علی آرمان ۴۰۲۳۶۲۴۰۰۲

در پیاده سازی ابتدا یک بار کد ها نوشته شد و در نهایت مرتب شد و به صورت بهینه کنار هم قرار گرفته.

در ابتدا نگارش کد ها گزارش شده و بعد از آن اجرای کد.

در ابتدا کتابخانه های لازم را ایمپورت میکنیم.

```
1. import warnings
2. import numpy as np
3. import pandas as pd
4. import seaborn as sns
5. import matplotlib.pyplot as plt
6. from sklearn.preprocessing import MinMaxScaler
7. from sklearn.model_selection import train_test_split
8. from sklearn.metrics import classification_report, confusion_matrix
```

مرحله ۱

در مرحله اول فایل مورد را باز کرده و ستون های مورد نظر را انتخاب میکنیم:

```
1. # Step 1: Read the dataset
2. def read_data(file_path):
3.     data = pd.read_csv(file_path)
4.     selected_data = data[['tenure', 'income', 'employ']]
5.     return selected_data
```

مرحله ۲

در این مرحله پیش پردازش متن را انجام میدهیم که در اینجا ابتدا مقادیر null با میانگین پر شده اند و داده های عددی با minmax scale نرمال سازی شده اند:

```
1. # Step 2: Preprocess the data
2. def preprocess_data(data):
3.     # Fill missing data with means
4.     data.fillna(data.mean(), inplace=True)
5.     # Normalize 'tenure', 'income', and 'employ' using Min-Max scaling
6.     scaler = MinMaxScaler()
7.         data[['tenure', 'income', 'employ']] = scaler.fit_transform(data[['tenure', 'income',
'employ']])
8.     return data
```

مرحله ۳

در این مرحله مطابق توضیحات ۸۰ درصد داده های برای آموزش جدا شده اند:

```
1. # Step 3: Split the data into training and testing sets
2. def split_data(data):
3.     X_train, X_test = train_test_split(data, test_size=0.2, shuffle=True, random_state=17)
4.     return X_train, X_test
```

مرحله ۴

در این مرحله به دلیل آزمایش های بیشتر، ۶ روش برای تعیین مرکز دسته ها انتخاب شد که این ۷ روش عبارت اند از:

۱- Random:

انتخاب مرکز دسته های به صورت رندم

۲- Kmeans++:

انتخاب مرکز دسته ها با استفاده از الگوریتم kmeans++ (که بهترین نتیجه را حاصل کرد)

۳- First-row-point:

انتخاب k مقدار اول به عنوان مرکز دسته

۴- Random-partition:

انتخاب به صورت پارتیشین بندی

۵- Random-subset:

انتخاب به صورت تصادفی از بین ساب ست ها

۶- Kmeans۲:

استفاده از الگوریتم kmeans|| که شباهت زیادی به kmeans++ دارد.

در آموزش از تعداد دور های مختلفی جهت پیدا کردن بهترین نتیجه استفاده شده که تعداد دور های آزمایش شده برابر است با: ۱۰۰ ۲۰۰ ۵۰۰ و ۱۰۰۰

در نهایت به تعداد دور های آموزش فاصله هر داده از هر مرکز داده (که با یکی از ۶ روش بدست آمده) محاسبه می شود و کمترین فاصله از بین فاصله بین هر داده با مراکز داده ها به دست می آید سپس لیبل آن دسته به داده اختصاص داده میشود.

```python
# Step 4: Implement the K-means algorithm
def k_means(data, k, initialization, num_iterations=100):
    # Convert the data array back to a pandas DataFrame
    data = pd.DataFrame(data)

    if initialization == 'random':
        # Initialize centroids randomly
        centroids = data.sample(n=k).values

    elif initialization == 'kmeans++':
        # Initialize centroids using KMeans++ algorithm
        centroids = [data.sample().values[0]]
        for _ in range(1, k):
          distances = np.array([min([np.inner(c-x,c-x) for c in centroids]) for x in data.values])
            probs = distances/distances.sum()
            cumprobs = probs.cumsum()
            r = np.random.rand()
            for j,p in enumerate(cumprobs):
                if r < p:
                    i = j
                    break
            centroids.append(data.values[i])

    elif initialization == 'first_few_points':
        # Initialize centroids using the first few data points
        centroids = data.head(k).values

    elif initialization == 'random_partition':
        # Initialize centroids using random partition method
        shuffled_indices = np.random.permutation(len(data))
        centroids = [data.values[i] for i in range(k)]

    elif initialization == 'random_subset':
        # Initialize centroids using a random subset of data points
        subset_indices = np.random.choice(len(data), k, replace=False)
        centroids = data.iloc[subset_indices].values

    elif initialization == 'kmeans2':
        # Initialize centroids using k-means|| algorithm
        centroids = [data.sample().values[0]]
        for _ in range(1, k):
          distances = np.array([min([np.inner(c-x,c-x) for c in centroids]) for x in data.values])
            probs = k * distances / sum(distances)
            cumprobs = probs.cumsum()
            r = np.random.rand()
            for j, p in enumerate(cumprobs):
                if r < p:
                    i = j
                    break
            centroids.append(data.values[i])

    for _ in range(num_iterations):
        distances = np.sqrt(((data.values[:, np.newaxis, :] - centroids)**2).sum(axis=2))
        # Assign each data point to the nearest centroid
        labels = np.argmin(distances, axis=1)
        # Update centroids based on the mean of data points in each cluster
        for i in range(k):
            centroids[i] = np.mean(data.values[labels == i], axis=0)
    return labels, centroids
```

مرحله ۵

در این مرحله جهت مشاهده و مقایسه بهتر داده ها از ۵ پلات استفاده شده که به شرح زیر می باشد:

در عنوان شماره تعداد دسته، تعداد دور آموزش، شماره آزمون و نوع انتخاب مرکز دسته ذکر شده

به ترتیب از سمت چپ

۱-نموار دسته ها به همراه مراکز

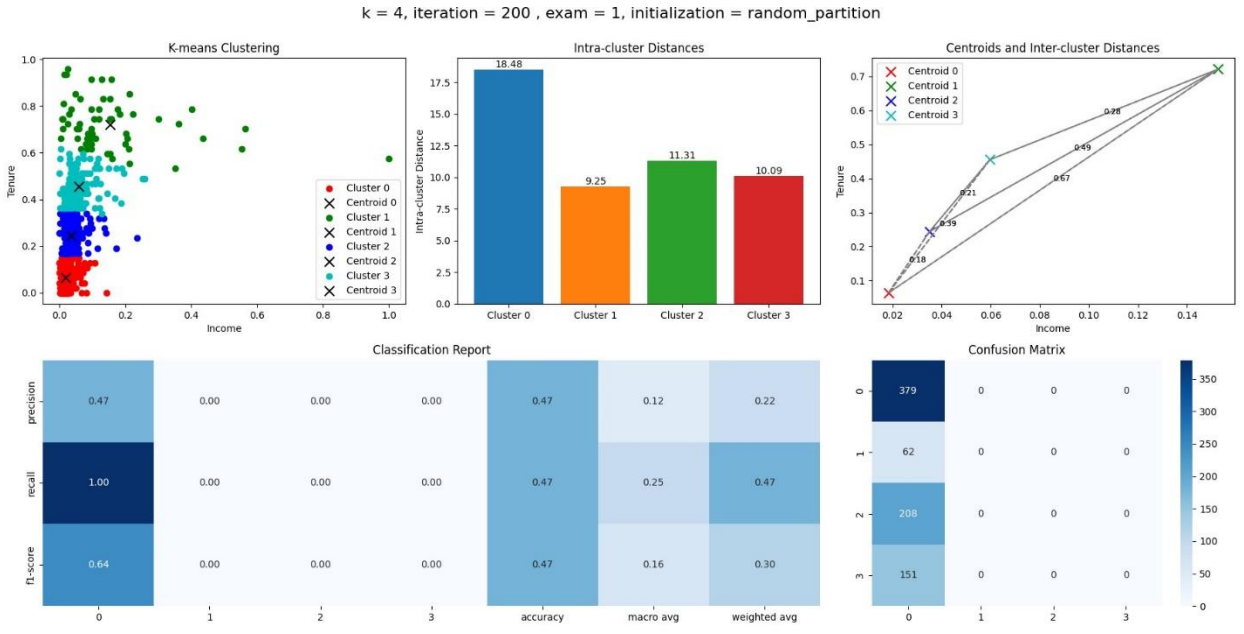
۲-نمودار فاصله درون خوشه ای برای هر خوشه

۳-فاصله مرکز دسته ها

۴- classification report

۵- confusion matrix

به عنوان مثال:



جهت ایجاد یک پلات ۲ در ۳ ایجاد شده و در هر ساب پلات داده مربوطه قرار داده شده برای محاسبه فاصله از intra_cluster_distance استفاده شده و برای فاصله بین مرکز دسته ها از در همان پلات هر مرکز دسته فاصله اش با سایرین حساب شده با خط در پلات قرار داده شده و مقادیر class_report_df و

confusion_matrix از تابع calculate_measures استفاده شده که در اجرا شرح داده خواهد شد. در نهایت هر پلات ایجاد شده برای بررسی ذخیره می شود.

کد مربوط به این مرحله:

```
# Step 5: Visualize the clusters
def visualize_clusters(data, labels, centroids, class_report_df, cm, k, it, exam, initialization):
    plt.figure(figsize=(18, 9))
    colors = ['r', 'g', 'b', 'c', 'm']

    plt.subplot(2, 3, 1)
    for i in range(len(centroids)):
        plt.scatter(data[labels == i][:, 0], data[labels == i][:, 1], color=colors[i], label=f'Cluster {i}')
        plt.scatter(centroids[i][0], centroids[i][1], color='k', marker='x', s=100, label=f'Centroid {i}')
    plt.xlabel('Income')
    plt.ylabel('Tenure')
    plt.title('K-means Clustering')
    plt.legend()

    plt.subplot(2, 3, 2)
    for i, intra_distance in enumerate(intra_cluster_distance(data, labels, centroids)):
        plt.bar(f'Cluster {i}', intra_distance)
        plt.text(i, intra_distance, f'{intra_distance:.2f}', ha='center', va='bottom')
    plt.ylabel('Intra-cluster Distance')
    plt.title('Intra-cluster Distances')

    plt.subplot(2, 3, 3)
    for i, centroid in enumerate(centroids):
        plt.scatter(centroid[0], centroid[1], color=colors[i], marker='x', s=100, label=f'Centroid {i}')
        for j, other_centroid in enumerate(centroids):
            if i != j:
                distance = np.sqrt(((centroid - other_centroid)**2).sum())
                plt.plot([centroid[0], other_centroid[0]], [centroid[1], other_centroid[1]], color='gray', linestyle='--')
                plt.text((centroid[0] + other_centroid[0]) / 2, (centroid[1] + other_centroid[1]) / 2, f'{distance:.2f}', fontsize=8)
    plt.xlabel('Income')
    plt.ylabel('Tenure')
    plt.title('Centroids and Inter-cluster Distances')
    plt.legend()

    plt.subplot(2, 3, (4, 5))
    sns.heatmap(class_report_df.iloc[:, :-1].T, annot=True, cmap='Blues', cbar=False, fmt=".2f")
    plt.title('Classification Report')

    plt.subplot(2, 3, 6)
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=np.unique(labels), yticklabels=np.unique(labels))
    plt.title('Confusion Matrix')

    plt.suptitle(f'k = {k}, iteration = {it} , exam = {exam}, initialization = {initialization}', fontsize=16, y=1)
    plt.tight_layout()

    filename = f'img/k{k}_it{it}_exam{exam}_{initialization}.png'
    plt.savefig(filename)
```

مرحله ۶

تابع calculate_measures، در این تابع لیبل ها و دسته بندی که توسط الگوریتم ایجاد شده به عنوان ورودی داده میشود و در نهایت خروجی confusion_matrix و classification_report را ایجاد میکند و بازگشت می دهد:

```
1. # Step 6: Calculate evaluation measures
2. def calculate_measures(labels, predictions):
3.     cm = confusion_matrix(labels, predictions)
4.     class_report_df = pd.DataFrame(classification_report(labels, predictions,
output_dict=True)).T
5.     return cm, class_report_df
```

مرحله ۷

محاسبه فاصله درون خوشه ای:

برای محاسبه تابع intra_cluster_distance نوشته شده که در هر دسته فاصله ی هر داده ها را حساب کرده و در نهایت تمام آن ها را با هم جمع میکند و داخل لیست فاصله ها قرار میدهد.

کد این مرحله:

```
1. # Step 7: Calculate intra-cluster distance
2. def intra_cluster_distance(data, labels, centroids):
3.     distances = []
4.     for i, centroid in enumerate(centroids):
5.         distance = np.sqrt(((data[labels == i] - centroid)**2).sum(axis=1)).sum()
6.         distances.append(distance)
7.     return distances
```

مرحله ۸

این قسمت برای محاسبه فاصله بین مراکز دسته ها می باشد که به صورت مستقیم داخل مرحله ۵ استفاده شده

```
1. # Step 8: Calculate inter-cluster distance
2. def inter_cluster_distance(centroids):
3.     distances = []
4.     k = len(centroids)
5.     for i in range(k):
6.         for j in range(i+1, k):
7.             distance = np.sqrt(((centroids[i] - centroids[j])**2).sum())
8.             distances.append(distance)
9.     return distances
```

مرحله اجرا

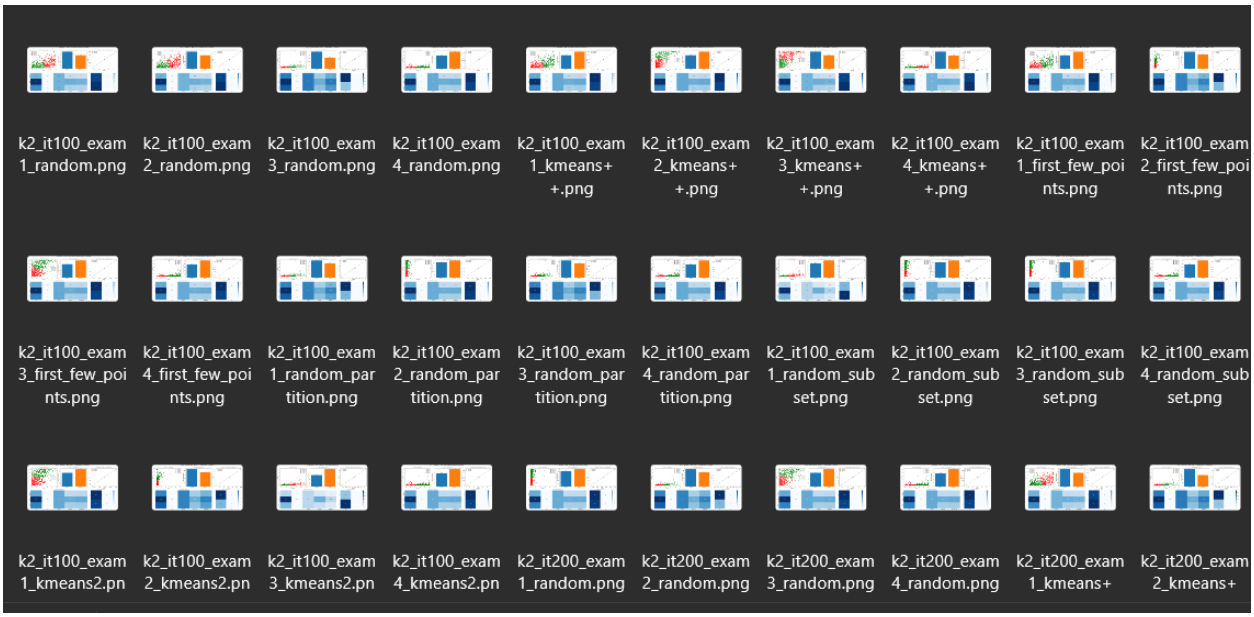
برای اجرا ابتدا داده فراخونی شده و مرحله پیش پردازش بر روی آن انجام شده، سپس داده تقسیم شده و وارد مرحله آموزش می شود.

جهت پیدا کردن بهترین نتیجه از ۴ حلقه تو در تو استفاده شده

۱- حلقه اول برای حرکت بین مقادیر k = ۱ ۲ ۳ ۴ ۵
۲- حلقه دوم برای حرکت بین مقادیر تعداد دور آموزش iterations_values = ۱۰۰ ۲۰۰ ۵۰۰ ۱۰۰۰
۳- حلقه سوم برای حرکت بین روش های انتخاب مراکز دسته (همان ۶ روش ذکر شده در مرحله ۴)
۴- حلقه چهارم برای ۴ آزمایش که در ۳ آزمایش اول ۲ ستون به صورت تصادفی می باش

در داخلی ترین حلقه هر بار تابع k_means با مقادیر نسبت داده شده فراخوانی می شود و لیبل ها و مراکز دسته بازگشت داده میشود و با توجه به این مقادیر تابع calculate_measures فراخونی میشود و جهت نمایش تابع نوشته شده در مرحله ۵ (visualize_clusters) را فراخوانی میکنیم.

تعداد دور های اجرا برابر است با ۴*۴*۶*۴ = ۳۸۶ در نتیجه ۳۸۶ عکس ذخیره میشود



کد این مرحله:

```
def main():
    # Step 1:
    data = read_data('Telecust1.csv')

    # Step 2:
    data = preprocess_data(data)

    # Step 3
    X_train, X_test = split_data(data)

    k_values = [2, 3, 4, 5]
    iterations_values = [100, 200, 500, 1000]
    initialization = ['random', 'kmeans++', 'first_few_points', 'random_partition',
'random_subset', 'kmeans2']

    warnings.filterwarnings("ignore", category=UserWarning)
    counter = 1
    for k in k_values:
        for it in iterations_values:
            for init in initialization:
                for i in range(4):
                    if i < 3:
                        selected_columns = np.random.choice(range(len(data.columns)), size=2,
replace=False)
                        selected_data = X_train.iloc[:, selected_columns].values
                    else:
                        selected_data = X_train.values

                    labels, centroids = k_means(selected_data, k, initialization=init,
num_iterations = it)

                    cm, class_report_df = calculate_measures(labels, np.zeros_like(labels))

                    visualize_clusters(selected_data, labels, centroids, class_report_df, cm, k,
it, i+1, init)

                    print(f'{counter} of 386')
                    counter += 1

if __name__ == "__main__":
    main()
```

با بررسی تصاویر ذخیره شده، بهترین امتیاز به شرح زیر می باشد که در چندین حالت قابل مشاهده بوده:

Precision=۰.۷۲

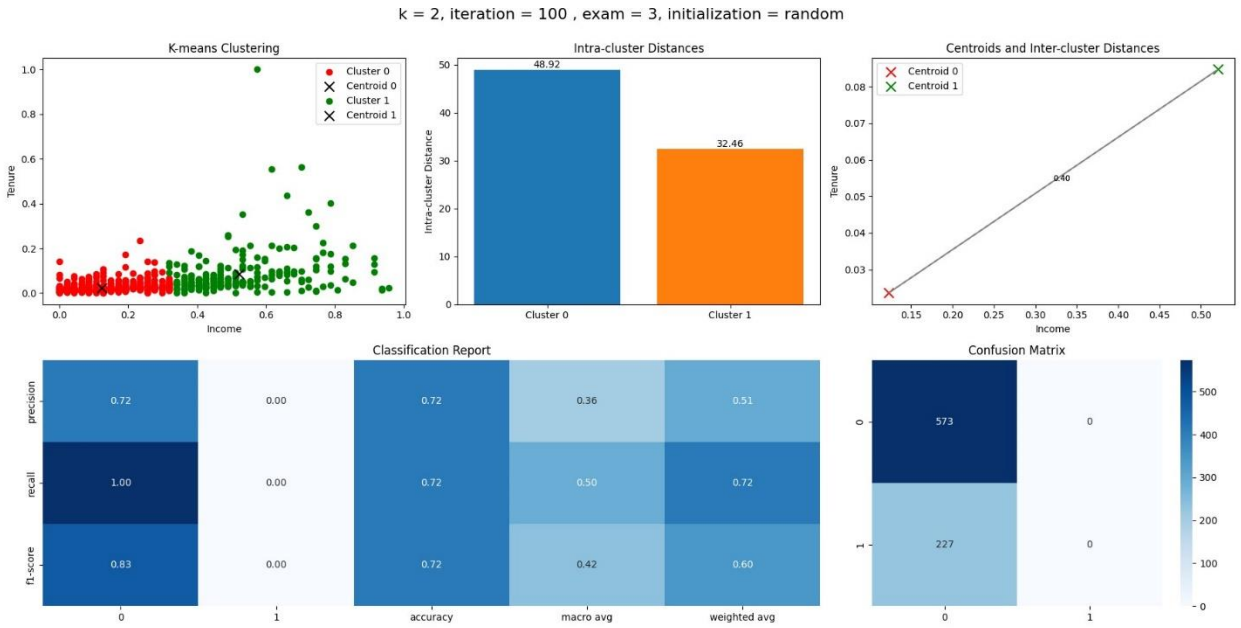
Recall=۱.۰۰

F۱-score=۰.۸۳

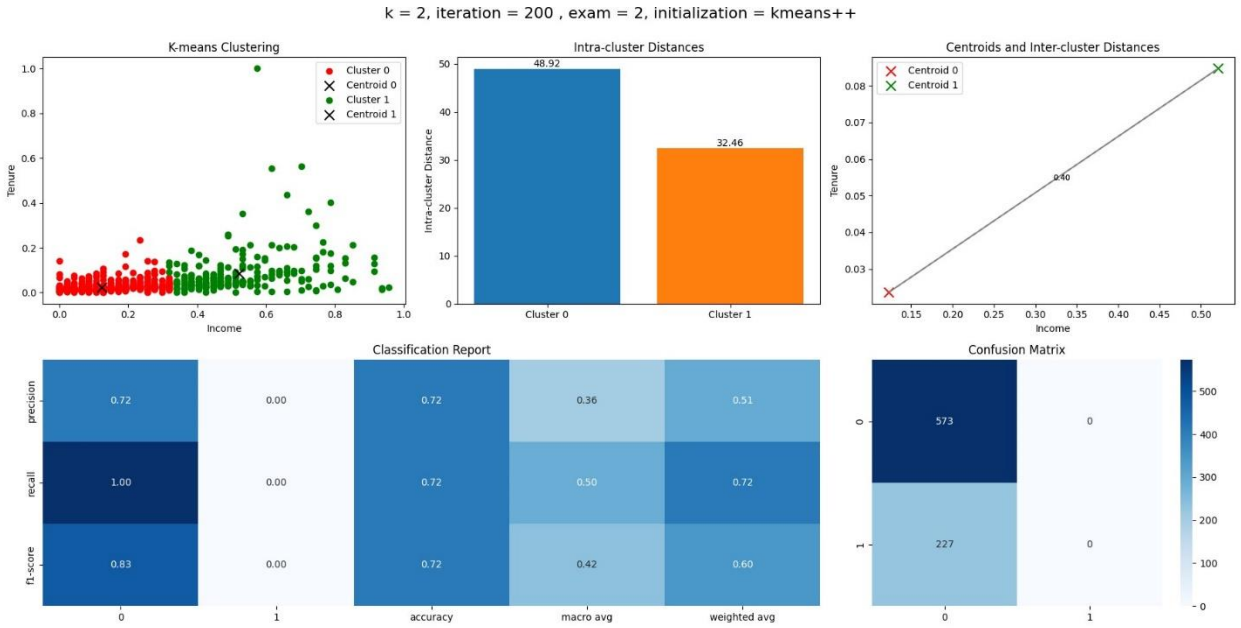
Accuracy=۰.۷۲

برخی از خروجی این داده ها:

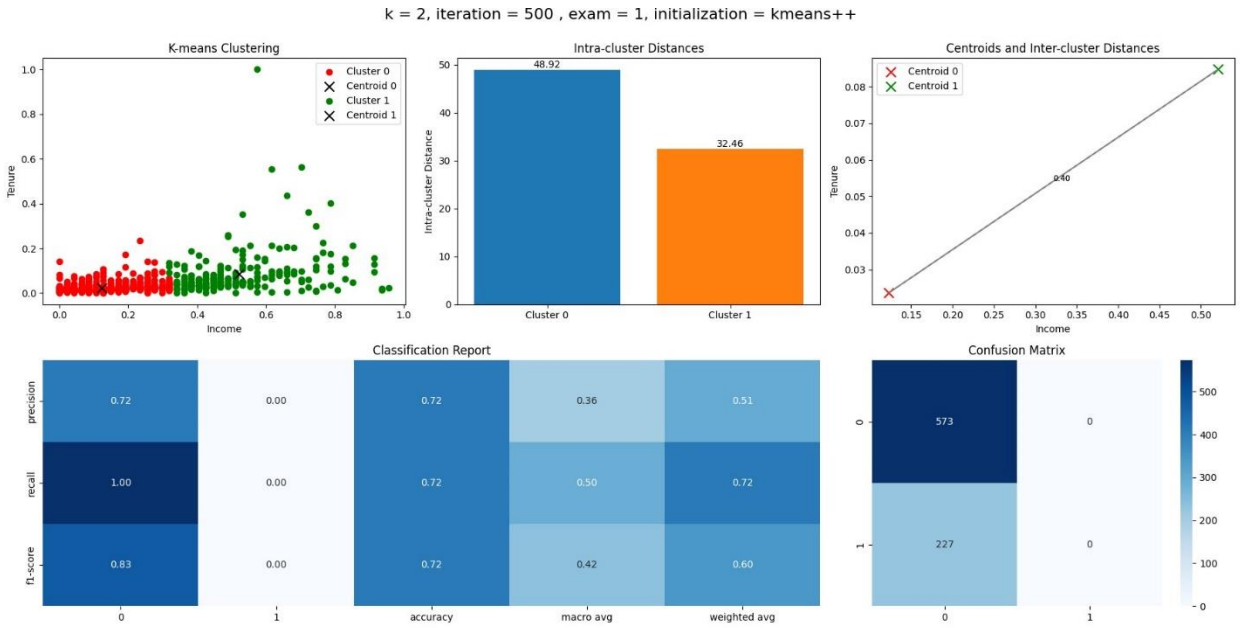
:k۲_it۱۰۰_exam۱_random



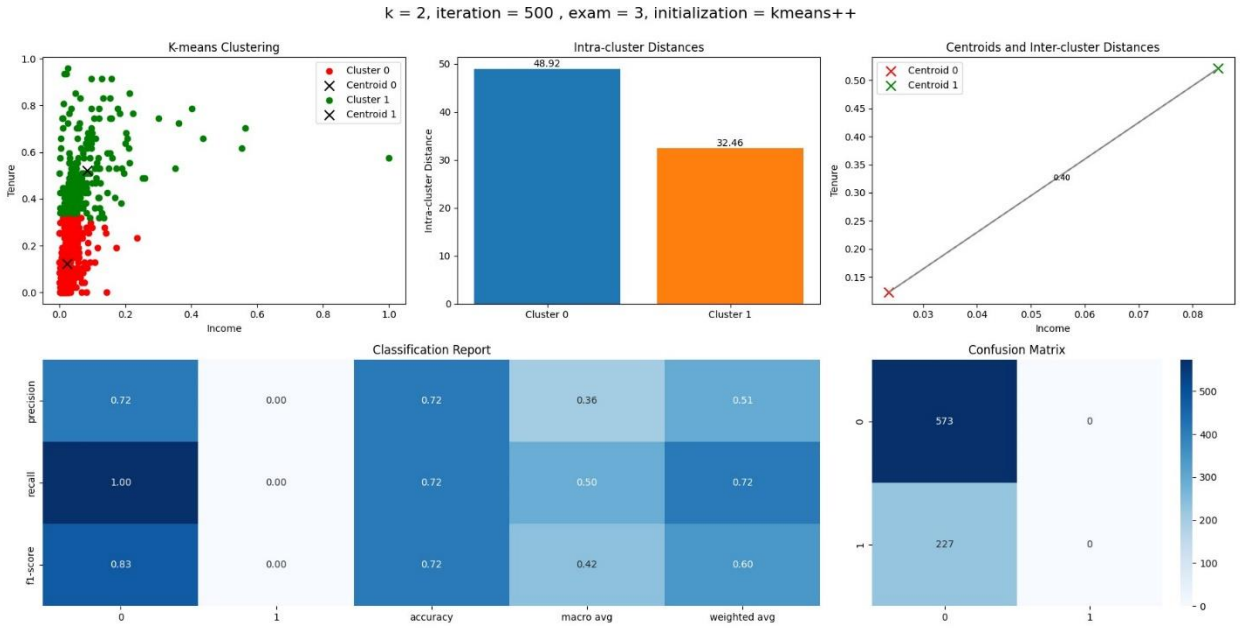
:k۲_it۲۰۰_exam۲_kmeans++



:k۲_it۵۰۰_exam۱_kmeans++



:k۲_it۵۰۰_exam۳_kmeans++



در کل بهترین نتیجه ها در تعداد خوشه برابر ۲ و الگوریتم kmeans++ با تعداد دور کمتر از ۱۰۰۰ نتیجه میدهد.