_pred)

print("Mean Squared Error:", mse)

================================================

#Random forest

from sklearn.datasets import load\_iris

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

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score

# Load the Iris dataset

iris = load\_iris()

X = iris.data

y = iris.target

# Split the 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 the RandomForestClassifier model

rf\_classifier = RandomForestClassifier(n\_estimators=100, random\_state=42)

rf\_classifier.fit(X\_train, y\_train)

# Make predictions on the testing set

y\_pred = rf\_classifier.predict(X\_test)

# Calculate accuracy

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

print("Accuracy:", accuracy)

from sklearn.datasets import fetch\_california\_housing

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

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean\_squared\_error

# Load the California housing dataset

housing = fetch\_california\_housing()

X = housing.data

y = housing.target

# Split the 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 the RandomForestRegressor model

rf\_regressor = RandomForestRegressor(n\_estimators=100, random\_state=42)

rf\_regressor.fit(X\_train, y\_train)

# Make predictions on the testing set

y\_pred = rf\_regressor.predict(X\_test)

# Calculate Mean Squared Error

mse = mean\_squared\_error(y\_test, y\_pred)

print("Mean Squared Error:", mse)

==================================================

#voting

from sklearn.datasets import load\_iris

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

from sklearn.ensemble import RandomForestClassifier, VotingClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score

iris = load\_iris()

X = iris.data

y = iris.target

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

log\_reg = LogisticRegression(max\_iter=1000)

rnd\_forest = RandomForestClassifier(n\_estimators=100, random\_state=42)

svm = SVC(probability=True)

voting\_clf = VotingClassifier(estimators=[('lr', log\_reg), ('rf', rnd\_forest), ('svc', svm)], voting='soft')

voting\_clf.fit(X\_train, y\_train)

y\_pred = voting\_clf.predict(X\_test)

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

print("Voting Classifier Accuracy:", accuracy)

=================================================

#collaborative filtering

!pip install scikit-surprise

from surprise import Dataset, Reader, KNNBasic

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

from surprise.accuracy import rmse

# Load the dataset

data = Dataset.load\_builtin('ml-100k')

# Define the rating scale

reader = Reader(rating\_scale=(1, 5))

# Build the full trainset

dataset = data.build\_full\_trainset()

# Split the dataset into train and test sets

trainset, testset = train\_test\_split(data, test\_size=0.2)

# Initialize the algorithm

algo = KNNBasic(sim\_options={'user\_based': True})

# Fit the algorithm to the trainset

algo.fit(trainset)

# Make predictions on the testset

predictions = algo.test(testset)

# Calculate RMSE

accuracy = rmse(predictions)

print("RMSE:", accuracy)

===========================================

#Accuracy, Precision, Recall, F1 Score, MSE, RMSE, ROC Curve, Confusion Matrix

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, roc\_auc\_score, confusion\_matrix

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error, r2\_score

import numpy as np

# Classification Metrics

# Define true and predicted labels for classification

y\_true\_classification = [0, 1, 1, 0, 1] # Example true labels

y\_pred\_classification = [0, 1, 0, 0, 1] # Example predicted labels

# Calculate classification metrics

accuracy = accuracy\_score(y\_true\_classification, y\_pred\_classification)

precision = precision\_score(y\_true\_classification, y\_pred\_classification)

recall = recall\_score(y\_true\_classification, y\_pred\_classification)

f1 = f1\_score(y\_true\_classification, y\_pred\_classification)

roc\_auc = roc\_auc\_score(y\_true\_classification, y\_pred\_classification) # Assuming binary classification with predicted class labels

conf\_matrix = confusion\_matrix(y\_true\_classification, y\_pred\_classification)

# Print the classification metrics

print("Classification Metrics:")

print("Accuracy:", accuracy)

print("Precision:", precision)

print("Recall:", recall)

print("F1 Score:", f1)

print("ROC AUC Score:", roc\_auc)

print("Confusion Matrix:\n", conf\_matrix)

# Regression Metrics

# Define true and predicted values for regression

y\_true\_regression = [2.5, 3.7, 4.2, 5.0, 6.2] # Example true values

y\_pred\_regression = [2.0, 3.5, 4.0, 5.2, 6.0] # Example predicted values

# Calculate regression metrics

mse = mean\_squared\_error(y\_true\_regression, y\_pred\_regression)

rmse = np.sqrt(mse)

mae = mean\_absolute\_error(y\_true\_regression, y\_pred\_regression)

r2 = r2\_score(y\_true\_regression, y\_pred\_regression)

# Print the regression metrics

print("\nRegression Metrics:")

print("Mean Squared Error:", mse)

print("Root Mean Squared Error:", rmse)

print("Mean Absolute Error:", mae)

print("R-squared:", r2)

===============================================

#Scatter plot

import matplotlib.pyplot as plt

from sklearn.datasets import make\_blobs

from sklearn.cluster import KMeans

# Generate sample data for demonstration

X, y = make\_blobs(n\_samples=300, centers=4, random\_state=42)

# Scatter plot

plt.scatter(X[:, 0], X[:, 1], c=y, cmap='viridis')

plt.xlabel('Feature 1')

plt.ylabel('Feature 2')

plt.title('Scatter Plot')

plt.colorbar(label='Target')

plt.show()

# Assuming y\_true and y\_pred are defined elsewhere in your code

# If not, you need to define them with your actual true and predicted labels

# Confusion Matrix

from sklearn.metrics import confusion\_matrix

import seaborn as sns

# Example of confusion matrix calculation

y\_true = [0, 1, 1, 0, 1] # Example true labels

y\_pred = [0, 1, 0, 0, 1] # Example predicted labels

conf\_matrix = confusion\_matrix(y\_true, y\_pred)

# Plot confusion matrix

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues')

plt.xlabel('Predicted')

plt.ylabel('True')

plt.title('Confusion Matrix')

plt.show()

# Clusters

# Example clustering using KMeans

kmeans = KMeans(n\_clusters=4, random\_state=42)

labels = kmeans.fit\_predict(X)

# Plot clustered data

plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis')

plt.xlabel('Feature 1')

plt.ylabel('Feature 2')

plt.title('Clustered Data')

plt.colorbar(label='Cluster')

plt.show()

=============================================

#apriori

from mlxtend.frequent\_patterns import apriori

from mlxtend.frequent\_patterns import association\_rules

import pandas as pd

dataset = [['Milk', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt'],

['Dill', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt'],

['Milk', 'Apple', 'Kidney Beans', 'Eggs'],

['Milk', 'Unicorn', 'Corn', 'Kidney Beans', 'Yogurt'],

['Corn', 'Onion', 'Onion', 'Kidney Beans', 'Ice cream', 'Eggs']]

df = pd.DataFrame(dataset)

df\_encoded = pd.get\_dummies(df, prefix='', prefix\_sep='')

frequent\_itemsets = apriori(df\_encoded, min\_support=0.5, use\_colnames=True)

rules = association\_rules(frequent\_itemsets, metric='lift', min\_threshold=1)

print("Frequent Itemsets:")

print(frequent\_itemsets)

print("\nAssociation Rules:")

print(rules)

============================================

#all classification

from sklearn.datasets import load\_iris

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

from sklearn.naive\_bayes import GaussianNB

from sklearn.svm import SVC

from sklearn.tree import DecisionTreeClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score

iris = load\_iris()

X = iris.data

y = iris.target

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

nb\_classifier = GaussianNB()

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

nb\_pred = nb\_classifier.predict(X\_test)

nb\_accuracy = accuracy\_score(y\_test, nb\_pred)

print("Naive Bayes Classifier Accuracy:", nb\_accuracy)

svm\_classifier = SVC()

svm\_classifier.fit(X\_train, y\_train)

svm\_pred = svm\_classifier.predict(X\_test)

svm\_accuracy = accuracy\_score(y\_test, svm\_pred)

print("SVM Classifier Accuracy:", svm\_accuracy)

dt\_classifier = DecisionTreeClassifier()

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

dt\_pred = dt\_classifier.predict(X\_test)

dt\_accuracy = accuracy\_score(y\_test, dt\_pred)

print("Decision Tree Classifier Accuracy:", dt\_accuracy)

knn\_classifier = KNeighborsClassifier()

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

knn\_pred = knn\_classifier.predict(X\_test)

knn\_accuracy = accuracy\_score(y\_test, knn\_pred)

print("KNN Classifier Accuracy:", knn\_accuracy)

rf\_classifier = RandomForestClassifier()

rf\_classifier.fit(X\_train, y\_train)

rf\_pred = rf\_classifier.predict(X\_test)

rf\_accuracy = accuracy\_score(y\_test, rf\_pred)

print("Random Forest Classifier Accuracy:", rf\_accuracy)

==================================================

#preprocessing

import pandas as pd

import numpy as np

raw\_data = pd.read\_csv("/content/sample\_data/iris1.data.csv")

raw\_data['numerical\_column'].fillna(raw\_data['numerical\_column'].mean(), inplace=True)

raw\_data['categorical\_column'].fillna(raw\_data['categorical\_column'].mode()[0], inplace=True)

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

raw\_data[['numerical\_feature1', 'numerical\_feature2']] = scaler.fit\_transform(raw\_data[['numerical\_feature1', 'numerical\_feature2']])

encoded\_data = pd.get\_dummies(raw\_data, columns=['categorical\_feature'])

encoded\_data.drop(['irrelevant\_column1', 'irrelevant\_column2'], axis=1, inplace=True)

encoded\_data['year'] = pd.to\_datetime(encoded\_data['date\_column']).dt.year

from nltk.tokenize import word\_tokenize

from nltk.stem import PorterStemmer

stemmer = PorterStemmer()

def tokenize\_and\_stem(text):

tokens = word\_tokenize(text)

stemmed\_tokens = [stemmer.stem(token) for token in tokens]

return ' '.join(stemmed\_tokens)

encoded\_data['processed\_text\_column'] = encoded\_data['text\_column'].apply(tokenize\_and\_stem)

X = encoded\_data.drop('target\_variable', axis=1)

y = encoded\_data['target\_variable']

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

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

==================================