به نام خدا

سید پوریا احمدی 9723002

بخش یک:

برای حل این بخش از این لینک کمک گرفته شده.

(https://www.youtube.com/watch?v=vtuH4VRq1AU&list=PLqnslRFeH2Upcrywf-u2etjdxxkL8nl7E&index=12)

ابتدا کتابخانه های لازم را import میکنیم.

کتابخانه matplotlib برای رسم و pandas برای خواندن دیتا

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
```

سپس یک کلاس به نام kmeans درست میکنیم که کل الگوریتم را در آن پیاده سازی میکنیم.

ابتدا توضیحات توابع داده می شود و در نهایت عکس این کلاس نیز ضمیمه می شود.

ابتدا در init

default برخی از متغیر هارا مشخص میکنیم که مشخص است.

مثلا تعداد خوشه ها 5 و حداکثر تعداد iteration هم 50 باشد و همچنین مرحله به مرحله خوشه هارا نمایش ندهد.

سپس خوشه ها و centroid های خودمان را می سازیم که در ابتدا خالی هستند.

```
def __init__(self, K=5, max_iters=50, plot_steps=False):
    self.K = K
    self.max_iters = max_iters
    self.plot_steps = plot_steps
    self.clusters = [[] for _ in range(self.K)]
    self.centroids = []
```

حال به تابع predict می رسیم.

ابتدا centroid هارا مقدار دهی اولیه میکنیم و همانطور که میدانیم باید رندوم باشد

سپس در یک حلقه به تعداد max iteration کلاستر ها را آپدیت میکنیم (با تابع create cluster)

در مرحله بعد از خوشه های جدید centroid های آنهارا حساب میکنیم. (با تابع get centroid)

سپس باید چک کنیم که converged شده است یا خیر. (با تابع is converged)

در ایم مرحله طبق خواسته سوال چون حداقل باید 15 مرتبه تکرار شود تا به 15 نرسیده است این حلقه جلو میرود و بعد از آن با چک کردن همگرایی آنرا قطع میکند ولی برای آنکه مشخص شود در چه شماره ای همگرا شده اند یک شرط گذاشته شده که مشخص میکند در کدام iter همگرا شده است و آنرا در انتها پرینت میکند.

همچنین یک شرط هست که مشخص کند در هر مرحله خوشه ها نشان داده شود یا نه که در واقع تابع plotرا فراخوانی میکند.

```
def predict(self, X):
    self.X = X
    self.centroids = [self.X[index] for index in np.random.choice(X.shape[0], self.K, replace=False)]
    flag_converged = False
    for i in range(self.max_iters):
        self.clusters = self.create_clusters(self.centroids)
        centroids_old = self.centroids
        self.centroids = self.get_centroids(self.clusters)
        if self.is_converged(centroids_old, self.centroids) and not flag_converged:
            iterr  = i
            flag_converged = True
        if self.is_converged(centroids_old, self.centroids) and i >=14:
            print("converged with {} iterations".format(iterr))
            break
        if self.plot_steps:
            self.plot()
    return self.get_cluster_labels(self.clusters)
```

حال تابع get cluster label :

در این تابع برای همه داده ها در یک حلقه کلاستر و خوشه آنرا بر میگرداند. (که به آن در سوال ما نیازی نیست اما اگر فقط بخواهیم label هر داده را بدانیم از این تابع می توان استفاده کرد)

```
def get_cluster_labels(self, clusters):
    labels = np.empty(X.shape[0])
    for cluster_idx, cluster in enumerate(clusters):
        for sample_index in cluster:
            labels[sample_index] = cluster_idx
    return labels
```

تابع create clusters :

در این تابع هر نمونه ای که داریم داریم با استفاده از closest centroid نزدیک ترین centroid به آن را پیدا میکنیم و آن داده را در لیست آن خوشه قرار میدهیم.

```
def create_clusters(self, centroids):
    clusters = [[] for _ in range(self.K)]
    for index, sample in enumerate(self.X):
        clusters[self.closest_centroid(sample, centroids)].append(index)
    return clusters

def closest_centroid(self, sample, centroids):
```

حال تابع closest centroid :

در این تابع ابتدا مراکز به تابع داده می شود و سپس در لیستی به نام distances فاصله اقلیدسی هر داده با تمام centroid ها حساب شده و کمترین فاصله را برمیگرداند.

```
def closest_centroid(self, sample, centroids):
    return np.argmin([euclidean_distance(sample, point) for point in centroids])
```

تابع get centroids :

در این تابع ابتدا centroid ها را در ابعاد k و تعداد feature های x یعنی x.shape[1] صفر قرار می دهیم تا مراکز خوشه هارا حساب کنیم.

حال به ازای هر کلاستر در یک حلقه میانگین میانگین تمام داده های در آن کلاستر را حساب کرده و centroid آن کلاستر را برابر میانگین بدست آمده میکنیم.

```
def get_centroids(self, clusters):
    centroids = np.zeros((self.K, X.shape[1]))
    for cluster_idx, cluster in enumerate(clusters):
        centroids[cluster_idx] = np.mean(self.X[cluster], axis=0)
    return centroids
```

تابع is converged :

در این تابع اگر به ازای تمام خوشه ها اگر اختلاف مراکز بدست آمده(با فاصله اقلیدسی) در مرحله کنونی با مرحله قبلی برابر صفر شود یعنی همگرا شده ایم و true برمیگرداند.

```
def is_converged(self, centroids_old, centroids):
    return sum(euclidean_distance(centroids_old[i], centroids[i]) for i in range(self.K)) == 0
```

تابع plot :

در این تابع نیز برای هر کلاستر تمام نقاط آنرا با یک رنگ خاص پلات میکند و سپس در یک حلقه دیگر centroidهارا با x مشخص میکند

```
def plot(self):
    fig, ax = plt.subplots(figsize=(12, 8))
    for i, index in enumerate(self.clusters):
        point = self.X[index].T
        ax.scatter(*point)
    for point in self.centroids:
        ax.scatter(*point, marker="x", color="black", linewidth=2)
    plt.show()
```

تابع فاصله اقلیدسی واضح است :

```
def euclidean_distance(x1, x2):
    return np.sqrt(np.sum((x1 - x2) ** 2))
```

عکس کل کلاس:

```
class KMeans:
    def __init__(self, K=5, max_iters=50, plot_steps=False):
        self.K = K
        self.max_iters = max_iters
        self.plot_steps = plot_steps
        self.clusters = [[] for _ in range(self.K)]
        self.centroids = []

    def predict(self, X):
        self.X = X
        self.centroids = [self.X[index] for index in np.random.choice(X.shape[0], self.K, replace=False)]
        flag_converged = False
        for i in range(self.max_iters):
            self.clusters = self.create_clusters(self.centroids)
            centroids_old = self.centroids
            self.centroids = self.get_centroids(self.clusters)
            if self.is_converged(centroids_old, self.centroids) and not flag_converged:
                iterr  = i
                flag_converged = True
            if self.is_converged(centroids_old, self.centroids) and i >=14:
                print("converged with {} iterations".format(iterr))
                break
            if self.plot_steps:
                self.plot()
        return self.get_cluster_labels(self.clusters)

    def get_cluster_labels(self, clusters):
        labels = np.empty(X.shape[0])
        for cluster_idx, cluster in enumerate(clusters):
            for sample_index in cluster:
                labels[sample_index] = cluster_idx
        return labels

    def create_clusters(self, centroids):
        clusters = [[] for _ in range(self.K)]
        for index, sample in enumerate(self.X):
            clusters[self.closest_centroid(sample, centroids)].append(index)
        return clusters

    def closest_centroid(self, sample, centroids):
        return np.argmin([euclidean_distance(sample, point) for point in centroids])

    def get_centroids(self, clusters):
        centroids = np.zeros((self.K, X.shape[1]))
        for cluster_idx, cluster in enumerate(clusters):
            centroids[cluster_idx] = np.mean(self.X[cluster], axis=0)
        return centroids

    def is_converged(self, centroids_old, centroids):
        return sum(euclidean_distance(centroids_old[i], centroids[i]) for i in range(self.K)) == 0

    def plot(self):
        fig, ax = plt.subplots(figsize=(12, 8))
        for i, index in enumerate(self.clusters):
            point = self.X[index].T
            ax.scatter(*point)
        for point in self.centroids:
            ax.scatter(*point, marker="x", color="black", linewidth=2)
        plt.show()
```

حال در تابع main :

ابتدا dataset را میخوانیم و محتوای آنرا تبدیل به numpy array میکنیم.

سپس از کلاس kmeans یک شی می سازیم به نام k که به آن داده های خود را میدهیم(k.predict) و الگوریتم روی آن انجام می شود.

در ساختن کلاس می توان همانطور که گفته شد k و max iter و اینکه در هر مرحله کلاستر ها پلات شوند یا نه را مشخص کرد

همچنین در انتها یکبار دیگر شکل نهایی بدست آمده را پلات میکنیم.

```
if __name__ == "__main__":
    df = pd.read_csv("Dataset1.csv")
    X = df.to_numpy()
    k = KMeans(K=2, max_iters=100, plot_steps=True)
    k.plot()
```

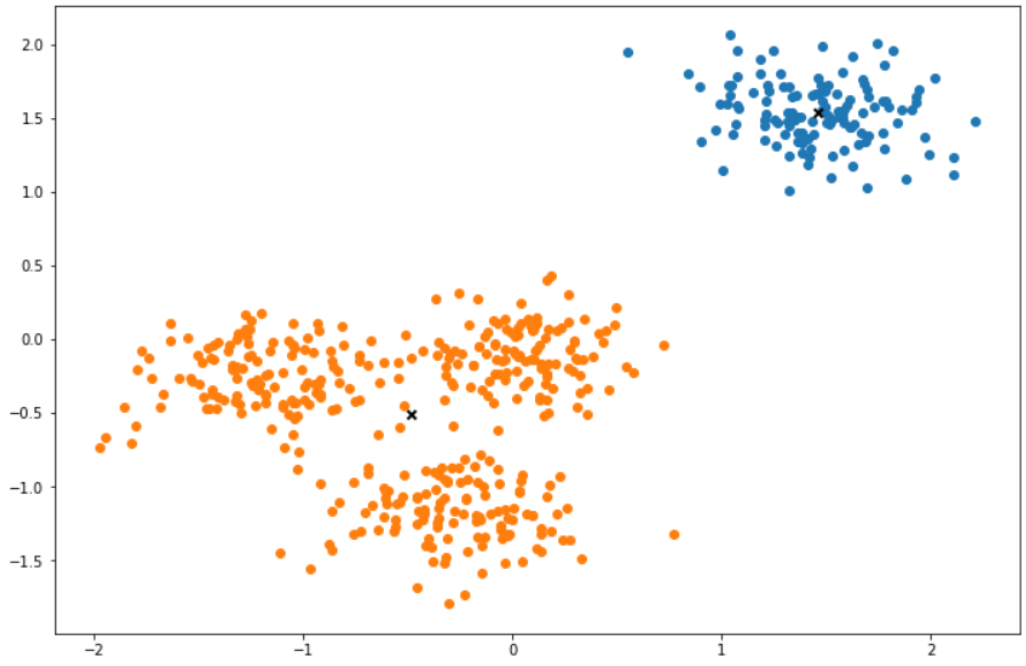
حال موارد خواسته شده سوال را انجام می دهیم:

دقت شود فقط شکل نهایی در گزارش نشان داده می شود به همراه شماره iter که در آن همگرا شده ایم

بقیه موارد مثل خوشه ها در ها تکرار حلقه در کد قابل مشاهده است. (در هر مرحله centroid ها با xمشخص شده و تغییرات آنها نیز قابل مشاهده خواهد بود)

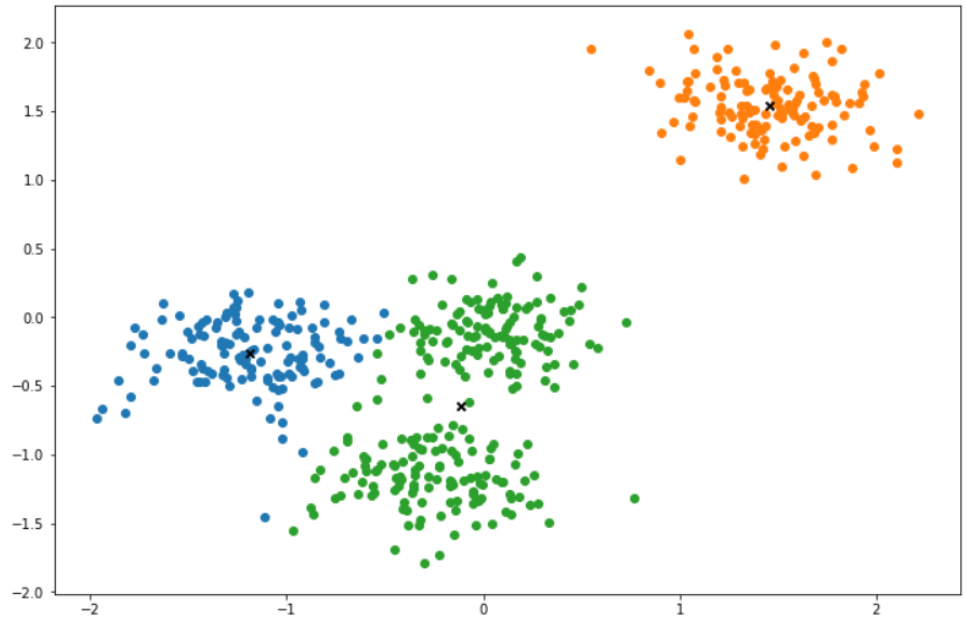
برای k = 2 :

```
converged with 5 iterations
```



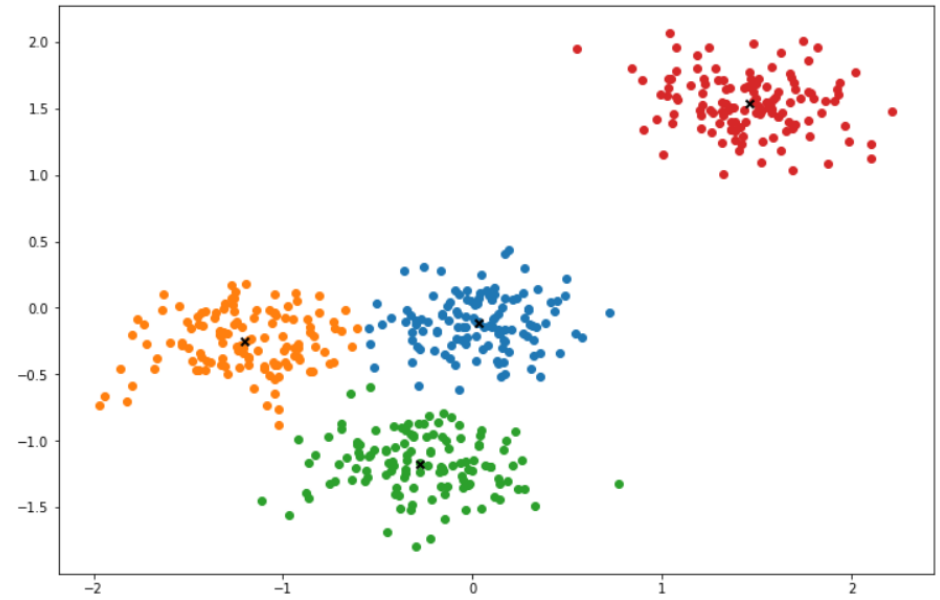
: K=3

```
converged with 7 iterations
```



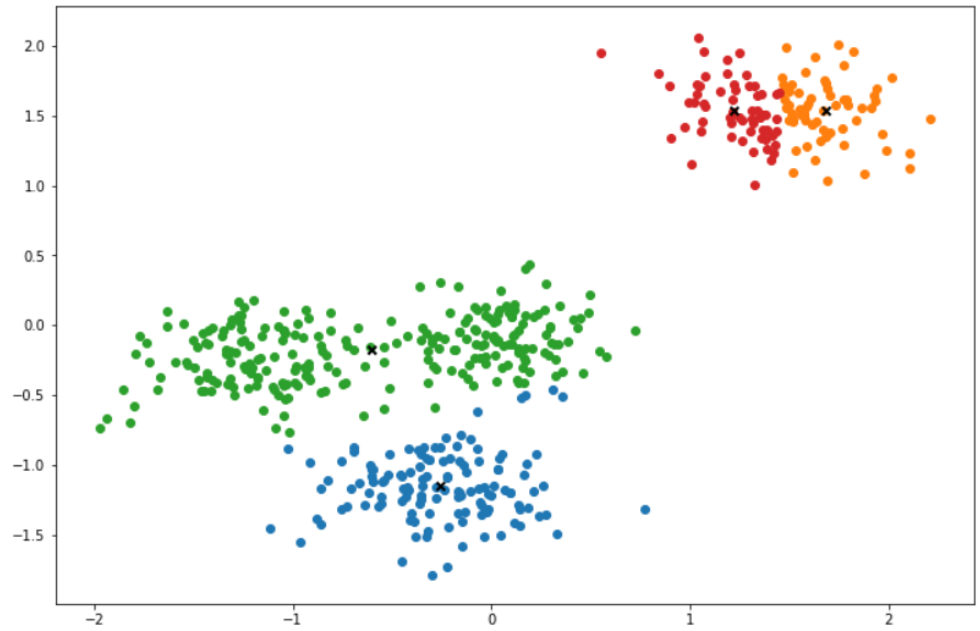
: K= 4

```
converged with 6 iterations
```



حالت دیگر برای k =4 که نشان میدهد انتخاب اولیه مراکز میتواند در خوشه ها تاثیر گذار باشد.

```
converged with 5 iterations
```



بخش دوم.

برای این بخش از لینک زیر کمک گرفته شده است

https://www.youtube.com/watch?v=52d7ha-GdV8&list=PLqnslRFeH2Upcrywf-u2etjdxxkL8nl7E&index=11

برای این بخش ابتدا کتابخانه های لازم را import میکنیم.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn import svm
```

سپس کلاس PCA را داریم که

```
class PCA:
    def __init__(self, n_components):
        self.n_components = n_components
        self.components = None
        self.mean = None
```

همانطور که مشخص است تعداد component ها که در صورت پروژه خواسته شده برابر 2 قرار دهیم را میتوان در این قسمت مقدار دهی کرد.

سپس تابع fit را داریم که به شکل زیر است.

```
def fit(self, X):
    # Mean centering
    X -= np.mean(X, axis=0)
    # covariance, function needs samples as columns
    cov = np.cov(X.T)
    eigenvalues, eigenvectors = np.linalg.eig(cov)
    eigenvectors = eigenvectors.T
    eigenvalues = eigenvalues[np.argsort(eigenvalues)[::-1]]
    eigenvectors = eigenvectors[np.argsort(eigenvalues)[::-1]]
    # store first n eigenvectors
    self.components = eigenvectors[0 : self.n_components]
```

در این تابع ابتدا میانگین را حساب میکنیم و طبق فرمول کوواریانس و تابع آن یعنی np.cov طبق الگوریتم PCA کوواریانس transpose داده را کوواریانسش را حساب می کنیم. دلیل transpose این است که داده ها را به صورت column ورودی میگیرد .

سپس eigenvalues , eigenvector هارا به وسیله np.linalg.eig(cov) بدست می آوریم اما باید دقت کرد که eigenvector به صورت ستونی برگردانده می شود و لازم است که transpose آنرا حساب کنیم.

حال np.argsort(eigenvalues) در واقع اندیس eigenvalues را به صورت decreasing برای ما صورت میکند.

یعنی الان ما eigenvalues و eigenvector هایی که داریم همه سورت شده میباشند

سپس component های ما به تعدادی که مشخص کرده ایم (در اینجا 2) ، 2 عضو ایتدایی(بزرگترین هارا) eigenvector که قبلا سورت شده است را انتخاب میکنیم .

به این صورت تابع fitبه اتمام میرسد.

حال به تابع transform میرسیم:

ابتدا میانگین را از X کم میکنیم و سپس داده را project میکنیم.

اینکار توسط dot کردن داده و transpose شده componentها انجام میشود. دوباره دلیل transpose این است که یک column vector میخواهیم که dot شود.

```
def transform(self, X):
    X -= np.mean(X, axis=0)
    return np.dot(X, self.components.T)
```

عکس از کل این کلاس :

```
class PCA:
    def __init__(self, n_components):
        self.n_components = n_components
        self.components = None
        self.mean = None
    def fit(self, X):
        # Mean centering
        X -= np.mean(X, axis=0)
        # covariance, function needs samples as columns
        cov = np.cov(X.T)
        eigenvalues, eigenvectors = np.linalg.eig(cov)
        eigenvectors = eigenvectors.T
        eigenvalues = eigenvalues[np.argsort(eigenvalues)[::-1]]
        eigenvectors = eigenvectors[np.argsort(eigenvalues)[::-1]]
        # store first n eigenvectors
        self.components = eigenvectors[0 : self.n_components]

    def transform(self, X):
        X -= np.mean(X, axis=0)
        return np.dot(X, self.components.T)
```

حال به تابع main میرسیم .

ابتدا توسط pandas داده را خوانده و ستون تارگت را از داده های جدا کرده و داده های به جز تارگت را X و تارگت را y می نامیم .

طبق خواسته سوال آنهارا scale کرده و y را تبدیل با آرایه numpy کرده و یک شی از کلاس pca که تعداد component های آن 2 باشد ساخته و داده هارا به آن میدهیم تا fit شوند همچنین تعدا ابعاد داده ها قبل و بعد از pca پرینت شده اند .

این داده را که project شده اند را plot میکنیم.

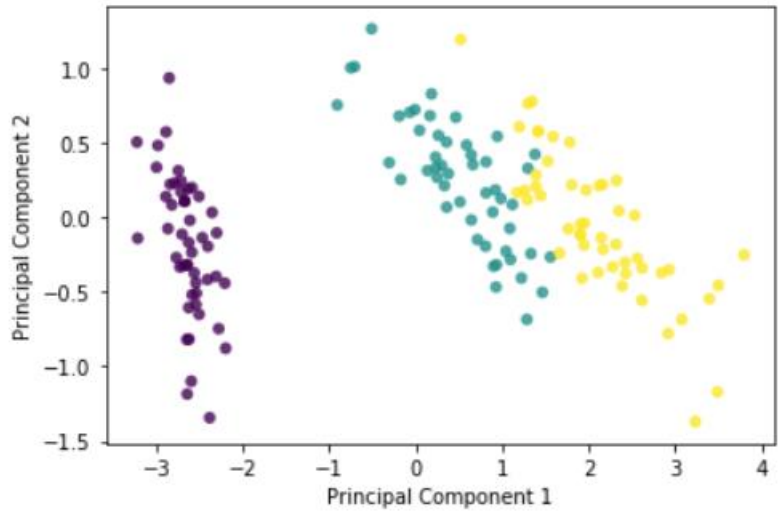
سپس طبق خواسته سوال داده های تست و آموزش را با نسبت 0.2 , 0.8جدا کرده و به کمک SVM در کتابخانه sklearn سبق خواسته صورت پروژه آنها را آموزش میدهیم و در نهایت Accuracy آنرا پرینت میکنیم.

```python
if __name__ == "__main__":
    df = pd.read_excel("dataset2.xlsx")
    X = df.drop('class', axis=1)
    y = df['class']
    scaler = StandardScaler()
    X = scaler.fit_transform(X)
    y = y.to_numpy()
    pca = PCA(2)
    pca.fit(X)
    X_projected = pca.transform(X)
    print("Shape of X:", X.shape)
    print("Shape of transformed X:", X_projected.shape)
    x1 = X_projected[:, 0]
    x2 = X_projected[:, 1]
    plt.scatter(
        x1, x2, c=y, edgecolor="none", alpha=0.8, cmap=plt.cm.get_cmap("viridis")
    )
    plt.xlabel("Principal Component 1")
    plt.ylabel("Principal Component 2")

    plt.show()
    X_train, X_test, y_train, y_test = train_test_split(X_projected, y, test_size=0.2)
    clf = svm.SVC()
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    print("Accuracy is : {}".format(accuracy_score(y_test, y_pred)))
```

توجه شود در کد فرستاده شده چون در صورت پروژه خواسته نشده بود که داده ها را پلات کنیم آن قسمت کامنت شده است.

```
Shape of X: (150, 4)
Shape of transformed X: (150, 2)
```




```
Accuracy is : 0.9666666666666667
```