Lab Assignment: 5

Objective: To implement K-Means algorithm and apply on a dataset.

Name: Aakash Verma

Reg. No.: 24-08-26

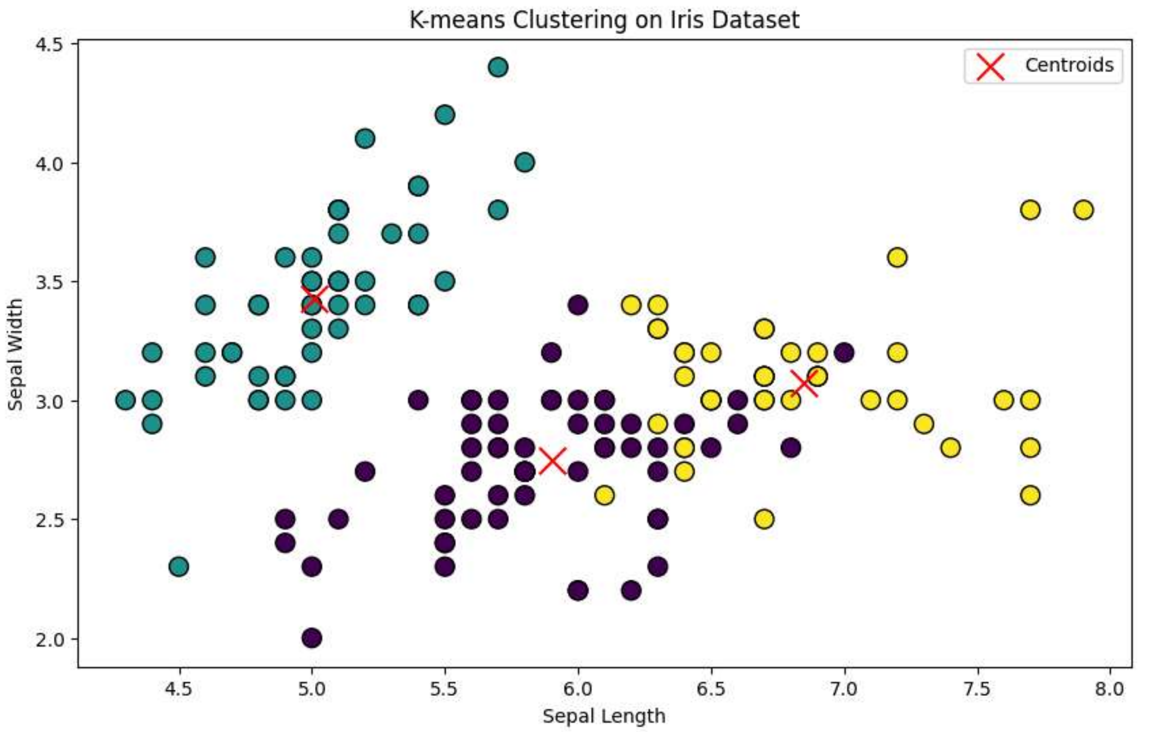
Course: M.Tech.(Cyber Security)

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

In [2]:
```
# Load the Iris dataset
iris = load_iris()
data = iris.data
labels = iris.target
```

In [3]:
```
# K-means implementation
class KMeans:
    def __init__(self, k=3, max_iter=100, tol=1e-4):
        self.k = k
        self.max_iter = max_iter
        self.tol = tol

    def fit(self, points):
        # Randomly initialize centroids
        np.random.seed(42)
        random_indices = np.random.choice(points.shape[0], self.k, replace=
        self.centroids = points[random_indices]

        for _ in range(self.max_iter):
            # Assign clusters
            distances = self._compute_distances(points)
            self.assignments = np.argmin(distances, axis=1)

            # Update centroids
            new_centroids = np.array([points[self.assignments == i].mean(ax

            # Check for convergence
            if np.all(np.abs(new_centroids - self.centroids) < self.tol):
                break

            self.centroids = new_centroids

    def _compute_distances(self, points):
        # Compute Euclidean distance from points to centroids
        return np.linalg.norm(points[:, np.newaxis] - self.centroids, axis=

    def predict(self, points):
        distances = self._compute_distances(points)
        return np.argmin(distances, axis=1)
```

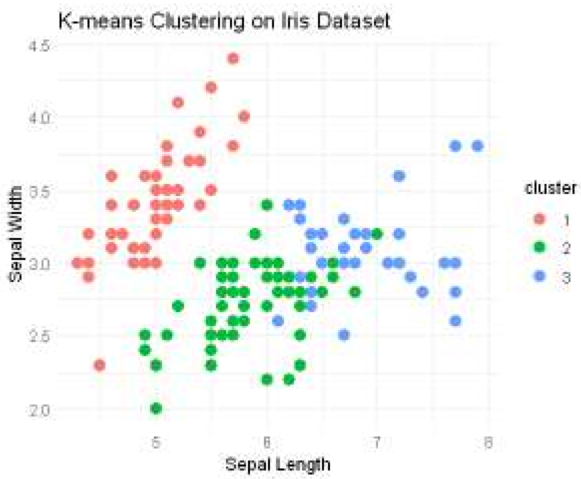
In [4]:
```
# Apply K-means on the Iris dataset
kmeans = KMeans(k=3)
kmeans.fit(data)

# Plot the results
plt.figure(figsize=(10, 6))
plt.scatter(data[:, 0], data[:, 1], c=kmeans.assignments, cmap='viridis', ma
plt.scatter(kmeans.centroids[:, 0], kmeans.centroids[:, 1], c='red', marker=
plt.title('K-means Clustering on Iris Dataset')
plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')
plt.legend()
plt.show()

# Show the assigned labels and centroids
print("Assigned cluster labels:\n", kmeans.assignments)
print("Centroids:\n", kmeans.centroids)
```




```
Assigned cluster labels:
 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1
 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 2 2 2 2 0 2 2 2 2
 2 2 0 0 2 2 2 2 0 2 0 2 0 2 2 0 0 2 2 2 2 2 0 2 2 2 2 0 2 2 2 0 2 2 2 0 2
 2 0]
Centroids:
 [[5.9016129  2.7483871  4.39354839 1.43387097]
 [5.006      3.428      1.462      0.246     ]
 [6.85       3.07368421 5.74210526 2.07105263]]
```

R code

In [ ]:
```
# Set plot size
options(repr.plot.width = 5, repr.plot.height = 4)  # Set width and height t

# Load necessary libraries
library(ggplot2)

# Load the iris dataset
data(iris)

# Set seed for reproducibility
set.seed(123)

# Perform K-means clustering
kmeans_result <- kmeans(iris[, -5], centers = 3)

# Add cluster assignments to the original dataset
iris$cluster <- as.factor(kmeans_result$cluster)

# Visualize the clusters using ggplot2
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width, color = cluster)) +
  geom_point(size = 3) +
  labs(title = "K-means Clustering on Iris Dataset",
       x = "Sepal Length",
       y = "Sepal Width") +
  theme_minimal()
```



Conclusion:

1. Naive Bayes effectively classified Iris species based on conditional probabilities, achieving strong accuracy due to its reliance on the assumption of feature independence.

2. The implementation highlighted Naive Bayes' simplicity and computational efficiency, making it an excellent choice for quick and effective classification tasks in machine learning.