In [3]:

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report,confusion_

# Load the Iris dataset
df = pd.read_csv('iris.csv')
df.info()

df.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   sepal.length  150 non-null    float64
 1   sepal.width   150 non-null    float64
 2   petal.length  150 non-null    float64
 3   petal.width   143 non-null    float64
 4   variety       150 non-null    object 
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

Out[3]:

	sepal.length	sepal.width	petal.length	petal.width	variety
0	5.1	3.5	1.4	NaN	Setosa
1	4.9	3.0	1.4	0.2	Setosa
2	4.7	3.2	1.3	0.2	Setosa
3	4.6	3.1	1.5	0.2	Setosa
4	5.0	3.6	1.4	0.2	Setosa

In [5]:

```python
print(df.isnull().sum())
```

```
sepal.length    0
sepal.width     0
petal.length    0
petal.width     7
variety         0
dtype: int64
```

In [6]:

```
# Handling missing values by replacing them with the column mean
imputer = SimpleImputer(strategy='mean')
df[['petal.width']] = imputer.fit_transform(df[['petal.width']])

#print("\nData after handling missing values:\n", df.head(15))

df.isnull().sum()
```

Out[6]:

```
sepal.length    0
sepal.width     0
petal.length    0
petal.width     0
variety         0
dtype: int64
```

In [7]:

```
# Prepare features and labels
X = df.drop(columns=['variety'])  # Features (sepal and petal measurements)
y = df['variety']  # Target variable (species of iris)

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




```

In [10]:

```
# standarise the features to scale values between -1 and 1
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
X_train
```

Out[10]:

```
array([[-1.47393679,  1.20365799, -1.56253475, -1.37183001],
       [-0.13307079,  2.99237573, -1.27600637, -1.0990868 ],
       [ 1.08589829,  0.08570939,  0.38585821, -0.00429936],
       [-1.23014297,  0.75647855, -1.2187007 , -0.00429936],
       [-1.7177306 ,  0.30929911, -1.39061772, -1.37183001],
       [ 0.59831066, -1.25582892,  0.72969227,  0.94648729],
       [ 0.72020757,  0.30929911,  0.44316389,  0.40100087],
       [-0.74255534,  0.98006827, -1.27600637, -1.37183001],
       [-0.98634915,  1.20365799, -1.33331205, -1.37183001],
       [-0.74255534,  2.32160658, -1.27600637, -1.50820162],
       [-0.01117388, -0.80864948,  0.78699794,  0.94648729],
       [ 0.23261993,  0.75647855,  0.44316389,  0.53737247],
       [ 1.08589829,  0.08570939,  0.55777524,  0.40100087],
       [-0.49876152,  1.87442714, -1.39061772, -1.0990868 ],
       [-0.49876152,  1.4272477 , -1.27600637, -1.37183001],
       [-0.37686461, -1.47941864, -0.01528151, -0.28085716],
       [ 0.59831066, -0.58505976,  0.78699794,  0.40100087],
       [ 0.72020757,  0.08570939,  1.01622064,  0.81011568],
       [ 0.96400139, -0.13788033,  0.38585821,  0.26462926],
       [ 1.69528284,  1.20365799,  1.3609547 ,  1.76471603],
```

In [11]:

```
# Create and train the KNN model (K=5)
knn = KNeighborsClassifier(n_neighbors=5,metric='euclidean')
knn.fit(X_train, y_train)
```

Out[11]:

```
KNeighborsClassifier(metric='euclidean')
```

In [12]:

```
# Predict on the test set
y_pred = knn.predict(X_test)
y_pred
```

Out[12]:

```
array(['Versicolor', 'Setosa', 'Virginica', 'Versicolor', 'Versicolor',
       'Setosa', 'Versicolor', 'Virginica', 'Versicolor', 'Versicolor',
       'Virginica', 'Setosa', 'Setosa', 'Setosa', 'Setosa', 'Versicolor',
       'Virginica', 'Versicolor', 'Versicolor', 'Virginica', 'Setosa',
       'Virginica', 'Setosa', 'Virginica', 'Virginica', 'Versicolor',
       'Virginica', 'Virginica', 'Setosa', 'Setosa'], dtype=object)
```

In [14]:

```
confusion_matrix(y_test,y_pred)
```

Out[14]:

```
array([[10,  0,  0],
       [ 0,  9,  0],
       [ 0,  1, 10]], dtype=int64)
```

In [17]:

```
# Evaluate model performance
accuracy = accuracy_score(y_test, y_pred)
print(f"\nModel Accuracy: {accuracy:.2f}")

```

```

Model Accuracy: 0.97
```

In [16]:

```
# Display detailed classification report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

      Setosa       1.00      1.00      1.00        10
  Versicolor       0.90      1.00      0.95         9
   Virginica       1.00      0.91      0.95        11

    accuracy                           0.97        30
   macro avg       0.97      0.97      0.97        30
weighted avg       0.97      0.97      0.97        30
```

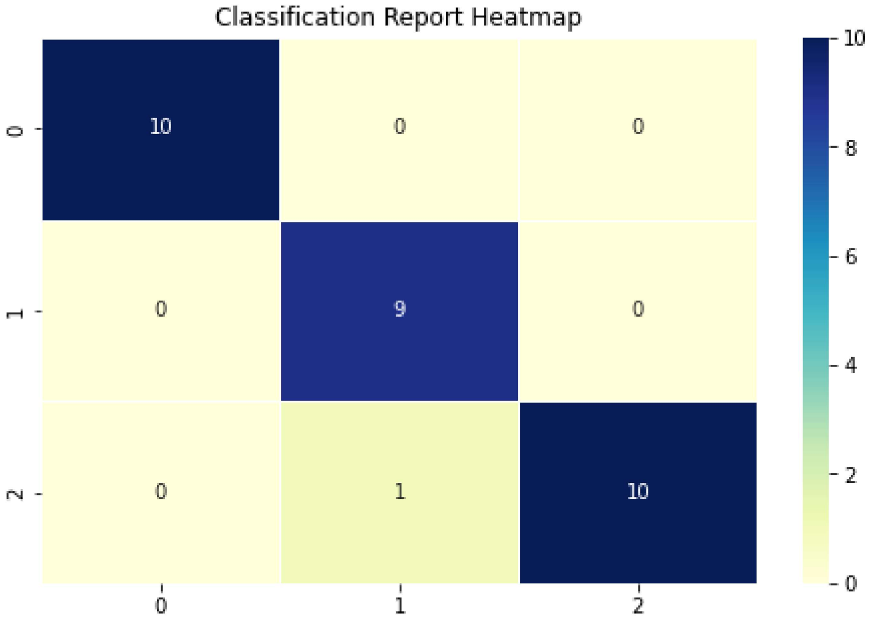
In [10]:

```python
import seaborn as sns
import matplotlib.pyplot as plt
# Generate classification report as a dictionary
class_report_dict = classification_report(y_test, y_pred, output_dict=True)

# Convert classification report to DataFrame
class_report_df = pd.DataFrame(class_report_dict).transpose()
```

In [20]:

```python
# Generate heatmap for classification report
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize=(8, 5))
sns.heatmap(confusion_matrix(y_test,y_pred), annot=True, cmap="YlGnBu", line
plt.title("Classification Report Heatmap")
plt.show()
```



In [28]:

```python
#predicting using custom input
sample= [[1.2,2.4,1.7,0.3]]
pred = knn.predict(sample)
pred
```

Out[28]:

```
array(['Virginica'], dtype=object)
```

In [ ]: