Exercise 2.1

1.1 Problem Statement:

Implement CART algorithm for decision tree learning. Use an iris data set for building the decision tree and apply this knowledge to classify a new sample.

1.2 Description of Machine learning Algorithm:

Decision Trees are an important type of algorithm for predictive modelling machine learning. Decision Trees are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.

CART is a decision tree algorithm and it stands for Classification and Regression Trees. CART can be used for classification or regression predictive modelling problems. The representation for the CART model is a binary tree. It builds decision trees based on the attribute selection measure "Gini's Impurity Index".
CART is an umbrella word that refers to the following types of decision trees:

Classification Trees: These are used to forecast the value of a categorical target variable.

Regression trees: These are used to forecast the value of a continuous target variable.

Advantages of Decision Trees:

- Decision trees are simple to understand, interpret, visualize.
- Decision trees implicitly perform variable screening or feature selection.
- Decision trees require little data preparation.
- Decision trees can handle both numerical and categorical data.
- Decision trees can also handle single-class and multi-class classification problems.
- Nonlinear relationships between parameters do not affect tree performance.
- Uses a white box model. If a given situation is observable in a model, the explanation for the condition is easily explained by Boolean logic.
- The cost of using the tree (i.e., predicting data) is logarithmic in the number of data points used to train the tree.
- Possible to validate a model using statistical tests. That makes it possible to account for the reliability of the model.
- Performs well even if its assumptions are somewhat violated by the true model from which the data were generated.

Disadvantages of Decision Trees:

- Decision-tree learners can create over-complex trees that do not generalize the data well. This is called overfitting. Mechanisms such as pruning, setting the minimum number of samples required at a leaf node or setting the maximum depth of the tree are necessary to avoid this problem.
- Decision trees can be unstable because small variations in the data might result in a completely different tree being generated. This is called variance, which needs to be lowered by methods like bagging, boosting etc.

- Predictions of decision trees are neither smooth nor continuous, but are piecewise constant approximations. Therefore, they are not good at extrapolation.
- The problem of learning an optimal decision tree is known to be NP-complete under several aspects of optimality and even for simple concepts. Consequently, practical decision-tree learning algorithms are based on heuristic algorithms such as the greedy algorithm where locally optimal decisions are made at each node. Such algorithms cannot guarantee to return the globally optimal decision tree. This can be mitigated by training multiple trees in an ensemble learner, where the features and samples are randomly sampled with replacement.
- There are concepts that are hard to learn because decision trees do not express them easily, such as XOR, parity or multiplexer problems.
- Decision tree learners create biased trees if some classes dominate. It is therefore recommended to balance the data set prior to fitting with the decision tree.

Advantages of CART algorithm:

- CART can handle missing values in the data.
- CART is not significantly impacted by outliers in the input variables.
- CART is non-parametric; Thus it does not depend on information from a certain sort of distribution.
- CART supports growing a full decision tree and further post-pruning the decision tree so that the probability that important structure in the data set will be overlooked by stopping too soon is minimized.
- CART combines both testing with a test data set and cross-validation to more precisely measure the goodness of fit.
- CART allows to utilize the same attributes many times in various regions of the tree. This skill is capable of revealing intricate interdependencies between groups of variables.
- CART can be used in conjunction with other prediction methods to select the input set of variables.

1.3 Description of the Dataset:

Title of the dataset: Iris Plants Database

The Iris Dataset contains information of three species of Iris flowers (Iris setosa, Iris virginica and Iris versicolor). The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant. One class is linearly separable from the other 2; the latter are NOT linearly separable from each other.

Data Set Characteristics: Multivariate

Area: Life Sciences

Number of samples (or instances) in the dataset: 150

Number of attributes (or features): 05 Attribute

Information:

1. sepal length in cm
2. sepal width in cm
3. petal length in cm
4. petal width in cm

5. class:
   - -- Iris Setosa
   - -- Iris Versicolour
   - -- Iris Virginica
6. Number of samples of each species of iris flowers:

   Class Distribution: 33.3% for each of 3 classes.

   50 (Setosa), 50 (Versicolor), 50 (Virginica)
7. Predicted attribute: class of iris plant.
8. Missing Attribute Values: None

Feature Name	Units of measurement	Datatype	Description
sepal length	Centimeters	Real (Numerical)	Length of Iris flower's sepal
sepal width length	Centimeters	Real (Numerical)	Width of Iris flower's sepal
petal length	Centimeters	Real (Numerical)	Length of Iris flower's petal
petal width length	Centimeters	Real (Numerical)	Width of Iris flower's petal
variety	Variety of species [Setosa, Virginica, Versicolor]	Object (Categorical)	Variety of the species of Iris flower

1.4 Data Preprocessing and Exploratory Data Analysis (EDA):

Data preprocessing is the process of transforming raw data into an understandable format. It is also an important step in data mining as we cannot work with raw data. The quality of the data should be checked before applying machine learning or data mining algorithms.

Major Tasks in Data Preprocessing:

1. Data cleaning
2. Data integration
3. Data reduction
4. Data transformation

Exploratory data analysis (EDA) is a technique that data professionals can use to understand a dataset before they start to model it. Some people refer to EDA as data exploration. The goal of conducting EDA is to determine the characteristics of the dataset. Conducting EDA can help data analysts make predictions and assumptions about data. Often, EDA involves data visualization, including creating graphs like histograms, scatter plots and box plots.

Major Tasks in EDA:

1. Observe your dataset
2. Find any missing values
3. Categorize your values
4. Find the shape of your dataset
5. Identify relationships in your dataset
6. Locate any outliers in your dataset

1.5 Machine Learning Package Used for Model building:

The scikit-learn (formerly scikits.learn and also known as sklearn) is a free software machine learning library for the Python programming language. It provides simple and efficient tools for predictive data analysis. It features various classification, regression and clustering algorithms including support-vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

The sklearn.model_selection.train_test_split() method splits arrays or matrices into random train and test subsets. The parameters for the method are as follows:

```
sklearn.model_selection.train_test_split(*arrays, test_size=None, train_size=None, random_state=Non e,
shuffle=True, stratify=None)
```

It returns lists containing train-test split of inputs.

The sklearn.tree.DecisionTreeClassifier is a class capable of performing multi-class classification on a dataset. It takes as input two arrays: an array X, holding the training samples, and an array Y holding the class labels for the training samples. In case that there are multiple classes with the same and highest probability, the classifier will predict the class with the lowest index amongst those classes. As an alternative to outputting a specific class, the probability of each class can be predicted, which is the fraction of training samples of the class in a leaf. DecisionTreeClassifier is capable of both binary (where the labels are [-1, 1]) classification and multiclass (where the labels are [0, …, K-1]) classification. The parameters of the DecisionTreeClassifier class are as follows:

```
class sklearn.tree.DecisionTreeClassifier(*, criterion='gini', splitter='best', max_depth=None, min_samp
les_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=N one,
max_leaf_nodes=None, min_impurity_decrease=0.0, class_weight=None, ccp_alpha=0.0)
```

The default values for the parameters controlling the size of the trees (e.g. max_depth, min_samples_leaf, etc.) lead to fully grown and unpruned trees which can potentially be very large on some data sets.

Limitations of sklearn.tree.DecisionTreeClassifier class:

- scikit-learn implementation does not support categorical variables for now.
- scikit-learn implementation can build only CART trees for now.

The DecisionTreeClassifier.fit() method builds a decision tree classifier from the training set (X, y).
The parameters of the fit() method are as follows:

```
fit(X, y, sample_weight=None, check_input=True, X_idx_sorted='deprecated')
```

It returns an fitted Decision Tree estimator.
The DecisionTreeClassifier.predict() method predicts class or regression value for X. The parameters of the predict() method are as follows:

```
predict(X, check_input=True)
```

For a classification model, the predicted class for each sample in X is returned. For a regression model, the predicted value based on X is returned.

The sklearn.tree class supports visualization of decision trees. Once a Decision Tree Classifier is built it can be visualized in test representation using sklearn.tree.export_text() method. The parameters for the method are as follows:

sklearn.tree.export_text(decision_tree, *, feature_names=None, max_depth=10, spacing=3, decimals=2 , show_weights=False)

Alternatively, the decision tree can be plotted sklearn.tree.plot_tree() method. The parameters for the method are as follows:

sklearn.tree.plot_tree(decision_tree, *, max_depth=None, feature_names=None, class_names=None, label='all', filled=False, impurity=True, node_ids=False, proportion=False, rounded=False, precision=3, ax=None, fontsize=None)

Once a Decision Tree Classifier is built, it can be evaluated for performance. The sklearn.metrics module implements several loss, score, and utility functions to measure classification performance. The sklearn.metrics.accuracy_score() method computes subset accuracy: the set of labels predicted for a sample must exactly match the corresponding set of labels in y_true. The parameters for the method are as follows:

sklearn.metrics.accuracy_score(y_true, y_pred, *, normalize=True, sample_weight=None)

1.6 Implementation:

```
from google.colab import files
uploaded = files.upload()
#import Database
import pandas as pd
df=pd.read_csv('Iris.csv')
df
```

```
   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
 ...   ...    ...    ...    ...    ...             ...
145    146    6.7    3.0    5.2    2.3  Iris-virginica
146    147    6.3    2.5    5.0    1.9  Iris-virginica
147    148    6.5    3.0    5.2    2.0  Iris-virginica
148    149    6.2    3.4    5.4    2.3  Iris-virginica
149    150    5.9    3.0    5.1    1.8  Iris-virginica
150 rows × 6 columns
```

```
df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             150 non-null    int64
 1   SepalLengthCm  150 non-null    float64
 2   SepalWidthCm   150 non-null    float64
 3   PetalLengthCm  150 non-null    float64
 4   PetalWidthCm   150 non-null    float64
 5   Species        150 non-null    object
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
```

Defining columns and target values:

```
from sklearn.datasets import load_iris
iris = load_iris()
X = iris.data
y = iris.target
feature_names = iris.feature_names
target_names = iris.target_names
print("Feature names:", feature_names)
print("Target names:", target_names)
print("\nFirst 10 rows of X:\n", X[:10])
Feature names: ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)',
'petal width (cm)']
Target names: ['setosa' 'versicolor' 'virginica']
First 10 rows of X:
 [[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]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]]
```

Splitting the dataset into training and testing with 70% and 30%:

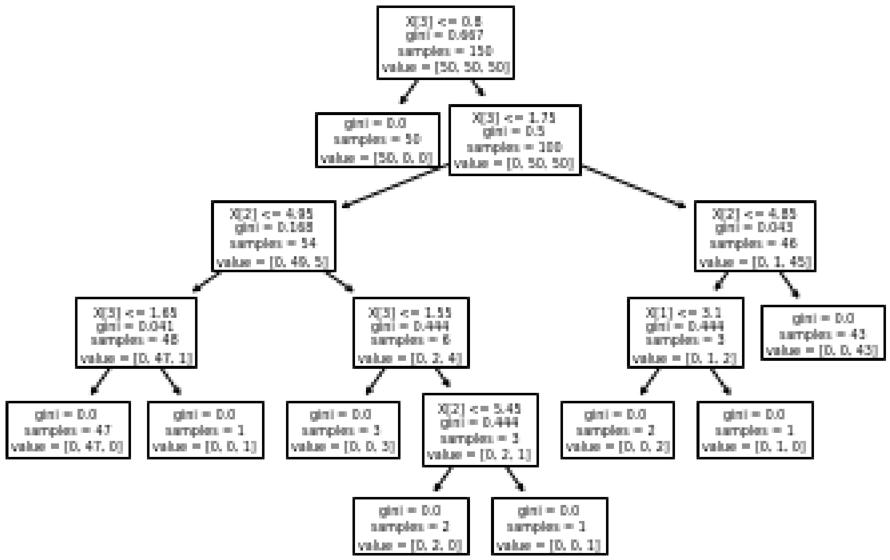
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
   X, y, test_size = 0.3, random_state = 1
)

print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(105, 4)
(45, 4)
(105,)
(45,)
```

```
from sklearn.datasets import load_iris
data = load_iris()
data.target[[10, 25, 50]]
list(data.target_names
['setosa', 'versicolor', 'virginica']
```

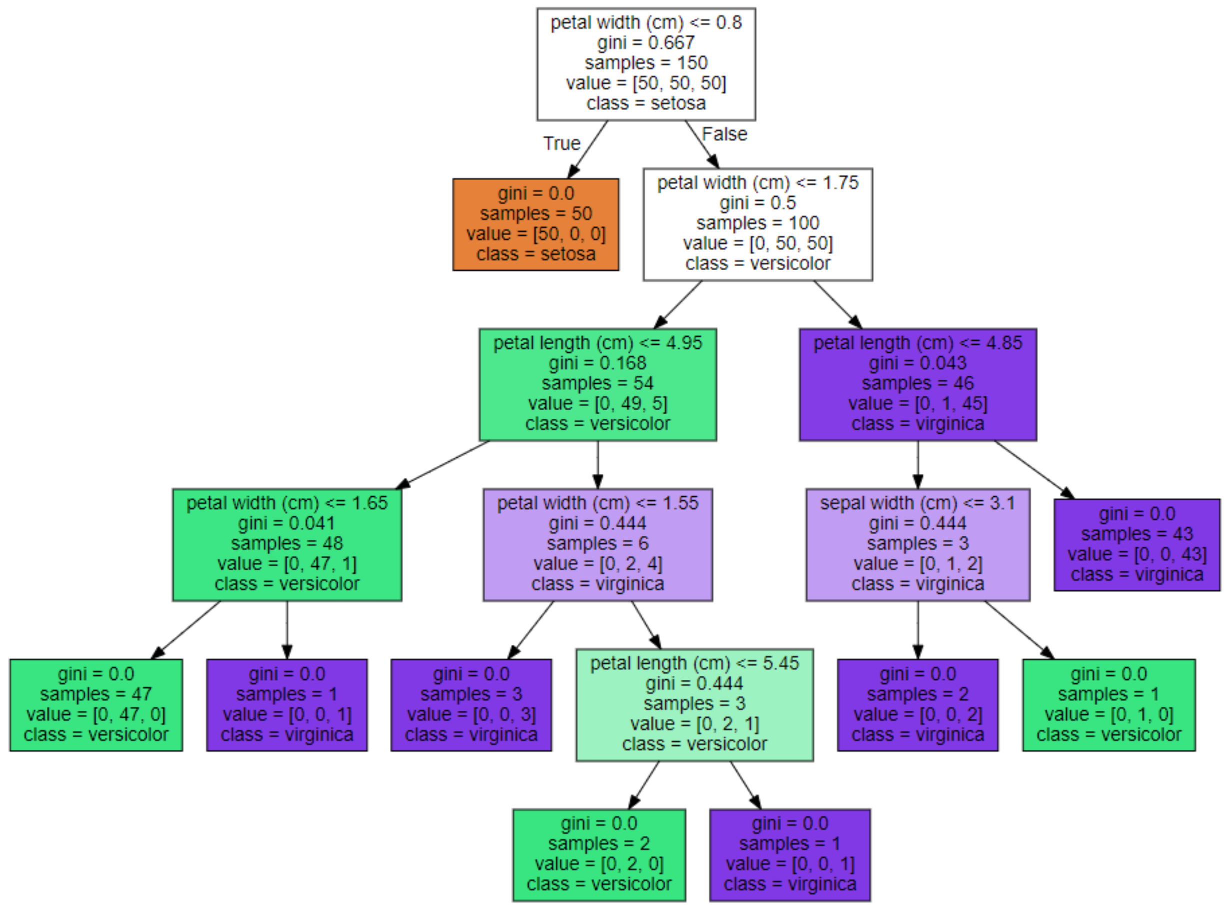
Making the dataset into Decision Classifier Tree:

```
from sklearn import tree
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
   X, y, test_size = 0.3, random_state = 1)

clf = tree.DecisionTreeClassifier()
clf = clf.fit(X, y)

tree.plot_tree(clf)
[Text(0.5, 0.9166666666666666, 'X[3] <= 0.8\ngini = 0.667\nsamples = 150\nvalue =
[50, 50, 50]'),
 Text(0.4230769230769231, 0.75, 'gini = 0.0\nsamples = 50\nvalue = [50, 0, 0]'),
 Text(0.5769230769230769, 0.75, 'X[3] <= 1.75\ngini = 0.5\nsamples = 100\nvalue =
[0, 50, 50]'),
 Text(0.3076923076923077, 0.5833333333333334, 'X[2] <= 4.95\ngini = 0.168\nsamples
= 54\nvalue = [0, 49, 5]'),
 Text(0.15384615384615385, 0.4166666666666667, 'X[3] <= 1.65\ngini =
0.041\nsamples = 48\nvalue = [0, 47, 1]'),
 Text(0.07692307692307693, 0.25, 'gini = 0.0\nsamples = 47\nvalue = [0, 47, 0]'),
 Text(0.23076923076923078, 0.25, 'gini = 0.0\nsamples = 1\nvalue = [0, 0, 1]'),
 Text(0.46153846153846156, 0.4166666666666667, 'X[3] <= 1.55\ngini =
0.444\nsamples = 6\nvalue = [0, 2, 4]'),
 Text(0.38461538461538464, 0.25, 'gini = 0.0\nsamples = 3\nvalue = [0, 0, 3]'),
 Text(0.5384615384615384, 0.25, 'X[2] <= 5.45\ngini = 0.444\nsamples = 3\nvalue =
[0, 2, 1]'),
 Text(0.46153846153846156, 0.08333333333333333, 'gini = 0.0\nsamples = 2\nvalue =
[0, 2, 0]'),
 Text(0.6153846153846154, 0.08333333333333333, 'gini = 0.0\nsamples = 1\nvalue =
[0, 0, 1]'),
 Text(0.8461538461538461, 0.5833333333333334, 'X[2] <= 4.85\ngini = 0.043\nsamples
= 46\nvalue = [0, 1, 45]'),
 Text(0.7692307692307693, 0.4166666666666667, 'X[1] <= 3.1\ngini = 0.444\nsamples
= 3\nvalue = [0, 1, 2]'),
 Text(0.6923076923076923, 0.25, 'gini = 0.0\nsamples = 2\nvalue = [0, 0, 2]'),
 Text(0.8461538461538461, 0.25, 'gini = 0.0\nsamples = 1\nvalue = [0, 1, 0]'),
 Text(0.9230769230769231, 0.4166666666666667, 'gini = 0.0\nsamples = 43\nvalue =
[0, 0, 43]')]
```



Graphical Representation:

```
#graph
import graphviz
dot_data = tree.export_graphviz(clf, out_file=None,
                                feature_names=iris.feature_names,
                                class_names=iris.target_names,
                                filled=True)
graph = graphviz.Source(dot_data, format="png")
graph
```



Testing the accuracy:

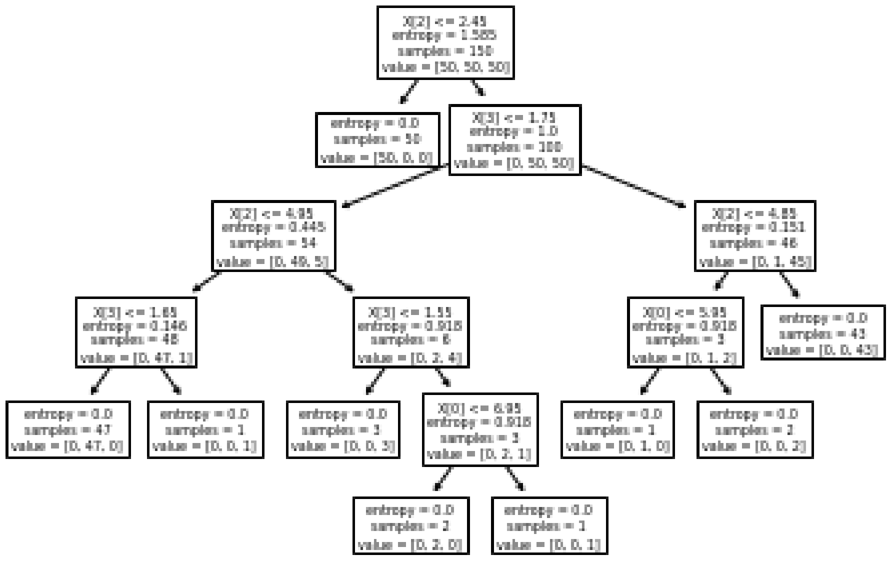
```
#Model accuracy
# To test whether the classifier is correct or not
from sklearn import metrics
y_pred = clf.predict(X_test)
print("accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
accuracy: 1.0
```

Train the Decision tree classifier using entropy:

```
clf = tree.DecisionTreeClassifier(criterion='entropy')
clf = clf.fit(X, y)
tree.plot_tree(clf)
y_pred = clf.predict(X_test)
#Model Accuracy
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 1.0
```



Tree Text Representation:

```
|--- feature_2 <= 2.45
|   |--- class: Setosa
|--- feature_2 >  2.45
|   |--- feature_3 <= 1.75
|   |   |--- feature_3 <= 1.45
|   |   |   |--- class: Versicolor
|   |   |--- feature_3 >  1.45
|   |   |   |--- feature_1 <= 2.60
|   |   |   |   |--- feature_0 <= 6.10
|   |   |   |   |   |--- class: Virginica
|   |   |   |   |--- feature_0 >  6.10
|   |   |   |   |   |--- class: Versicolor
|   |   |   |--- feature_1 >  2.60
|   |   |   |   |--- feature_0 <= 7.05
|   |   |   |   |   |--- class: Versicolor
|   |   |   |   |--- feature_0 >  7.05
|   |   |   |   |   |--- class: Virginica
|   |--- feature_3 >  1.75
|   |   |--- class: Virginica
```

Printing Accuracy:

```
% of Accuracy on training data:  100.0
% of Accuracy on test data:  92.10526315789474
```

1.7 Results and Discussion

Hence, to Implement CART algorithm for decision tree learning using an appropriate data set for building the decision tree and applying this knowledge to classify a new sample is successfully executed.