week10

October 11, 2024

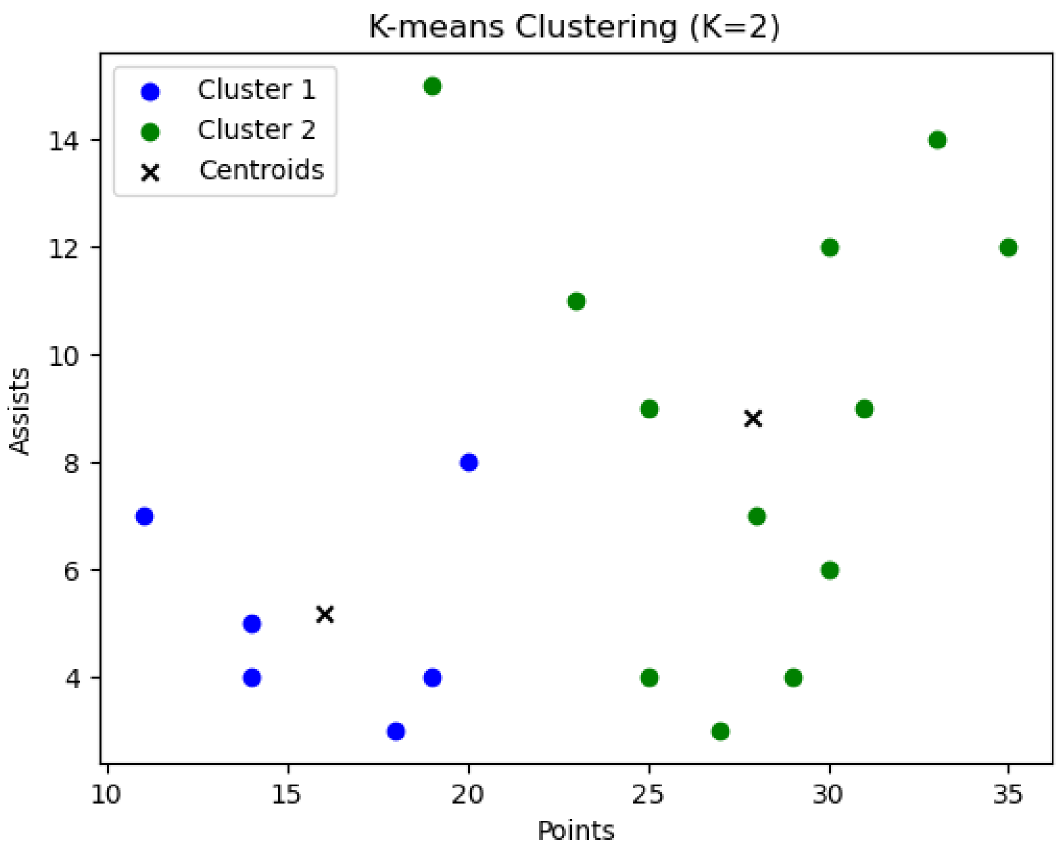
[1]:
```
import pandas as pd

data = {
    'Points': [18.0, 19.0, 14.0, 14.0, 11.0, 20.0, 28.0, 30.0, 31.0, 35.0, 33.
 ↪0, 25.0, 25.0, 27.0, 29.0, 30.0, 19.0, 23.0],
    'Assists': [3.0, 4.0, 5.0, 4.0, 7.0, 8.0, 7.0, 6.0, 9.0, 12.0, 14.0, 9.0, 4.
 ↪0, 3.0, 4.0, 12.0, 15.0, 11.0],
    'Rebounds': [15, 14, 10, 8, 14, 13, 9, 5, 4, 11, 6, 5, 3, 8, 12, 7, 6, 5]
}

df = pd.DataFrame(data)
print(df)
```

```
    Points  Assists  Rebounds
0     18.0      3.0        15
1     19.0      4.0        14
2     14.0      5.0        10
3     14.0      4.0         8
4     11.0      7.0        14
5     20.0      8.0        13
6     28.0      7.0         9
7     30.0      6.0         5
8     31.0      9.0         4
9     35.0     12.0        11
10    33.0     14.0         6
11    25.0      9.0         5
12    25.0      4.0         3
13    27.0      3.0         8
14    29.0      4.0        12
15    30.0     12.0         7
16    19.0     15.0         6
17    23.0     11.0         5
```

[2]:
```
import numpy as np
import matplotlib.pyplot as plt

# Function to calculate Euclidean distance
def euclidean_distance(a, b):
```

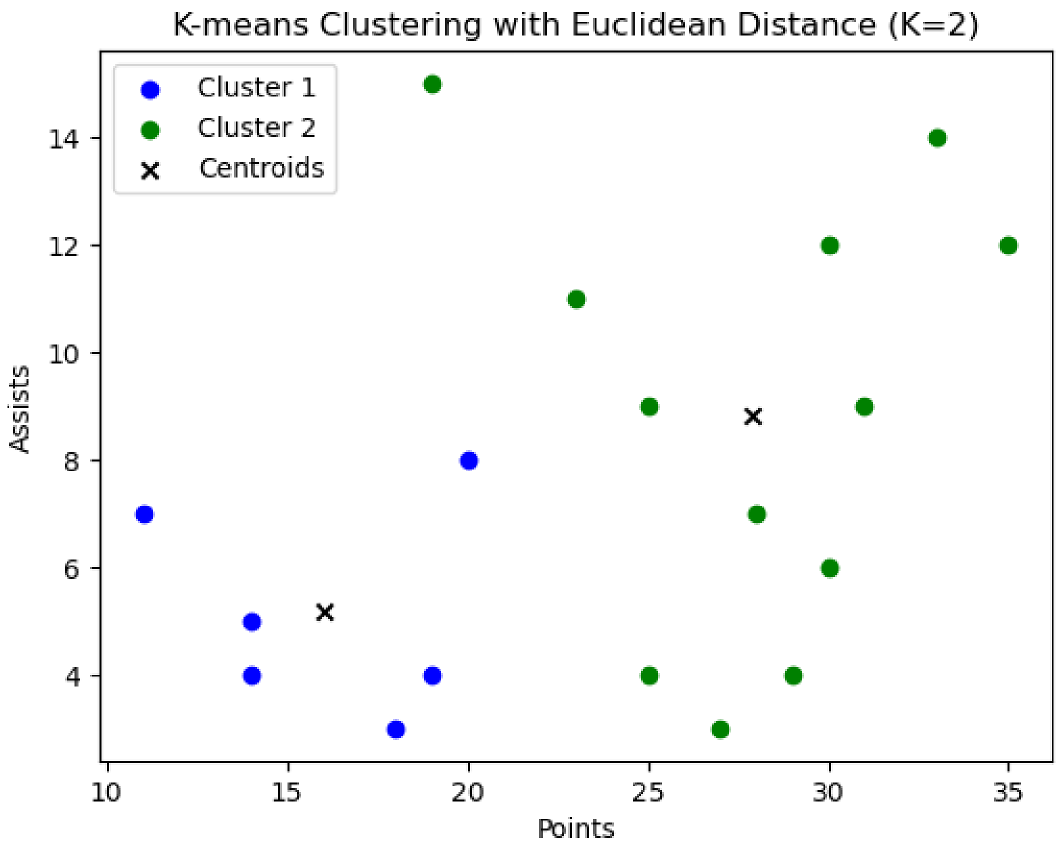
```
    return np.sqrt(np.sum((a - b) ** 2))

# K-means clustering algorithm
def kmeans_manual(df, k=2, iterations=10):
    # Randomly initialize centroids
    np.random.seed(42)
    centroids = df.sample(n=k).values

    for _ in range(iterations):
        clusters = {}

        # Assign each point to the nearest centroid
        for idx, row in df.iterrows():
            distances = [euclidean_distance(row.values, centroid) for centroid␣
↪in centroids]
            cluster = np.argmin(distances)

            if cluster not in clusters:
                clusters[cluster] = []
            clusters[cluster].append(row.values)

        # Update centroids
        new_centroids = []
        for cluster in clusters:
            new_centroid = np.mean(clusters[cluster], axis=0)
            new_centroids.append(new_centroid)

        centroids = new_centroids

    return clusters, centroids

# Perform K-means clustering with K=2
clusters, centroids = kmeans_manual(df, k=2, iterations=10)

# Function to plot clusters
def plot_clusters(df, clusters, centroids):
    colors = ['blue', 'green', 'red', 'purple']

    for cluster_idx, points in clusters.items():
        points = np.array(points)
        plt.scatter(points[:, 0], points[:, 1], color=colors[cluster_idx],␣
↪label=f'Cluster {cluster_idx+1}')

    centroids = np.array(centroids)
    plt.scatter(centroids[:, 0], centroids[:, 1], color='black', marker='x',␣
↪label='Centroids')
    plt.xlabel('Points')
```

```
    plt.ylabel('Assists')
    plt.legend()
    plt.title('K-means Clustering (K=2)')
    plt.show()

# Plot the results
plot_clusters(df, clusters, centroids)
```



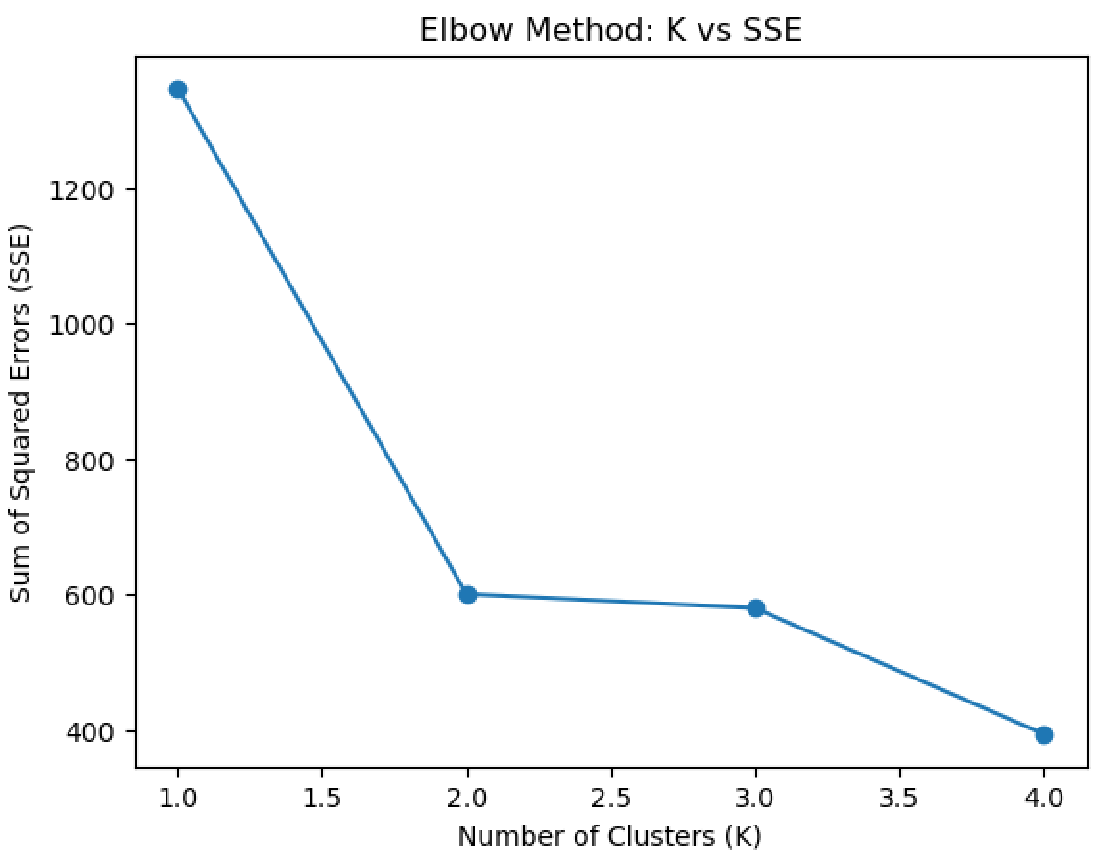
[3]:
```
def plot_clusters(df, clusters, centroids, title):
    colors = ['blue', 'green', 'red', 'purple']

    for cluster_idx, points in clusters.items():
        points = np.array(points)
        plt.scatter(points[:, 0], points[:, 1], color=colors[cluster_idx %␣
↪len(colors)], label=f'Cluster {cluster_idx+1}')

    centroids = np.array(centroids)
    plt.scatter(centroids[:, 0], centroids[:, 1], color='black', marker='x',␣
↪label='Centroids')
```

```
    plt.xlabel('Points')
    plt.ylabel('Assists')
    plt.legend()
    plt.title(title)
    plt.show()

# Example usage for K=2 (you can change K for different plots):
clusters, centroids = kmeans_manual(df, k=2, iterations=10)
plot_clusters(df, clusters, centroids, title="K-means Clustering with Euclidean␣
↪Distance (K=2)")
```



[4]:
```
# SSE calculation function
def calculate_sse(clusters, centroids, distance_func):
    sse = 0
    for cluster_idx, points in clusters.items():
        centroid = centroids[cluster_idx]
        for point in points:
            sse += distance_func(point, centroid) ** 2
    return sse
```

[6]:
```python
def elbow_method(df, max_k=4, distance_func=euclidean_distance):
    sse_values = []
    k_values = range(1, max_k + 1)

    for k in k_values:
        clusters, centroids = kmeans_manual(df, k=k, iterations=10)  # Use␣
↪relevant distance function
        sse = calculate_sse(clusters, centroids, distance_func)
        sse_values.append(sse)

    # Plotting K vs SSE
    plt.plot(k_values, sse_values, marker='o')
    plt.xlabel('Number of Clusters (K)')
    plt.ylabel('Sum of Squared Errors (SSE)')
    plt.title('Elbow Method: K vs SSE')
    plt.show()

    return sse_values

# Example usage with Manhattan distance:
elbow_method(df, max_k=4, distance_func=euclidean_distance)
```

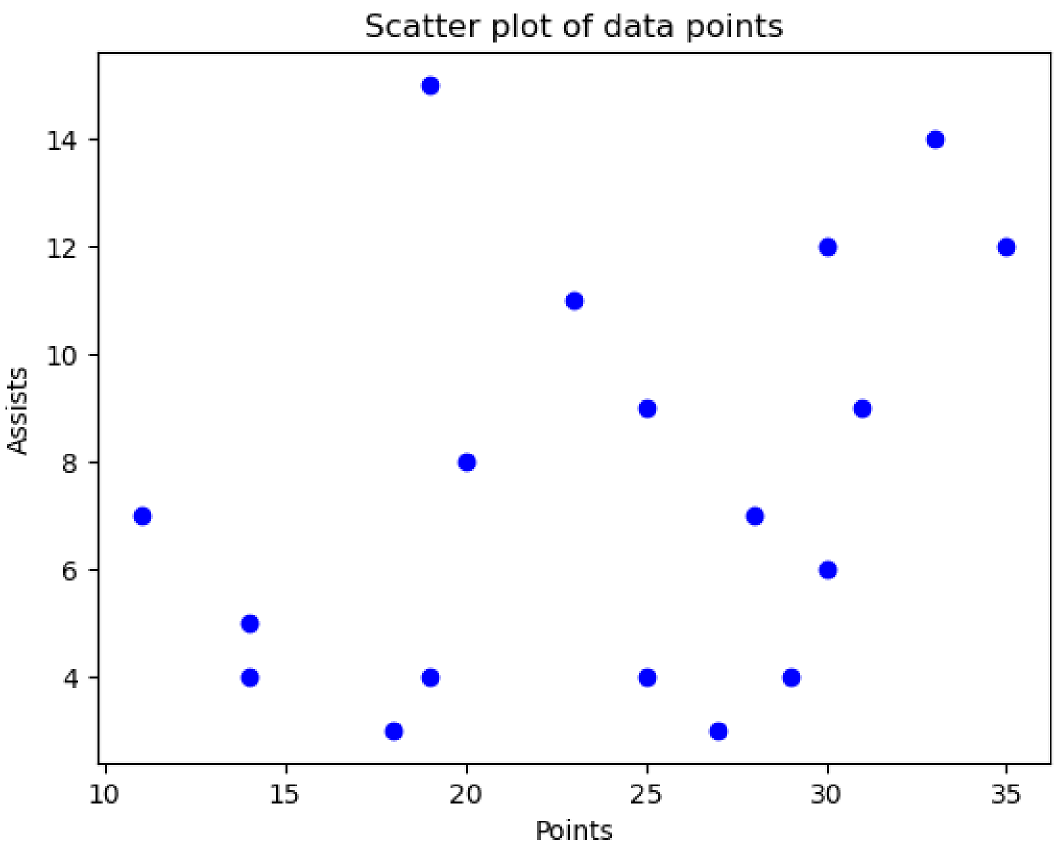


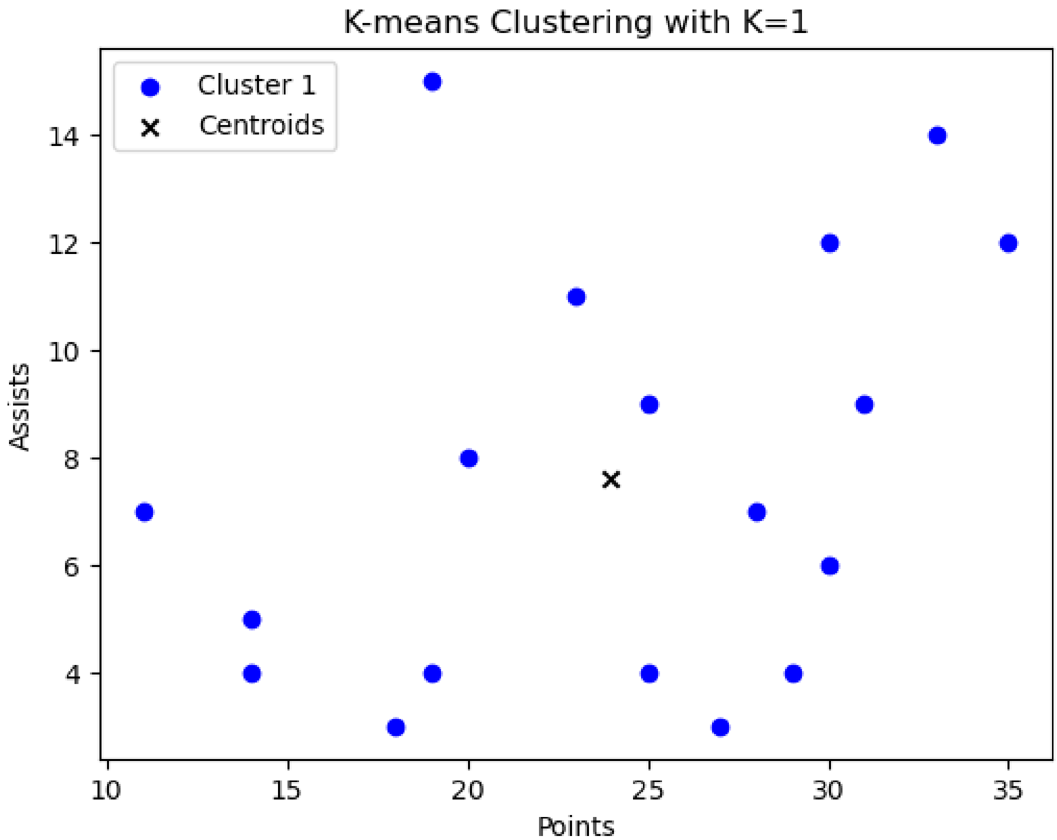
[6]: [1347.5, 601.0, 580.0833333333333, 394.1]

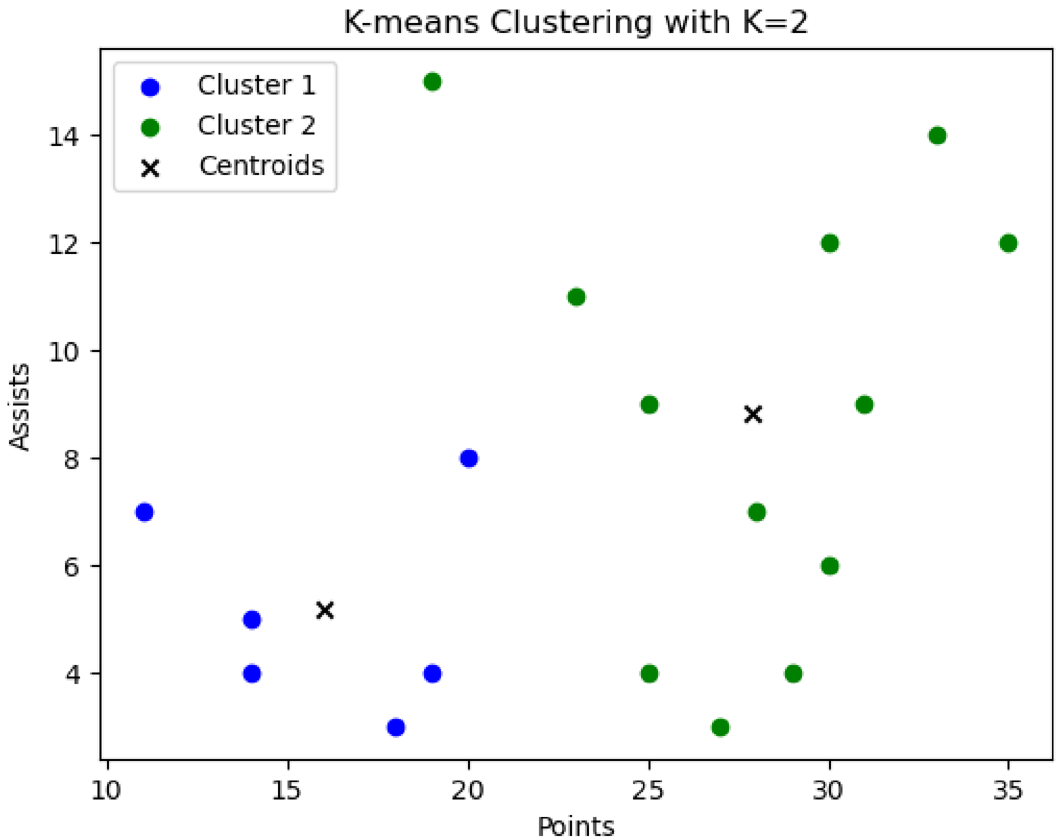
[7]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

def create_data():
    data = {
        'Points': [18.0, 19.0, 14.0, 14.0, 11.0, 20.0, 28.0, 30.0, 31.0, 35.0,␣
↪33.0, 25.0, 25.0, 27.0, 29.0, 30.0, 19.0, 23.0],
        'Assists': [3.0, 4.0, 5.0, 4.0, 7.0, 8.0, 7.0, 6.0, 9.0, 12.0, 14.0, 9.
↪0, 4.0, 3.0, 4.0, 12.0, 15.0, 11.0],
        'Rebounds': [15, 14, 10, 8, 14, 13, 9, 5, 4, 11, 6, 5, 3, 8, 12, 7, 6,␣
↪5]
    }
    df = pd.DataFrame(data)
    return df

def euclidean_distance(a, b):
    return np.sqrt(np.sum((a - b) ** 2))

def kmeans_manual(df, k=2, iterations=10):
    np.random.seed(42)
    centroids = df.sample(n=k).values

    for _ in range(iterations):
        clusters = {}

        for idx, row in df.iterrows():
            distances = [euclidean_distance(row.values, centroid) for centroid␣
↪in centroids]
            cluster = np.argmin(distances)

            if cluster not in clusters:
                clusters[cluster] = []
            clusters[cluster].append(row.values)

        new_centroids = []
        for cluster in clusters:
            new_centroid = np.mean(clusters[cluster], axis=0)
            new_centroids.append(new_centroid)

        centroids = new_centroids

    return clusters, centroids
```

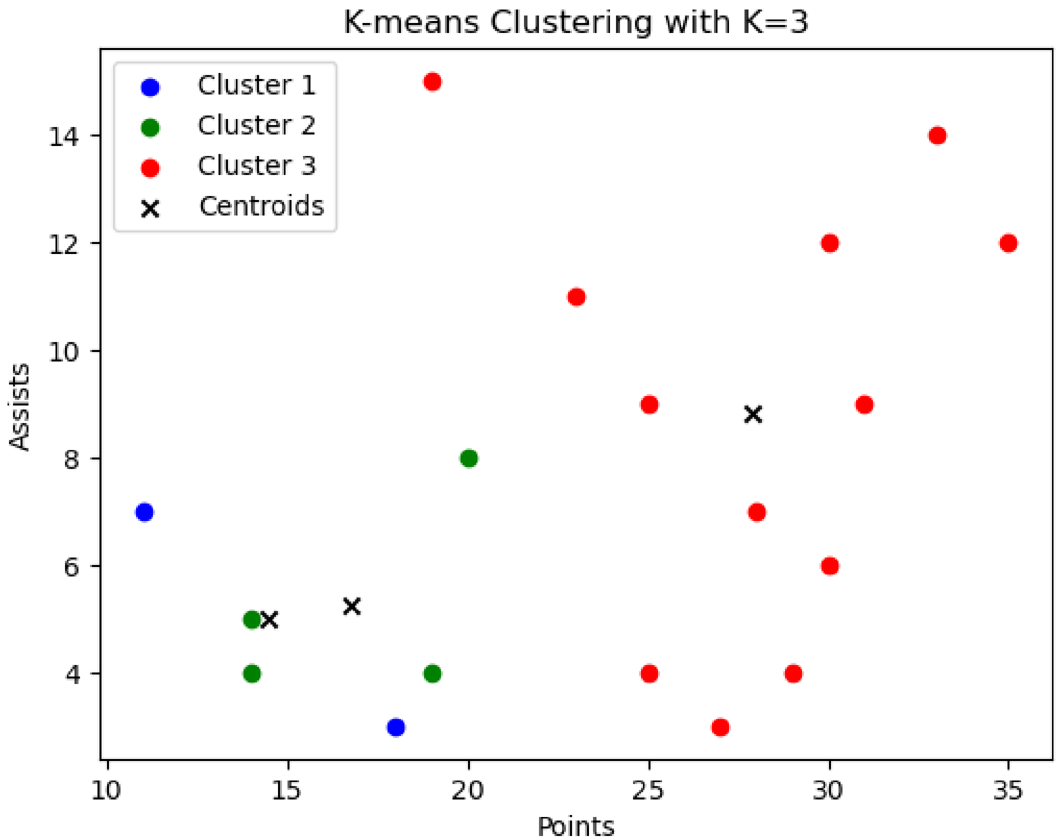
```python
def plot_clusters(df, clusters, centroids, title):
    colors = ['blue', 'green', 'red', 'purple']

    for cluster_idx, points in clusters.items():
        points = np.array(points)
        plt.scatter(points[:, 0], points[:, 1], color=colors[cluster_idx %
 ↪len(colors)], label=f'Cluster {cluster_idx+1}')

    centroids = np.array(centroids)
    plt.scatter(centroids[:, 0], centroids[:, 1], color='black', marker='x',
 ↪label='Centroids')
    plt.xlabel('Points')
    plt.ylabel('Assists')
    plt.legend()
    plt.title(title)
    plt.show()

# Step 5: SSE calculation function
def calculate_sse(clusters, centroids, distance_func):
    sse = 0
    for cluster_idx, points in clusters.items():
        centroid = centroids[cluster_idx]
        for point in points:
            sse += distance_func(point, centroid) ** 2
    return sse

# Step 6: Elbow method for K vs SSE plot
def elbow_method(df, max_k=4, distance_func=euclidean_distance):
    sse_values = []
    k_values = range(1, max_k + 1)

    for k in k_values:
        clusters, centroids = kmeans_manual(df, k=k, iterations=10)  # Using
 ↪Euclidean distance
        sse = calculate_sse(clusters, centroids, distance_func)
        sse_values.append(sse)

    # Plotting K vs SSE
    plt.plot(k_values, sse_values, marker='o')
    plt.xlabel('Number of Clusters (K)')
    plt.ylabel('Sum of Squared Errors (SSE)')
    plt.title('Elbow Method: K vs SSE')
    plt.grid(True)
    plt.show()

    return sse_values
```

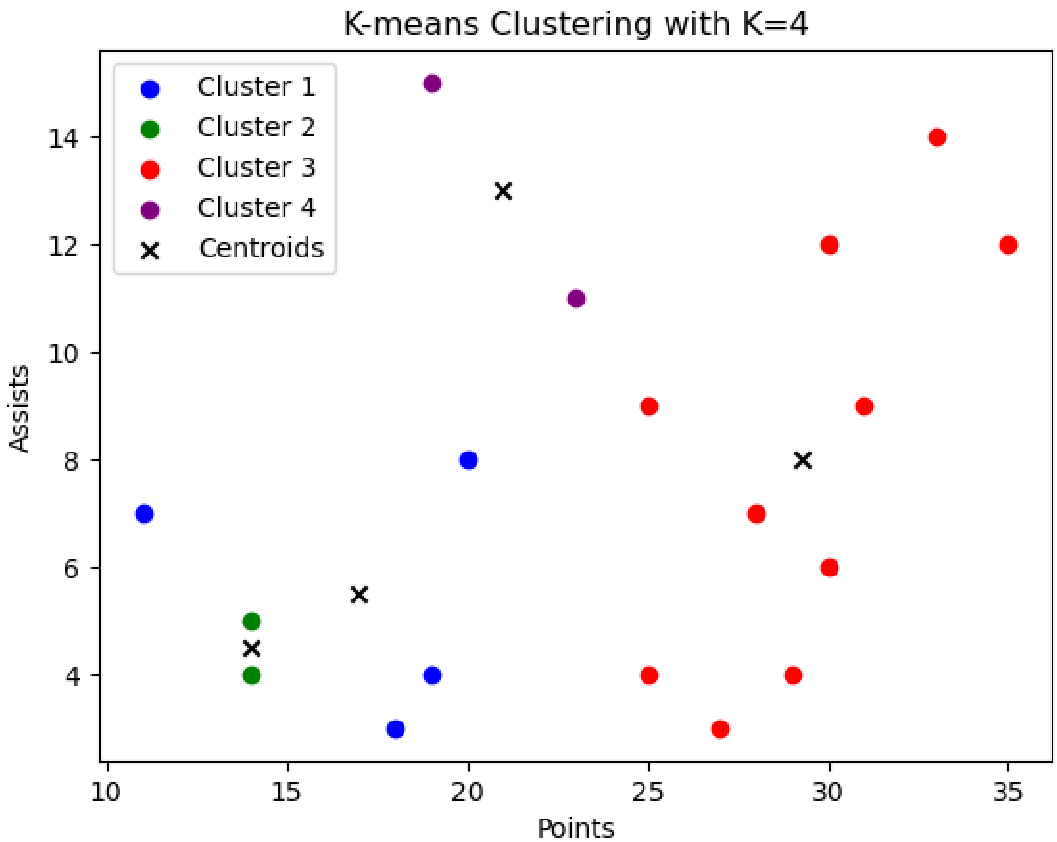
```
df = create_data()

plt.scatter(df['Points'], df['Assists'], color='blue')
plt.xlabel('Points')
plt.ylabel('Assists')
plt.title('Scatter plot of data points')
plt.show()

# Perform K-means for K=1, 2, 3, 4 and plot clusters
for k in range(1, 5):
    clusters, centroids = kmeans_manual(df, k=k, iterations=10)
    plot_clusters(df, clusters, centroids, title=f'K-means Clustering with␣
 ↪K={k}')

# Apply Elbow method to determine the optimal K
elbow_method(df, max_k=4, distance_func=euclidean_distance)
```

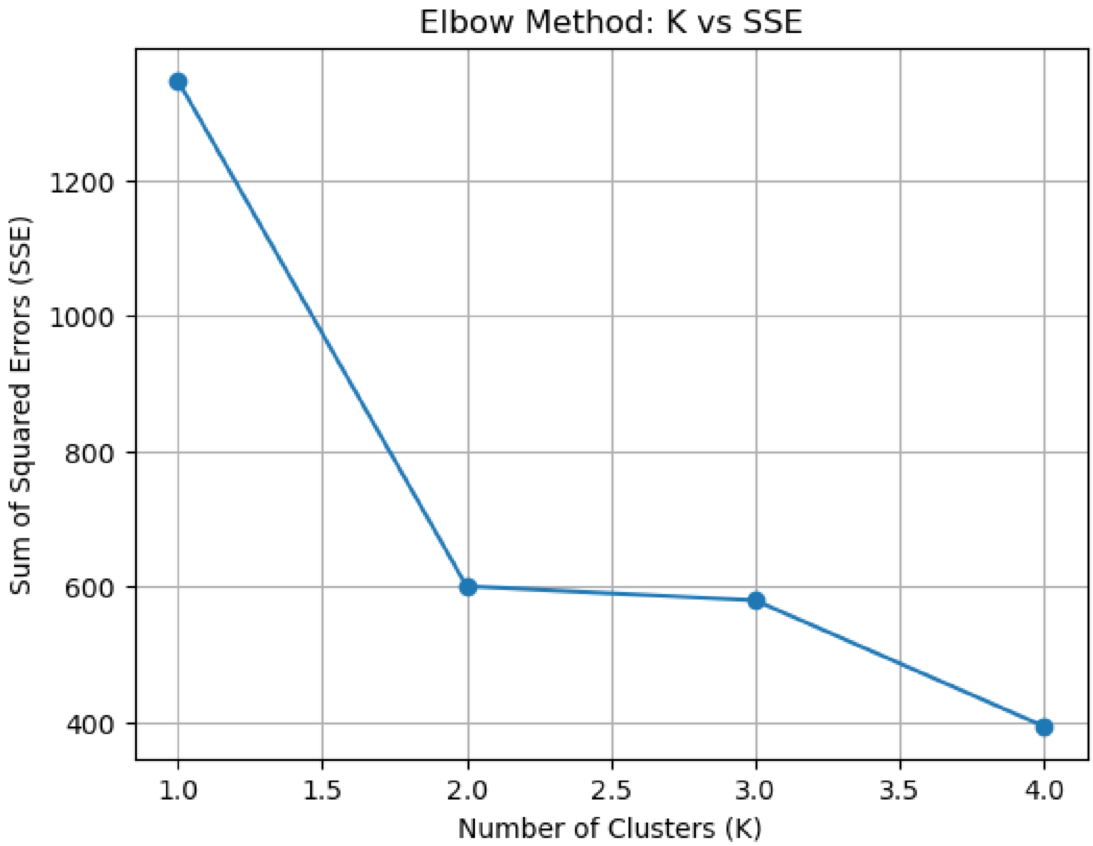



K-means Clustering with K=1
Cluster 1
Centroids
Assists
Points
14
12
10
8
6
4
10
15
20
25
30
35


K-means Clustering with K=2
Cluster 1
Cluster 2
Centroids
Assists
Points
14
12
10
8
6
4
10
15
20
25
30
35


K-means Clustering with K=3
Cluster 1
Cluster 2
Cluster 3
Centroids
Assists
Points
14
12
10
8
6
4
10
15
20
25
30
35


K-means Clustering with K=4
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Centroids
Assists
Points
14
12
10
8
6
4
10
15
20
25
30
35



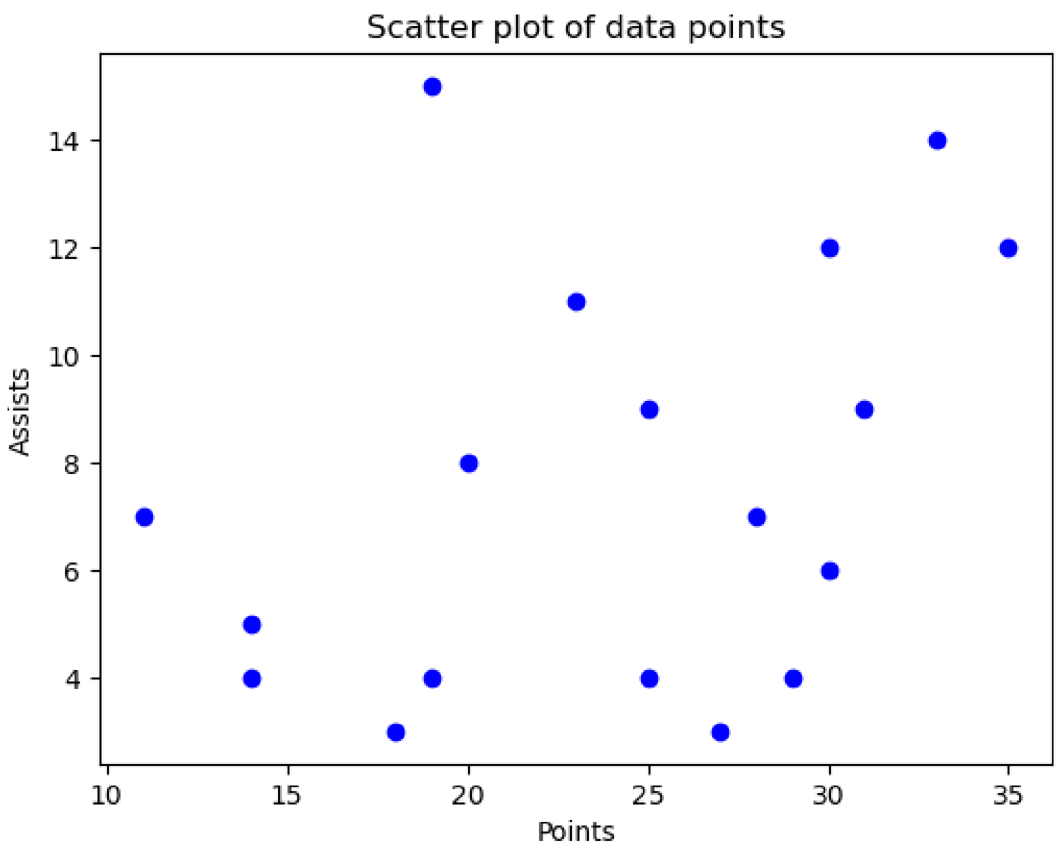
[7]: [1347.5, 601.0, 580.0833333333333, 394.1]

[8]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

def create_data():
    data = {
        'Points': [18.0, 19.0, 14.0, 14.0, 11.0, 20.0, 28.0, 30.0, 31.0, 35.0,␣
 ↪33.0, 25.0, 25.0, 27.0, 29.0, 30.0, 19.0, 23.0],
        'Assists': [3.0, 4.0, 5.0, 4.0, 7.0, 8.0, 7.0, 6.0, 9.0, 12.0, 14.0, 9.
 ↪0, 4.0, 3.0, 4.0, 12.0, 15.0, 11.0],
        'Rebounds': [15, 14, 10, 8, 14, 13, 9, 5, 4, 11, 6, 5, 3, 8, 12, 7, 6,␣
 ↪5]
    }
    df = pd.DataFrame(data)
    return df
```

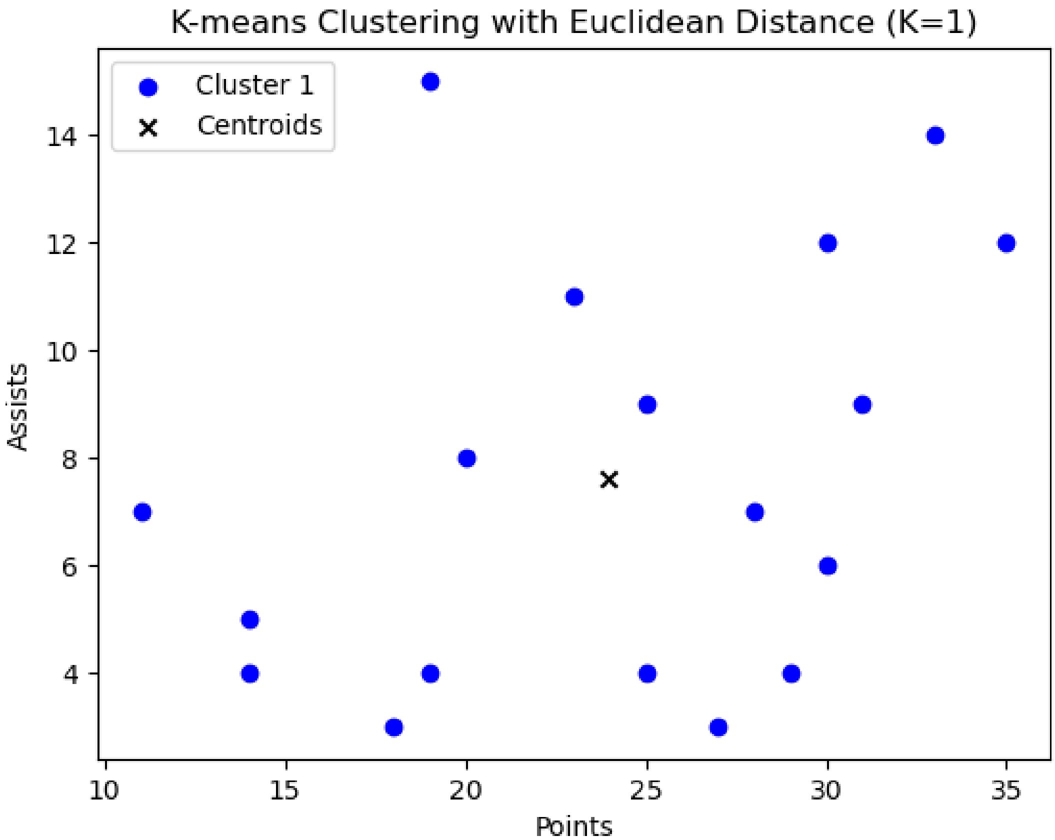
```python
def euclidean_distance(a, b):
    return np.sqrt(np.sum((a - b) ** 2))

def manhattan_distance(a, b):
    return np.sum(np.abs(a - b))

def minkowski_distance(a, b, p=3):  # Using p=3 for this example
    return np.power(np.sum(np.abs(a - b) ** p), 1 / p)

def kmeans_manual(df, k=2, distance_func=euclidean_distance, iterations=10):
    np.random.seed(42)
    centroids = df.sample(n=k).values

    for _ in range(iterations):
        clusters = {}

        for idx, row in df.iterrows():
            distances = [distance_func(row.values, centroid) for centroid in␣
 ↪centroids]
            cluster = np.argmin(distances)

            if cluster not in clusters:
                clusters[cluster] = []
            clusters[cluster].append(row.values)

        # Update centroids
        new_centroids = []
        for cluster in clusters:
            new_centroid = np.mean(clusters[cluster], axis=0)
            new_centroids.append(new_centroid)

        centroids = new_centroids

    return clusters, centroids

def plot_clusters(df, clusters, centroids, title):
    colors = ['blue', 'green', 'red', 'purple']

    for cluster_idx, points in clusters.items():
        points = np.array(points)
        plt.scatter(points[:, 0], points[:, 1], color=colors[cluster_idx %␣
 ↪len(colors)], label=f'Cluster {cluster_idx+1}')

    centroids = np.array(centroids)
    plt.scatter(centroids[:, 0], centroids[:, 1], color='black', marker='x',␣
 ↪label='Centroids')
    plt.xlabel('Points')
```

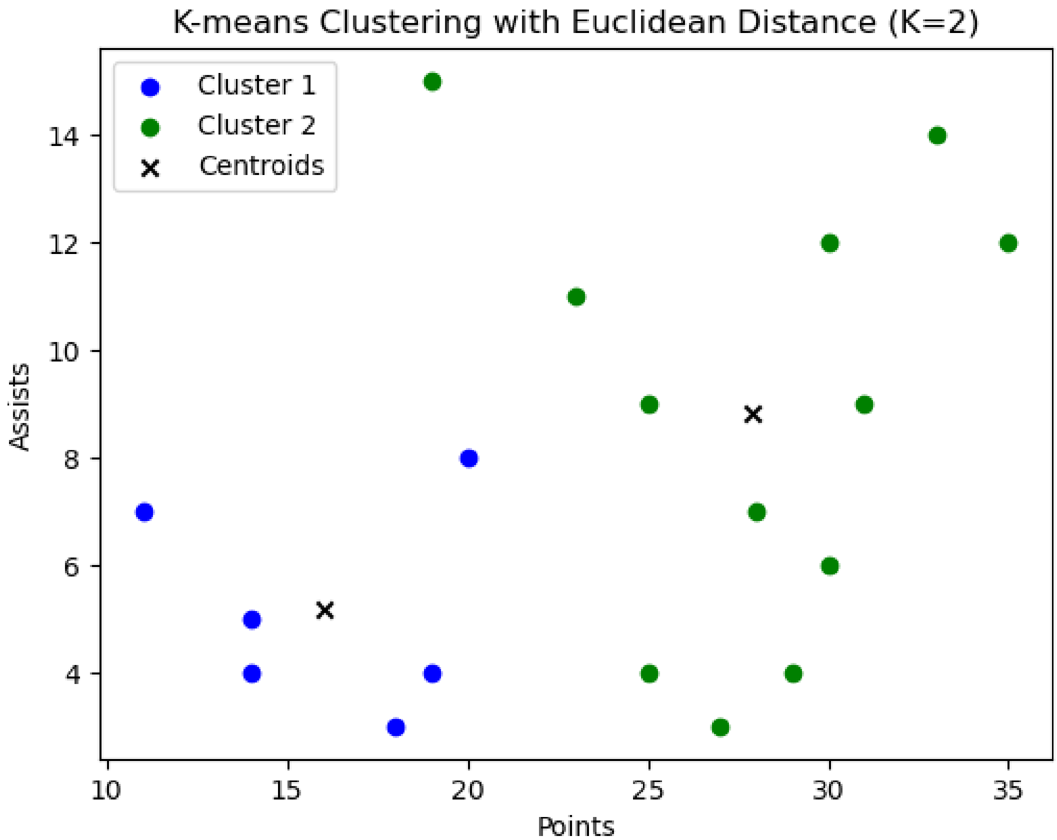
```
    plt.ylabel('Assists')
    plt.legend()
    plt.title(title)
    plt.show()

def calculate_sse(clusters, centroids, distance_func):
    sse = 0
    for cluster_idx, points in clusters.items():
        centroid = centroids[cluster_idx]
        for point in points:
            sse += distance_func(point, centroid) ** 2
    return sse

def elbow_method(df, max_k=4, distance_func=euclidean_distance):
    sse_values = []
    k_values = range(1, max_k + 1)

    for k in k_values:
        clusters, centroids = kmeans_manual(df, k=k,␣
↪distance_func=distance_func, iterations=10)
        sse = calculate_sse(clusters, centroids, distance_func)
        sse_values.append(sse)

    plt.plot(k_values, sse_values, marker='o')
    plt.xlabel('Number of Clusters (K)')
    plt.ylabel('Sum of Squared Errors (SSE)')
    plt.title(f'Elbow Method: K vs SSE ({distance_func.__name__})')
    plt.grid(True)
    plt.show()

    return sse_values

df = create_data()

plt.scatter(df['Points'], df['Assists'], color='blue')
plt.xlabel('Points')
plt.ylabel('Assists')
plt.title('Scatter plot of data points')
plt.show()


print("Clustering with Euclidean Distance")
for k in range(1, 5):
    clusters, centroids = kmeans_manual(df, k=k,␣
↪distance_func=euclidean_distance, iterations=10)
    plot_clusters(df, clusters, centroids, title=f'K-means Clustering with␣
↪Euclidean Distance (K={k})')
```

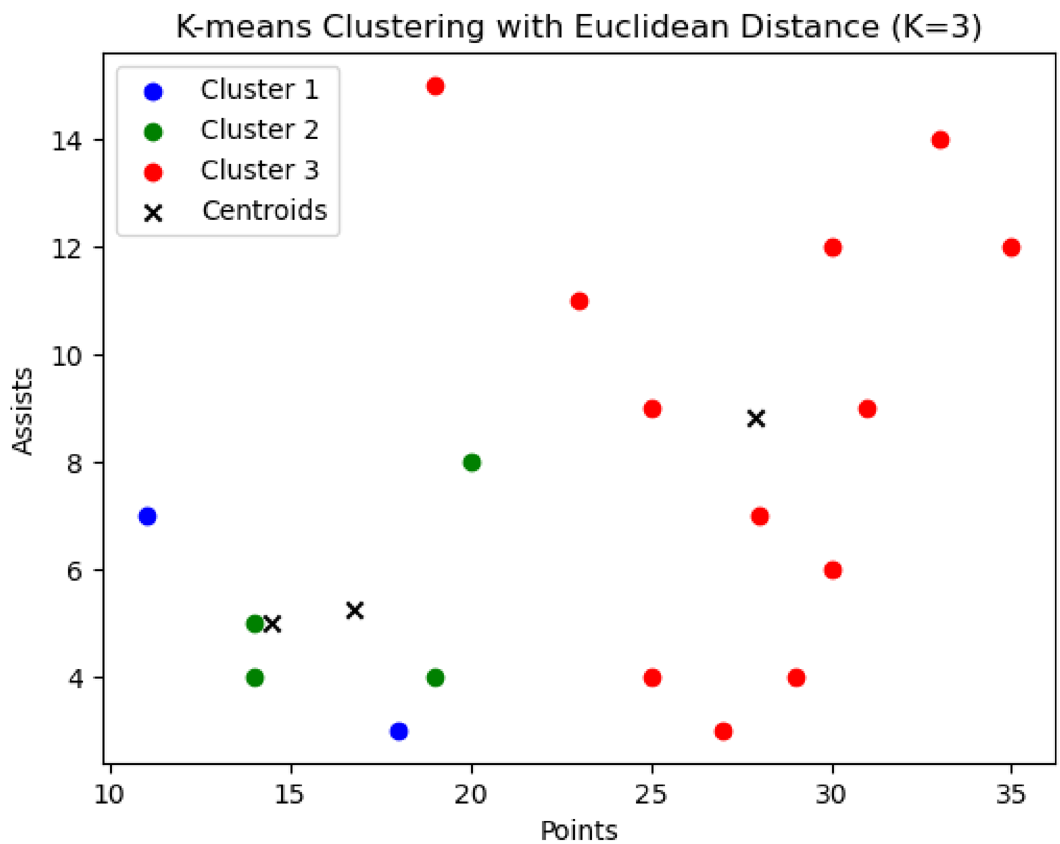
```
elbow_method(df, max_k=4, distance_func=euclidean_distance)

print("Clustering with Manhattan Distance")
for k in range(1, 5):
    clusters, centroids = kmeans_manual(df, k=k,␣
 ↪distance_func=manhattan_distance, iterations=10)
    plot_clusters(df, clusters, centroids, title=f'K-means Clustering with␣
 ↪Manhattan Distance (K={k})')
elbow_method(df, max_k=4, distance_func=manhattan_distance)

print("Clustering with Minkowski Distance (p=3)")
for k in range(1, 5):
    clusters, centroids = kmeans_manual(df, k=k, distance_func=lambda a, b:␣
 ↪minkowski_distance(a, b, p=3), iterations=10)
    plot_clusters(df, clusters, centroids, title=f'K-means Clustering with␣
 ↪Minkowski Distance (p=3) (K={k})')
elbow_method(df, max_k=4, distance_func=lambda a, b: minkowski_distance(a, b,␣
 ↪p=3))
```

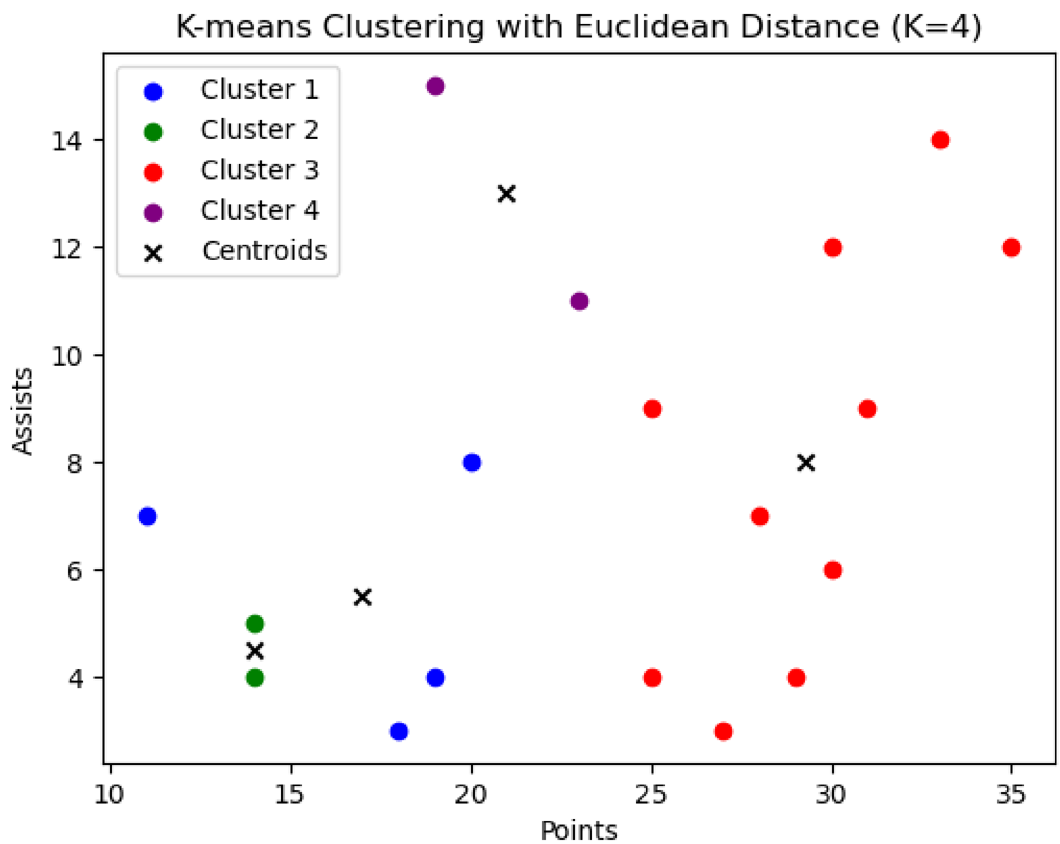


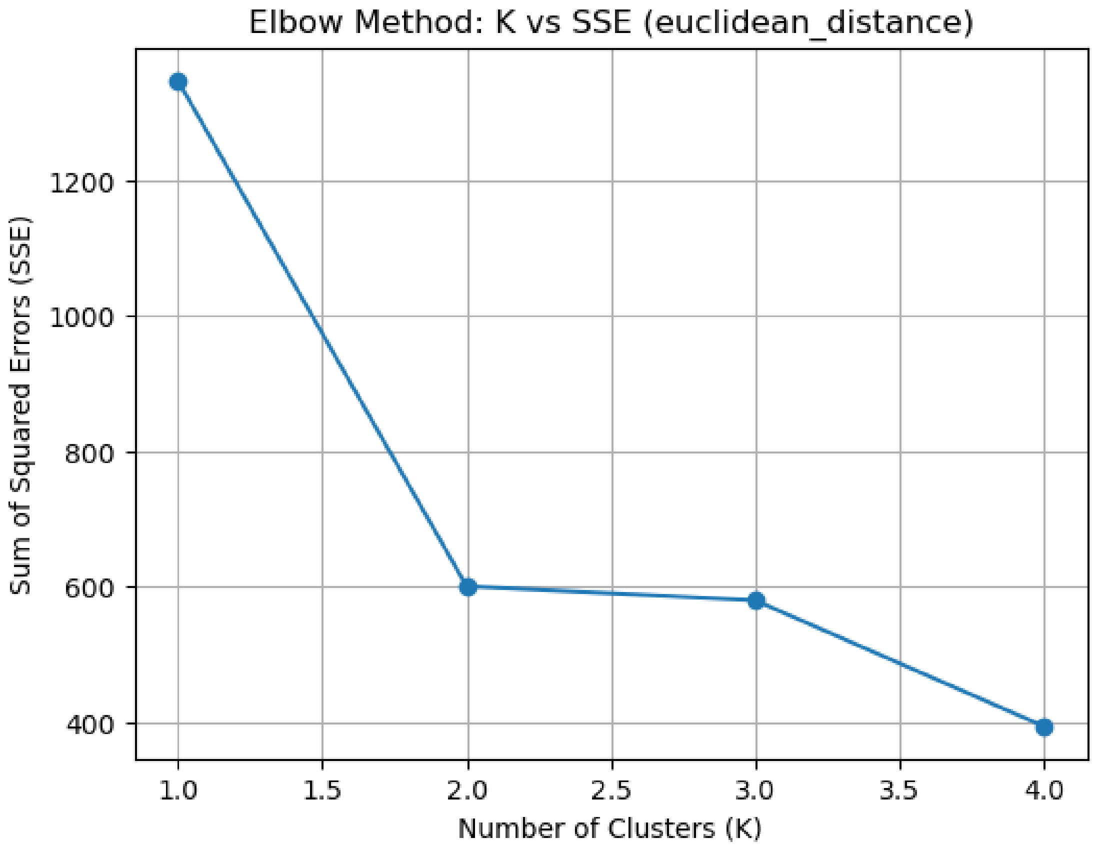

```
Clustering with Euclidean Distance
```


K-means Clustering with Euclidean Distance (K=1)
Cluster 1
Centroids
Assists
Points
14
12
10
8
6
4
10
15
20
25
30
35

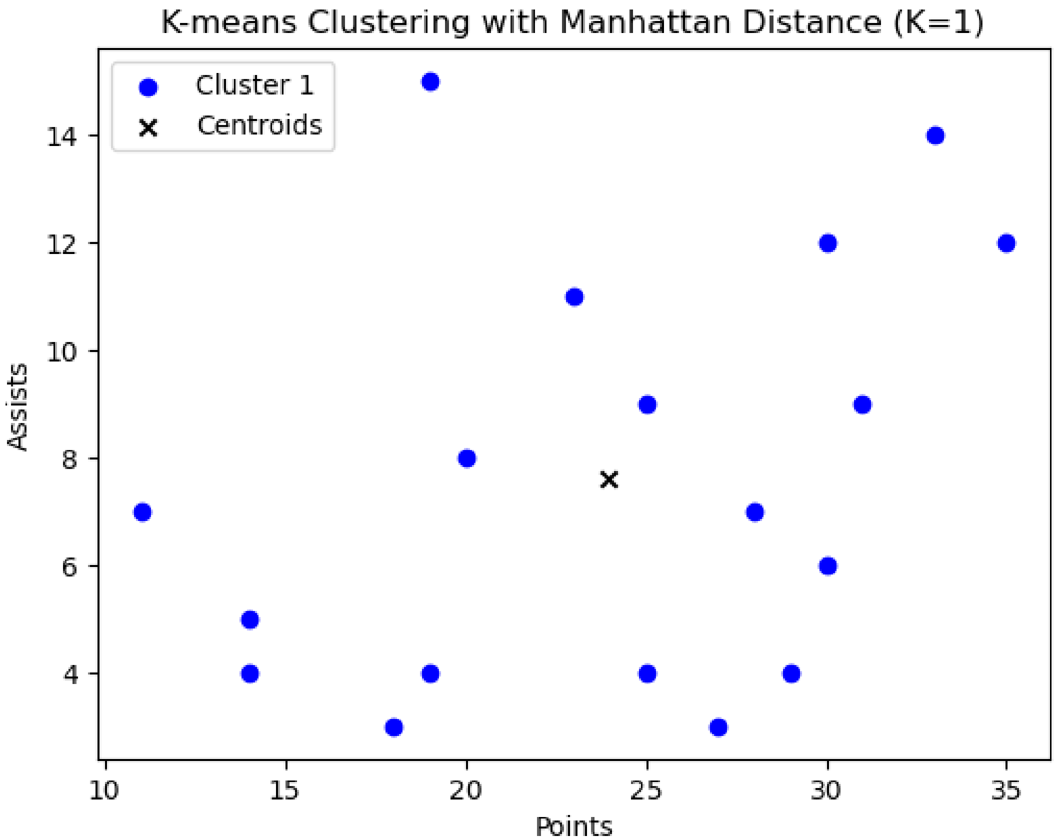

K-means Clustering with Euclidean Distance (K=2)
Cluster 1
Cluster 2
Centroids
Assists
Points
14
12
10
8
6
4
10
15
20
25
30
35

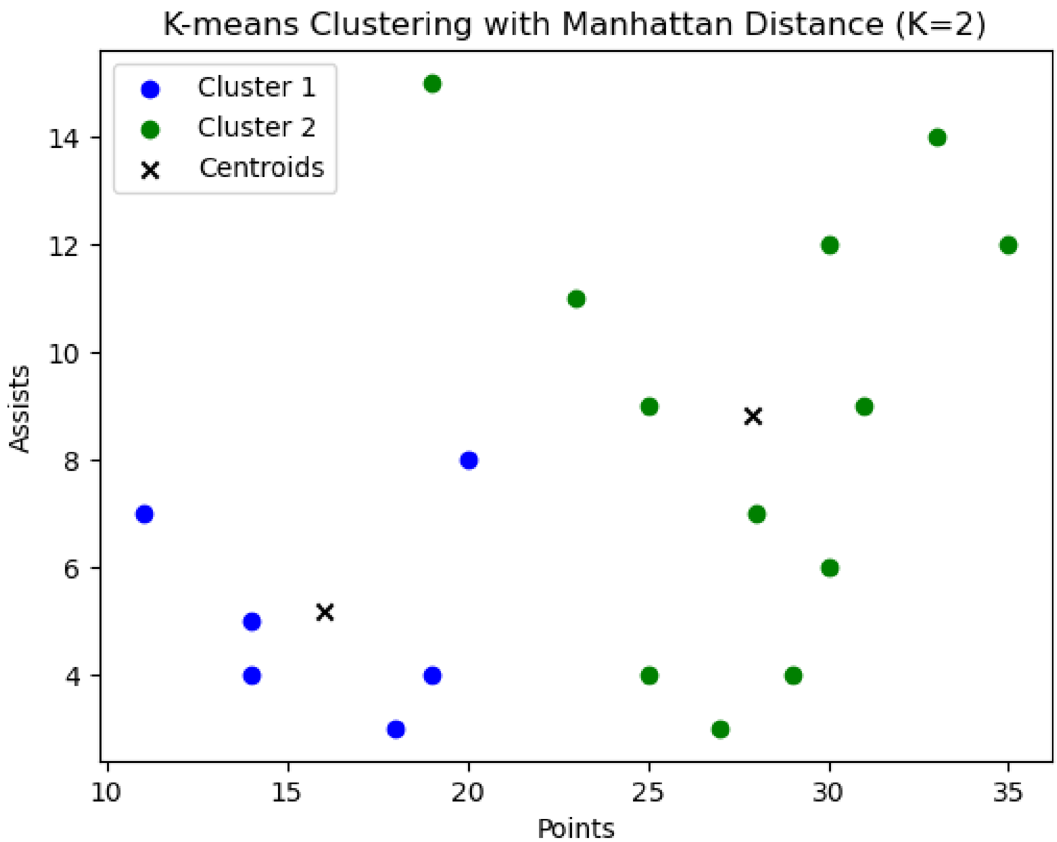

K-means Clustering with Euclidean Distance (K=3)
Cluster 1
Cluster 2
Cluster 3
Centroids
Assists
Points
14
12
10
8
6
4
10
15
20
25
30
35

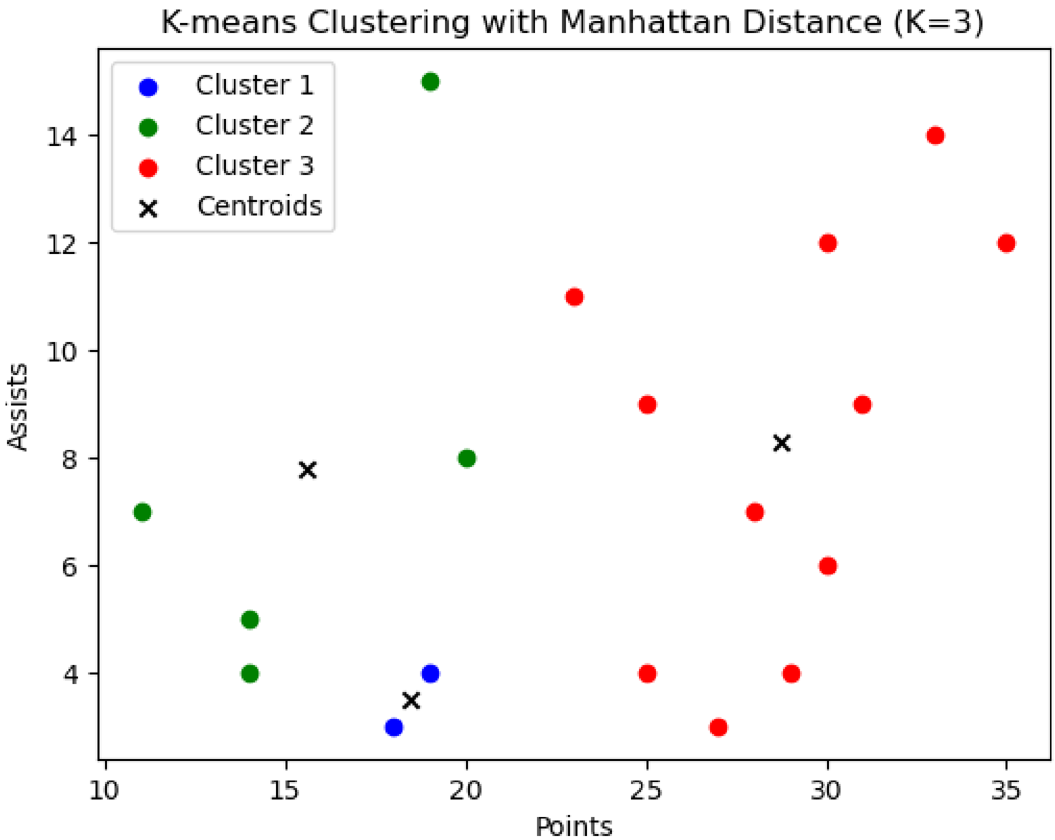

K-means Clustering with Euclidean Distance (K=4)
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Centroids
Assists
Points
14
12
10
8
6
4
10
15
20
25
30
35

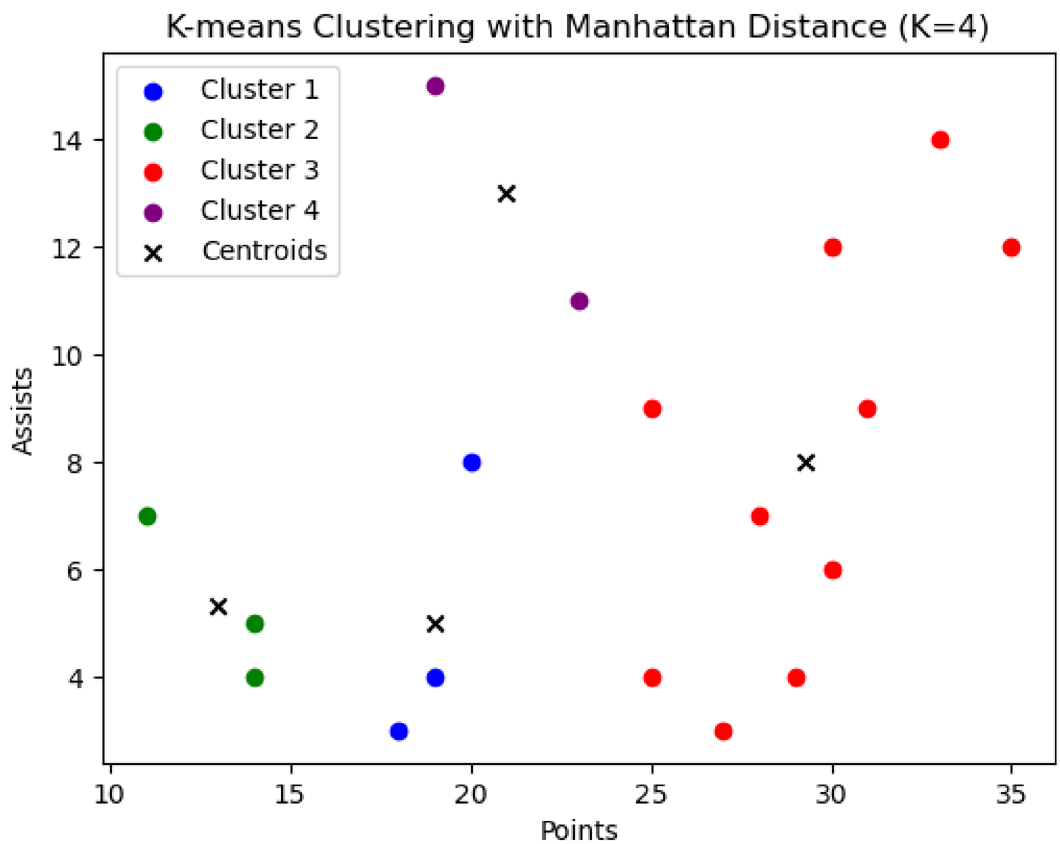


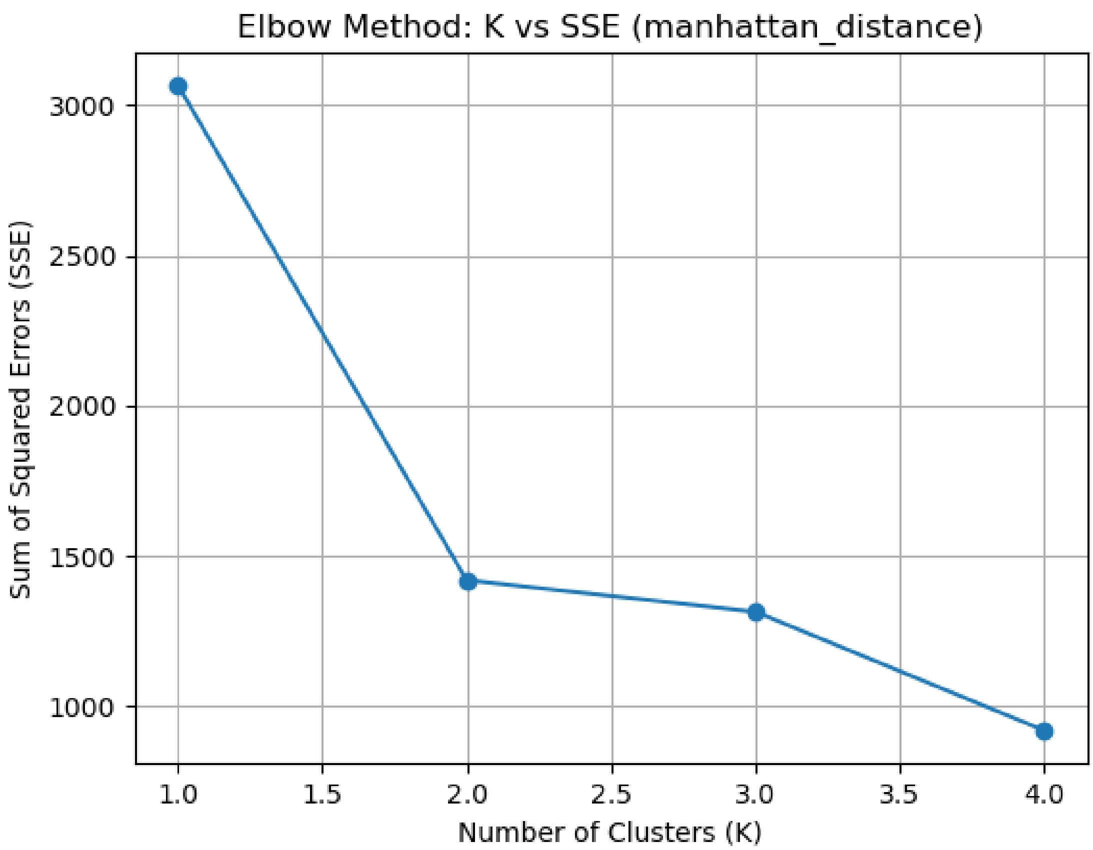
Clustering with Manhattan Distance


K-means Clustering with Manhattan Distance (K=1)
Cluster 1
Centroids
Assists
Points
14
12
10
8
6
4
10
15
20
25
30
35

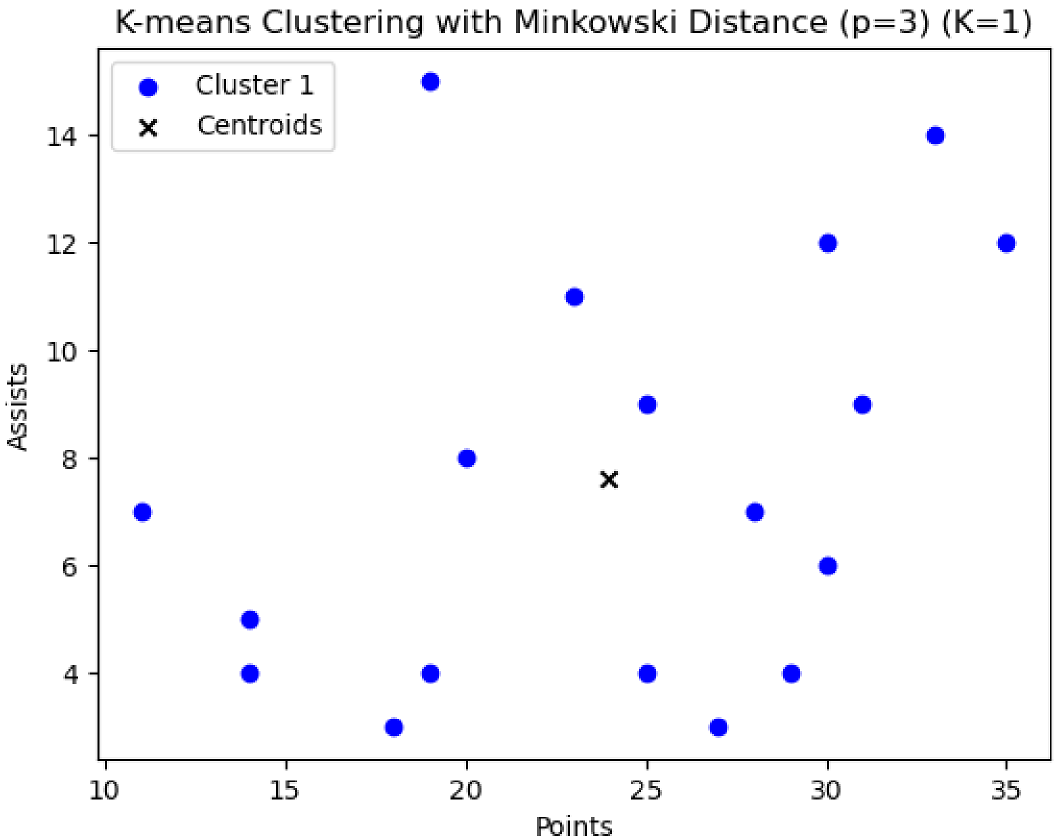

K-means Clustering with Manhattan Distance (K=2)
Cluster 1
Cluster 2
Centroids
Assists
Points
14
12
10
8
6
4
10
15
20
25
30
35

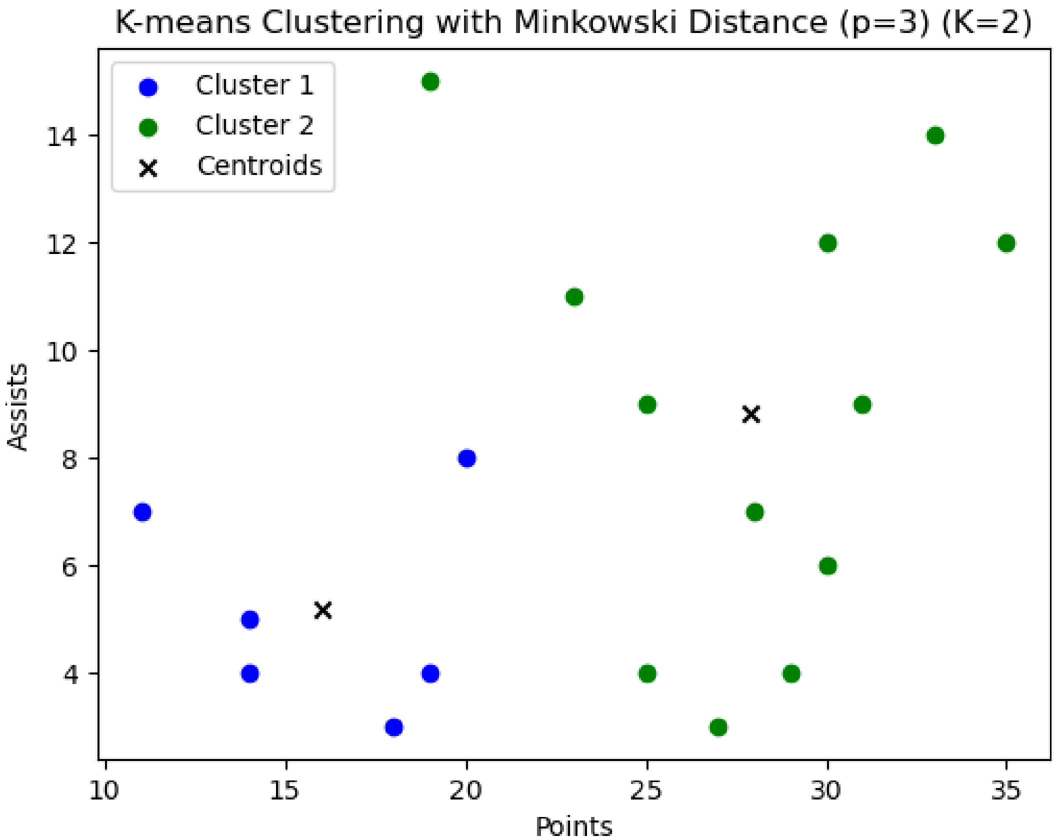

K-means Clustering with Manhattan Distance (K=3)
Cluster 1
Cluster 2
Cluster 3
Centroids
Assists
Points
14
12
10
8
6
4
10
15
20
25
30
35

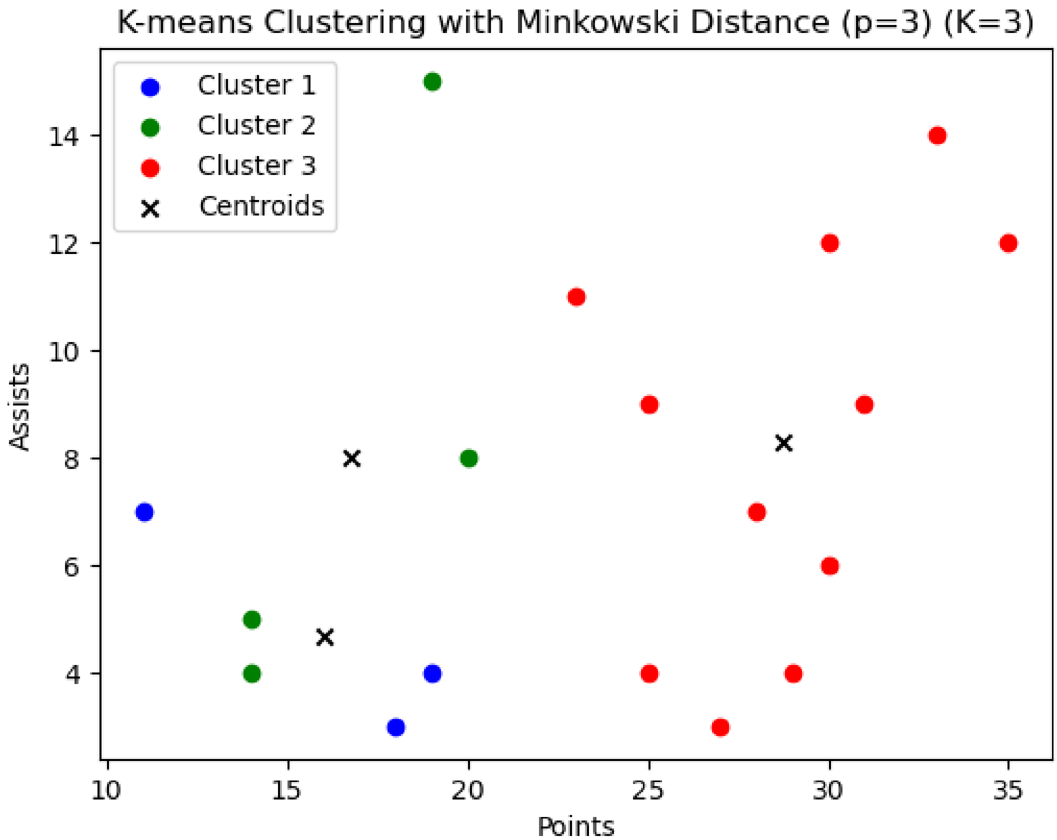

K-means Clustering with Manhattan Distance (K=4)
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Centroids
Assists
Points
10
15
20
25
30
35
4
6
8
10
12
14

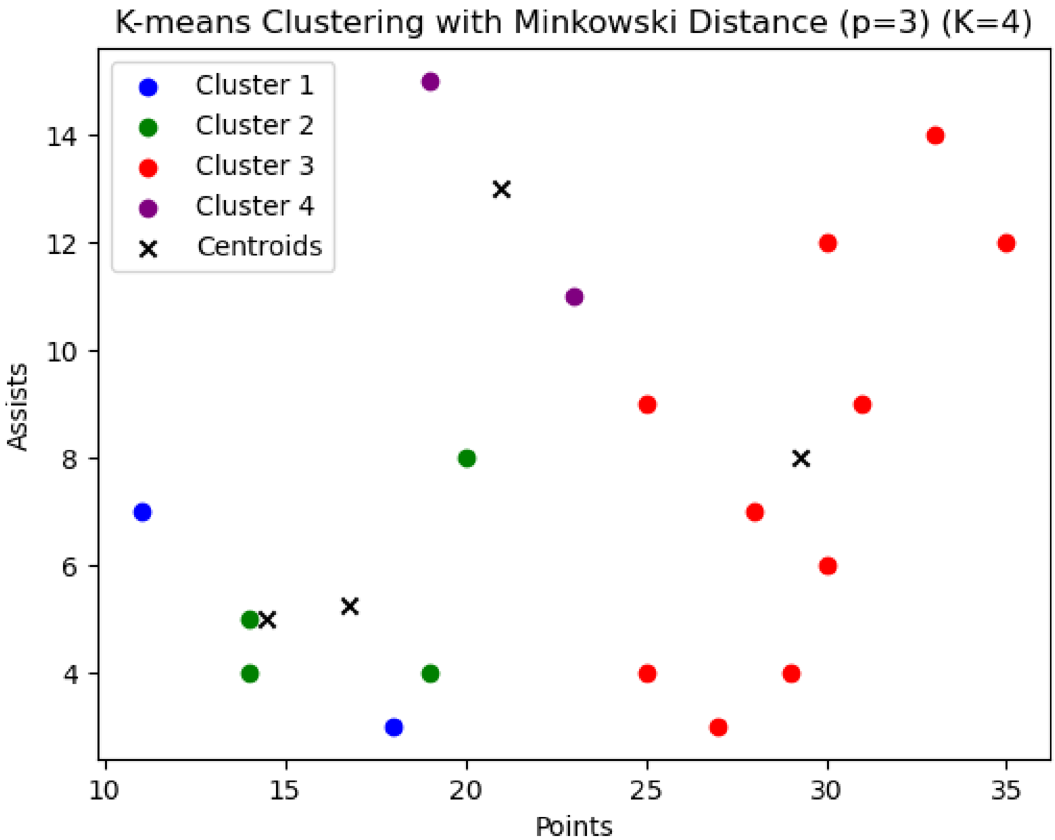


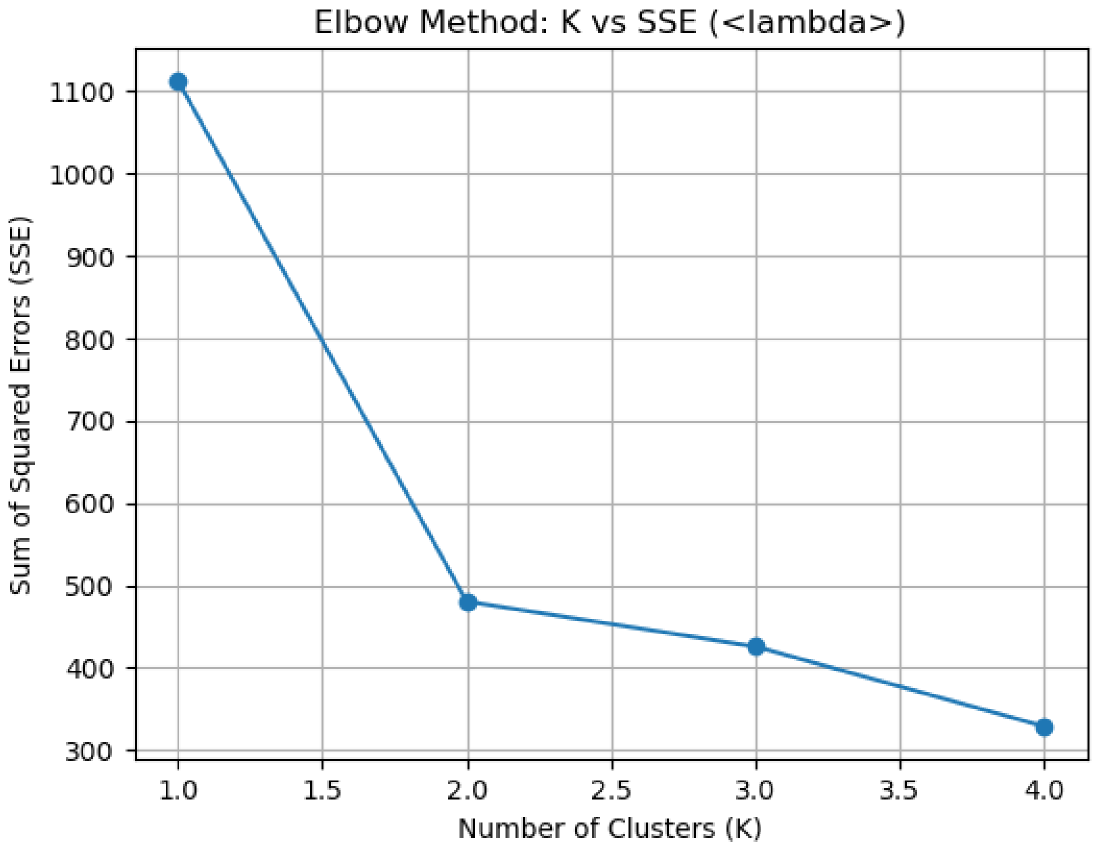
Clustering with Minkowski Distance (p=3)


K-means Clustering with Minkowski Distance (p=3) (K=1)
Cluster 1
Centroids
Assists
Points
14
12
10
8
6
4
10
15
20
25
30
35


K-means Clustering with Minkowski Distance (p=3) (K=2)
Cluster 1
Cluster 2
Centroids
Assists
Points
14
12
10
8
6
4
10
15
20
25
30
35


K-means Clustering with Minkowski Distance (p=3) (K=3)
Cluster 1
Cluster 2
Cluster 3
Centroids
Assists
Points
14
12
10
8
6
4
10
15
20
25
30
35


K-means Clustering with Minkowski Distance (p=3) (K=4)
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Centroids
Assists
Points
14
12
10
8
6
4
10
15
20
25
30
35



[8]: [1112.276058060871, 480.5059730403577, 425.8816414448997, 329.05570880436517]

[ ]: