What does this program do?

This program is a customer analysis tool that allows users to perform three different tasks:

1-Split customers into groups (clusters) based on their total spending and age using the k-means clustering algorithm.

2-Determine the optimal number of clusters for the customer data using the elbow method.

3-Mine association rules from a transactional dataset using the Apriori algorithm.

For task 1, the user is prompted to select a CSV file containing customer data and enter the number of clusters to split the customers into. The program loads the data from the CSV file, preprocesses it, and uses k-means clustering to split the customers into the specified number of clusters. It then displays a scatter plot of the data points, coloring each point according to its cluster label, and a table of customer names, ages, total spending, and cluster labels in a separate window.

For task 2, the user is prompted to select a CSV file containing customer data. The program loads the data from the CSV file, preprocesses it, and uses the elbow method to determine the optimal number of clusters for the data. It then displays a plot of the sum of squared distances as a function of the number of clusters.

For task 3, the user is prompted to select a CSV file containing transactional data. The program loads the data from the CSV file and uses the Apriori algorithm to mine association rules from the transactions. It then displays the mined association rules in a separate window.

Explanation of the code:

This code imports the necessary libraries.

tkinter is a library for creating GUI applications in Python.

filedialog is a module within **tkinter** that provides functions for opening file selection dialogs.

```
import tkinter as tk
from tkinter import *
import tkinter.ttk as ttk
from tkinter import filedialog
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from PIL import ImageTk,Image
from apyori import  apriori
```

tkinter: a library for creating graphical user interfaces (GUIs) in Python.

filedialog: a module in tkinter that provides functions for opening and saving files.

pandas: a library for data manipulation and analysis.

numpy: a library for scientific computing with Python.

matplotlib.pyplot: a module in matplotlib for creating plots and charts.

sklearn.cluster: a module in the scikit-learn library for clustering algorithms.

PIL.ImageTk and PIL.Image: modules in the PIL (Python Imaging Library) for working with images.

apyori: a library for mining association rules.

This code defines the split_customers() function, which takes a CSV file dataset and an integer n as arguments.

The function loads the CSV file into a pandas data frame and extracts the customer, age, and total columns. It stores the values in the customer_names, customer_ages, and customer_totals variables, respectively.

```
def split_customers(dataset,n):
    data = pd.read_csv(dataset)
    customer_names = data["customer"].values
    customer_ages = data["age"].values
    customer_totals = data["total"].values
    customer_data = np.column_stack([customer_totals, customer_ages])
    kmeans = KMeans(n_clusters=n, n_init=20)
    kmeans.fit(customer_data)
```

It then combines the customer_totals and customer_ages arrays into a single array customer_data using np.column_stack() from numpy.

Also this code uses the KMeans class from sklearn.cluster to perform k-means clustering on the customer_data array. The n_clusters parameter specifies the number of clusters to create (n in this case), and the n_init parameter specifies the number of times the algorithm will be run with different centroid seeds.

The fit() method fits the k-means model to the data.

```
window = tk.Toplevel(root)
window.config(bg="#001253")
custom_font = ("Arial", 14)
bg_color = "#fff"
text_color = "#000"
button_color = "#00f"
```

This code defines variables for a custom font and color scheme to be used in the GUI. custom_font is a tuple with the font family and size, bg_color is the background color, text_color is the text color, and button_color is the button color.

```
def display_plot():
    labels = kmeans.labels_
    points = customer_data
    plt.scatter(points[:, 0], points[:, 1], c=labels, cmap='viridis')
    plt.xlabel('Total')
    plt.ylabel('Age')
    plt.title('K-Means Clustering')
    plt.show()
```

This code defines a function display_plot() that extracts the cluster labels and data points from the k-means model, and creates a scatter plot of the data points using matplotlib.pyplot. The c parameter specifies the cluster labels for each point, and the cmap parameter specifies the color map to use. The xlabel(), ylabel(), and title() functions add labels and a title to the plot. The show() function displays the plot.

```
tk.Button(window, text="Show plot", command=display_plot, font=custom_font, bg=button_color, fg="white").pack()
scrollbar = tk.Scrollbar(window)
scrollbar.pack(side="right", fill="y")
text = tk.Text(window, yscrollcommand=scrollbar.set, width=31, font=custom_font, bg=bg_color, fg=text_color)
text.pack(side="left", fill="both")
scrollbar.config(command=text.yview)
```

1-A themed button is created using the Button method from tkinter, with the display_plot function as its command. This button, when clicked, will display the scatter plot of the data points

2-A scrollbar is created using the Scrollbar method from tkinter

3-A themed text box is created using the Text method from tkinter, and the scrollbar is configured to work with the text box

---

```
text.insert("end", "Name\tAge\tTotal\tCluster\n")
for i in range(len(customer_data)):
    name = customer_names[i]
    age = customer_ages[i]
    total = customer_totals[i]
    cluster = kmeans.labels_[i]
    text.insert("end", f"{name}\t{age}\t{total}\t{cluster}\n")
```

The output of the k-means clustering process is inserted into the text box. For each customer, the function inserts the customer's name, age, total spending, and cluster label into the text box

---

```
def find_optimal_clusters(dataset):
    data = pd.read_csv(dataset)
    inertias = []
    for k in range(1, 15):
        km = KMeans(n_clusters=k, n_init=20)
        km = km.fit(data[['age', 'total']])
        inertias.append(km.inertia_)

    plt.plot(range(1, 15), inertias, 'bo-')
    plt.xlabel('Number of clusters')
    plt.ylabel('Inertias')
    plt.title('Elbow Method For Optimal k')
    plt.show()
```

This code defines the **find_optimal_clusters()** function, which takes a CSV file **dataset** as an argument.

It then uses a loop to perform k-means clustering with different numbers of clusters (1 to 15), and stores the resulting inertias in the **inertias** list. The **inertia_** attribute of the **KMeans** object returns the sum of squared distances of samples to their closest cluster center.

The code then plots the inertias against the number of clusters using **matplotlib.pyplot**.

```
def mine_association_rules(dataset):
    custom_font = ("Arial", 14)
    bg_color = "#fff"
    text_color = "#000"
    button_color = "#00f"
    df1 = pd.read_csv(dataset, header=None)
    df1.columns = ["items", "count", "total", "rnd", "customer", "age", "city", "paymentType"]
    records = []
    itemDataSet = pd.DataFrame(df1['items'])
    itemDataSet = pd.concat([itemDataSet['items'], itemDataSet['items'].str.split(',', expand=True)], axis=1)
    for i in range(0, 9835):
        records.append([str(itemDataSet.values[i, j]) for j in range(0, 33)])
    new_records = []
    temp = []
    for i in range(0, 9835):
        for j in range(0, len(records[i])):
            if records[i][j] != 'None':
                temp.append(records[i][j])
        new_records.append(temp)
        temp = []
    del records
    del temp
    window = tk.Toplevel(root)
    window.config(bg="#001253")
    window.geometry("350x200")
    tk.Label(window, text="Minimum Support:",bg="#001253",fg="white",font=custom_font).grid(row=0, column=1, padx=5, pady=5)
    support_entry = tk.Entry(window)
    support_entry.grid(row=0, column=2, padx=5, pady=5)
    tk.Label(window, text="Minimum Confidence:",bg="#001253",fg="white",font=custom_font).grid(row=1, column=1, padx=5, pady=5)
    confidence_entry = tk.Entry(window)
    confidence_entry.grid(row=1, column=2, padx=5, pady=5)
    tk.Label(window, text="Minimum Lift:", bg="#001253", fg="white", font=custom_font).grid(row=2, column=1, padx=5, pady=5)
    min_lift_entry = tk.Entry(window)
    min_lift_entry.grid(row=2, column=2, padx=5, pady=5)
    tk.Label(window, text="Minimum Length:", bg="#001253", fg="white", font=custom_font).grid(row=3, column=1, padx=5, pady=5)
    min_length_entry = tk.Entry(window)
    min_length_entry.grid(row=3, column=2, padx=5, pady=5)
```

This code loads the CSV file into a pandas data frame and extracts the customer and item columns. It then creates a list of transactions, where each transaction is a list of strings representing the customer and item for that transaction then uses the apriori() function from the apyori library to mine association rules from the transactions list. The min_support parameter specifies the minimum support for an itemset to be considered frequent. The min_confidence parameter specifies the minimum confidence for an association rule to be considered valid. The min_lift parameter specifies the minimum lift for an association rule to be considered interesting. The min_length parameter specifies the minimum number of items in an itemset.

The mined association rules are stored in the rules list.

then we create labels and inputs for Minimum_Support,Minimum_Confidence,Minimum_lift,Minimum_Length.

```
    def inspect(output):
        lhs = [tuple(result[2][0][0])[0] for result in output]
        rhs = [tuple(result[2][0][1])[0] for result in output]
        support = [result[1] for result in output]
        confidence = [result[2][0][2] for result in output]
        lift = [result[2][0][3] for result in output]
        return list(zip(lhs, rhs, support, confidence, lift))

    def display_rules(rules):
        window = tk.Toplevel(root)
        window.geometry("850x230")
        window.title("Association Rules")
        tree = ttk.Treeview(window, columns=("lhs", "rhs", "support", "confidence", "lift"), show="headings")
        tree.pack(side="left")
        tree.column("lhs", width=200, anchor="center")
        tree.column("rhs", width=200, anchor="center")
        tree.column("support", width=150, anchor="center")
        tree.column("confidence", width=150, anchor="center")
        tree.column("lift", width=150, anchor="center")
        tree.heading("lhs", text="Left Hand Side")
        tree.heading("rhs", text="Right Hand Side")
        tree.heading("support", text="Support")
        tree.heading("confidence", text="Confidence")
        tree.heading("lift", text="Lift")
        for i, rule in enumerate(rules):
            tree.insert("", "end", values=rule)
    display_rules(inspect(rules))
tk.Button(window, text="Mine association rules", command=mine, font=custom_font, bg=button_color, fg="white").place(x=80,y=160)
```

The inspect function takes in a list output and returns a list of tuples. It does this by creating four lists:

lhs: A list of the left-hand side items of the association rules in output. This is created by iterating over output and extracting the left-hand side item of each rule.

rhs: A list of the right-hand side items of the association rules in output. This is created in a similar way to lhs, but extracting the right-hand side item of each rule.

support: A list of the support values of the association rules in output. This is created by iterating over output and extracting the support value of each rule.

confidence: A list of the confidence values of the association rules in output. This is created by iterating over output and extracting the confidence value of each rule.

lift: A list of the lift values of the association rules in output. This is created in a similar way to confidence, but extracting the lift value of each rule.

Once these lists have been created, the function returns a list of tuples, where each tuple contains the corresponding elements from the lhs, rhs, support, confidence, and lift lists.

The display_rules function takes in a list of rules and displays them in a Treeview widget. It does this by creating a new window and adding a Treeview widget to it. The Treeview widget has five columns: "lhs", "rhs", "support", "confidence", and "lift". These columns are set up with the appropriate headings and widths.

Next, the function iterates over the list of rules and inserts each rule into the Treeview widget.

The values of each rule are inserted into the corresponding columns of the Treeview widget.

Finally, there is a button that, when clicked, calls the mine function and then calls display_rules with the result of calling inspect on the output of the mine function.

```
# Create Tkinter GUI
root = tk.Tk()

root.iconbitmap("logo.ico")
root.title("Data Analysis")
image_0=Image.open("background.jpg")
bck_end=ImageTk.PhotoImage(image_0)
lbl=Label(root,image=bck_end)
lbl.place(x=-2,y=0)
    # Create a custom font and color scheme
custom_font = ("Arial", 12)
# Add a label for the dataset file
tk.Label(root, text="Dataset file:",font="white",bg="#001253",fg="white").grid(row=0, column=0, padx=5, pady=5)

# Add a text box for dataset file path
file_input = tk.Entry(root,width=30)
file_input.grid(row=0, column=1, padx=5, pady=5)

# Add a button to browse for a dataset file
tk.Button(root, text="Browse", command=lambda: file_input.insert(0, filedialog.askopenfilename()), font=custom_font, bg=button_color, fg="white").grid(row=0, column=2, padx=5, pady=5)
# Add a label and entry field for the min_threshold value
Split_Clusters_label = tk.Label(text="Number of Cluster:",font=custom_font,fg="white",bg="#001253")
Split_Clusters_label.grid(row=2, column=0, padx=5, pady=5)
Split_Clusters_entry = tk.Entry()
Split_Clusters_entry.grid(row=2, column=1, padx=5, pady=5)

# Add a button to find optimal number of clusters
Optimal_Clusters_button=tk.Button(text="Optimal Clusters", command=lambda:find_optimal_clusters(file_input.get()), font=custom_font, fg="white",bg=button_color)
Optimal_Clusters_button.grid(row=1, column=0, columnspan=3, padx=5, pady=5)
# Add a button to split customers into groups
tk.Button(root, text="Split Customers", command=lambda: split_customers(file_input.get(),int(Split_Clusters_entry.get())), font=custom_font, bg=button_color, fg="white").grid(row=2, column=2, columnspan=3, padx=5,

# Add a button to run the 'mine_association_rules' function
association_button = tk.Button(text="Mine Association Rules", command=lambda:mine_association_rules(file_input.get()), font=custom_font, bg=button_color, fg="white")
association_button.grid(row=3, column=0, columnspan=3, padx=5, pady=5)
# Run the tkinter event loop
root.mainloop()
```

For the last part:This code creates a graphical user interface (GUI) using the tkinter library in Python. The GUI has several components:

1-A label that says "Dataset file:" and a text box for entering the file path of a dataset.

2-A "Browse" button that allows the user to select a file from their computer using a file browser.

3-A label that says "Number of Clusters" and an entry field for entering a number of clusters.

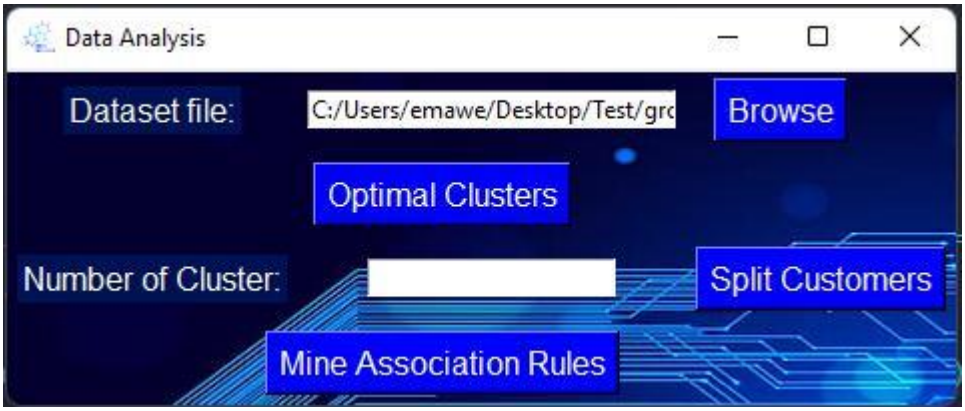
4-A "Optimal Clusters" button that, when clicked, runs a function called find_optimal_clusters and passes the file path entered in the text box as an argument.

5-A "Split Customers" button that, when clicked, runs a function called split_customers and passes the file path and the number of clusters entered in the entry field as arguments.

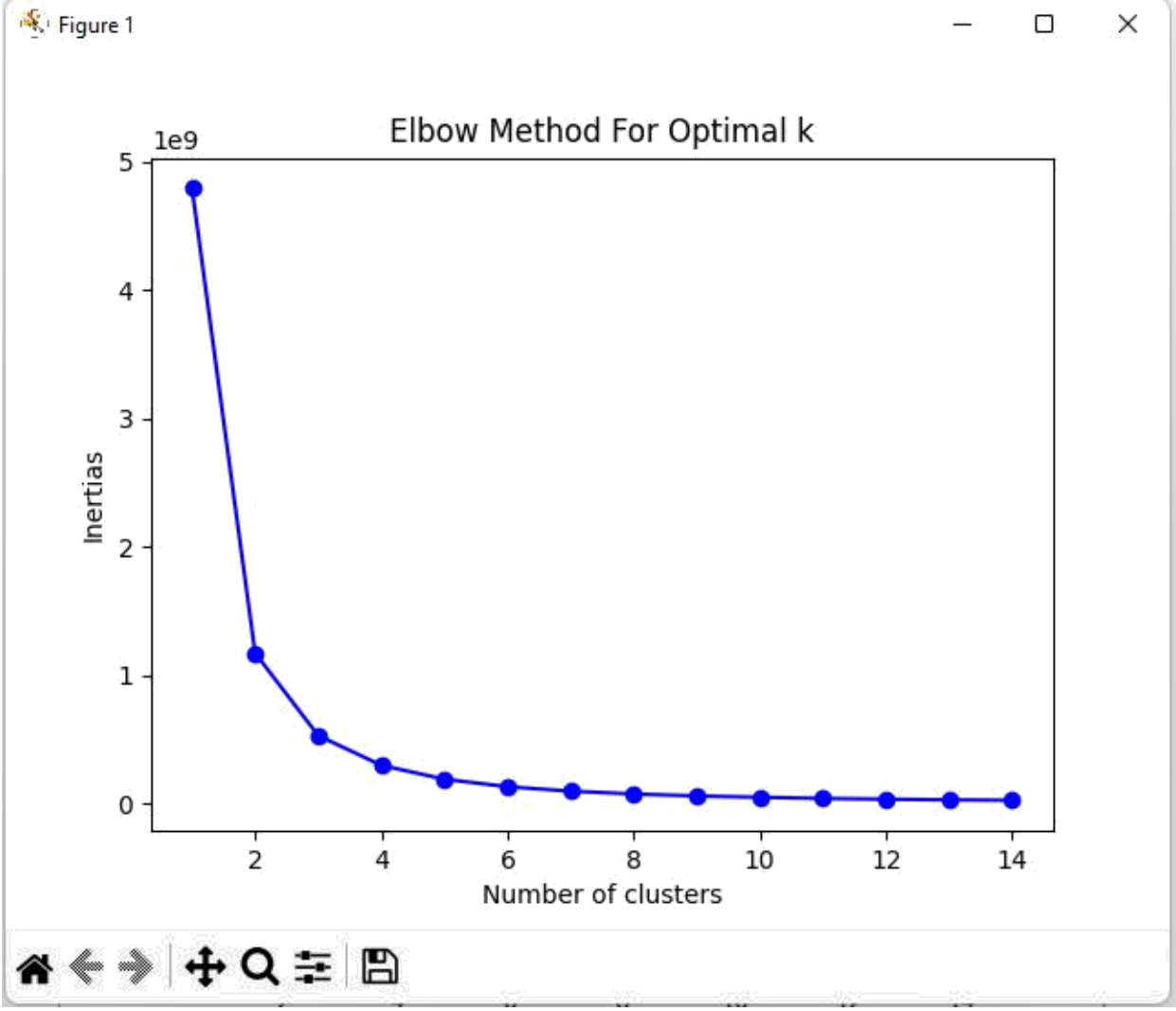
6-A "Mine Association Rules" button that, when clicked, runs a function called mine_association_rules and passes the file path entered in the text box as an argument.

7-When the GUI is launched, the mainloop function is called, which allows the GUI to run and respond to events (e.g., clicking buttons) until the user closes it.

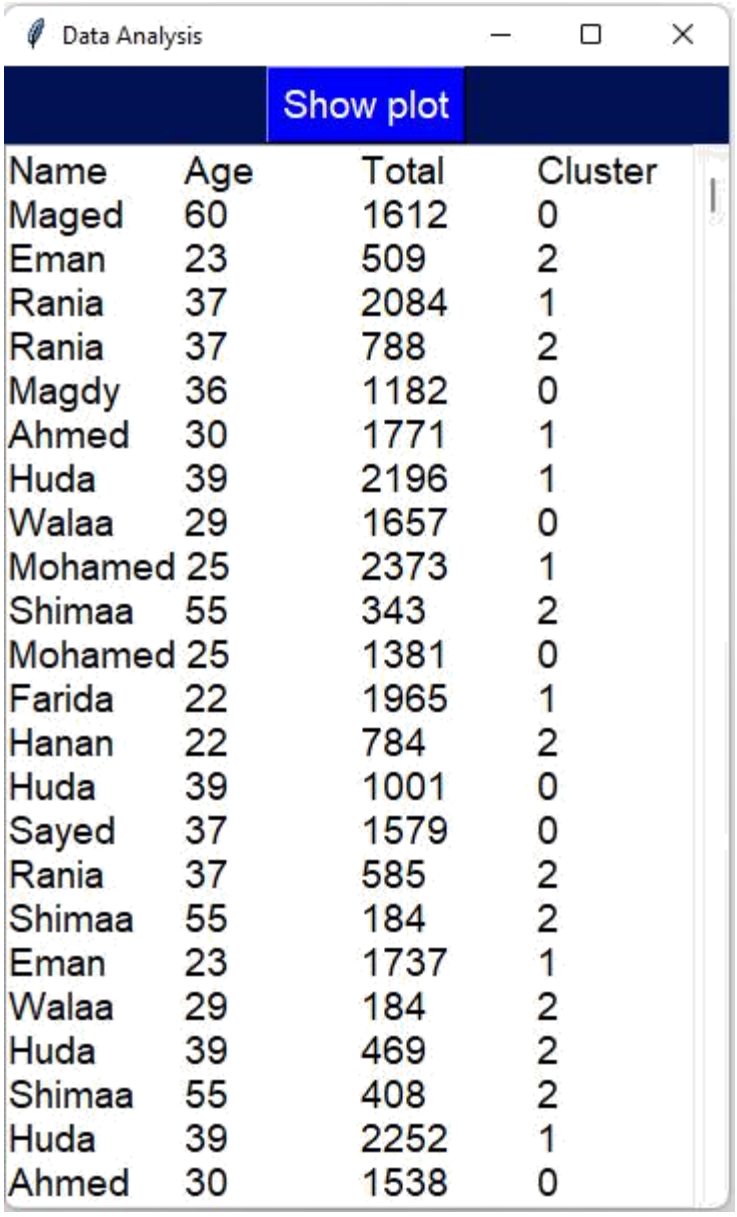
Expected output:



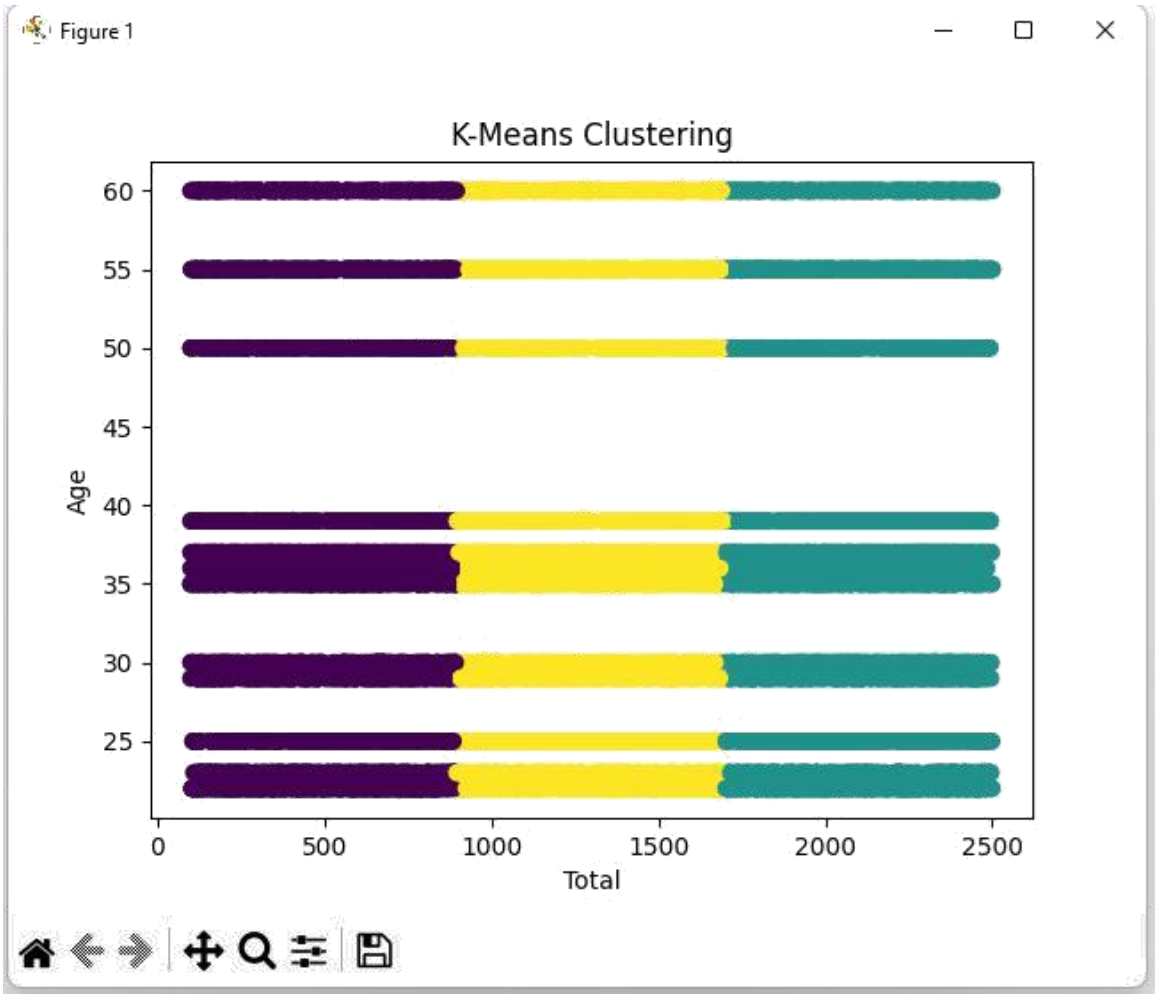
If you click on Optimal Clusters it should show this output:



If you input number of clusters and click on Optimal Clusters it should show this output:



If you click on show plot it should show this output:



If you click on mine association rules it should show this output:

If you input minimum support and minimum confidence it should show this output:

Association Rules

Left Hand Side	Right Hand Side	Support	Confidence	Lift
beef	root vegetables	0.017386883579054397	0.3313953488372093	3.0403668431100312
berries	yogurt	0.010574478901881037	0.3180428134556575	2.279847718904075
butter	root vegetables	0.012913065582104729	0.23302752293577983	2.137897097083391
chicken	other vegetables	0.017793594306049824	0.41567695961995255	2.1494126697488083
chicken	root vegetables	0.010879511947127605	0.25415676959619954	2.331746109121849
citrus fruit	tropical fruit	0.0199288256227758	0.2407862407862408	2.2969279128347995
cream cheese	yogurt	0.012404677173360447	0.3128205128205128	2.2424123495552064
curd	yogurt	0.01728520589730554	0.3244274809160305	2.325615360648076
domestic eggs	root vegetables	0.014336553126588714	0.22596153846153844	2.0730706443742823
frozen vegetables	root vegetables	0.011591255719369599	0.24101479915433405	2.211175885898205