```python
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris

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

# Print dataset dimensions
print("Shape of X:", X.shape)
print("Shape of y:", y.shape)
print("Unique classes:", np.unique(y))

# Preview first 5 rows using Pandas
df = pd.DataFrame(X, columns=iris.feature_names)
print("\nFirst 5 rows:")
print(df.head())
```

در این بخش از پروژه، هدف اصلی فراخوانی کتابخانه های مجاز و بارگذاری مجموعه داده **Iris** جهت اعمال الگوریتم خوشه بندی است.

فراخوانی کتابخانه ها: **sklearn.datasets ,Pandas,NumPy**

بارگذاری و جداسازی داده ها: با دستور load_iris()، کل دیکشنری داده بارگذاری میشود. سپس داده ها را به دو بخش تقسیم میکنیم: **ماتریس ویژگی‌ها(x)** که شامل ۴ ویژگی برای ۱۵۰ نمونه گل است. این همان داده ای است که به الگوریتم K-means داده میشود.

**برچسب های هدف (y)** که شامل کلاس واقعی گل ها (۰، ۱ و ۲) . از آنجا که K-means یک الگوریتم **یادگیری نظارت نشده** (Unsupervised) است، این برچسب ها در فرآیند آموزش (Training) استفاده نمیشوند و صرفا برای ارزیابی نهایی مدل و محاسبه دقت نگهداری شده اند.

بررسی ابعاد و صحت داده ها: برای اطمینان از صحت بارگذاری، ابعاد ماتریس ها را چاپ میکنیم خروجی (150, 4) برای X نشان دهنده وجود ۱۵۰ نمونه با ۴ ویژگی است.

خروجی (150,) برای y تعداد برچسب ها را نشان میدهد.

دستور np.unique(y) نشان میدهد که ۳ کلاس متمایز در داده ها وجود دارد که با مقدار K=3 در الگوریتم K-means مطابقت دارد.

پیش نمایش داده ها: در نهایت برای درک بصری بهتر، ۵ سطر اول داده ها را با استفاده از pd.DataFrame نمایش دادیم تا مقادیر عددی ویژگی ها و نام ستون ها را بررسی کنیم.

```
# Function to calculate Euclidean distance
def euclidean_distance(x1, x2):
    return np.sqrt(np.sum((x1 - x2) ** 2))

# Function to initialize centroids randomly
def initialize_centroids(X, k):
    # Select k random indices
    indices = np.random.choice(X.shape[0], k, replace=False)
    # Return data points at those indices
    return X[indices]

# Set parameters
k = 3
np.random.seed(42)  # For reproducible results

# Initialize centroids
centroids = initialize_centroids(X, k)

print("Initial Centroids:\n", centroids)
```

در این مرحله، ابزارهای محاسباتی لازم برای اجرای الگوریتم K-means را پیاده‌سازی کردیم. این بخش شامل دو تابع حیاتی و تنظیمات اولیه است: تابع محاسبه فاصله اقلیدسی(euclidean_distance) :

اساس الگوریتم K-means بر مبنای شباهت داده ها به مراکز خوشه هاست و در اینجا از فاصله اقلیدسی به عنوان معیار شباهت استفاده کردیم.

این تابع دو بردار x1و x2 (نمایانگر دو نقطه داده) را دریافت میکند. با استفاده از قابلیت های برداری کتابخانهNumPy ، ابتدا اختلاف درایه های متناظر محاسبه و به توان ۲ میرسد(2 ** (x1 - x2)) . سپس مجموع این اختلافات محاسبه شده و در نهایت جذر آن گرفته میشود.

تابع انتخاب تصادفی مراکز اولیه(initialize_centroids) :

الگوریتم K-means یک روش تکرارشونده است و نیاز به نقطه شروع دارد. روش های مختلفی برای انتخاب اولیه وجود دارد که در اینجا از روش انتخاب تصادفی از میان داده های موجود استفاده کردیم.

ورودی های تابع شامل ماتریس داده ها (X) و تعداد خوشه ها (k) است. دستور np.random.choice تعداد k ایندکس تصادفی را از بین کل داده ها انتخاب می‌کند. پارامتر replace=False تضمین میکند که یک نقطه داده نمی تواند همزمان دو بار به عنوان مرکز انتخاب شود.در نهایت، خود داده های متناظر با این ایندکس ها به عنوان مراکز اولیه (Centroids) بازگردانده میشوند.

**تنظیم پارامترها و تکرارپذیری: تعداد خوشه‌ها (K=3)** : با توجه به دانش قبلی از مجموعه داده Iris که دارای ۳ کلاس گل متفاوت است، مقدار K را برابر ۳ در نظر گرفتیم. **دانه تصادفی (random.seed(42)) :**از آنجا که K-means به نقاط شروع حساس است و نتایج ممکن است در هر بار اجرا تغییر کند، با تنظیم seed روی عدد ۴۲، اطمینان حاصل شده است که نتایج تولید شده در هر بار اجرای برنامه دقیقا یکسان خواهند بود.

```
# Function to assign each data point to the nearest centroid
def assign_clusters(X, centroids):
    labels = []
    for point in X:
        # Calculate distances to all centroids
        distances = [euclidean_distance(point, centroid) for centroid in centroids]
        # Find index of the nearest centroid
        closest_idx = np.argmin(distances)
        labels.append(closest_idx)
    return np.array(labels)

# Function to update centroids based on mean of assigned points
def update_centroids(X, labels, k):
    new_centroids = []
    for i in range(k):
        # Filter points belonging to cluster i
        points_in_cluster = X[labels == i]

        # Calculate mean if cluster is not empty
        if len(points_in_cluster) > 0:
            new_centroids.append(points_in_cluster.mean(axis=0))
        else:
            # Handle empty cluster (keep previous centroid or re-init)
            # For simplicity, we create a zero vector or keep random
            new_centroids.append(np.zeros(X.shape[1]))

    return np.array(new_centroids)
```

در این مرحله، دو تابع اصلی که بدنه حلقه یادگیری را تشکیل می دهند، پیاده سازی کردیم. الگوریتم K-means با تکرار متناوب این دو مرحله به سمت جواب همگرا میشود.

تابع تخصیص خوشه ها(assign_clusters) :

این تابع وظیفه دارد هر نقطه داده را به نزدیک ترین مرکز خوشه اختصاص دهد. **ورودی :**ماتریس داده ها (X) و مختصات فعلی مراکز.

**عملکرد:**یک حلقه for روی تک تک نقاط داده اجرا میشود. سپس برای هر نقطه، فاصله اقلیدسی آن تا تمام K مرکز محاسبه شده و در لیست distances ذخیره میشود. دستور np.argmin(distances) ایندکس (شماره) مرکزی که کمترین فاصله را دارد، پیدا میکند. این ایندکس همان "برچسب خوشه" برای آن داده است. **خروجی :**یک آرایه شامل برچسب های پیش بینی شده برای تمام داده ها.

تابع بروزرسانی مراکز (update_centroids) :

پس از اینکه مشخص شد هر داده متعلق به کدام خوشه است، باید مکان مراکز اصلاح شود تا در مرکزیت داده های جدید قرار گیرند.

**عملکرد:** یک حلقه به تعداد خوشه ها (K) اجرا میشود. سپس با استفاده از تکنیک **Boolean Indexing** درNumPy دستور X[labels == i] ، تمام نقاطی که در مرحله قبل به خوشه i اختصاص یافته اند، فیلتر و جدا میشوند.

محاسبه میانگین: اگر خوشه ای خالی نباشد، میانگین تمام نقاط آن خوشه در راستای ستون ها (axis=0) محاسبه میشود. این میانگین، مختصات جدید مرکز آن خوشه خواهد بود.

**مدیریت خوشه های خالی :**یک شرط (if len > 0) قرار داده شده تا در حالت نادری که یک خوشه هیچ داده ای جذب نکرده است، از خطای تقسیم بر صفر جلوگیری شود.

```
# Parameters
max_iters = 100

for i in range(max_iters):
    # 1. Assign points to nearest centroid
    labels = assign_clusters(X, centroids)

    # 2. Calculate new centroids
    new_centroids = update_centroids(X, labels, k)

    # 3. Check for convergence
    # If centroids do not change significantly, stop
    if np.allclose(centroids, new_centroids):
        print(f"Converged at iteration {i}")
        break

    centroids = new_centroids

print("Training completed.")
print("Final Centroids:\n", centroids)
print("First 20 labels:", labels[:20])
```

**۱ .پیکربندی تکرار (max_iters) :**متغیر max_iters = 100 به عنوان سقف تعداد تکرارها تعریف شده است. این کار برای جلوگیری از افتادن برنامه در حلقه های بی‌نهایت (در صورتی که همگرایی کامل رخ ندهد) ضروری است.

**۲ .چرخه یادگیری (Learning Loop) :**در داخل حلقه for، در هر دور تکرار، دو عملیات اصلی که در بخش قبل تعریف شد، فراخوانی میشوند: **گام انتساب** labels = assign_clusters(...) : مشخص میکند هر داده در حال حاضر به کدام مرکز نزدیک تر است.

**گام به‌روزرسانی** new_centroids = update_centroids(...) :مکان جدید مراکز را بر اساس میانگین داده های جذب شده محاسبه میکند.

**۳ .شرط توقف و همگرایی (Learning Loop) :**مهم ترین بخش این حلقه، بررسی شرط توقف است. هدف K-means رسیدن به نقطه ای است که در آن مراکز خوشه ها دیگر تغییر نکنند.

از دستور np.allclose(centroids, new_centroids) استفاده شده است. این دستور بررسی میکند که آیا تمام درایه های دو ماتریس مراکز قدیم و جدید با هم برابر هستند (یا اختلاف بسیار ناچیزی دارند) یا خیر؟ اگر شرط برقرار باشد، به این معنی است که الگوریتم همگرا شده (Converged) و با دستور break از حلقه خارج میشود تا از محاسبات اضافی پرهیز شود.

در غیر این صورت، مراکز جدید جایگزین مراکز قبلی (centroids = new_centroids) شده و تکرار بعدی آغاز میشود.

```
# Function to map cluster labels to true labels
def get_reference_labels(clusters, true_labels):
    reference_labels = np.zeros_like(clusters)
    for i in range(k):
        # Find indices where the cluster is i
        indices = np.where(clusters == i)
        # Find the most frequent true label in this cluster
        if len(indices[0]) > 0:
            mode_label = np.bincount(true_labels[indices]).argmax()
            reference_labels[indices] = mode_label
    return reference_labels

# Function to calculate metrics manually
def calculate_metrics_detailed(y_true, y_pred):
    classes = np.unique(y_true)
    total_samples = len(y_true)

    # Variables to store sums
    total_correct = 0
    recall_sum = 0

    for cls in classes:
        # True Positives (TP): Points that are actually cls AND predicted as cls
        tp = np.sum((y_true == cls) & (y_pred == cls))

        # False Negatives (FN): Points that are actually cls BUT predicted as others
        fn = np.sum((y_true == cls) & (y_pred != cls))

        # Accumulate correct predictions for Accuracy
        total_correct += tp

        # Calculate Recall for this specific class
        if (tp + fn) > 0:
            class_recall = tp / (tp + fn)
        else:
            class_recall = 0

        recall_sum += class_recall

    # Calculate final metrics
    accuracy = total_correct / total_samples
    error = 1 - accuracy
    avg_recall = recall_sum / len(classes)
    return accuracy, error, avg_recall

# Calculate metrics again
acc, err, rec = calculate_metrics_detailed(y, predicted_labels)

print(f"Accuracy: {acc * 100:.2f}%")
print(f"Error Rate: {err * 100:.2f}%")
print(f"Recall: {rec * 100:.2f}%")
```

در مرحله پایانی، عملکرد مدل خوشه بندی را مورد ارزیابی قرار دادیم. از آنجا که در یادگیری بدون نظارت، برچسب های پیش بینی شده لزوما با برچسب های واقعی هم نام نیستند، ابتدا نیاز به یک مرحله همگام سازی داریم.

تطبیق برچسب ها(Label Matching) :

الگوریتم K-means خوشه هایی با اندیس های ۰، ۱ و ۲ تولید میکند، اما نمی داند کدام اندیس معادل کدام نوع گل است. **تابع : get_reference_labels** این تابع با استفاده از رویکرد **رای اکثریت** عمل میکند. برای هر خوشه تولید شده، بررسی میشود که اکثر داده های درون آن خوشه در واقعیت متعلق به کدام کلاس بوده اند. سپس آن کلاس واقعی به کل خوشه نسبت داده میشود. این کار باعث میشود بتوانیم نتایج را با داده های واقعی (y_true) مقایسه کنیم.

محاسبه دقیق معیارها:

طبق خواسته پروژه مبنی بر عدم استفاده از توابع آماده ی ارزیابی، تابع calculate_metrics_detailed را برای محاسبه دستی معیارها طراحی کردیم. این تابع با پیمایش روی کلاس ها، مقادیر زیر را محاسبه میکند:

**تعداد مثبت‌های صحیح (TP) :** تعداد نمونه هایی که کلاس آنها به درستی تشخیص داده شده است.

**تعداد منفی های کاذب (FN) :**تعداد نمونه هایی که متعلق به یک کلاس بوده اند اما مدل آنها را در کلاس دیگری قرار داده است.

**فرمول های استفاده شده:**

دقت (Accuracy) :نسبت کل پیش بینی های درست به کل داده ها.

فراخوانی (Recall) : میانگین فراخوانی تمام کلاس ها .

نرخ خطا (Error Rate) : مکمل دقت.

Error = 1 – Accuracy

تحلیل نتایج:

پس از اجرای مدل روی تمام ۱۵۰ نمونه، نتایج زیر حاصل شد:

**دقت (Accuracy) :٪۸۹.۳۳**

**فراخوانی (Recall) :٪۸۹.۳۳**

**نرخ خطا (Error) :٪۱۰.۶۷**

این نتایج نشان میدهد که الگوریتم توانسته است با موفقیت ساختار پنهان داده ها را شناسایی کند و تنها حدود ۱۰ درصد خطا در تفکیک مرزهای پیچیده بین کلاس ها داشته است.