BDA HOLIDAY ASSIGNMENT -1

2211CS020319
AIML-DELTA

In [23]:
```
!pip install imbalanced-learn
```

```
Requirement already satisfied: imbalanced-learn in c:\users\chait\anaconda3\lib\site-packages (0.12.4)
Requirement already satisfied: scikit-learn>=1.0.2 in c:\users\chait\anaconda3\lib\site-packages (from imba
lanced-learn) (1.0.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\chait\anaconda3\lib\site-packages (from imb
alanced-learn) (3.5.0)
Requirement already satisfied: scipy>=1.5.0 in c:\users\chait\anaconda3\lib\site-packages (from imbalanced-
learn) (1.9.1)
Requirement already satisfied: joblib>=1.1.1 in c:\users\chait\anaconda3\lib\site-packages (from imbalanced
-learn) (1.4.2)
Requirement already satisfied: numpy>=1.17.3 in c:\users\chait\anaconda3\lib\site-packages (from imbalanced
-learn) (1.24.4)

WARNING: Ignoring invalid distribution -rotobuf (c:\users\chait\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -rotobuf (c:\users\chait\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -rotobuf (c:\users\chait\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -rotobuf (c:\users\chait\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -rotobuf (c:\users\chait\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -rotobuf (c:\users\chait\anaconda3\lib\site-packages)
```

In [4]:

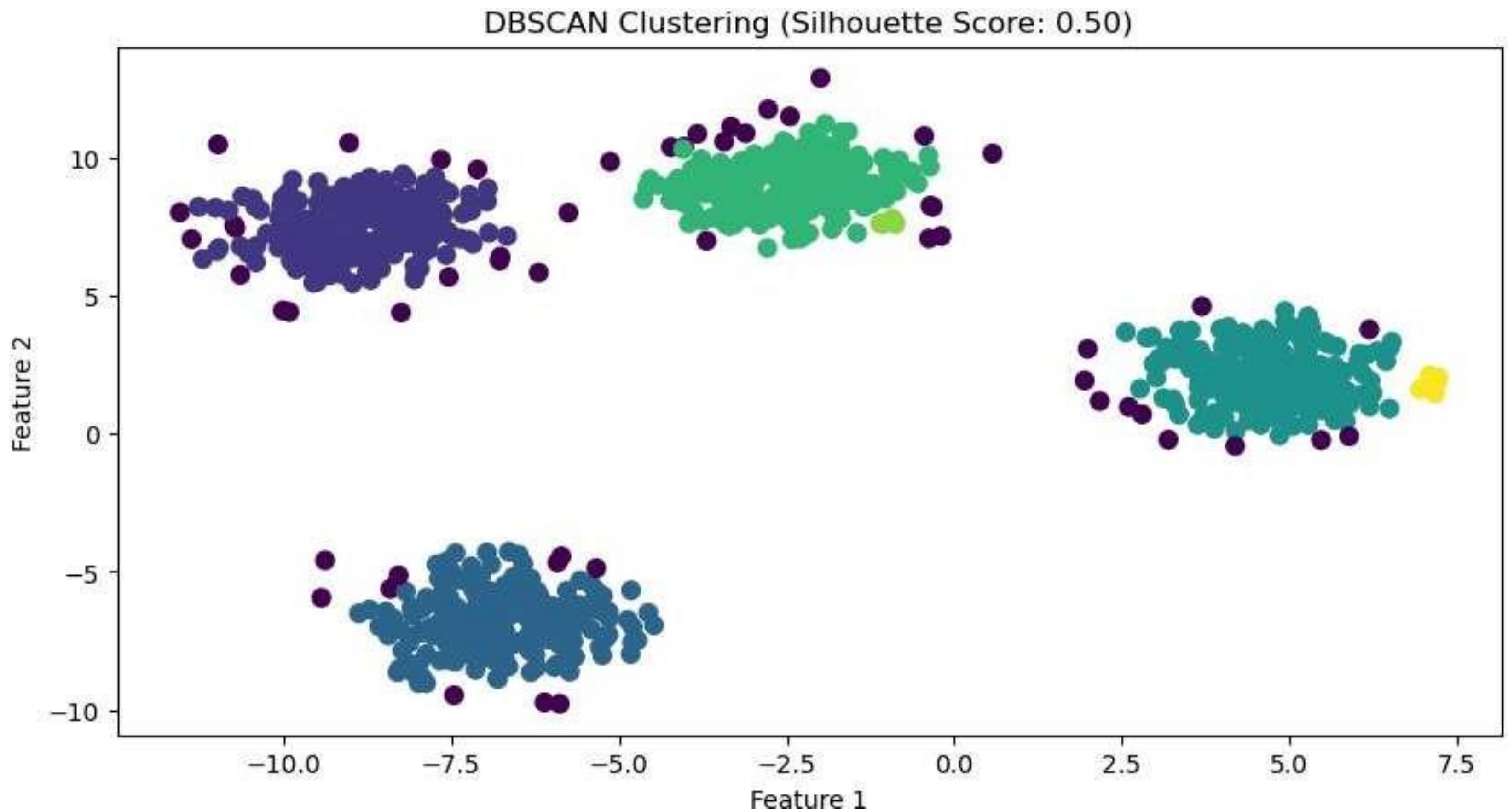
```python
#Question 1: Handling Imbalanced Datasets

from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
from imblearn.over_sampling import SMOTE
from sklearn.utils.class_weight import compute_class_weight

# Generate a synthetic imbalanced dataset
X, y = make_classification(n_samples=1000, n_features=20, n_classes=2,
                           weights=[0.9, 0.1], random_state=42)

# Split the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Apply SMOTE for oversampling the minority class
smote = SMOTE(random_state=42)
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)

# Compute class weights to adjust for imbalance
class_weights = compute_class_weight('balanced', classes=[0, 1], y=y_train)
class_weights_dict = {0: class_weights[0], 1: class_weights[1]}

# Train a RandomForestClassifier using the resampled data
model = RandomForestClassifier(class_weight=class_weights_dict, random_state=42)
model.fit(X_train_smote, y_train_smote)

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

# Print classification report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.96      0.95      0.96       270
           1       0.59      0.63      0.61        30

    accuracy                           0.92       300
   macro avg       0.78      0.79      0.78       300
weighted avg       0.92      0.92      0.92       300
```

```
!pip install --upgrade threadpoolctl
```

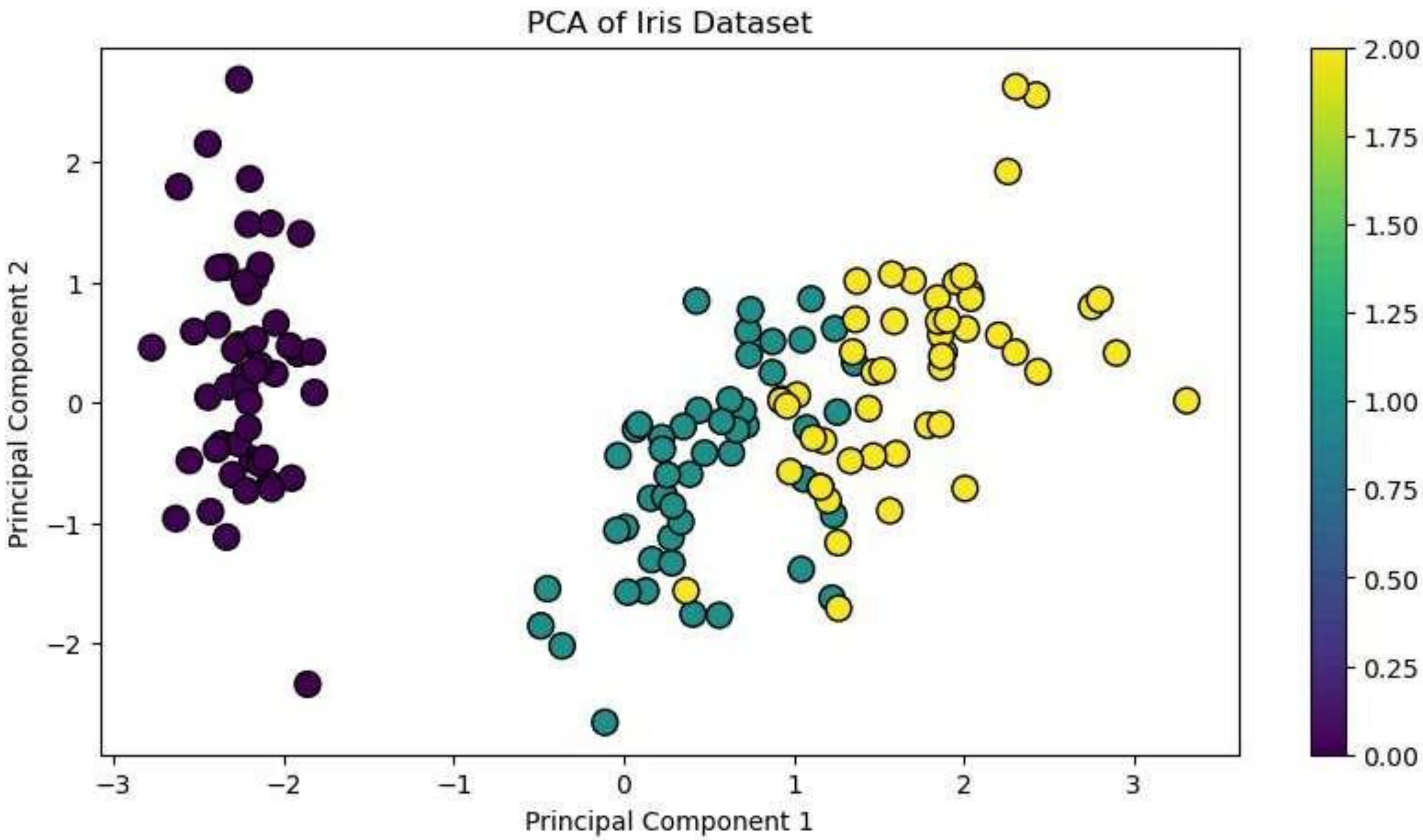
In [10]:

```python
#Question 2: Optimal Clusters for K-means

from sklearn.datasets import make_blobs
from sklearn.cluster import DBSCAN
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt

# Generate synthetic data
X, _ = make_blobs(n_samples=1000, n_features=2, centers=4, cluster_std=1.0, random_state=42)

# Apply DBSCAN
dbscan = DBSCAN(eps=0.5, min_samples=5).fit(X)
labels = dbscan.labels_

# Calculate the silhouette score
score = silhouette_score(X, labels)

# Plot the clusters
plt.figure(figsize=(10, 5))
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', s=50)
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title(f'DBSCAN Clustering (Silhouette Score: {score:.2f})')
plt.show()
```


DBSCAN Clustering (Silhouette Score: 0.50)
Feature 2
Feature 1
10
5
0
−5
−10
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5

In [11]:
```python
#Question 3:Dimensionality reduction
import numpy as np
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

# Load the dataset
data = load_iris()
X = data.data
y = data.target

# Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Explained variance
explained_variance = pca.explained_variance_ratio_
print(f"Explained variance by component: {explained_variance}")

# Plot the transformed data
plt.figure(figsize=(10, 5))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, cmap='viridis', edgecolor='k', s=100)
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('PCA of Iris Dataset')
plt.colorbar()
plt.show()
```

```
Explained variance by component: [0.72962445 0.22850762]
```

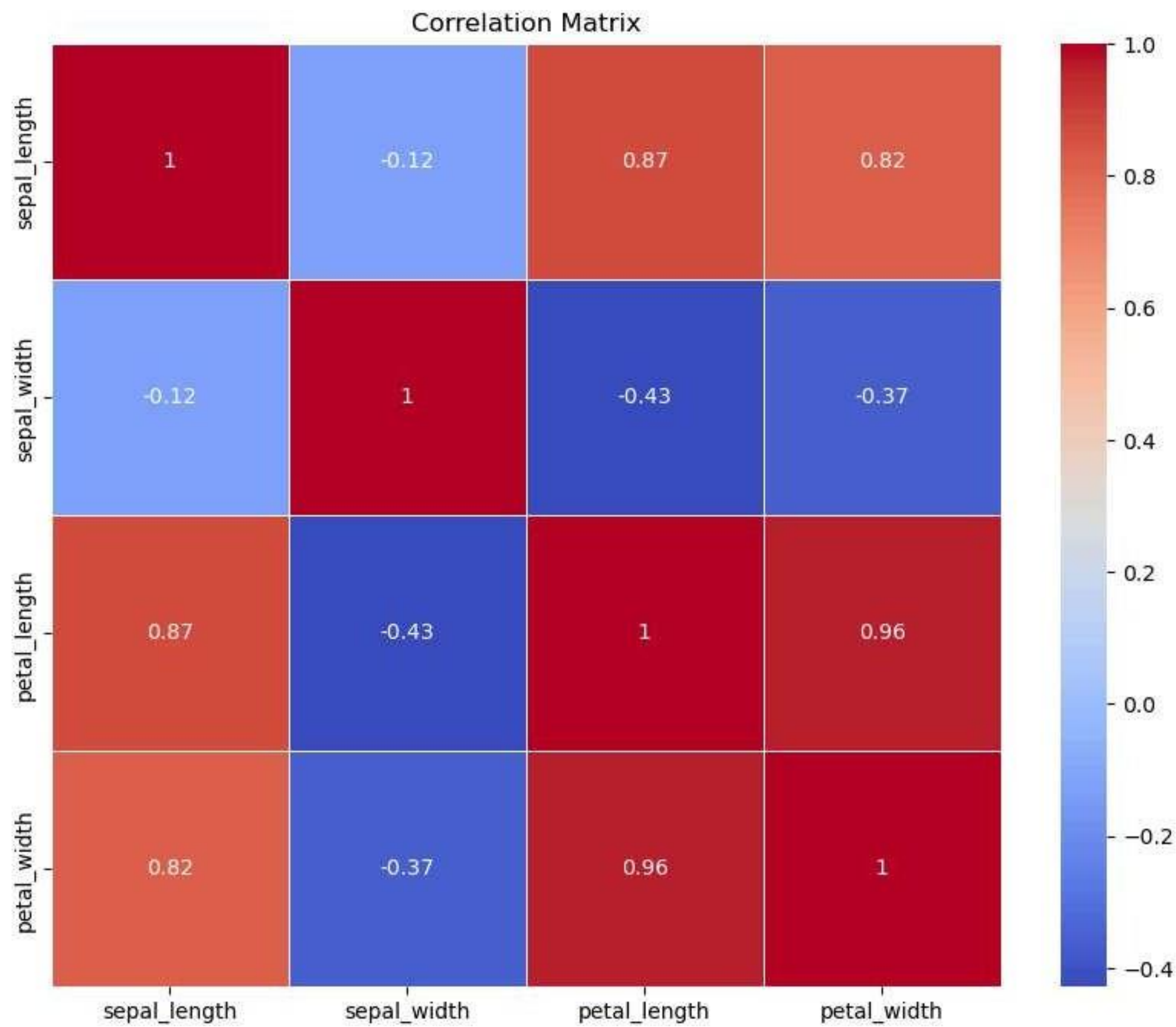


PCA of Iris Dataset
Principal Component 1
Principal Component 2
−3
−2
−1
0
1
2
3
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00

In [12]:

```python
#Question 4: Correlations in a Dataset
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Load a sample dataset
df = sns.load_dataset('iris')

# Calculate correlation matrix
correlation_matrix = df.corr()

# Print the correlation matrix
print(correlation_matrix)

# Visualize the correlation matrix using a heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Matrix')
plt.show()
```

```
              sepal_length  sepal_width  petal_length  petal_width
sepal_length      1.000000    -0.117570      0.871754     0.817941
sepal_width      -0.117570     1.000000     -0.428440    -0.366126
petal_length      0.871754    -0.428440      1.000000     0.962865
petal_width       0.817941    -0.366126      0.962865     1.000000
```



Correlation Matrix
sepal_length
sepal_width
petal_length
petal_width
1
-0.12
0.87
0.82
-0.12
1
-0.43
-0.37
0.87
-0.43
1
0.96
0.82
-0.37
0.96
1
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4

In [13]:

```python
#Question 5: Handling Missing Values
import pandas as pd
from sklearn.impute import SimpleImputer, KNNImputer

# Load a sample dataset with missing values
data = {'A': [1, 2, None, 4, 5],
        'B': [None, 2, 3, None, 5],
        'C': [1, None, None, 4, 5]}
df = pd.DataFrame(data)

# Mean Imputation using pandas
df_mean_imputed = df.fillna(df.mean())
print("Mean Imputation:\n", df_mean_imputed)

# KNN Imputation using scikit-learn
knn_imputer = KNNImputer(n_neighbors=2)
df_knn_imputed = pd.DataFrame(knn_imputer.fit_transform(df), columns=df.columns)
print("KNN Imputation:\n", df_knn_imputed)
```

```
Mean Imputation:
      A         B         C
0  1.0  3.333333  1.000000
1  2.0  2.000000  3.333333
2  3.0  3.000000  3.333333
3  4.0  3.333333  4.000000
4  5.0  5.000000  5.000000
KNN Imputation:
      A    B    C
0  1.0  3.5  1.0
1  2.0  2.0  2.5
2  3.5  3.0  4.5
3  4.0  3.5  4.0
4  5.0  5.0  5.0
```

In [15]:
```python
#Question 6: Detect and Remove Duplicates
#import pandas as pd

# Sample dataset with duplicates
data = {'A': [1, 2, 2, 4, 5],
        'B': [5, 6, 6, 8, 9],
        'C': [1, 2, 2, 4, 5]}
df = pd.DataFrame(data)

# Detect duplicates
duplicates = df[df.duplicated()]
print("Detected Duplicates:\n", duplicates)

# Remove duplicates
df_cleaned = df.drop_duplicates()
print("DataFrame after Removing Duplicates:\n", df_cleaned)
```

```
Detected Duplicates:
    A  B  C
2  2  6  2
DataFrame after Removing Duplicates:
    A  B  C
0  1  5  1
1  2  6  2
3  4  8  4
4  5  9  5
```

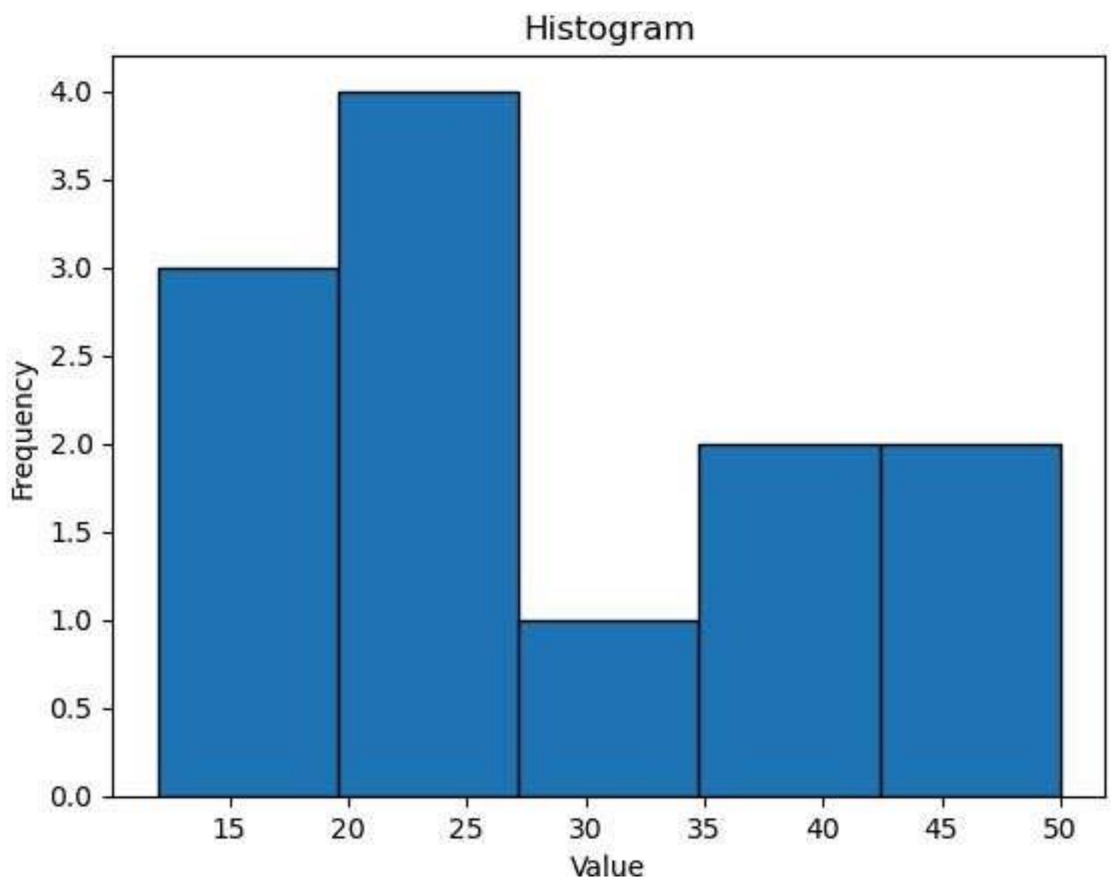
In [16]:
```python
#Question 7: Random Forest Regression for Housing Prices
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.datasets import load_boston

# Load the dataset
boston = load_boston()
df = pd.DataFrame(boston.data, columns=boston.feature_names)
df['PRICE'] = boston.target

# Handle missing values if any (for demonstration, Boston dataset has no missing values)
# df.fillna(df.mean(), inplace=True)

# Split the data into training and test sets
X = df.drop('PRICE', axis=1)
y = df['PRICE']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train the Random Forest Regressor
model = RandomForestRegressor(n_estimators=100, random_state=42)
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(f"Mean Squared Error: {mse}")
print(f"R^2 Score: {r2}")

# Feature importance
importances = model.feature_importances_
feature_importances = pd.DataFrame({'Feature': X.columns, 'Importance': importances})
feature_importances = feature_importances.sort_values(by='Importance', ascending=False)

print("Feature Importances:\n", feature_importances)
```

```
C:\Users\chait\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning:  Function load_bo
ston is deprecated; `load_boston` is deprecated in 1.0 and will be removed in 1.2.

    The Boston housing prices dataset has an ethical problem. You can refer to
    the documentation of this function for further details.

    The scikit-learn maintainers therefore strongly discourage the use of this
    dataset unless the purpose of the code is to study and educate about
    ethical issues in data science and machine learning.

    In this special case, you can fetch the dataset from the original
    source::

        import pandas as pd
        import numpy as np


        data_url = "http://lib.stat.cmu.edu/datasets/boston"
        raw_df = pd.read_csv(data_url, sep="\s+", skiprows=22, header=None)
        data = np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :2]])
        target = raw_df.values[1::2, 2]

    Alternative datasets include the California housing dataset (i.e.
    :func:`~sklearn.datasets.fetch_california_housing`) and the Ames housing
    dataset. You can load the datasets as follows::

        from sklearn.datasets import fetch_california_housing
        housing = fetch_california_housing()

    for the California housing dataset and::

        from sklearn.datasets import fetch_openml
        housing = fetch_openml(name="house_prices", as_frame=True)

    for the Ames housing dataset.

  warnings.warn(msg, category=FutureWarning)
```

```
Mean Squared Error: 7.901513892156864
R^2 Score: 0.8922527442109116
Feature Importances:
     Feature  Importance
5         RM    0.503845
12     LSTAT    0.309509
7        DIS    0.060549
0       CRIM    0.038062
10   PTRATIO    0.016313
9        TAX    0.015661
4        NOX    0.015544
6        AGE    0.013840
11         B    0.012154
2      INDUS    0.007953
8        RAD    0.003811
1         ZN    0.001756
3       CHAS    0.001004
```

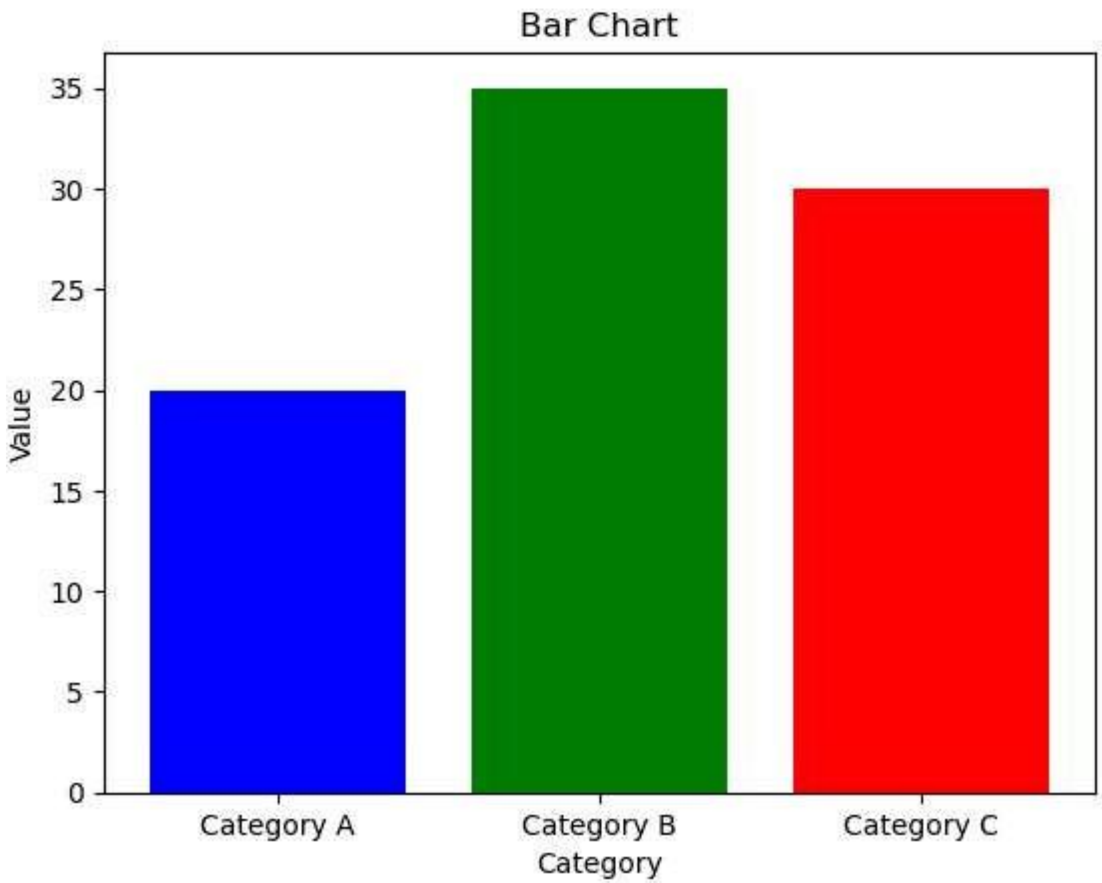
In [17]:

```python
#Question 8: Histogram, Bar Chart, and Pie Chart
import matplotlib.pyplot as plt
import pandas as pd

# Sample data
data = [12, 15, 17, 20, 22, 25, 27, 30, 35, 40, 45, 50]

# Plot histogram
plt.hist(data, bins=5, edgecolor='black')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.title('Histogram')
plt.show()
```

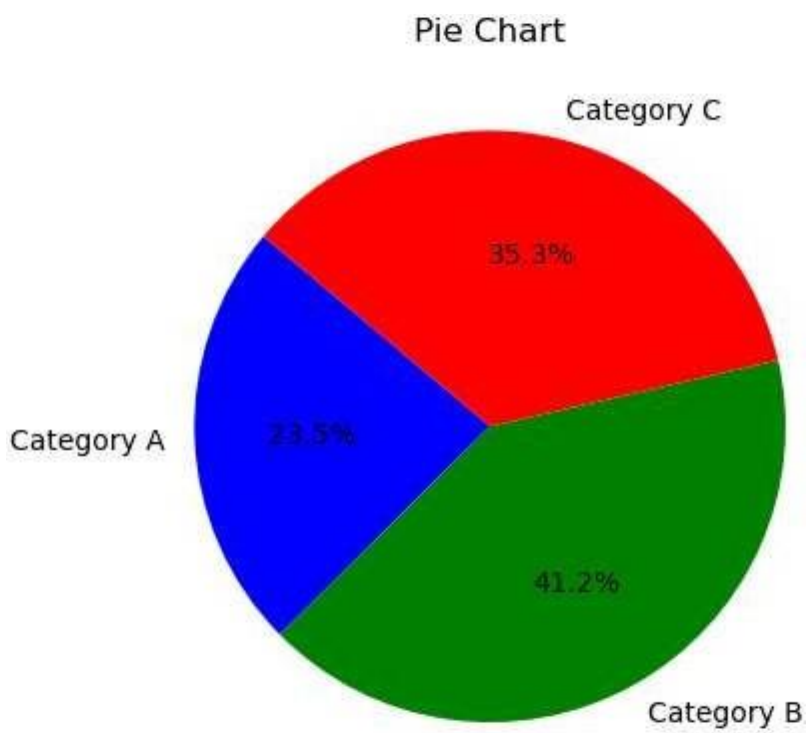


In [18]:

```
# Sample data
categories = ['Category A', 'Category B', 'Category C']
values = [20, 35, 30]

# Plot bar chart
plt.bar(categories, values, color=['blue', 'green', 'red'])
plt.xlabel('Category')
plt.ylabel('Value')
plt.title('Bar Chart')
plt.show()
```



In [19]:
```
# Sample data
categories = ['Category A', 'Category B', 'Category C']
values = [20, 35, 30]

# Plot pie chart
plt.pie(values, labels=categories, autopct='%1.1f%%', colors=['blue', 'green', 'red'], startangle=140)
plt.title('Pie Chart')
plt.show()
```



In [20]:

```python
#Question 9: Linear and Logistic Regression
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Sample dataset
data = {'Area': [650, 785, 1200, 1600, 2100],
        'Price': [80000, 105000, 180000, 230000, 300000]}
df = pd.DataFrame(data)

# Features and target variable
X = df[['Area']]
y = df['Price']

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train the model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions
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(f"Mean Squared Error: {mse}")
print(f"R^2 Score: {r2}")
```

```
Mean Squared Error: 4306026.030872749
R^2 Score: nan

C:\Users\chait\anaconda3\lib\site-packages\sklearn\metrics\_regression.py:796: UndefinedMetricWarning: R^2
score is not well-defined with less than two samples.
  warnings.warn(msg, UndefinedMetricWarning)
```

In [21]:
```python
#Question 10: Lag Features for Time-Series Data
import pandas as pd

# Sample time-series data
data = {'Date': pd.date_range(start='2022-01-01', periods=10, freq='D'),
        'Value': [10, 12, 14, 13, 15, 18, 20, 19, 21, 23]}
df = pd.DataFrame(data)
df.set_index('Date', inplace=True)

# Create lag features
df['Lag_1'] = df['Value'].shift(1)
df['Lag_2'] = df['Value'].shift(2)

# Handle missing values by filling with a suitable method (e.g., forward fill)
df.fillna(method='bfill', inplace=True)

print(df)
```

```
            Value  Lag_1  Lag_2
Date
2022-01-01     10   10.0   10.0
2022-01-02     12   10.0   10.0
2022-01-03     14   12.0   10.0
2022-01-04     13   14.0   12.0
2022-01-05     15   13.0   14.0
2022-01-06     18   15.0   13.0
2022-01-07     20   18.0   15.0
2022-01-08     19   20.0   18.0
2022-01-09     21   19.0   20.0
2022-01-10     23   21.0   19.0
```

In [ ]: