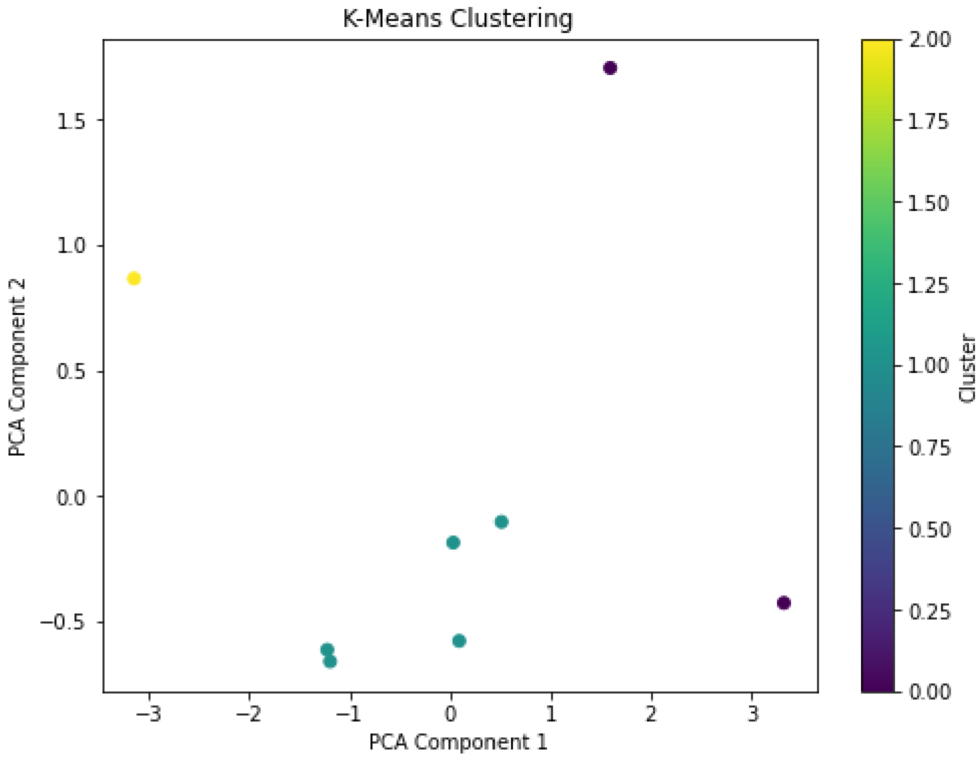
In [1]:
```
# Step 1: Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_score

# Step 2: Sample Data (you can replace this with your own dataset)
# For this example, we're assuming three subjects: Data Mining, Web Designin
# Columns might represent metrics such as "Interest", "Difficulty", "Hours S
data = {
    'Data Mining': [7, 8, 6, 9, 7, 5, 8, 7],
    'Web Designing': [5, 6, 7, 8, 6, 4, 5, 6],
    'Data Analytics': [8, 9, 7, 8, 9, 6, 7, 8],
    'Interest': [8, 7, 6, 9, 8, 5, 7, 6],
    'Difficulty': [6, 7, 5, 8, 7, 6, 6, 7],
}

# Create a DataFrame from the dictionary
df = pd.DataFrame(data)

# Step 3: Preprocess the data (optional but important)
# Normalize the features so that all features contribute equally to the clus
scaler = StandardScaler()
scaled_data = scaler.fit_transform(df)

# Step 4: Apply K-Means Clustering
# Choose the number of clusters (k)
k = 3
kmeans = KMeans(n_clusters=k, random_state=42)
kmeans.fit(scaled_data)

# Step 5: Add the cluster labels to the DataFrame
df['Cluster'] = kmeans.labels_

# Step 6: Visualize the clusters (assuming 2D data for simplicity)
# Using PCA to reduce the data to 2D for visualization purposes
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
reduced_data = pca.fit_transform(scaled_data)

# Plotting the clusters
plt.figure(figsize=(8, 6))
plt.scatter(reduced_data[:, 0], reduced_data[:, 1], c=kmeans.labels_, cmap='
plt.title('K-Means Clustering')
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.colorbar(label='Cluster')
plt.show()

# Step 7: Evaluate clustering (Optional but recommended)
silhouette_avg = silhouette_score(scaled_data, kmeans.labels_)
print(f'Silhouette Score: {silhouette_avg}')

# Show the resulting clusters in the original DataFrame
print(df)
```




```
Silhouette Score: 0.24980500434159675
   Data Mining  Web Designing  Data Analytics  Interest  Difficulty  Clust
er
0            7              5               8         8           6
1
1            8              6               9         7           7
1
2            6              7               7         6           5
0
3            9              8               8         9           8
2
4            7              6               9         8           7
1
5            5              4               6         5           6
0
6            8              5               7         7           6
1
7            7              6               8         6           7
1
```

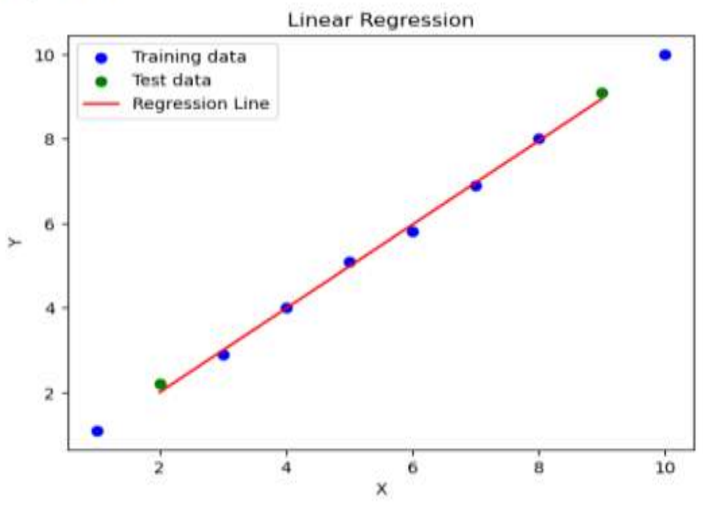
In [ ]:

In [8]:
```
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
from sklearn.datasets import load_iris

# Load the Iris dataset
data = load_iris()
X = data.data  # Features
y = data.target  # Target labels

# Split the dataset into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a Random Forest Classifier
clf = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the classifier
clf.fit(X_train, y_train)

# Make predictions on the test set
y_pred = clf.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")

# Generate a classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
Accuracy: 100.00%
```

In [1]:
```python
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt

# Example dataset
data = {
    'X': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],  # Independent variable
    'Y': [1.1, 2.2, 2.9, 4.0, 5.1, 5.8, 6.9, 8.0, 9.1, 10.0]  # Dependent variable
}

# Create DataFrame
df = pd.DataFrame(data)

# Split the data into features and target
X = df[['X']]  # Features (independent variable)
Y = df['Y']    # Target (dependent variable)

# Split the data into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=42)

# Create and train the model
model = LinearRegression()
model.fit(X_train, Y_train)

# Make predictions on the test set
Y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(Y_test, Y_pred)
r2 = r2_score(Y_test, Y_pred)

# Print evaluation metrics
print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')

# Visualize the results
plt.scatter(X_train, Y_train, color='blue', label='Training data')
plt.scatter(X_test, Y_test, color='green', label='Test data')
plt.plot(X_test, Y_pred, color='red', label='Regression Line')
plt.title('Linear Regression')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.show()
```

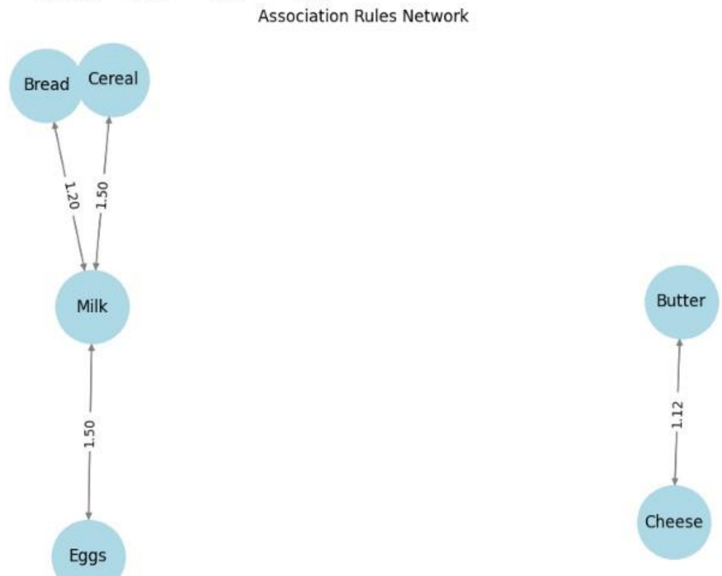
```
Mean Squared Error: 0.031018133174791754
R-squared: 0.9973939816698347
```



8. Association Rule Mining using Apriori.

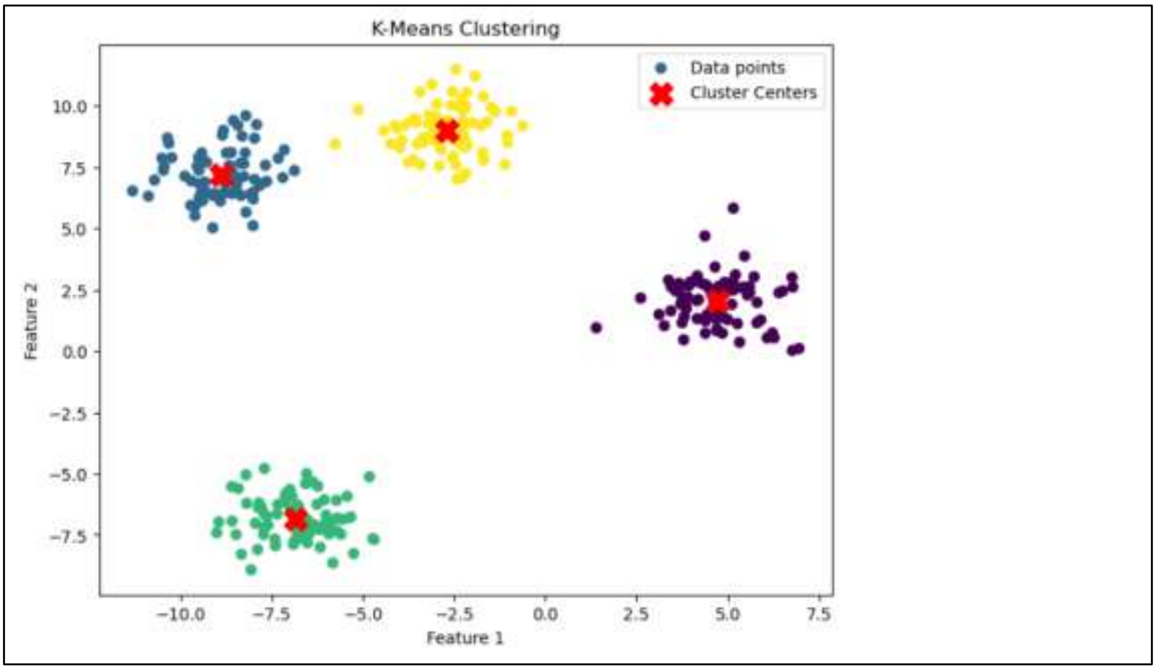
```
--import pandas as pd
import matplotlib.pyplot as plt
import networkx as nx
from mlxtend.frequent_patterns import apriori, association_rules

# Sample dataset (Market Basket Data)
data = {
    'Milk': [1, 0, 1, 1, 0, 1],
    'Bread': [1, 1, 1, 0, 1, 1],
    'Butter': [0, 1, 1, 1, 1, 0],
    'Eggs': [1, 0, 0, 1, 0, 1],
    'Cheese': [0, 1, 0, 1, 1, 1],
    'Cereal': [1, 0, 1, 0, 1, 1]
}

df = pd.DataFrame(data)

# Apply the Apriori algorithm to find frequent itemsets
frequent_itemsets = apriori(df, min_support=0.4, use_colnames=True)
print("Frequent Itemsets:")
print(frequent_itemsets)

# Generate association rules
rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1.0)
print("\nAssociation Rules:")
print(rules)

# Visualization of Association Rules
def draw_graph(rules):
    G = nx.DiGraph()
    for i, row in rules.iterrows():
        for antecedent in row['antecedents']:
            for consequent in row['consequents']:
                G.add_edge(antecedent, consequent, weight=row['lift'])

    plt.figure(figsize=(8, 6))
    pos = nx.spring_layout(G, k=0.5)
    edges = G.edges(data=True)
    nx.draw(G, pos, with_labels=True, node_size=3000, node_color="lightblue", edge_color="gray")
    labels = {(u, v): f"{d['weight']:.2f}" for u, v, d in edges}
    nx.draw_networkx_edge_labels(G, pos, edge_labels=labels)
    plt.title("Association Rules Network")
    plt.show()

# Draw the graph of association rules
draw_graph(rules)
```

```
/usr/local/lib/python3.11/dist-packages/mlxtend/frequent_patterns/fpcommon.py:161: DeprecationWarning: DataFram
  warnings.warn(
Frequent Itemsets:
     support               itemsets
0   0.666667                 (Milk)
1   0.833333                (Bread)
2   0.666667               (Butter)
3   0.500000                 (Eggs)
4   0.666667               (Cheese)
5   0.666667               (Cereal)
6   0.500000          (Milk, Bread)
7   0.500000           (Milk, Eggs)
8   0.500000         (Milk, Cereal)
9   0.500000        (Butter, Bread)
10  0.500000        (Cheese, Bread)
11  0.666667        (Bread, Cereal)
12  0.500000       (Butter, Cheese)
13  0.500000  (Milk, Bread, Cereal)

Association Rules:
       antecedents      consequents  antecedent support  consequent support  \
0           (Milk)           (Eggs)            0.666667            0.500000
1           (Eggs)           (Milk)            0.500000            0.666667
2           (Milk)         (Cereal)            0.666667            0.666667
3         (Cereal)           (Milk)            0.666667            0.666667
4          (Bread)         (Cereal)            0.833333            0.666667
5         (Cereal)          (Bread)            0.666667            0.833333
6         (Butter)         (Cheese)            0.666667            0.666667
7         (Cheese)         (Butter)            0.666667            0.666667
8    (Milk, Bread)         (Cereal)            0.500000            0.666667
9   (Milk, Cereal)          (Bread)            0.500000            0.833333
10 (Bread, Cereal)           (Milk)            0.666667            0.666667
11          (Milk)  (Bread, Cereal)            0.666667            0.666667
12         (Bread)   (Milk, Cereal)            0.833333            0.500000
13        (Cereal)    (Milk, Bread)            0.666667            0.500000

     support  confidence   lift  representativity  leverage  conviction  \
0   0.500000        0.75  1.500               1.0  0.166667    2.000000
1   0.500000        1.00  1.500               1.0  0.166667         inf
2   0.500000        0.75  1.125               1.0  0.055556    1.333333
3   0.500000        0.75  1.125               1.0  0.055556    1.333333
4   0.666667        0.80  1.200               1.0  0.111111    1.666667
5   0.666667        1.00  1.200               1.0  0.111111         inf
6   0.500000        0.75  1.125               1.0  0.055556    1.333333
7   0.500000        0.75  1.125               1.0  0.055556    1.333333
8   0.500000        1.00  1.500               1.0  0.166667         inf
9   0.500000        1.00  1.200               1.0  0.083333         inf
10  0.500000        0.75  1.125               1.0  0.055556    1.333333
11  0.500000        0.75  1.125               1.0  0.055556    1.333333
12  0.500000        0.60  1.200               1.0  0.083333    1.250000
13  0.500000        0.75  1.500               1.0  0.166667    2.000000


    zhangs_metric  jaccard  certainty  kulczynski
0        1.000000     0.75       0.50       0.875
1        0.666667     0.75       1.00       0.875
2        0.333333     0.60       0.25       0.750
3        0.333333     0.60       0.25       0.750
4        1.000000     0.80       0.40       0.900
5        0.500000     0.80       1.00       0.900
6        0.333333     0.60       0.25       0.750
7        0.333333     0.60       0.25       0.750
8        0.666667     0.75       1.00       0.875
9        0.333333     0.60       1.00       0.800
10       0.333333     0.60       0.25       0.750
11       0.333333     0.60       0.25       0.750
12       1.000000     0.60       0.20       0.800
13       1.000000     0.75       0.50       0.875
```



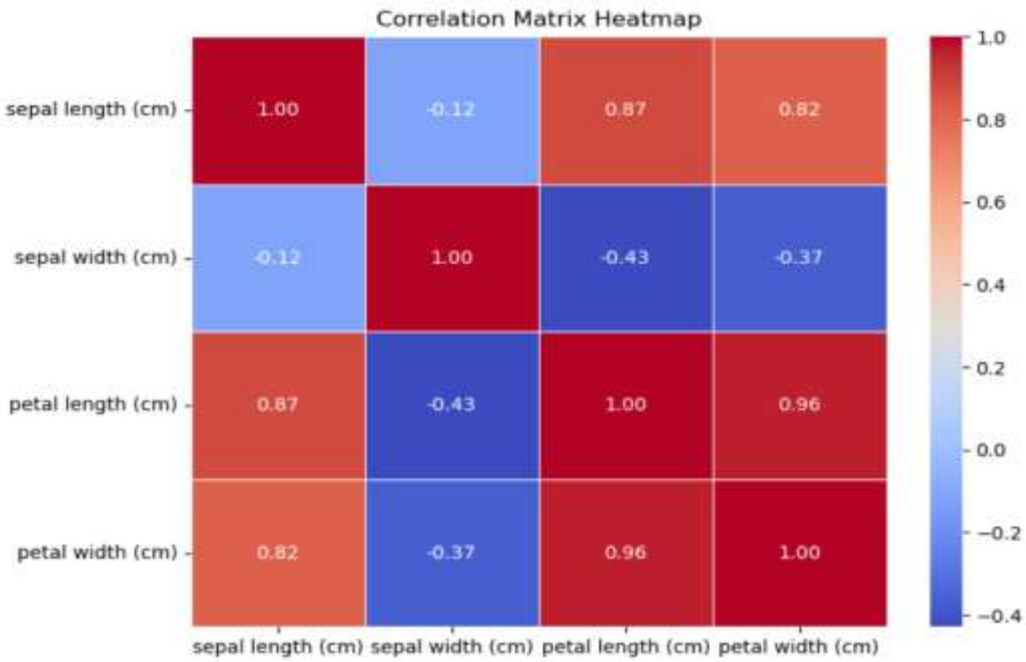
Practical 9 visualize the result of the clustering and compare.

In [1]:
```
import numpy as np
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt

# Step 1: Generate synthetic data
X, y = make_blobs(n_samples=300, centers=4, random_state=42)

# Step 2: Apply KMeans clustering
kmeans = KMeans(n_clusters=4, random_state=42)
kmeans.fit(X)

# Step 3: Get the labels and centers
labels = kmeans.labels_
centers = kmeans.cluster_centers_

# Step 4: Visualize the results
plt.figure(figsize=(8, 6))

# Plot data points with color based on the labels
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', label='Data points')

# Plot the cluster centers
plt.scatter(centers[:, 0], centers[:, 1], s=200, c='red', marker='X', label='Cluster Centers')

plt.title('K-Means Clustering')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.show()
```



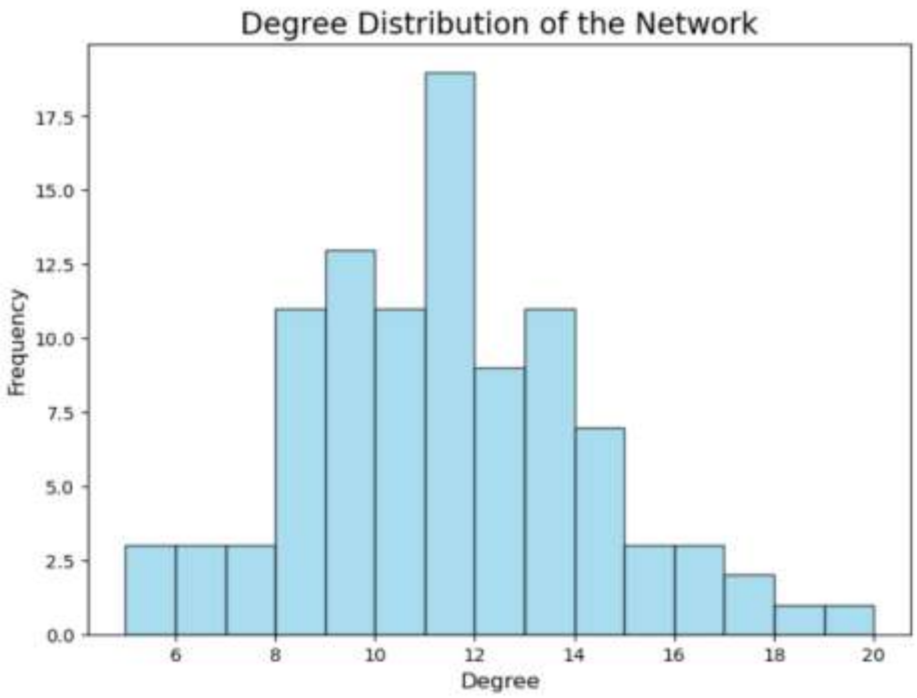
Practical-10: Visualize the correlation matrix using a pseudocolor plot.

In [1]:
```
# Step 1: Import necessary libraries
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Step 2: Load or create your dataset (Example using the Iris dataset)
from sklearn.datasets import load_iris
import numpy as np

# Load the Iris dataset as an example
data = load_iris()
df = pd.DataFrame(data=data.data, columns=data.feature_names)

# Step 3: Compute the correlation matrix
corr_matrix = df.corr()

# Step 4: Plot the correlation matrix using a heatmap
plt.figure(figsize=(8, 6))  # Set the figure size
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5)
plt.title('Correlation Matrix Heatmap')
plt.show()
```



Practical-11: Use of degrees distribution of a network.

```
In [3]: import networkx as nx
        import matplotlib.pyplot as plt

        # Step 1: Create a network (Graph)
        G = nx.erdos_renyi_graph(100, 0.1)  # Example: Create a random graph with 100 nodes and edge probability 0.1

        # Step 2: Calculate the degree of each node
        degree_sequence = [d for n, d in G.degree()]

        # Step 3: Plot the degree distribution
        plt.figure(figsize=(8, 6))
        plt.hist(degree_sequence, bins=range(min(degree_sequence), max(degree_sequence) + 1), color='skyblue', edgecolor='black',
                 alpha=0.7)
        plt.title("Degree Distribution of the Network", fontsize=16)
        plt.xlabel("Degree", fontsize=12)
        plt.ylabel("Frequency", fontsize=12)
        plt.show()
```



Practical-12: Graph visualization of a network using maximum, minimum, median, first quartile and third quartile.

In [10]:
```python
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np

# Step 1: Create a network (Graph)
G = nx.erdos_renyi_graph(100, 0.1)  # Create a random graph with 100 nodes and edge probability 0.1

# Step 2: Calculate the degree of each node
degree_sequence = [d for n, d in G.degree()]

# Step 3: Calculate the statistical measures
min_degree = np.min(degree_sequence)
max_degree = np.max(degree_sequence)
median_degree = np.median(degree_sequence)
Q1 = np.percentile(degree_sequence, 25)
Q3 = np.percentile(degree_sequence, 75)

# Print the statistics
print(f"Min degree: {min_degree}")
print(f"Max degree: {max_degree}")
print(f"Median degree: {median_degree}")
print(f"1st Quartile (Q1): {Q1}")
print(f"3rd Quartile (Q3): {Q3}")

# Step 4: Visualize the network, with node color based on degree
plt.figure(figsize=(10, 8))

# Choose node color based on degree
node_color = [d for n, d in G.degree()]

# Create a spring layout for the graph
pos = nx.spring_layout(G)

# Draw the graph with node size proportional to degree
nx.draw(G, pos, with_labels=True, node_size=[d * 50 for d in degree_sequence],
        node_color=node_color, cmap=plt.cm.Blues, edge_color='gray', font_size=10)

# Add labels for the statistics on the plot
plt.title(f"Network Graph\nMin:{min_degree},Max: {max_degree}, Median: {median_degree}\nQ1: {Q1}, Q3: {Q3}", fontsize=14)

plt.show()
```

```
Min degree: 2
Max degree: 19
Median degree: 10.0
1st Quartile (Q1): 7.0
3rd Quartile (Q3): 12.0
```