Aim:

Demonstrate and analyse the results of classification based on KNN Algorithm.

Program:

Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.

```python
import pandas as pd

from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy_score


# Load the Iris dataset

iris = load_iris()

data = pd.DataFrame(data=iris.data, columns=iris.feature_names)

data['target'] = iris.target


# Split the data into training and testing sets

X = data.drop('target', axis=1)

y = data['target']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)


# Train the k-NN classifier

k = 3  # Number of neighbors

knn = KNeighborsClassifier(n_neighbors=k)

knn.fit(X_train, y_train)
```

```
Out[5]:
KNeighborsClassifier(n_neighbors=3)
In [6]:
# Make predictions
y_pred = knn.predict(X_test)
In [7]:
# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
Accuracy: 1.00
In [9]:
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix: \n", cm)
Confusion Matrix:
 [[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
In [8]:
# Print both correct and wrong predictions
correct_predictions = []
wrong_predictions = []

for i in range(len(y_test)):
    if y_test.iloc[i] == y_pred[i]:
        correct_predictions.append((X_test.iloc[i].tolist(), y_test.iloc[i], y_pred[i]))
    else:
```

```
        wrong_predictions.append((X_test.iloc[i].tolist(), y_test.iloc[i], y_pred[i]))

print("\nCorrect Predictions:")
for cp in correct_predictions:
    print(f"Features: {cp[0]}, True Label: {cp[1]}, Predicted Label: {cp[2]}")

print("\nWrong Predictions:")
for wp in wrong_predictions:
    print(f"Features: {wp[0]}, True Label: {wp[1]}, Predicted Label: {wp[2]}")
```

```
Correct Predictions:
Features: [6.1, 2.8, 4.7, 1.2], True Label: 1, Predicted Label: 1
Features: [5.7, 3.8, 1.7, 0.3], True Label: 0, Predicted Label: 0
Features: [7.7, 2.6, 6.9, 2.3], True Label: 2, Predicted Label: 2
Features: [6.0, 2.9, 4.5, 1.5], True Label: 1, Predicted Label: 1
Features: [6.8, 2.8, 4.8, 1.4], True Label: 1, Predicted Label: 1
Features: [5.4, 3.4, 1.5, 0.4], True Label: 0, Predicted Label: 0
Features: [5.6, 2.9, 3.6, 1.3], True Label: 1, Predicted Label: 1
Features: [6.9, 3.1, 5.1, 2.3], True Label: 2, Predicted Label: 2
Features: [6.2, 2.2, 4.5, 1.5], True Label: 1, Predicted Label: 1
Features: [5.8, 2.7, 3.9, 1.2], True Label: 1, Predicted Label: 1
Features: [6.5, 3.2, 5.1, 2.0], True Label: 2, Predicted Label: 2
Features: [4.8, 3.0, 1.4, 0.1], True Label: 0, Predicted Label: 0
Features: [5.5, 3.5, 1.3, 0.2], True Label: 0, Predicted Label: 0
Features: [4.9, 3.1, 1.5, 0.1], True Label: 0, Predicted Label: 0
Features: [5.1, 3.8, 1.5, 0.3], True Label: 0, Predicted Label: 0
Features: [6.3, 3.3, 4.7, 1.6], True Label: 1, Predicted Label: 1
Features: [6.5, 3.0, 5.8, 2.2], True Label: 2, Predicted Label: 2
```

Features: [5.6, 2.5, 3.9, 1.1], True Label: 1, Predicted Label: 1

Features: [5.7, 2.8, 4.5, 1.3], True Label: 1, Predicted Label: 1

Features: [6.4, 2.8, 5.6, 2.2], True Label: 2, Predicted Label: 2

Features: [4.7, 3.2, 1.6, 0.2], True Label: 0, Predicted Label: 0

Features: [6.1, 3.0, 4.9, 1.8], True Label: 2, Predicted Label: 2

Features: [5.0, 3.4, 1.6, 0.4], True Label: 0, Predicted Label: 0

Features: [6.4, 2.8, 5.6, 2.1], True Label: 2, Predicted Label: 2

Features: [7.9, 3.8, 6.4, 2.0], True Label: 2, Predicted Label: 2

Features: [6.7, 3.0, 5.2, 2.3], True Label: 2, Predicted Label: 2

Features: [6.7, 2.5, 5.8, 1.8], True Label: 2, Predicted Label: 2

Features: [6.8, 3.2, 5.9, 2.3], True Label: 2, Predicted Label: 2

Features: [4.8, 3.0, 1.4, 0.3], True Label: 0, Predicted Label: 0

Features: [4.8, 3.1, 1.6, 0.2], True Label: 0, Predicted Label: 0

Wrong Predictions:

In [ ]: