9. 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.

**K-Nearest Neighbor Algorithm**

Training algorithm:
- For each training example \((x, f(x))\), add the example to the list training examples

Classification algorithm:
- Given a query instance \(x_q\) to be classified,
  - Let \(x_1 \ldots x_k\) denote the \(k\) instances from training examples that are nearest to \(x_q\)
  - Return

\[
\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} f(x_i)}{k}
\]

- Where, \(f(x_i)\) function to calculate the mean value of the \(k\) nearest training examples.

**Data Set:**

Iris Plants Dataset: Dataset contains 150 instances (50 in each of three classes)
Number of Attributes: 4 numeric, predictive attributes and the Class

<table>
<thead>
<tr>
<th>sepal-length</th>
<th>sepal-width</th>
<th>petal-length</th>
<th>petal-width</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5.1</td>
<td>3.5</td>
<td>1.4</td>
<td>0.2</td>
</tr>
<tr>
<td>1</td>
<td>4.9</td>
<td>3.0</td>
<td>1.4</td>
<td>0.2</td>
</tr>
<tr>
<td>2</td>
<td>4.7</td>
<td>3.2</td>
<td>1.3</td>
<td>0.2</td>
</tr>
<tr>
<td>3</td>
<td>4.6</td>
<td>3.1</td>
<td>1.5</td>
<td>0.2</td>
</tr>
<tr>
<td>4</td>
<td>5.0</td>
<td>3.6</td>
<td>1.4</td>
<td>0.2</td>
</tr>
</tbody>
</table>
Program:

```python
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn import datasets

# Iris Plants Dataset, dataset contains 150 (50 in each of three classes) Number of Attributes: 4 numeric, predictive attributes and the Class
iris=datasets.load_iris()

# The x variable contains the first four columns of the dataset (i.e. attributes) while y contains the labels.
x = iris.data
y = iris.target

print ('sepal-length', 'sepal-width', 'petal-length', 'petal-width')
print(x)
print('class: 0-Iris-Setosa, 1- Iris-Versicolour, 2- Iris-Virginica')
print(y)

# Splits the dataset into 70% train data and 30% test data. This means that out of total 150 records, the training set will contain 105 records and the test set contains 45 of those records
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.3)

# To Training the model and Nearest neighbors K=5
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(x_train, y_train)

# to make predictions on our test data
y_pred=classifier.predict(x_test)

# For evaluating an algorithm, confusion matrix, precision, recall and f1 score are the most commonly used metrics.
print('Confusion Matrix')
print(confusion_matrix(y_test,y_pred))
print('Accuracy Metrics')
print(classification_report(y_test,y_pred))
```
Output:

```
sepal-length sepal-width petal-length petal-width
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 .  .  .  .
 .  .  .  .
[6.2 3.4 5.4 2.3]
[5.9 3.  5.1 1.8]]

class: 0-Iris-Setosa, 1- Iris-Versicolour, 2- Iris-Virginica
[0 0 0 ........0 0 1 1 1 ........1 1 2 2 2 .......... 2 2]

Confusion Matrix
[[20  0  0]
 [ 0 10  0]
 [ 0  1 14]]

Accuracy Metrics

<table>
<thead>
<tr>
<th></th>
<th>Precision</th>
<th>recall</th>
<th>f1-score</th>
<th>support</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>20</td>
</tr>
<tr>
<td>1</td>
<td>0.91</td>
<td>1.00</td>
<td>0.95</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>1.00</td>
<td>0.93</td>
<td>0.97</td>
<td>15</td>
</tr>
<tr>
<td>avg / total</td>
<td>0.98</td>
<td>0.98</td>
<td>0.98</td>
<td>45</td>
</tr>
</tbody>
</table>
```
**Basic knowledge**

**Confusion Matrix**

- **True positives**: data points labelled as positive that are actually positive
- **False positives**: data points labelled as positive that are actually negative
- **True negatives**: data points labelled as negative that are actually negative
- **False negatives**: data points labelled as negative that are actually positive

**Accuracy**: how often is the classifier correct?

\[
\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{Total}}
\]

**F1-Score**:

\[
\text{F1 Score} = \frac{2 \times \text{TP}}{2 \times \text{TP} + \text{FP} + \text{FN}}
\]

**Support**: Total Predicted of Class.

\[
\text{Support} = \text{TP} + \text{FN}
\]
Example:

<table>
<thead>
<tr>
<th></th>
<th>GoldLabel_A</th>
<th>GoldLabel_B</th>
<th>GoldLabel_C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Predicted_A</td>
<td>30</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>Predicted_B</td>
<td>50</td>
<td>60</td>
<td>20</td>
</tr>
<tr>
<td>Predicted_C</td>
<td>20</td>
<td>20</td>
<td>80</td>
</tr>
<tr>
<td>TotalGoldLabel_A=100</td>
<td>TotalGoldLabel_B=100</td>
<td>TotalGoldLabel_C=100</td>
<td></td>
</tr>
<tr>
<td>TotalPredicted_A=60</td>
<td>TotalPredicted_B=120</td>
<td>TotalPredicted_C=120</td>
<td></td>
</tr>
</tbody>
</table>

This is an example confusion matrix for 3 labels: A, B and C

- Now, let us compute recall for Label A:
  \[ \text{Recall}_A = \frac{\text{TP}_A}{\text{TP}_A + \text{FN}_A} \]
  \[ = \frac{30}{30 + (20 + 10)} \]
  \[ = \frac{30}{60} \]
  \[ = 0.5 \]

- Now, let us compute precision for Label A:
  \[ \text{Precision}_A = \frac{\text{TP}_A}{\text{TP}_A + \text{FP}_A} \]
  \[ = \frac{30}{30 + 60} \]
  \[ = 0.33 \]

- Now, let us compute F1-score for Label A:
  \[ F1 \text{ Score} = \frac{2 \times \text{TP}}{2 \times \text{TP} + \text{FP} + \text{FN}} \]
  \[ = \frac{2 \times 30}{2 \times 30 + 60 + 100} \]
  \[ = 0.27 \]

- Support _A = \text{TP}_A + \text{FN}_A
  \[ = 30 + (20 + 10) \]
  \[ = 60 \]