Svm

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix

# Load CSV file
data = pd.read_csv("your_dataset.csv")  # change file name

# Manually specify features (X) and target (y)
X = data[["feature1", "feature2", "feature3"]]  # replace with your feature columns
y = data["target_column"]                    # replace with your target column

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create SVM model (default kernel = 'rbf')
model = SVC()
model.fit(X_train, y_train)

# Predictions
y_pred = model.predict(X_test)

# Evaluate
print("✅ Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

LR

```
import pandas as pd
from sklearn.model_selection import train_test_split
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, confusion_matrix, roc_curve, auc
import matplotlib.pyplot as plt

# Load CSV file
data = pd.read_csv("your_dataset.csv")  # change file name

# Manually specify features (X) and target (y)
X = data[["feature1", "feature2", "feature3"]]  # replace with your feature columns
y = data["target_column"]                      # replace with your target column

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create model
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)

# Predictions
y_pred = model.predict(X_test)
y_prob = model.predict_proba(X_test)[:, 1]  # probability scores for ROC

# Evaluation
print("✅ Accuracy :", accuracy_score(y_test, y_pred))
print("✅ Precision:", precision_score(y_test, y_pred, average="binary"))
print("✅ Recall   :", recall_score(y_test, y_pred, average="binary"))
print("✅ Confusion Matrix:\n", confusion_matrix(y_test, y_pred))

# ROC Curve
fpr, tpr, _ = roc_curve(y_test, y_prob)
```

```
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(6, 6))
plt.plot(fpr, tpr, color="blue", label=f"ROC curve (AUC = {roc_auc:.2f})")
plt.plot([0, 1], [0, 1], color="red", linestyle="--")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve - Logistic Regression")
plt.legend(loc="lower right")
plt.show()
```

kmeans

```
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

iris = load_iris()
X = iris.data

kmeans = KMeans(n_clusters=3, random_state=42)
labels = kmeans.fit_predict(X)

plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], c='red', marker='X', s=200)
plt.title("K-Means Clustering (Iris)")
plt.show()
```

dbscan

```
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import DBSCAN
import matplotlib.pyplot as plt

# Load dataset
iris = load_iris()
X = iris.data

# Standardize the features (important for DBSCAN)
X_scaled = StandardScaler().fit_transform(X)

# Apply DBSCAN
db = DBSCAN(eps=0.6, min_samples=5)
labels = db.fit_predict(X_scaled)

# Plot the clusters (using first two features)
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=labels, cmap='viridis')
plt.title("DBSCAN Clustering on Iris Dataset")
plt.xlabel("Feature 1 (scaled)")
plt.ylabel("Feature 2 (scaled)")
plt.show()

# Print cluster labels
print("Cluster labels:", set(labels))
```

gaussian

```
from sklearn.datasets import load_iris
from sklearn.mixture import GaussianMixture
```

```
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

iris = load_iris()
X = StandardScaler().fit_transform(iris.data)

gmm = GaussianMixture(n_components=3, random_state=0)
labels = gmm.fit_predict(X)

plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis')
plt.title("Gaussian Mixture Model (Iris)")
plt.show()
```

adaboost

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Load dataset
iris = load_iris()
X, y = iris.data, iris.target

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Initialize AdaBoost with Decision Tree base learner
```

```
model = AdaBoostClassifier(DecisionTreeClassifier(max_depth=1), n_estimators=50,
random_state=42)

# Train model
model.fit(X_train, y_train)

# Predict and evaluate
y_pred = model.predict(X_test)
print("AdaBoost Accuracy:", accuracy_score(y_test, y_pred))
```

bagging

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Load dataset
iris = load_iris()
X, y = iris.data, iris.target

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Initialize Bagging with Decision Tree base learner
model = BaggingClassifier(DecisionTreeClassifier(), n_estimators=50, random_state=42)

# Train model
model.fit(X_train, y_train)
```

```
# Predict and evaluate
y_pred = model.predict(X_test)
print("Bagging Accuracy:", accuracy_score(y_test, y_pred))
```

text p

```
import re
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import nltk

# Download resources (run once)
nltk.download('punkt')
nltk.download('stopwords')

# Sample text
text = "Machine Learning is amazing!!! It helps in AI and Data Science 2025."

# Clean text
text = text.lower()                    # lowercase
text = re.sub(r'[^a-z\s]', '', text)      # remove punctuation & numbers

# Tokenize
words = word_tokenize(text)

# Remove stopwords
stop_words = set(stopwords.words('english'))
filtered = [w for w in words if w not in stop_words]

print("Original:", text)
print("Tokens:", words)
```

```python
print("After Stopword Removal:", filtered)
```

image p

```python
import cv2
import matplotlib.pyplot as plt

# Load the image
img = cv2.imread("image.jpg")  # <-- replace with your image file
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)  # Convert BGR to RGB

# Convert to grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Apply Gaussian Blur
blur = cv2.GaussianBlur(gray, (5, 5), 0)

# Edge detection using Canny
edges = cv2.Canny(blur, 100, 200)

# Display results
plt.figure(figsize=(10, 6))

plt.subplot(2, 2, 1)
plt.imshow(img_rgb)
plt.title("Original Image")
plt.axis("off")

plt.subplot(2, 2, 2)
plt.imshow(gray, cmap='gray')
plt.title("Grayscale Image")
plt.axis("off")
```

```
plt.subplot(2, 2, 3)
plt.imshow(blur, cmap='gray')
plt.title("Blurred Image")
plt.axis("off")

plt.subplot(2, 2, 4)
plt.imshow(edges, cmap='gray')
plt.title("Edge Detection (Canny)")
plt.axis("off")

plt.tight_layout()
plt.show()
```

audio

```
import librosa
import librosa.display
import matplotlib.pyplot as plt

# Load an audio file
y, sr = librosa.load("audio.wav")   # <-- replace with your file name

# Display basic info
print("Sampling rate:", sr)
print("Audio duration (seconds):", librosa.get_duration(y=y, sr=sr))

# Plot the waveform
plt.figure(figsize=(10, 4))
librosa.display.waveshow(y, sr=sr)
plt.title("Audio Waveform")
```

```
plt.xlabel("Time (s)")
plt.ylabel("Amplitude")
plt.show()

# Compute MFCC (Mel-Frequency Cepstral Coefficients)
mfcc = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=13)
plt.figure(figsize=(8, 4))
librosa.display.specshow(mfcc, x_axis='time', sr=sr)
plt.colorbar()
plt.title("MFCC Features")
plt.tight_layout()
plt.show()
```

hash f

```
# Basic Hash Table using Python dictionary
class HashTable:
    def _init_(self):
        self.table = {}

    def insert(self, key, value):
        self.table[key] = value

    def get(self, key):
        return self.table.get(key, None)

    def remove(self, key):
        if key in self.table:
            del self.table[key]

# Example
```

```
h = HashTable()
h.insert("A", 10)
h.insert("B", 20)
print("A ->", h.get("A"))
h.remove("A")
print("After removal:", h.get("A"))
```

cuckkooo

```
class CuckooHashing:
    def _init_(self, size=11):
        self.size = size
        self.table1 = [None] * size
        self.table2 = [None] * size

    def _hash1(self, key):
        return hash(key) % self.size

    def _hash2(self, key):
        return (hash(key) // self.size) % self.size

    def insert(self, key, value):
        pos1 = self._hash1(key)
        if self.table1[pos1] is None:
            self.table1[pos1] = (key, value)
            return
        key, value, self.table1[pos1] = self.table1[pos1][0], self.table1[pos1][1], (key, value)
        pos2 = self._hash2(key)
        if self.table2[pos2] is None:
            self.table2[pos2] = (key, value)
        else:
```

```
            print(f"Rehash needed for key {key}")

    def search(self, key):
        pos1, pos2 = self._hash1(key), self._hash2(key)
        if self.table1[pos1] and self.table1[pos1][0] == key:
            return self.table1[pos1][1]
        if self.table2[pos2] and self.table2[pos2][0] == key:
            return self.table2[pos2][1]
        return None

# Example
cuckoo = CuckooHashing()
cuckoo.insert("A", 100)
cuckoo.insert("B", 200)
print("Search A ->", cuckoo.search("A"))
print("Table1:", cuckoo.table1)
print("Table2:", cuckoo.table2)
```