```
1 import numpy as np
2 import pandas as pd
```

```
1 df = pd.read_csv('Iris.csv')
2 df.head()
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

```
1 df.groupby('Species').size()
```

```
Species
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
dtype: int64
```

```
1 feature_columns = ['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm','PetalWidthCm']
2 X = df[feature_columns].values
3 y = df['Species'].values
```

```
1 # Change Species coloumn to numbers
2
3 from sklearn.preprocessing import LabelEncoder
4 le = LabelEncoder()
5 y = le.fit_transform(y)
6 df['Species'] = y
```

```
1 # Split data into training and test
2
3 from sklearn.model_selection import train_test_split
4 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_stat
```

```
1 # Fitting clasifier to the Training set
2 # Loading libraries
3 from sklearn.neighbors import KNeighborsClassifier
4 from sklearn.metrics import confusion_matrix, accuracy_score
5 from sklearn.model_selection import cross_val_score
6
7 # Instantiate learning model (k = 3)
8 classifier = KNeighborsClassifier(n_neighbors=3)
9
```

```
# Fitting the model
classifier.fit(X_train, y_train)

# Predicting the Test set results
y_pred = classifier.predict(X_test)

print(y_pred)
```

```
[2 1 0 2 0 2 0 1 1 1 2 1 1 1 2 0 1 1 0 0 2 1 0 0 2 0 0 1 1 0]
```

```
# Define a function module to print results of ML Classifier Score

from sklearn.metrics import accuracy_score, confusion_matrix, precision_score, recall_

def print_score(clf, x_train, y_train, x_test, y_test, train = True):
    if train:
        pred = clf.predict(x_train)
        print("Train Result:\ ================================================")
        print(f"accuracy score: {accuracy_score(y_train, pred):.4f}\n")
        print("Classification Data:")
        print(f"Precision: {precision_score(y_train, pred, average=None, zero_divisior
        print (f"Recall Score: {recall_score(y_train, pred, average=None, zero_divisic
        print(f"Confusion_matrix:\n {confusion_matrix(y_train, clf.predict(x_train))}\
    elif train == False:
        pred = clf.predict(x_test)
        print("Test Result:\ ================================================")
        print(f"accuracy score: {accuracy_score(y_test, pred)}\n")
        print("Classification Data:")
        print(f"Precision: {precision_score(y_test, pred, average=None, zero_division=
        print (f"Recall Score: {recall_score(y_test, pred, average=None, zero_divisior
        print(f"Confusion_matrix:\n {confusion_matrix(y_test, clf.predict(x_test))}\n'
```

```
# Print results

print_score(classifier, X_train, y_train, X_test, y_test, train = True)
print_score(classifier, X_train, y_train, X_test, y_test, train = False)
```

```
Train Result:\ ================================================
accuracy score: 0.9500

Classification Data:
Precision: [1.         0.91891892 0.93181818]

Recall Score: [1.         0.91891892 0.93181818]

Confusion_matrix:
 [[39  0  0]
 [ 0 34  3]
 [ 0  3 41]]

Test Result:\ ================================================
accuracy score: 0.9666666666666667

Classification Data:
Precision: [1.         1.         0.85714286]
```

```
Recall Score: [1.         0.92307692 1.        ]

Confusion_matrix:
 [[11  0  0]
 [ 0 12  1]
 [ 0  0  6]]
```

```
# Using cross-validation for parameter tuning:

# creating list of K for KNN
k_list = list(range(1,50,2))

# creating list of cv scores
cv_scores = []

# perform 10-fold cross validation
for k in k_list:
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, X_train, y_train, cv=10, scoring='accuracy')
    cv_scores.append(scores.mean())

    print(cv_scores)
```

```
[0.95]
[0.95, 0.925]
[0.95, 0.925, 0.9333333333333333]
[0.95, 0.925, 0.9333333333333333, 0.95]
[0.95, 0.925, 0.9333333333333333, 0.95, 0.9583333333333333]
[0.95, 0.925, 0.9333333333333333, 0.95, 0.9583333333333333, 0.9583333333333333]
[0.95, 0.925, 0.9333333333333333, 0.95, 0.9583333333333333, 0.9583333333333333, 0.95]
[0.95, 0.925, 0.9333333333333333, 0.95, 0.9583333333333333, 0.9583333333333333, 0.95,
[0.95, 0.925, 0.9333333333333333, 0.95, 0.9583333333333333, 0.9583333333333333, 0.95,
[0.95, 0.925, 0.9333333333333333, 0.95, 0.9583333333333333, 0.9583333333333333, 0.95,
[0.95, 0.925, 0.9333333333333333, 0.95, 0.9583333333333333, 0.9583333333333333, 0.95,
[0.95, 0.925, 0.9333333333333333, 0.95, 0.9583333333333333, 0.9583333333333333, 0.95,
[0.95, 0.925, 0.9333333333333333, 0.95, 0.9583333333333333, 0.9583333333333333, 0.95,
[0.95, 0.925, 0.9333333333333333, 0.95, 0.9583333333333333, 0.9583333333333333, 0.95,
[0.95, 0.925, 0.9333333333333333, 0.95, 0.9583333333333333, 0.9583333333333333, 0.95,
[0.95, 0.925, 0.9333333333333333, 0.95, 0.9583333333333333, 0.9583333333333333, 0.95,
[0.95, 0.925, 0.9333333333333333, 0.95, 0.9583333333333333, 0.9583333333333333, 0.95,
[0.95, 0.925, 0.9333333333333333, 0.95, 0.9583333333333333, 0.9583333333333333, 0.95,
[0.95, 0.925, 0.9333333333333333, 0.95, 0.9583333333333333, 0.9583333333333333, 0.95,
[0.95, 0.925, 0.9333333333333333, 0.95, 0.9583333333333333, 0.9583333333333333, 0.95,
[0.95, 0.925, 0.9333333333333333, 0.95, 0.9583333333333333, 0.9583333333333333, 0.95,
[0.95, 0.925, 0.9333333333333333, 0.95, 0.9583333333333333, 0.9583333333333333, 0.95,
[0.95, 0.925, 0.9333333333333333, 0.95, 0.9583333333333333, 0.9583333333333333, 0.95,
[0.95, 0.925, 0.9333333333333333, 0.95, 0.9583333333333333, 0.9583333333333333, 0.95,
[0.95, 0.925, 0.9333333333333333, 0.95, 0.9583333333333333, 0.9583333333333333, 0.95,
```

```
# changing to misclassification error
import matplotlib.pyplot as plt
import seaborn as sns
MSE = [1 - x for x in cv_scores]

```

```
plt.figure()
plt.figure(figsize=(15,10))
plt.title('The optimal number of neighbors', fontsize=20, fontweight='bold')
plt.xlabel('Number of Neighbors K', fontsize=15)
plt.ylabel('Misclassification Error', fontsize=15)
sns.set_style("whitegrid")
plt.plot(k_list, MSE)

plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```