In [ ]:

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
import matplotlib.pyplot as plt
```

In [ ]:

```
# Load dataset (Iris dataset for this example)
data = load_iris()
X = data.data
y = data.target
```

In [ ]:

```
X.shape
```

In [ ]:

```
y.shape
```

In [ ]:

```
df = pd.DataFrame(X, columns=data['feature_names'])
df['target'] = y
```

In [ ]:

```
df.head()
```

In [ ]:

```
df.tail()
```

In [ ]:

```
df.describe()
```

count: The number of samples for each column.

mean: The average value for each feature.

std: Standard Deviation:A measure of data dispersion or variability.

Features like petal length (cm) have higher variability (1.765) compared to sepal width (cm) (0.435), indicating more spread in petal length.

min&max: The smallest and largest values in each column.

25%, 50%, 75% (Percentiles):

-> 25% (1st quartile): 25% of the data is less than this value.

-> 50% (Median): The middle value, where half the data is less than or equal to th
is.

-> 75% (3rd quartile): 75% of the data is less than this value.

For example, 50% of flowers have a sepal length less than 5.8 cm, and 75% have less than 6.4 cm.

In [ ]:

```
# Let's now take a look at the number of instances (rows) that belong to each class. We c
an view this as an absolute count.
df.groupby('target').size()
```

In [ ]:

```
X = data.data
y = data.target
```

In [ ]:

```
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

In [ ]:

```
# Feature scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [ ]:

```
X_train[:10]
```

In [ ]:

```
X_test[:10]
```

In [ ]:

```
# prompt: one-hot encoding y_train and y_test

from sklearn.preprocessing import OneHotEncoder

# Initialize OneHotEncoder
enc = OneHotEncoder(handle_unknown='ignore')

# Fit and transform the training data
y_train_encoded = enc.fit_transform(y_train.reshape(-1,1)).toarray()

# Transform the testing data
y_test_encoded = enc.transform(y_test.reshape(-1,1)).toarray()
```

In [ ]:

```
y_train_encoded[:10]
```

In [ ]:

```
y_test_encoded[:10]
```

In [ ]:

```
# KNN Classifier
```

```
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
```

In [ ]:

```
# Evaluation: Confusion Matrix and Classification Report
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)
print("\nClassification Report:")
print(classification_report(y_test, y_pred, target_names=data.target_names))
```

In confusion matrix,each row corresponds to the actual class, and each column corresponds to the predicted class.

Classification Report

$$
\begin{aligned}
precision &= \frac{TP}{TP + FP} \\
recall &= \frac{TP}{TP + FN} \\
F1 &= \frac{2 \times precision \times recall}{precision + recall} \\
accuracy &= \frac{TP + TN}{TP + FN + TN + FP} \\
specificity &= \frac{TN}{TN + FP}
\end{aligned}
$$

Support: The number of actual occurrences of the class in the dataset.

In [ ]:

```
cm = confusion_matrix(y_test, y_pred) # Assuming y_test and y_pred are defined

plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```

In [ ]:

```
#prediction
X_new = np.array([[5.1, 3.5, 1.4, 0.2]]) # Example new data point
y_pred_new = knn.predict(X_new)

print(f"Predicted class for new data point: {y_pred_new[0]}")
```

In [ ]:

```
# For multiple predictions:

X_new_multiple = np.array([[5.1, 3.5, 1.4, 0.2], [6.2, 2.9, 4.3, 1.3]])
y_pred_multiple = knn.predict(X_new_multiple)

print(f"Predicted classes for multiple data points: {y_pred_multiple}")
```