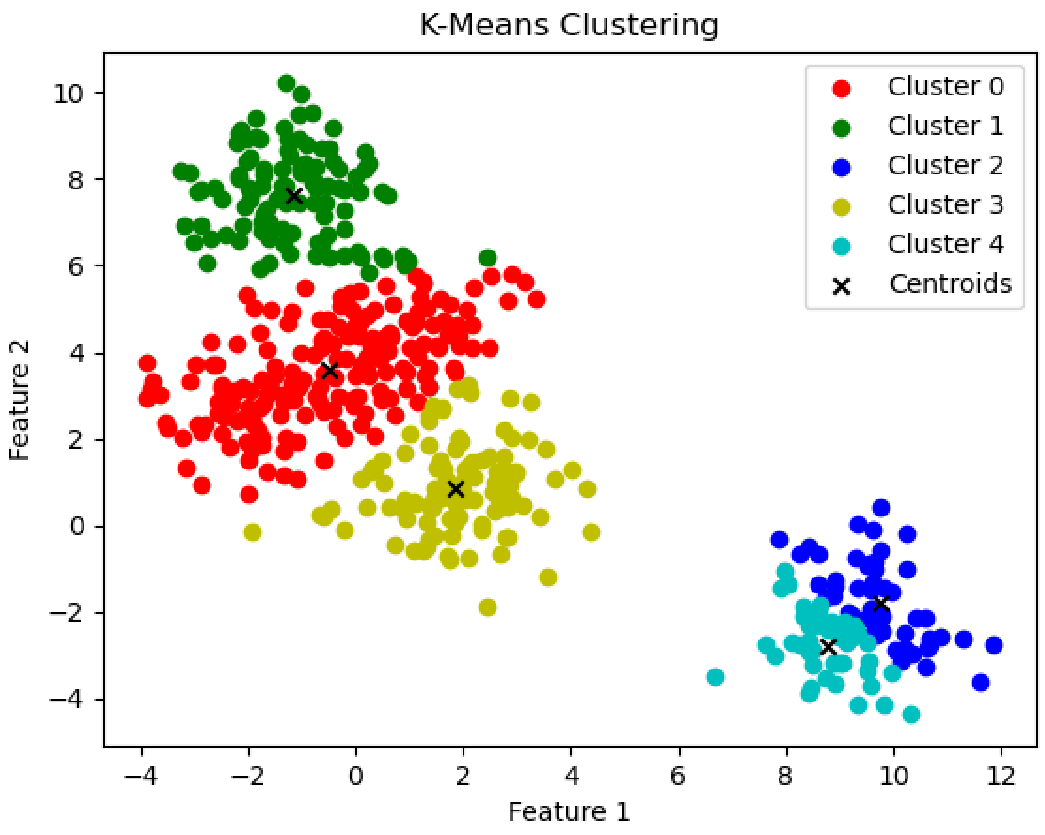
# K-MEANS CLUSTERING

```
import csv
import random
import matplotlib.pyplot as plt
def load_data(file_path):
    data = []
    with open(file_path, 'r') as file:
        reader = csv.reader(file)
        next(reader)  # Skip header
        for row in reader:
            data.append([float(row[0]), float(row[1])])
    return data
def initialize_centroids(data, k):
    return random.sample(data, k)
def euclidean_distance(point1, point2):
    return sum((x - y) ** 2 for x, y in zip(point1, point2)) ** 0.5
def assign_clusters(data, centroids):
    clusters = [[] for _ in centroids]
    for point in data:
        distances = [euclidean_distance(point, centroid) for centroid in centroids]
        closest_centroid_index = distances.index(min(distances))
        clusters[closest_centroid_index].append(point)
    return clusters
def recalculate_centroids(clusters):
    new_centroids = []
    for cluster in clusters:
        new_centroid = [sum(dim)/len(cluster) for dim in zip(*cluster)]
        new_centroids.append(new_centroid)
    return new_centroids
def has_converged(old_centroids, new_centroids, threshold=1e-4):
    total_movement = sum(euclidean_distance(old, new) for old, new in zip(old_centroids,
new_centroids))
```

```
    return total_movement < threshold


def k_means(data, k, max_iterations=100):
    centroids = initialize_centroids(data, k)
    for _ in range(max_iterations):
        clusters = assign_clusters(data, centroids)
        new_centroids = recalculate_centroids(clusters)
        if has_converged(centroids, new_centroids):
            break
        centroids = new_centroids
    return clusters, centroids
def plot_clusters(clusters, centroids):
    colors = ['r', 'g', 'b', 'y', 'c', 'm']
    for i, cluster in enumerate(clusters):
        cluster_points = list(zip(*cluster))
        plt.scatter(cluster_points[0], cluster_points[1], c=colors[i], label=f'Cluster {i}')
    centroid_points = list(zip(*centroids))
    plt.scatter(centroid_points[0], centroid_points[1], c='k', marker='x', label='Centroids')
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.legend()
    plt.title('K-Means Clustering')
    plt.show()
if __name__ == "__main__":
    data = load_data('kmeansdataset.csv')
    k =5
    clusters, centroids = k_means(data, k)
    print(f"Final centroids: {centroids}")
    for i, cluster in enumerate(clusters):
        print(f"Cluster {i}: {cluster}")
    plot_clusters(clusters, centroids)
```

OUTPUT:


K-Means Clustering
Cluster 0
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Centroids
Feature 1
Feature 2
−4
−2
0
2
4
6
8
10
12

# FINDING THE MISSING VALUES AND ACCURANCY OF TWO DIFFERENT DATASET

```
import pandas as pd
import numpy as np
def count_missing_values(df):
    missing_counts = df.isnull().sum()
    return missing_counts[missing_counts > 0]
def calculate_accuracy(y_true, y_pred):
    correct = np.sum(y_true == y_pred)
    total = len(y_true)
    accuracy = correct / total
    return accuracy
data = pd.read_csv('output.csv')
data.columns = data.columns.str.strip()
data['Anaemic'] = data['Anaemic'].map({'Yes': 1, 'No': 0})
missing_counts = count_missing_values(data)
if not missing_counts.empty:
    print("DATASET - 1 \n Missing Values:")
    print(missing_counts)
else:
    print("\n DATASET - 1 \n No missing values.")
X = data.drop(['Number', 'Sex', 'Anaemic'], axis=1)
y = data['Anaemic']
y_pred = np.zeros(len(y))
accuracy = calculate_accuracy(y, y_pred)
print("\nAccuracy:", accuracy)
def count_missing_values(df):
    missing_counts = df.isnull().sum()
    return missing_counts[missing_counts > 0]
def calculate_accuracy(y_true, y_pred):
    correct = np.sum(y_true == y_pred)
    total = len(y_true)
```

```
   accuracy = correct / total
    return accuracy
data = pd.read_csv('missing.csv')
data.columns = data.columns.str.strip()
data['Anaemic'] = data['Anaemic'].map({'Yes': 1, 'No': 0})
missing_counts = count_missing_values(data)
if not missing_counts.empty:
    print("DATASET - 2 \n Missing Values:")
    print(missing_counts)
else:
    print("No missing values.")
X = data.drop(['Number', 'Sex', 'Anaemic'], axis=1)
y = data['Anaemic']
y_pred = np.zeros(len(y))
accuracy = calculate_accuracy(y, y_pred)
print("Accuracy:", accuracy)
```

**OUTPUT:**

**DATASET - 1**

```
 No missing values.
Accuracy: 0.75
```

**DATASET - 2**

```
 Missing Values:
Sex            1
%Red Pixel      4
%Green pixel    4
%Blue pixel     4
Hb             3
Anaemic        3
dtype: int64
Accuracy: 0.7254901960784313
```

## REMOVING INCONSISTENT DATA FROM DATASET

```
import csv
import random
import matplotlib.pyplot as plt
def load_data(file_path):
    data = []
    with open(file_path, 'r') as file:
        reader = csv.reader(file)
        next(reader)  # Skip header
        for row in reader:
            data.append([float(row[0]), float(row[1])])
    return data
def initialize_centroids(data, k):
    return random.sample(data, k)
def euclidean_distance(point1, point2):
    return sum((x - y) ** 2 for x, y in zip(point1, point2)) ** 0
def assign_clusters(data, centroids):
    clusters = [[] for _ in centroids]
    for point in data:
        distances = [euclidean_distance(point, centroid) for centroid in centroids]
        closest_centroid_index = distances.index(min(distances))
        clusters[closest_centroid_index].append(point)
    return clusters
def recalculate_centroids(clusters):
    new_centroids = []
    for cluster in clusters:
        new_centroid = [sum(dim)/len(cluster) for dim in zip(*cluster)]
        new_centroids.append(new_centroid)
    return new_centroids
def has_converged(old_centroids, new_centroids, threshold=1e-4):
    total_movement = sum(euclidean_distance(old, new) for old, new in zip(old_centroids, 
new_centroids))
    return total_movement < threshold
```

```
def k_means(data, k, max_iterations=100):
    centroids = initialize_centroids(data, k)
    for _ in range(max_iterations):
        clusters = assign_clusters(data, centroids)
        new_centroids = recalculate_centroids(clusters)
        if has_converged(centroids, new_centroids):
            break
        centroids = new_centroids
    return clusters, centroids
def plot_clusters(clusters, centroids):
    colors = ['r', 'g', 'b', 'y', 'c', 'm']
    for i, cluster in enumerate(clusters):
        cluster_points = list(zip(*cluster))
        plt.scatter(cluster_points[0], cluster_points[1], c=colors[i], label=f'Cluster {i}')
    centroid_points = list(zip(*centroids))
    plt.scatter(centroid_points[0], centroid_points[1], c='k', marker='x', label='Centroids')
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.legend()
    plt.title('K-Means Clustering')
    plt.show()
if __name__ == "__main__":
    data = load_data('kmeansdataset.csv')
    k =5
    clusters, centroids = k_means(data, k)
    print(f"Final centroids: {centroids}")
    for i, cluster in enumerate(clusters):
        print(f"Cluster {i}: {cluster}")
    plot_clusters(clusters, centroids)
```

OUTPUT:

```
Cleaned data saved to “food_coded_cleaned_data.csv”
```

# BAYESIAN NETWORK FOR DIABETIC DATASET

```
import pandas as pd

import numpy as np

from scipy.stats import norm

data = pd.read_csv('diabetics.csv')

features = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI',
'DiabetesPedigreeFunction', 'Age']

target = 'Outcome'

def calculate_cpt(data, target, features):

    cpt = {}

    for feature in features:

        cpt[feature] = {}

        for value in data[target].unique():

            feature_values = data[data[target] == value][feature]

            mean, std = feature_values.mean(), feature_values.std()

            cpt[feature][value] = (mean, std)

    return cpt

cpt = calculate_cpt(data, target, features)

def predict_diabetes(evidence, cpt, data, target, features):

    probabilities = []

    for cls in data[target].unique():

        prob_cls = len(data[data[target] == cls]) / len(data)

        prob = prob_cls

        for feature in features:

            mean, std = cpt[feature][cls]

            value = evidence[feature]

            prob *= norm.pdf(value, loc=mean, scale=std)

        probabilities.append(prob)

    total_prob = sum(probabilities)

    normalized_probabilities = [prob / total_prob for prob in probabilities]

    return normalized_probabilities

def get_user_input():

    evidence = {}
    print("Please enter the following information:")
    for feature in features:
        while True:
            try:
                value = float(input(f"{feature}: "))
                evidence[feature] = value
                break
            except ValueError:
                print("Invalid input. Please enter a numerical value.")
    return evidence
evidence = get_user_input()
probabilities = predict_diabetes(evidence, cpt, data, target, features)
print(f"Probability of having diabetes: {probabilities[1]:.4f}")
print(f"Probability of not having diabetes: {probabilities[0]:.4f}")
predicted_class = np.argmax(probabilities)
print(f"Predicted class: {'Diabetic' if predicted_class == 1 else 'Not Diabetic'}")
```

OUTPUT:

```
Please enter the following information:
Pregnancies: 7
Glucose: 196
BloodPressure: 90
SkinThickness: 0
Insulin: 0
BMI: 39.8
DiabetesPedigreeFunction: 0.451
Age: 41
Probability of having diabetes: 0.0194
Probability of not having diabetes: 0.9806
Predicted class: Not Diabetic
```