Exercise 5a Date: 22/01/2025

K Means Clustering - Two-dimensional dataset

Aim: To apply the K-Means clustering algorithm to a two-dimensional dataset, to identify clusters within the data.

Algorithm:

K-Means is an unsupervised learning algorithm used for clustering.

The goal is to partition the data into distinct clusters based on similarity, with each cluster representing data points that are more similar to each other than to those in other clusters.

The K-Means algorithm works by partitioning the dataset into `k` clusters, where each data point belongs to the cluster with the nearest mean.

The algorithm follows these steps:

1. **Initialization**: Choose `k` initial cluster centroids randomly from the dataset.
2. **Assignment Step**: Assign each data point to the nearest centroid. This creates `k` clusters.
3. **Update Step**: Recalculate the centroids of each cluster based on the data points assigned to it.
4. **Convergence**: Repeat the assignment and update steps until the centroids no longer change significantly or a maximum number of iterations is reached.

Step 1: Import necessary libraries

- Import necessary Python libraries such as `pandas`, `sklearn`, and `matplotlib` to handle data manipulation, model training, and visualization.

Step 2: Load the Dataset

- Load the Iris dataset using `load_iris` from `sklearn.datasets`. Select the first two features (Sepal Length and Sepal Width) as `X` and set the target variable `y`.

Step 3: Preprocess the Data

- Scale the feature data using `StandardScaler` to standardize it, ensuring that all features are normalized and have equal importance.

Step 4: Apply KMeans clustering

- Initialize the KMeans model with `n_clusters=3` and fit it to the scaled data. The algorithm assigns each data point to one of the three clusters.

Step 5: Add cluster labels to the dataset

- After performing the clustering, add the cluster labels to the dataset to identify which data points belong to which cluster.

Step 6: Visualize the results

- Plot the clustering results using the first two features (Sepal Length and Sepal Width) with each point colored by its assigned cluster. The plot shows how the K-Means algorithm has grouped the data into three clusters.

Importing necessary libraries

In [5]:
```
import pandas as pd
import warnings
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
warnings.filterwarnings("ignore")
```

Load the Dataset

In [6]:
```
iris = load_iris()
X = iris.data[:, :2]
y = iris.target
feature_names = iris.feature_names[:2]
target_names = iris.target_names
```

Preprocess the Data

In [7]:
```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

Applying KMeans clustering

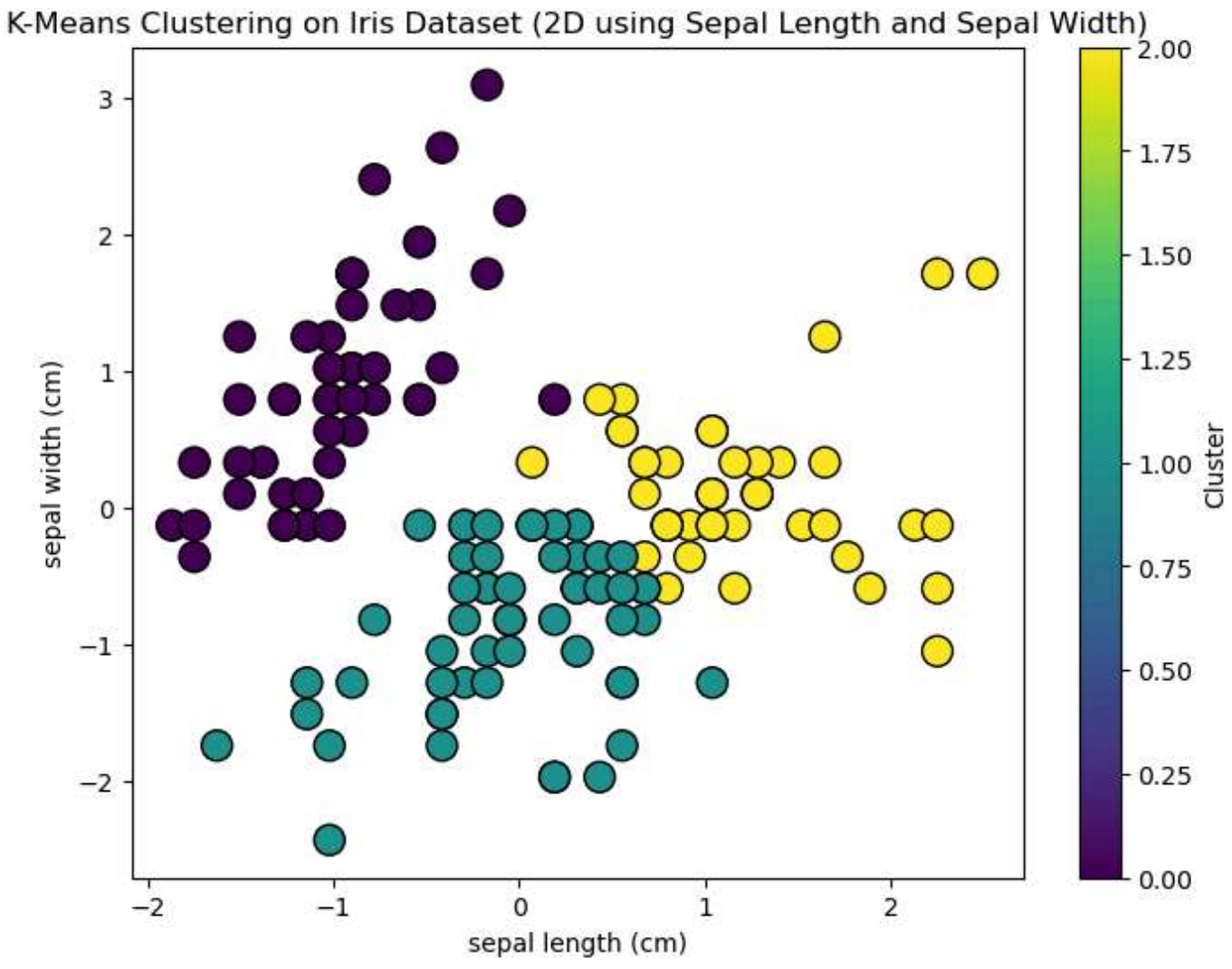
In [8]:
```
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X_scaled)
y_kmeans = kmeans.predict(X_scaled)
```

Adding cluster labels to the dataset

In [10]:
```
df = pd.DataFrame(X_scaled, columns=feature_names)
df['Cluster'] = y_kmeans
```

Plotting the clustering result (2D visualization)

In [11]:
```
plt.figure(figsize=(8, 6))
plt.scatter(df.iloc[:, 0], df.iloc[:, 1], c=df['Cluster'], cmap='viridis', edgecolor='k', s=150)
plt.xlabel(feature_names[0])
plt.ylabel(feature_names[1])
plt.title('K-Means Clustering on Iris Dataset (2D using Sepal Length and Sepal Width)')
plt.colorbar(label='Cluster')
plt.show()
```



Result

The K-Means algorithm successfully identified three clusters within the dataset that correspond to the three iris species.

Exercise 5b Date: 22/01/2025

K Means Clustering - Multidimensional dataset

Aim: To apply the K-Means clustering algorithm to a multidimensional dataset, to identify clusters within the data.

Algorithm:

K-Means is an unsupervised learning algorithm used for clustering.

The goal is to partition the data into distinct clusters based on similarity, with each cluster representing data points that are more similar to each other than to those in other clusters.

The K-Means algorithm works by partitioning the dataset into `k` clusters, where each data point belongs to the cluster with the nearest mean.

The algorithm follows these steps:

1. **Initialization**: Choose `k` initial cluster centroids randomly from the dataset.
2. **Assignment Step**: Assign each data point to the nearest centroid. This creates `k` clusters.
3. **Update Step**: Recalculate the centroids of each cluster based on the data points assigned to it.
4. **Convergence**: Repeat the assignment and update steps until the centroids no longer change significantly or a maximum number of iterations is reached.

Step 1: Import necessary libraries

- Import necessary Python libraries such as `pandas`, `sklearn`, and `matplotlib` to handle data manipulation, model training, and visualization.

Step 2: Load the dataset

- Load the Iris dataset using `load_iris` from `sklearn.datasets`. Split the data into features (`X`) and the target variable (`y`).

Step 3: Preprocess the data

- Scale the feature data to standardize it using `StandardScaler`, ensuring that all features are given equal importance by normalizing them.

Step 4: Apply K-Means clustering

- Initialize the KMeans model with `n_clusters=3` and fit it to the scaled data. The algorithm assigns each data point to one of the three clusters.

Step 5: Add cluster labels

- After the clustering is completed, add the cluster labels (assignments) to the dataset to identify which data points belong to each cluster.

Step 6: Visualize the results

- Plot the results of the clustering using the first two features of the Iris dataset, coloring each point according to its assigned cluster. The plot visualizes how the K-Means algorithm has grouped the data into three clusters.

Import necessary libraries

In [6]:
```
import pandas as pd
import warnings
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
warnings.filterwarnings("ignore")
```

Load the dataset

In [7]:
```
iris = load_iris()
X = iris.data
y = iris.target
feature_names = iris.feature_names
target_names = iris.target_names
```

Preprocess the data

In [8]:
```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

Applying K Means Clustering

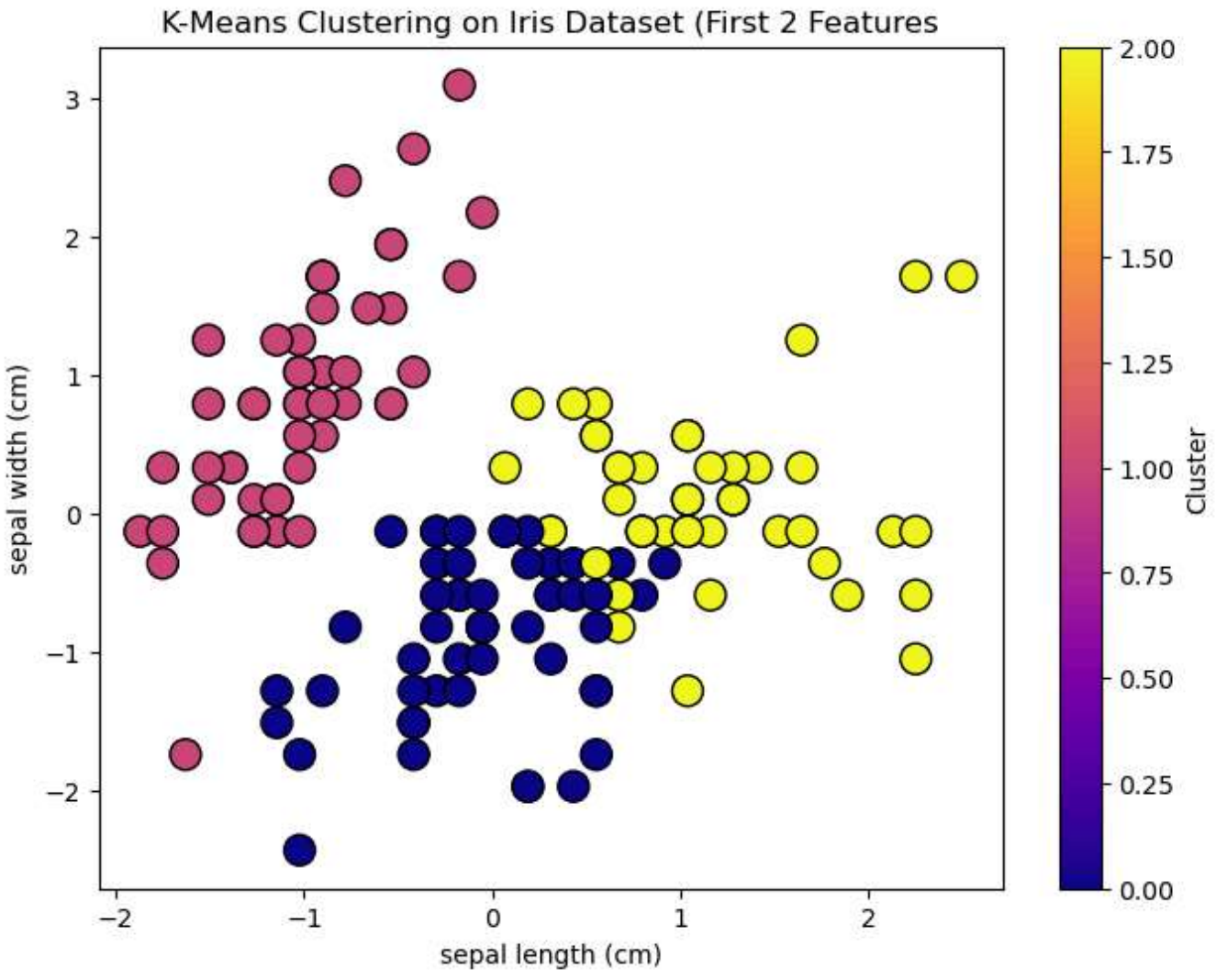
In [9]:
```
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X_scaled)
y_kmeans = kmeans.predict(X_scaled)
```

Adding cluster labels

In [10]:
```
df = pd.DataFrame(X_scaled, columns=feature_names)
df['Cluster'] = y_kmeans
```

Plot the clustering results

In [11]:
```
plt.figure(figsize=(8, 6))
plt.scatter(df.iloc[:, 0], df.iloc[:, 1], c=df['Cluster'], cmap='plasma', edgecolor='k', s=150)
plt.xlabel(feature_names[0])
plt.ylabel(feature_names[1])
plt.title('K-Means Clustering on Iris Dataset (First 2 Features')
plt.colorbar(label='Cluster')
plt.show()
```



Result

The K-Means algorithm successfully identified three clusters within the dataset that correspond to the three iris species.

Exercise 5c Date: 22/01/2025

Outlier Detection

Aim: To detect outliers in a dataset using the Interquartile Range (IQR) method and visualize them using a scatter plot.

Algorithm:

Outlier detection refers to the process of identifying data points that significantly differ from the rest of the dataset.

One common method of outlier detection is using the Interquartile Range (IQR).

Calculate the Interquartile Range (IQR): The IQR is the difference between the third quartile (Q3) and the first quartile (Q1). It measures the spread of the middle 50% of the data.

$$\text{IQR} = Q3 - Q1$$

Determine the lower and upper bounds for outliers: Outliers are defined as data points that lie below the lower bound or above the upper bound. These bounds are calculated as:

- **Lower Bound:**

$$\text{Lower Bound} = Q1 - 1.5 \times \text{IQR}$$

- **Upper Bound:**

$$\text{Upper Bound} = Q3 + 1.5 \times \text{IQR}$$

Identify outliers: Any data point that is less than the lower bound or greater than the upper bound is considered an outlier. These points are usually isolated and significantly different from the majority of the dataset.

Step 1: Import Libraries

- Import necessary Python libraries such as pandas for data manipulation, numpy for numerical operations, and matplotlib for visualization.

Step 2: Generate the Dataset

- Create a sample dataset using `numpy.random.randn()` to generate random data points and introduce outliers using `numpy.random.uniform()`.

Step 3: Prepare the Data

- Convert the data into a pandas DataFrame for easier manipulation and visualization. This allows for better handling of outliers and feature selection.

Step 4: Calculate the Quartiles

- Calculate the 25th percentile (Q1) and the 75th percentile (Q3) of the data using the `quantile()` function of pandas.

Step 5: Calculate the Interquartile Range (IQR)

- Calculate the IQR as the difference between Q3 and Q1:

$$\text{IQR} = Q3 - Q1$$

Step 6: Determine Lower and Upper Bounds for Outliers

- Calculate the lower and upper bounds using the IQR:
- Lower Bound:

$$\text{Lower Bound} = Q1 - 1.5 \times \text{IQR}$$

- Upper Bound:

$$\text{Upper Bound} = Q3 + 1.5 \times \text{IQR}$$

Step 7: Identify the Outliers

- Identify data points that fall outside of the lower and upper bounds as outliers. These are points that are significantly different from the majority of the dataset.

Step 8: Visualize the Data

- Plot the data points using `matplotlib` where inliers are shown in one color and outliers are shown in another color, visually identifying the outliers in the dataset.

Import necessary libraries

In [24]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

Generate a sample dataset

In [25]:
```
np.random.seed(42)
data = np.random.randn(700, 2) * 10 + 50
```

In [26]:
```
outliers = np.random.uniform(low=20, high=80, size=(10, 2))  # 10 random outliers
data_with_outliers = np.vstack([data, outliers])
```

Convert data to DataFrame

In [27]:
```
df = pd.DataFrame(data_with_outliers, columns=['Feature1', 'Feature2'])
```

In [28]:
```
df
```

Out[28]:

	Feature1	Feature2
0	54.967142	48.617357
1	56.476885	65.230299
2	47.658466	47.658630
3	65.792128	57.674347
4	45.305256	55.425600
...	...	...
705	37.726687	66.153389
706	57.479814	42.916378
707	32.341236	27.283185
708	56.900778	66.478027
709	58.634255	51.818128

710 rows × 2 columns

Calculate 25th and 75th percentile

In [29]:
```
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
```

Calculate Interquartile Range

In [30]:
```
IQR = Q3 - Q1
```

Set lower and upper bounds for outliers

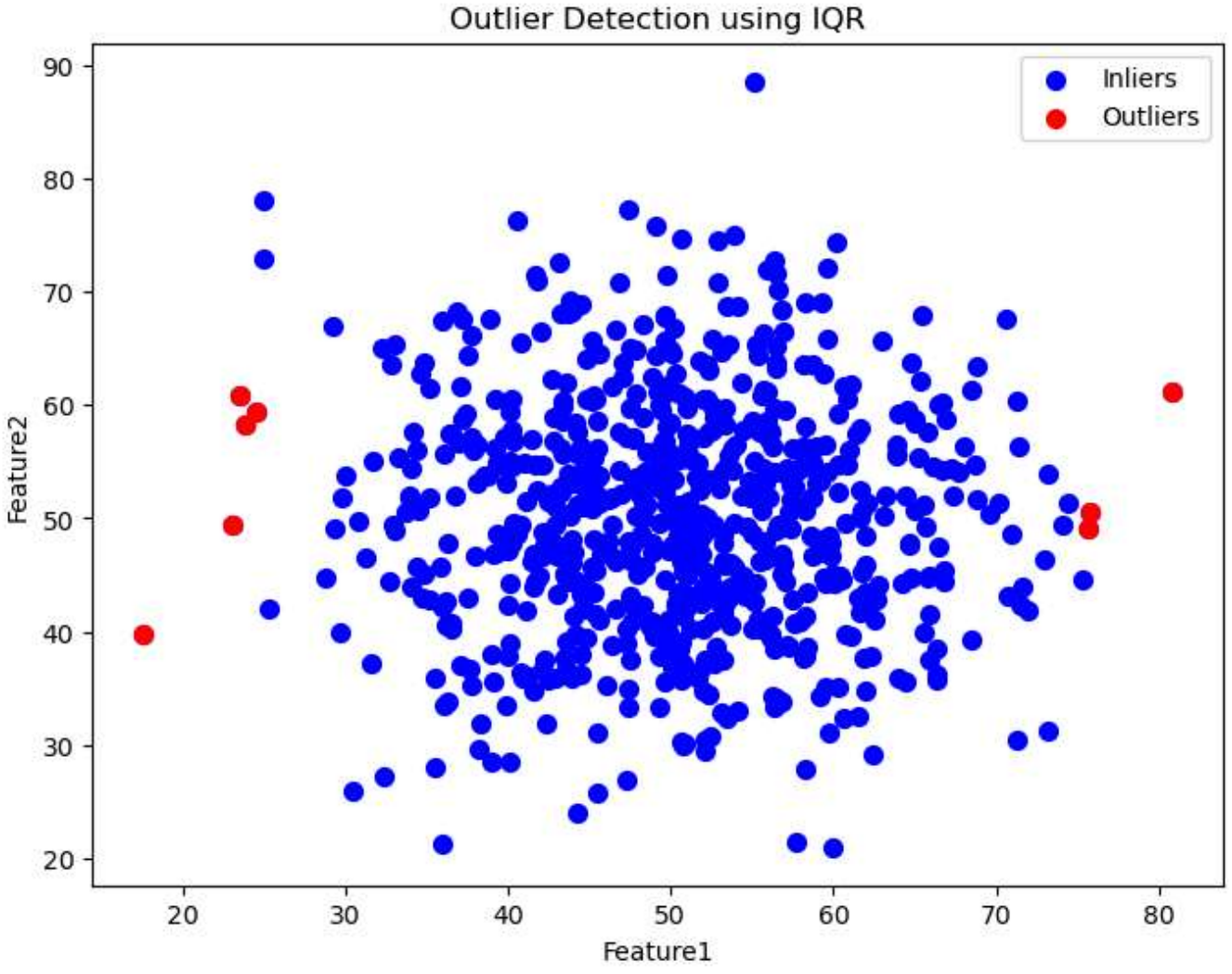
In [31]:
```
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
```

Identify the outliers

In [32]:
```
outliers = (df < lower_bound) | (df > upper_bound)
```

Plot the graph

In [33]:
```
plt.figure(figsize=(8, 6))
plt.scatter(df['Feature1'], df['Feature2'], color='blue', label='Inliers', s=50)
plt.scatter(df[outliers['Feature1']]['Feature1'], df[outliers['Feature1']]['Feature2'], color='red', label='Outliers', s=50)
plt.title('Outlier Detection using IQR')
plt.xlabel('Feature1')
plt.ylabel('Feature2')
plt.legend()
plt.show()
```



In [34]:
```
outliers_data = df[outliers.any(axis=1)]
print("Outliers detected:")
print(outliers_data)
```

```
Outliers detected:
      Feature1   Feature2
37   23.802549  58.219025
104  55.150477  88.527315
131  17.587327  39.756124
239  80.788808  61.195749
323  23.031134  49.457051
327  75.733598  50.592184
334  23.490302  60.915069
381  75.600845  49.039401
530  57.716987  21.514574
550  59.980101  21.037446
673  24.460789  59.343199
703  35.902786  21.256698
704  24.930300  78.071602
```

Result

Outlier detection was successfully performed on a dataset using the Interquartile Range (IQR) method. The identified outliers were isolated and visualized, highlighting 13 points as outliers.
