Experiment 3

Source Code-

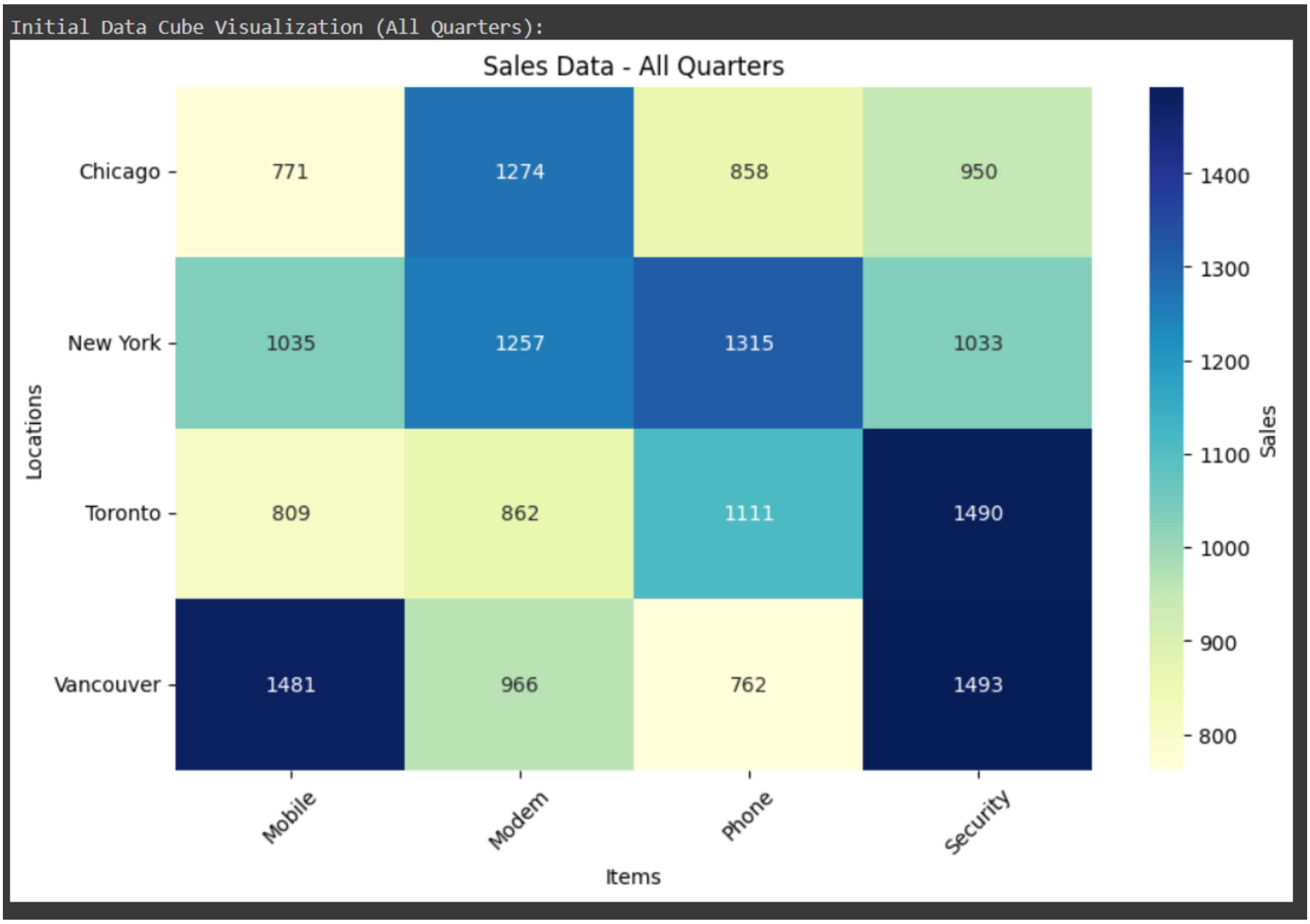
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Define the dimensions
locations = ['Chicago', 'New York', 'Toronto', 'Vancouver']
countries = ['USA', 'USA', 'Canada', 'Canada']  # Mapping for locations to countries
quarters = ['Q1', 'Q2', 'Q3', 'Q4']
months = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']
items = ['Mobile', 'Modem', 'Phone', 'Security']

# Create a 3D array to represent the data cube with integer sales numbers
# Random sales data for the sake of example
sales_cube = np.random.randint(100, 500, size=(len(locations), len(quarters), len(items)))

# Function to visualize 2D heatmaps
def visualize_2d_heatmap(data, x_labels, y_labels, title, annotate=False):
    plt.figure(figsize=(10, 6))
    sns.heatmap(data, annot=annotate, fmt="d", cmap="YlGnBu", cbar_kws={'label': 'Sales'})
    plt.xticks(ticks=np.arange(len(x_labels)) + 0.5, labels=x_labels, rotation=45)
    plt.yticks(ticks=np.arange(len(y_labels)) + 0.5, labels=y_labels, rotation=0)
    plt.title(title)
    plt.xlabel('Items')
    plt.ylabel('Locations')
    plt.show()

# Initial Cube Visualization (All Quarters)
```

```
print("Initial Data Cube Visualization (All Quarters):")
visualize_2d_heatmap(sales_cube.sum(axis=1).astype(int), items, locations, title="Sales Data
- All Quarters", annotate=True)

# Roll-Up: Aggregating from cities to countries
# Aggregate by countries (USA and Canada)
unique_countries = list(set(countries))
rollup_cube = np.zeros((len(unique_countries), len(items)))

for loc_index, location in enumerate(locations):
    country_index = unique_countries.index(countries[loc_index])
    rollup_cube[country_index] += sales_cube[loc_index].sum(axis=0)

# Convert rollup_cube to integers
rollup_cube = rollup_cube.astype(int)

# Visualize Cube after Roll-Up (Countries vs. Items for All Quarters)
print("\nData Cube after Roll-Up (Countries vs. Items for All Quarters):")
visualize_2d_heatmap(rollup_cube.astype(int), items, unique_countries, title="Sales Data
after Roll-Up - All Quarters", annotate=True)

# Drill-Down: Changing time from quarters to months
# Let's assume each quarter consists of 3 months
sales_months_cube = np.zeros((len(locations), len(months), len(items)))

# Fill the data based on quarters to months
for loc_index in range(len(locations)):
    for quarter_index in range(len(quarters)):
        for month_index in range(3):  # Each quarter has 3 months
            sales_months_cube[loc_index, quarter_index * 3 + month_index] =
sales_cube[loc_index, quarter_index]
```

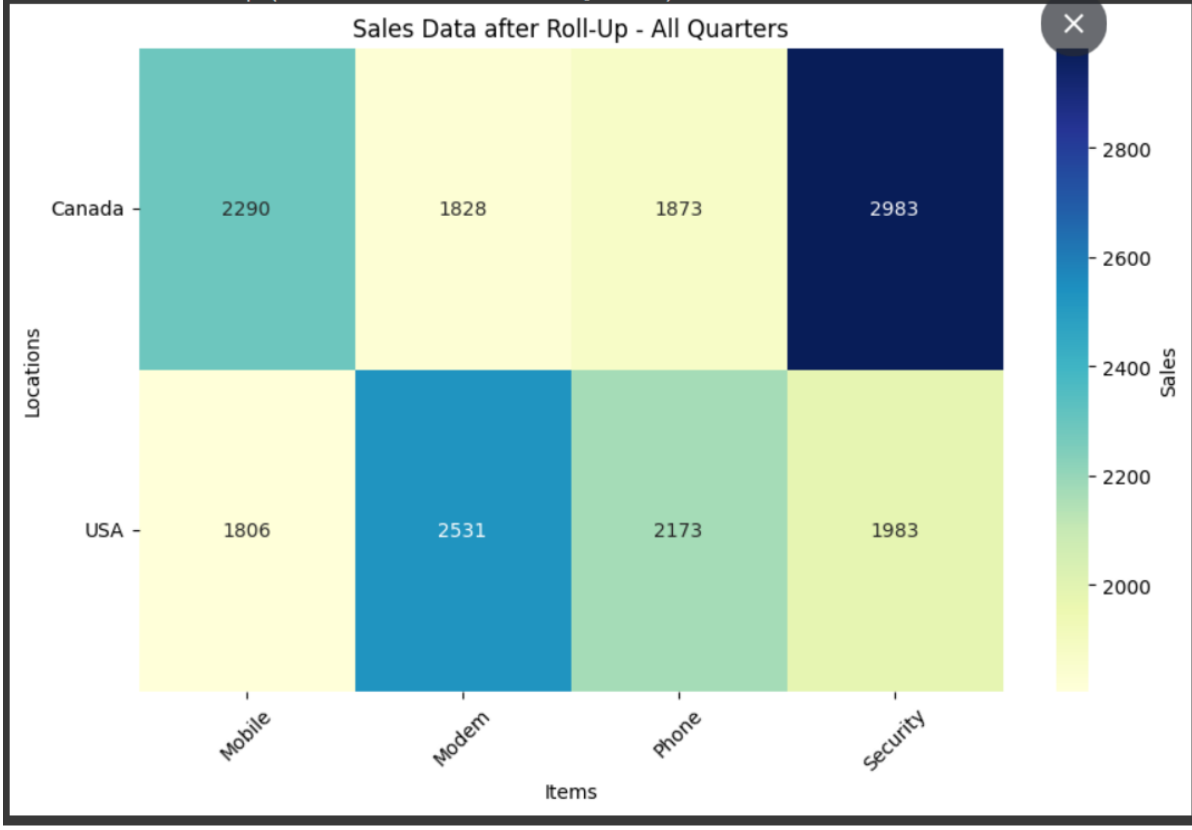
```
# Now we will drill down, removing the quarters and visualizing with months
# Aggregate monthly sales for each location and item
drill_down_cube = sales_months_cube.sum(axis=0).astype(int)  # Sum over the locations for each month

# Visualize Cube after Drill-Down (All Months)
print("\nData Cube after Drill-Down (All Months):")
visualize_2d_heatmap(drill_down_cube, items, months, title="Sales Data after Drill-Down - All Months", annotate=True)

# Slice: Cutting down the time dimension for all Q1
slice_op = sales_cube[:, 0, :]  # Slice for Q1 only (index 0 for Q1)

# Visualize Cube after Slice (for Q1 only)
print("\nData Cube after Slice (Q1 only):")
visualize_2d_heatmap(slice_op.astype(int), items, locations, title="Sales Data after Slice - Q1", annotate=True)

# Dice: Selecting criteria (Location = Toronto or Vancouver, Time = Q1 or Q2, Item = Mobile or Modem)
dice_locations_indices = [2, 3]  # Toronto and Vancouver
dice_quarters_indices = [0, 1]    # Q1 and Q2
dice_items_indices = [0, 1]       # Mobile and Modem

# Create the resulting cube after dice operation
dice_op = sales_cube[np.ix_(dice_locations_indices, dice_quarters_indices, dice_items_indices)]

# Visualize Cube after Dice
# We will visualize the sum across the quarter dimension for simplicity.
dice_visualization = dice_op.sum(axis=1).astype(int)  # Summing over the quarter dimension
```

```
# Now visualize the filtered data after dice operation
print("\nData Cube after Dice (Toronto & Vancouver, Q1 & Q2, Mobile & Modem):")
visualize_2d_heatmap(dice_visualization.astype(int), ['Mobile', 'Modem'], ['Toronto',
'Vancouver'], title="Sales Data after Dice (Filtered Data)", annotate=True)
```

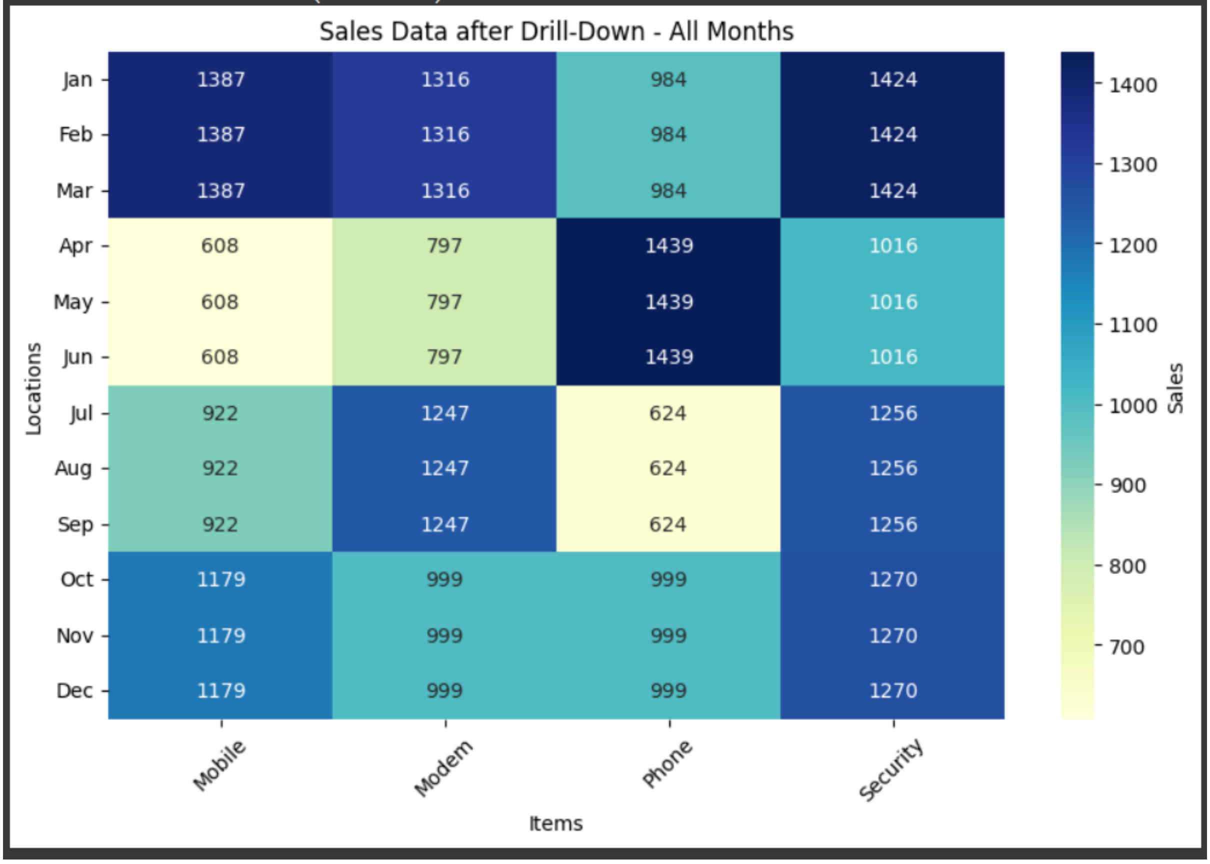
Output-



Data Cube after Roll-Up (Countries vs. Items for All Quarters):

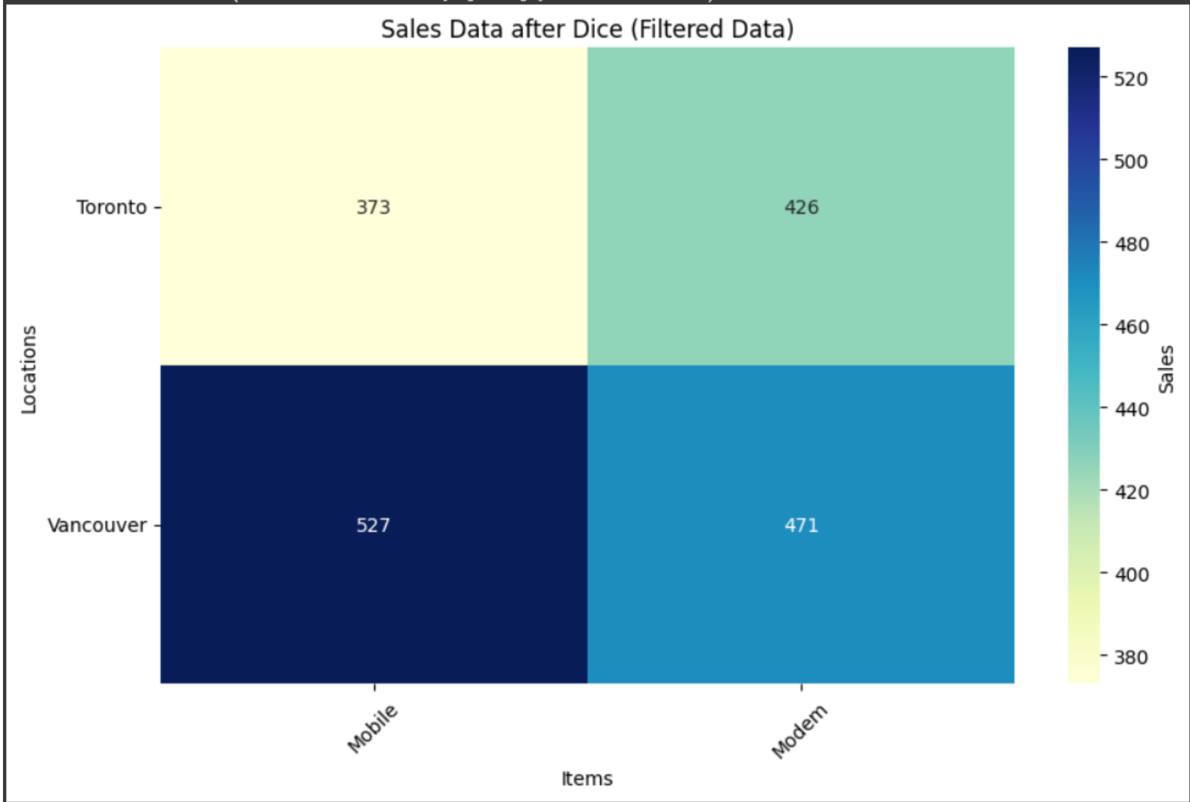


Data Cube after Drill-Down (All Months):



Data Cube after Slice (Q1 only):

Data Cube after Dice (Toronto & Vancouver, Q1 & Q2, Mobile & Modem):



Experiment 4

1. **Data Exploration: Compute Mean, Median, Mode, and Five-Number Summary**

Source Code-

```
import numpy as np
import statistics as stats

# Data for exploration
data = [13, 15, 16, 19, 20, 20, 21, 22, 22, 25, 25, 25, 25, 30, 35, 35, 35]

# Mean
mean = np.mean(data)
# Median
median = np.median(data)
# Mode
mode = stats.mode(data)

# Five-Number Summary
minimum = np.min(data)
maximum = np.max(data)
q1 = np.percentile(data, 25)
q3 = np.percentile(data, 75)
iqr = q3 - q1

print("Data Exploration:")
print(f"Mean: {mean}")
print(f"Median: {median}")
print(f"Mode: {mode}")
```

print(f"Minimum: {minimum}")

print(f"Q1: {q1}")

print(f"Q3: {q3}")

print(f"Maximum: {maximum}")

print(f"Interquartile Range (IQR): {iqr}")

Output-

```
Data Exploration:
Mean: 23.705882352941178
Median: 22.0
Mode: 25
Minimum: 13
Q1: 20.0
Q3: 25.0
Maximum: 35
Interquartile Range (IQR): 5.0
```

2.Data Preprocessing: Smoothing Using Binning Data: [8, 16, 9, 15, 21, 21, 24, 30, 26, 27, 30, 34]

A.Bin Mean:

Source Code-

```
# Data for Binning
data = [8, 16, 9, 15, 21, 21, 24, 30, 26, 27, 30, 34]

# Create bins (equal-width binning, with 3 bins)
# bins = [data[i:i+4] for i in range(0, len(data), 4)]
bins = []
for i in range (0, len(data), 4):
    bins.append(data[i:i+4])
print(f"Bins:{bins}")
# Calculate the mean of each bin
# Bin Mean
# bin_mean = [int(np.mean(bin)) for bin in bins]
bin_mean = []
for bin in bins:
    bin_mean.append(int(np.mean(bin)))
print(f"Bin Mean:{bin_mean}")

# Replace values in each bin by the mean of that bin
bin_mean_result = []
for bin, mean in zip(bins, bin_mean):
    bin_mean_result.extend([mean] * len(bin))

print("\nBinning by Mean:")
print(bin_mean_result)
```

Output-

```
Bins:[[8, 16, 9, 15], [21, 21, 24, 30], [26, 27, 30, 34]]
Bin Mean:[12, 24, 29]

Binning by Mean:
[12, 12, 12, 12, 24, 24, 24, 24, 29, 29, 29, 29]
```

B.Bin Boundaries:

Source Code-

```
# Bin Boundaries
bin_boundary_result = []
for bin in bins:
    min_boundary = min(bin)
    max_boundary = max(bin)
    boundary_bin = [min_boundary if abs(x - min_boundary) < abs(x - max_boundary) else max_boundary for x in bin]
    bin_boundary_result.extend(boundary_bin)


print("\nBinning by Boundaries:")
print(bin_boundary_result)
```

Output-

```
Binning by Boundaries:
[8, 16, 8, 16, 21, 21, 21, 30, 26, 26, 34, 34]
```

C.Bin Median:

Source Code-

```
# Bin Median
bin_median = [int(np.median(bin)) for bin in bins]

# Replace values in each bin by the median of that bin
bin_median_result = []
for bin, median in zip(bins, bin_median):
    bin_median_result.extend([median] * len(bin))

print("\nBinning by Median:")
print(bin_median_result)
```

Output-

```
Binning by Median:
[12, 12, 12, 12, 22, 22, 22, 22, 28, 28, 28, 28]
```

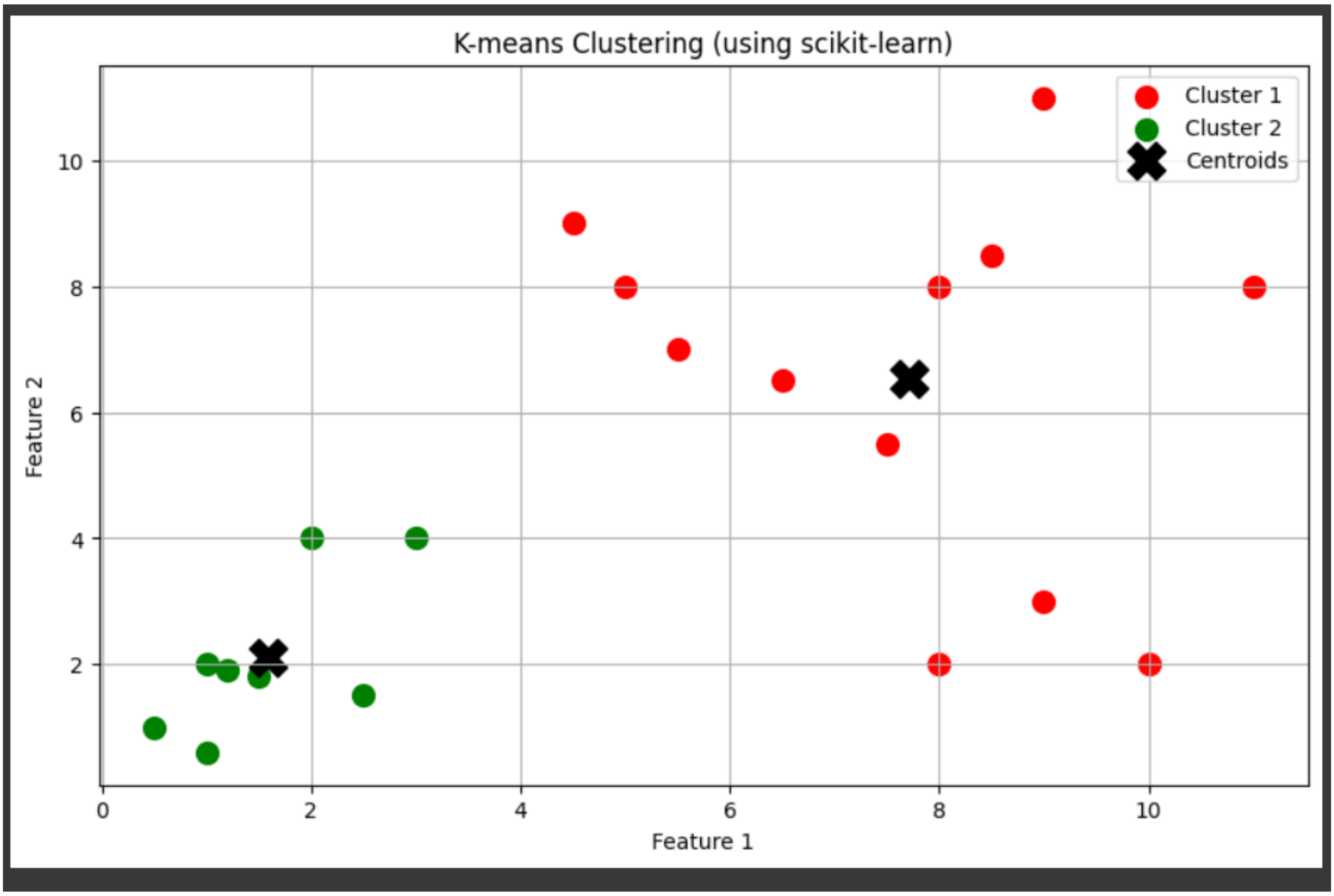
Experiment 6

Source Code-

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

# Step 4: Hardcoded dataset with more data points
# Example data points (more diverse)
X = np.array([
    [1.0, 2.0],
    [1.5, 1.8],
    [5.0, 8.0],
    [8.0, 8.0],
    [1.0, 0.6],
    [9.0, 11.0],
    [8.0, 2.0],
    [10.0, 2.0],
    [9.0, 3.0],
    [1.2, 1.9],
    [5.5, 7.0],
    [7.5, 5.5],
    [2.0, 4.0],
    [6.5, 6.5],
    [2.5, 1.5],
    [3.0, 4.0],
    [4.5, 9.0],
    [8.5, 8.5],
    [0.5, 1.0],
    [11.0, 8.0]
```

```
])

# Parameters
n_clusters = 2  # Set the number of clusters

# Apply K-means clustering using scikit-learn
kmeans = KMeans(n_clusters=n_clusters, random_state=42)
kmeans.fit(X)
labels = kmeans.labels_
centroids = kmeans.cluster_centers_

# Step 5: Visualize the results
plt.figure(figsize=(10, 6))
colors = ['r', 'g', 'b', 'y']  # Colors for clusters

# Plot each cluster
for i in range(n_clusters):
    plt.scatter(X[labels == i, 0], X[labels == i, 1], s=100, c=colors[i], label=f'Cluster {i + 1}')

# Plot centroids
plt.scatter(centroids[:, 0], centroids[:, 1], s=300, c='black', marker='X', label='Centroids')
plt.title('K-means Clustering (using scikit-learn)')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.grid()
plt.show()
```

Output-



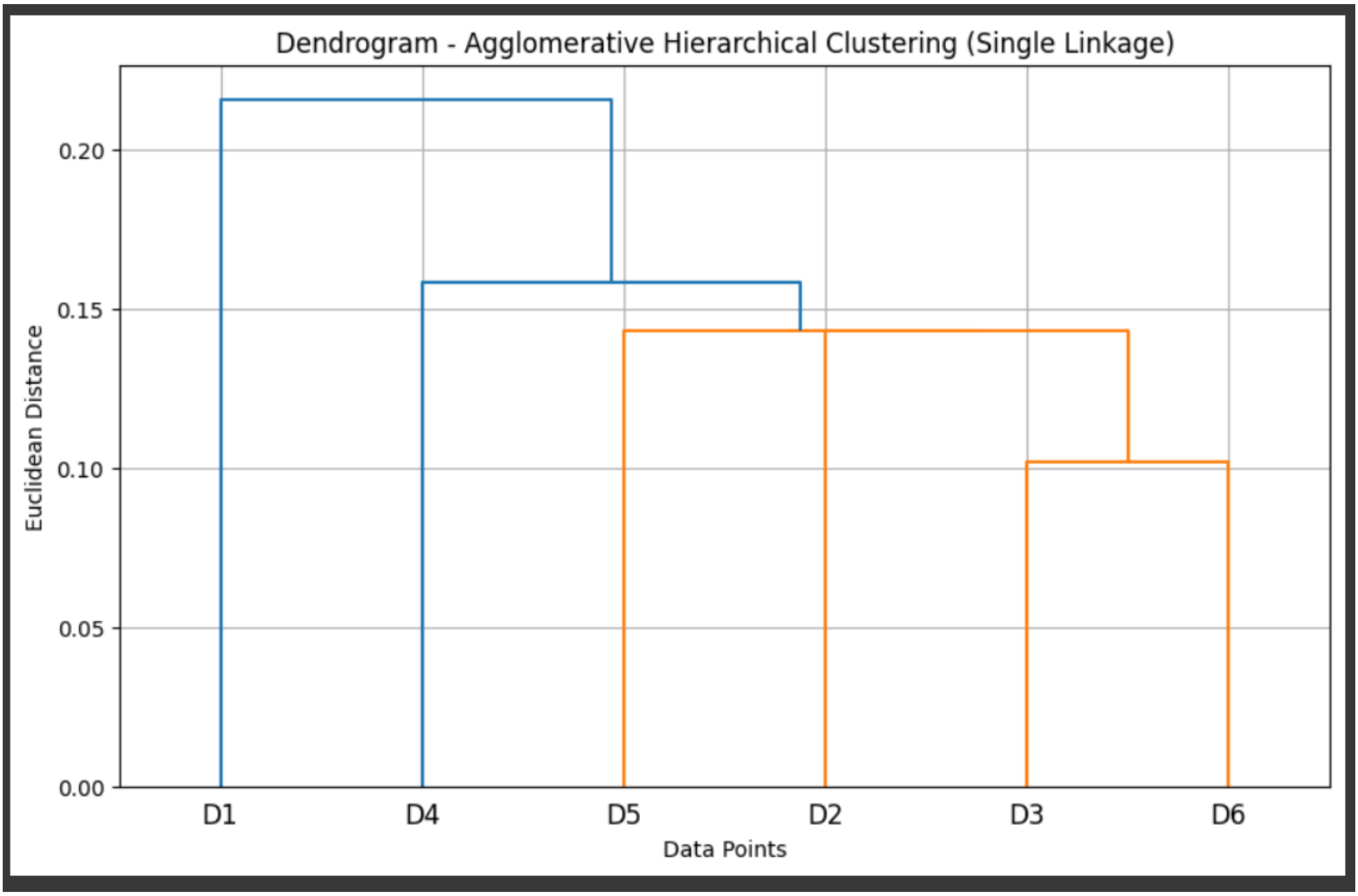
K-means Clustering (using scikit-learn)
Cluster 1
Cluster 2
Centroids
Feature 1
Feature 2
0
2
4
6
8
10

Experiment 7

Source Code-

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage

# Step 1: Define the data points
data_points = np.array([
    [0.4, 0.53],  # D1
    [0.22, 0.38], # D2
    [0.35, 0.32], # D3
    [0.26, 0.19], # D4
    [0.08, 0.41], # D5
    [0.45, 0.30]  # D6
])

# Step 2: Perform Agglomerative Hierarchical Clustering using Single Linkage
Z = linkage(data_points, method='single')

# Step 3: Plot the Dendrogram
plt.figure(figsize=(10, 6))
dendrogram(Z, labels=[f'D{i+1}' for i in range(len(data_points))])
plt.title('Dendrogram - Agglomerative Hierarchical Clustering (Single Linkage)')
plt.xlabel('Data Points')
plt.ylabel('Euclidean Distance')
plt.grid()
plt.show()
```

Output-



Dendrogram - Agglomerative Hierarchical Clustering (Single Linkage)
Euclidean Distance
0.20
0.15
0.10
0.05
0.00
D1
D4
D5
D2
D3
D6
Data Points

Experiment 8

Source Code-

```
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori, association_rules

# Sample transaction data (same as before)
data = {
    'TransactionID': [1, 2, 3, 4, 5, 6, 7, 8],
    'Items': [
        ['Milk', 'Bread', 'Diaper'],
        ['Milk', 'Bread'],
        ['Bread', 'Diaper', 'Eggs'],
        ['Milk', 'Diaper', 'Eggs'],
        ['Bread', 'Diaper'],
        ['Milk', 'Bread', 'Diaper', 'Eggs'],
        ['Bread'],
        ['Milk', 'Bread', 'Diaper', 'Eggs', 'Cola']
    ]
}

# Create a DataFrame
transactions = pd.DataFrame(data)

# Transform transactions into a one-hot encoded DataFrame
te = TransactionEncoder()
te_ary = te.fit(transactions['Items']).transform(transactions['Items'])
df = pd.DataFrame(te_ary, columns=te.columns_)
```

```
# Apply Apriori algorithm with mlxtend
frequent_itemsets = apriori(df, min_support=0.3, use_colnames=True)

# Generate association rules
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.6)

# Print frequent itemsets and association rules
print("Frequent Itemsets:")
print(frequent_itemsets)
print("\nAssociation Rules:")
print(rules)
```

Output-

```
Frequent Itemsets:
    support              itemsets
0     0.875               (Bread)
1     0.750              (Diaper)
2     0.500                (Eggs)
3     0.625                (Milk)
4     0.625       (Diaper, Bread)
5     0.375         (Eggs, Bread)
6     0.500         (Milk, Bread)
7     0.500        (Eggs, Diaper)
8     0.500        (Milk, Diaper)
9     0.375          (Eggs, Milk)
10    0.375 (Eggs, Diaper, Bread)
11    0.375 (Milk, Diaper, Bread)
12    0.375  (Eggs, Milk, Diaper)

Association Rules:
        antecedents      consequents  antecedent support  consequent support  \
0          (Diaper)          (Bread)               0.750               0.875
1           (Bread)         (Diaper)               0.875               0.750
2            (Eggs)          (Bread)               0.500               0.875
3            (Milk)          (Bread)               0.625               0.875
4            (Eggs)         (Diaper)               0.500               0.750
5          (Diaper)           (Eggs)               0.750               0.500
6            (Milk)         (Diaper)               0.625               0.750
7          (Diaper)           (Milk)               0.750               0.625
8            (Eggs)           (Milk)               0.500               0.625
9            (Milk)           (Eggs)               0.625               0.500
10   (Eggs, Diaper)          (Bread)               0.500               0.875
11    (Eggs, Bread)         (Diaper)               0.375               0.750
12  (Diaper, Bread)           (Eggs)               0.625               0.500
13           (Eggs)  (Diaper, Bread)               0.500               0.625
14   (Milk, Diaper)          (Bread)               0.500               0.875
15    (Milk, Bread)         (Diaper)               0.500               0.750
16  (Diaper, Bread)           (Milk)               0.625               0.625
17           (Milk)  (Diaper, Bread)               0.625               0.625
18     (Eggs, Milk)         (Diaper)               0.375               0.750
19   (Eggs, Diaper)           (Milk)               0.500               0.625
20   (Milk, Diaper)           (Eggs)               0.500               0.500
21           (Eggs)   (Milk, Diaper)               0.500               0.500
22           (Milk)   (Eggs, Diaper)               0.625               0.500

    support  confidence      lift  leverage  conviction  zhangs_metric
0     0.625    0.833333  0.952381 -0.031250      0.7500      -0.166667
1     0.625    0.714286  0.952381 -0.031250      0.8750      -0.285714
2     0.375    0.750000  0.857143 -0.062500      0.5000      -0.250000
3     0.500    0.800000  0.914286 -0.046875      0.6250      -0.200000
4     0.500    1.000000  1.333333  0.125000         inf       0.500000
5     0.500    0.666667  1.333333  0.125000      1.5000       1.000000
6     0.500    0.800000  1.066667  0.031250      1.2500       0.166667
7     0.500    0.666667  1.066667  0.031250      1.1250       0.250000
8     0.375    0.750000  1.200000  0.062500      1.5000       0.333333
9     0.375    0.600000  1.200000  0.062500      1.2500       0.444444
10    0.375    0.750000  0.857143 -0.062500      0.5000      -0.250000
11    0.375    1.000000  1.333333  0.093750         inf       0.400000
12    0.375    0.600000  1.200000  0.062500      1.2500       0.444444
13    0.375    0.750000  1.200000  0.062500      1.5000       0.333333
14    0.375    0.750000  0.857143 -0.062500      0.5000      -0.250000
15    0.375    0.750000  1.000000  0.000000      1.0000       0.000000
16    0.375    0.600000  0.960000 -0.015625      0.9375      -0.100000
17    0.375    0.600000  0.960000 -0.015625      0.9375      -0.100000
18    0.375    1.000000  1.333333  0.093750         inf       0.400000
19    0.375    0.750000  1.200000  0.062500      1.5000       0.333333
20    0.375    0.750000  1.500000  0.125000      2.0000       0.666667
21    0.375    0.750000  1.500000  0.125000      2.0000       0.666667
22    0.375    0.600000  1.200000  0.062500      1.2500       0.444444
```

Experimewnt 9

Source Code-

```
import numpy as np
from fractions import Fraction

def display_format(my_vector, my_decimal):
    return np.round((my_vector).astype(float), decimals=my_decimal)

my_dp = Fraction(1, 3)  # Define the equal probability

# Adjacency matrix (links between pages)
Mat = np.matrix([[Fraction(1, 3), Fraction(1, 2), 0],          # Page A links to Page B and C
                 [Fraction(1, 3), 0, Fraction(1, 2)],  # Page B links to Page A and C
                 [Fraction(1, 3), Fraction(1, 2), Fraction(1, 2)]]) # Page C links to A, B, and itself

# Matrix of equal probabilities for teleportation (stochastic jump)
Ex = np.zeros((3, 3))
Ex[:] = my_dp  # Each element has a value of 1/3 (equal probability of moving to any page)

beta = 0.7  # Damping factor (70% follows links, 30% random jump)

# Transition matrix combining both the teleportation and link-following behavior
Al = beta * Mat + ((1 - beta) * Ex)

# Initial rank vector (equal probability for all 3 pages)
r = np.matrix([my_dp, my_dp, my_dp])
r = np.transpose(r)

previous_r = r
```

```
# Iterate until convergence or up to 4 iterations
for i in range(1, 4):
    r = Al * r  # Update the rank vector
    print(f"Iteration {i}:")
    print(display_format(r, 3))

    # Check if the rank vector has converged (difference is negligible)
    if (previous_r == r).all():
        print("Converged!")
        break

    previous_r = r  # Update for the next iteration

# Final result
print("\nFinal PageRank Vector:\n", display_format(r, 3))
print("Sum of ranks:", np.sum(r))

# Find the highest rank page
highest_rank_index = np.argmax(r)  # Index of the page with the highest rank
highest_rank_value = r[highest_rank_index, 0]  # Get the highest rank value

# Map index to page names
pages = ['Page A', 'Page B', 'Page C']  # Corresponding page names
highest_page = pages[highest_rank_index]  # Name of the highest rank page

print(f"The highest rank page is: {highest_page} with a rank of {highest_rank_value:.3f}")
```

Output-

```
[[Fraction(1, 3) Fraction(1, 2) 0]
 [Fraction(1, 3) 0 Fraction(1, 2)]
 [Fraction(1, 3) Fraction(1, 2) Fraction(1, 2)]]
Iteration 1:
[[0.294]
 [0.294]
 [0.411]]
Iteration 2:
[[0.272]
 [0.313]
 [0.416]]
Iteration 3:
[[0.273]
 [0.309]
 [0.418]]

Final PageRank Vector:
 [[0.273]
 [0.309]
 [0.418]]
Sum of ranks: 0.9999999999999998
The highest rank page is: Page C with a rank of 0.418
```