EXPERIMENT 2.2

DECISION TREE CLASSIFIER

TITLE: *DECISION TREE CLASSIFIER USING PYTHON WITH SCIKIT-LEARN LIBRARY*

INDEX

NOTE: The report form must be filled up based on your understanding after completion of the experiment and have to be submitted after showing the results.

1.1 OBJECTIVE(S):

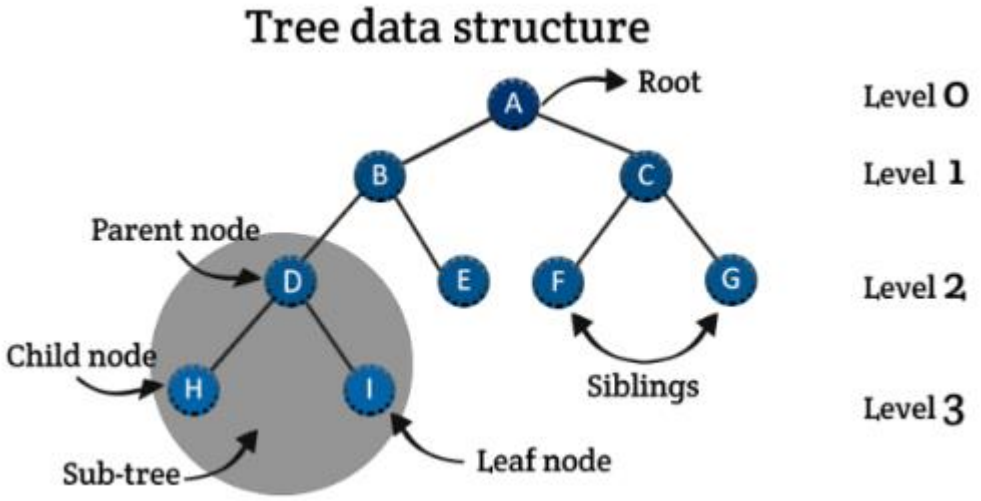
- In this tutorial project, you will understand the concept of a Decision Tree Classifier, learn how to preprocess data for training a Decision Tree Classifier, build and train a Decision Tree Classifier using Python, evaluate the model's performance, and visualize the results.

1.2 EQUIPMENT/SETUP:

- A Computer (PC) with Internet connection
- Operating system
- Any browser to use Google Colab (*https://colab.research.google.com*)
- To follow this tutorial, you will need a Jupyter notebook environment (e.g., Google Colab) with the following libraries installed: numpy, pandas, matplotlib, seaborn, and scikit-learn.

1.3 BACKGROUND:

A decision tree is a very specific type of probability tree that enables you to make a decision about some kind of process. It is used to break down complex problems or branches. Each branch of the decision tree could be a possible outcome.



- Supervised
- Classification
- Entropy
- Information Gain (IG)
- Gini Index

Problem Data Set

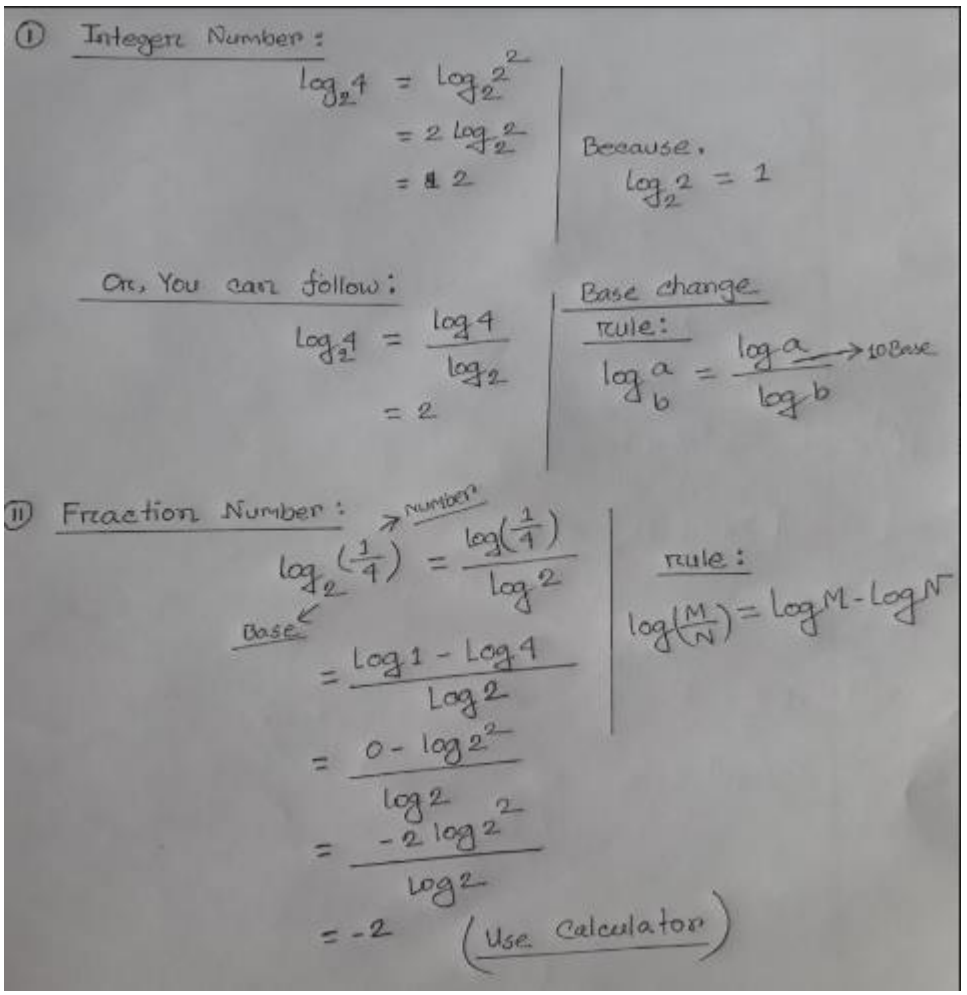
Class

Days	Outlook	Temperature	Routine	Wear Jacket?
1	Sunny	Cold	Indoor	No
2	Sunny	Warm	Outdoor	No
3	Cloudy	Warm	Indoor	No
4	Sunny	Warm	Indoor	No
5	Cloudy	Cold	Indoor	Yes
6	Cloudy	Cold	Outdoor	Yes
7	Sunny	Cold	Outdoor	Yes

$$E(S) = \sum_{i=1}^{c} - p_i \log_2 p_i$$

$$IG(Y, X) = E(Y) - E(Y|X)$$

$$\textit{Gini index} = 1 - \sum_{i=1}^{n} p_i^2$$



Wear Jacket?		
1	YES	3 Times
2	NO	4 Times

Entropy Before Partition:

Entropy of Wear Jacket:

= Entropy (4, 3)
= Entropy (– (Pi $\log_2$ Pi) + (– Pi $\log_2$ Pi))
= (-4/7 $\log_2$ 4/7) + (-3/7 $\log_2$ 3/7)
= (-.57 $\log_2$ .57) + (-.43 $\log_2$ .43)
= .985 (Entropy Before Partition)

$$E(S) = \sum_{i=1}^{c} - p_i \log_2 p_i$$

E(Y) = Entropy Before Partition
E(Y|X) = Entropy After Partition
Target, E(Y) >> E(Y|X)

$$E(S) = \sum_{i=1}^{c} - p_i \log_2 p_i$$

Problem Data Set

Days	Outlook	Temperature	Routine	Wear Jacket?
1	Sunny	Cold	Indoor	No
2	Sunny	Warm	Outdoor	No
3	Cloudy	Warm	Indoor	No
4	Sunny	Warm	Indoor	No
5	Cloudy	Cold	Indoor	Yes
6	Cloudy	Cold	Outdoor	Yes
7	Sunny	Cold	Outdoor	Yes

Outlook

E (Outlook, Sunny) =
-(1/4 $\log_2$ ¼ + ¾ $\log_2$ ¾)
= .812

E (Outlook, Cloudy) =
-(2/3 $\log_2$ 2/3 + 1/3 $\log_2$ 1/3)
= .918

Info Gain (S, Outlook) =
E(S) – (4/7 * .812) – (3/7 * .918)
= .985 – (4/7 * .812) – (3/7 * .918)
= .127

$$E(S) = \sum_{i=1}^{c} - p_i \log_2 p_i$$

Problem Data Set

Days	Outlook	Temperature	Routine	Wear Jacket?
1	Sunny	Cold	Indoor	No
2	Sunny	Warm	Outdoor	No
3	Cloudy	Warm	Indoor	No
4	Sunny	Warm	Indoor	No
5	Cloudy	Cold	Indoor	Yes
6	Cloudy	Cold	Outdoor	Yes
7	Sunny	Cold	Outdoor	Yes

Temperature

E (Temperature, Cold) =
-(1/4 $\log_2$ ¼ + ¾ $\log_2$ ¾)
= .812

E (Temperature, Warm) =
-(0/3 $\log_2$ 0/3 + 3/3 $\log_2$ 3/3)
= 0.00

Info Gain (S, Temperature) =
E(S) – (4/7 * .812) – (3/7 * 0)
= .985 – (4/7 * .812) – (3/7*0)
=.520

$$E(S) = \sum_{i=1}^{c} - p_i \log_2 p_i$$

Routine
E (Routine, Indoor) = -(1/4 $\log_2$ ¼ + ¾ $\log_2$ ¾) = .812
E (Routine, Outdoor) = -(2/3 $\log_2$ 2/3 + 1/3 $\log_2$ 1/3) = .918
Info Gain (S, Routine) = E(S) – (4/7*.812) – (3/7 * .918) = .985 – (4/7*.812) – (3/7 * .918) =.127

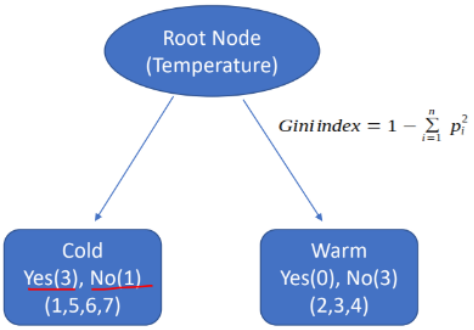
Problem Data Set

Days	Outlook	Temperature	Routine	Wear Jacket?
1	Sunny	Cold	Indoor	No
2	Sunny	Warm	Outdoor	No
3	Cloudy	Warm	Indoor	No
4	Sunny	Warm	Indoor	No
5	Cloudy	Cold	Indoor	Yes
6	Cloudy	Cold	Outdoor	Yes
7	Sunny	Cold	Outdoor	Yes

$$E(S) = \sum_{i=1}^{c} - p_i \log_2 p_i$$

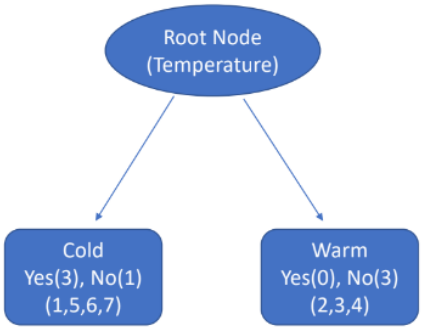
Root Node Selection Table

Outlook	Temperature	Routine
E (Outlook, Sunny) = -(1/4 $\log_2$ ¼ + ¾ $\log_2$ ¾) = .812	E (Temperature, Cold) = -(1/4 $\log_2$ ¼ + ¾ $\log_2$ ¾) = .812	E (Routine, Indoor) = -(1/4 $\log_2$ ¼ + ¾ $\log_2$ ¾) = .812
E (Outlook, Cloudy) = -(2/3 $\log_2$ 2/3 + 1/3 $\log_2$ 1/3) = .918	E (Temperature, Warm) = -(0/3 $\log_2$ 0/3 + 3/3 $\log_2$ 3/3) = 0.00	E (Routine, Outdoor) = -(2/3 $\log_2$ 2/3 + 1/3 $\log_2$ 1/3) = .918
Info Gain (S, Outlook) = E(S) – (4/7 * .812) – (3/7 * .918) = .985 – (4/7 * .812) – (3/7 * .918) = .127	Info Gain (S, Temperature) = E(S) – (4/7 * .812) – (3/7 * 0) = .985 – (4/7 * .812) – (3/7*0) =.520	Info Gain (S, Routine) = E(S) – (4/7*.812) – (3/7 * .918) = .985 – (4/7*.812) – (3/7 * .918) =.127



Problem Data Set

Days	Outlook	Temperature	Routine	Wear Jacket?
1	Sunny	Cold	Indoor	No
2	Sunny	Warm	Outdoor	No
3	Cloudy	Warm	Indoor	No
4	Sunny	Warm	Indoor	No
5	Cloudy	Cold	Indoor	Yes
6	Cloudy	Cold	Outdoor	Yes
7	Sunny	Cold	Outdoor	Yes



Problem Data Set

Days	Outlook	Temperature	Routine	Wear Jacket?
1	Sunny	Cold	Indoor	No
2	Sunny	Warm ✖	Outdoor	No
3	Cloudy	Warm ✖	Indoor	No
4	Sunny	Warm ✖	Indoor	No
5	Cloudy	Cold	Indoor	Yes
6	Cloudy	Cold	Outdoor	Yes
7	Sunny	Cold	Outdoor	Yes

Problem Data Set

Days	Outlook	Temperature	Routine	Wear Jacket?
1	Sunny	Cold	Indoor	No
2	Sunny	Warm	Outdoor	No
3	Cloudy	Warm	Indoor	No
4	Sunny	Warm	Indoor	No
5	Cloudy	Cold	Indoor	Yes
6	Cloudy	Cold	Outdoor	Yes
7	Sunny	Cold	Outdoor	Yes

Entropy of New Subset:

S2 = Entropy (1,3)
= Entropy (– (Pi $\log_2$ Pi) + (– Pi $\log_2$ Pi))
= (-1/4 $\log_2$ 1/4) + (-3/4 $\log_2$ 3/4)
= (-.25 $\log_2$ .25) + (-.75 $\log_2$ .75)
= .812 (Entropy for New Subset)

Problem Data Set

Days	Outlook	Temperature	Routine	Wear Jacket?
1	Sunny	Cold	Indoor	No
2	Sunny	Warm	Outdoor	No
3	Cloudy	Warm	Indoor	No
4	Sunny	Warm	Indoor	No
5	Cloudy	Cold	Indoor	Yes
6	Cloudy	Cold	Outdoor	Yes
7	Sunny	Cold	Outdoor	Yes

E (Routine, Indoor) =
-(1/2 $\log_2$ ½ + 1/2 $\log_2$ 1/2)
= 1

E (Routine, Outdoor) =
-(2/2 $\log_2$ 2/2 + 0/2 $\log_2$ 0/2)
= 0

Info Gain (S2, Routine) =
E(S2) – 2/4 * 1 – 2/4 *0
= .812 – 2/4 * 1 – 2/4 *0
= .312

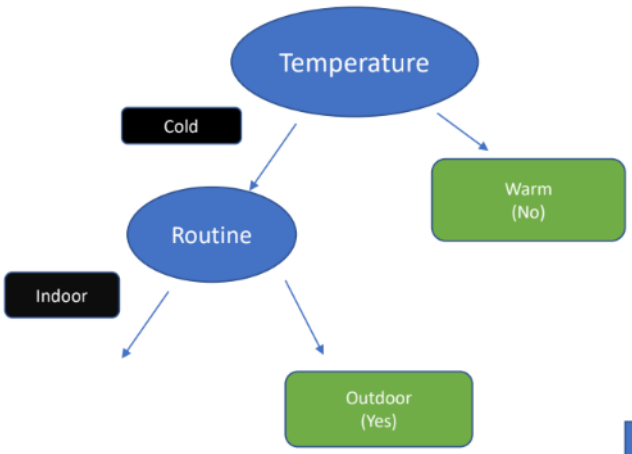
Problem Data Set

Days	Outlook	Temperature	Routine	Wear Jacket?
1	Sunny	Cold	Indoor	No
2	Sunny	Warm	Outdoor	No
3	Cloudy	Warm	Indoor	No
4	Sunny	Warm	Indoor	No
5	Cloudy	Cold	Indoor	Yes
6	Cloudy	Cold	Outdoor	Yes
7	Sunny	Cold	Outdoor	Yes

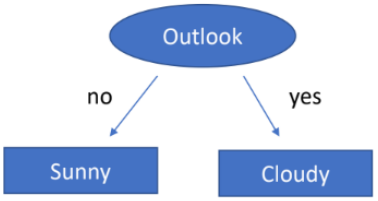
E (Outlook, Sunny) =
-(1/2 $\log_2$ ½ + 1/2 $\log_2$ 1/2)
= 1

E (Outlook, Cloudy) =
-(2/2 $\log_2$ 2/2 + 0/2 $\log_2$ 0/2)
= 0

Info Gain (S2, Outlook) =
E(S2) – 2/4 * 1 – 2/4 *0
= .812 – 2/4 * 1 – 2/4 *0
= .312

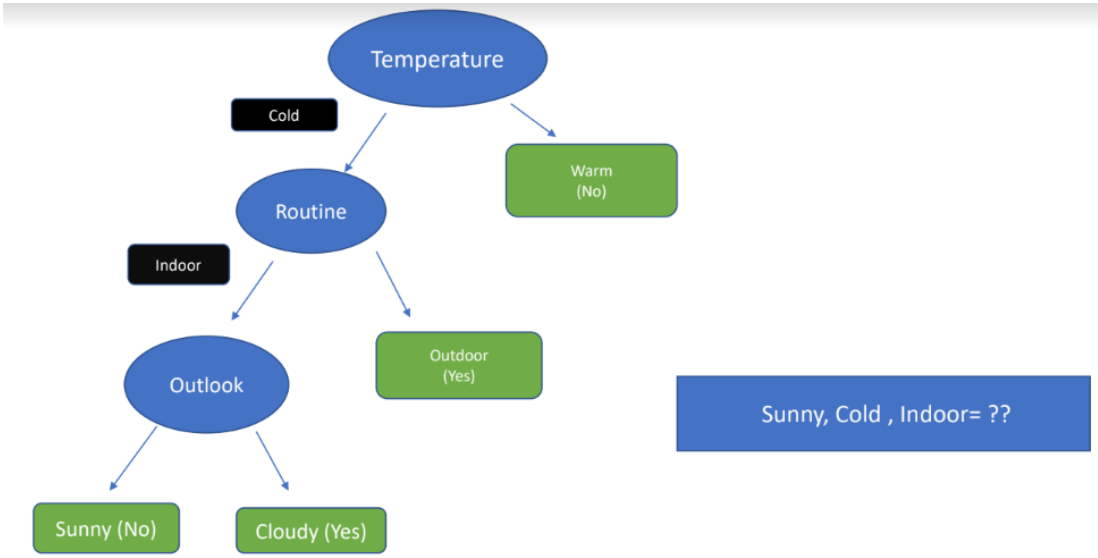


Sunny, Cold , Indoor= ??



Problem Data Set

Days	Outlook	Temperature	Routine	Wear Jacket?
1	Sunny	Cold	Indoor	No
2	Sunny	Warm	Outdoor	No
3	Cloudy	Warm	Indoor	No
4	Sunny	Warm	Indoor	No
5	Cloudy	Cold	Indoor	Yes
6	Cloudy	Cold	Outdoor	Yes
7	Sunny	Cold	Outdoor	Yes



1.4 DECISION TREE CLASSIFIER STRATEGY:

This project will follow a step-by-step approach:

- Data Loading and Exploration
- Data Preprocessing
- Model Building and Training
- Model Evaluation
- Visualization of Results

1.5 PROBLEM DESCRIPTION:

You have a dataset that contains information about the weather conditions (Outlook, Temperature), the daily routine (Indoor/Outdoor), and whether or not a person wears a jacket on a given day. The goal of this tutorial is to build a Decision Tree Classifier to predict whether a person will wear a jacket based on the weather conditions and daily routine.

1.6 PROCEDURE:

- Get the '**game data.csv**' and upload in your Google Drive.
- Open Google Colab (*https://colab.research.google.com/*).
- Create/rename your project as '*Decision Tree Classifier.ipynb*'.
- Follow the steps of next section (read and understand) and execute all the codes.

1.7 LAB PRACTICE/EXPERIMENT:

1.7.1 Getting the dataset

Download the dataset '**game data.csv**' from https://github.com/TITHI-KHAN/Decision-Tree and upload in your Google Drive. Please open the CSV file to see and understand the data.

1.7.2 Understanding the dataset

I've provided a dataset with five columns:

Features (Columns):

Days: A unique identifier for each day.

Outlook: The weather outlook for the day (e.g., Sunny or Cloudy).

Temperature: The temperature for the day (e.g., Cold or Warm).

Routine: Whether the daily routine is Indoor or Outdoor.

Target Variable:

Wear Jacket?: This is the variable we want to predict. It indicates whether a person will wear a jacket on that day (Yes or No).

Objective: The objective of this tutorial is to use the provided dataset to train a Decision Tree Classifier that can predict whether a person will wear a jacket based on the given weather conditions and daily routine. This can be a useful model for individuals planning their attire for the day, as it can help them make informed decisions based on the expected weather conditions and activities

1.7.3 Import Libraries in Google Colab

```
#Importing all the required libraries
import pandas as pd
import numpy as np
import sklearn
from pandas.core.dtypes.common import is_numeric_dtype
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
import matplotlib.pyplot as plt
import seaborn as sns
import warnings as wr #Ignores the warnings
wr.filterwarnings('ignore')

df=pd.read_csv("game data.csv")
df
```

The CSV file contains 8 records (small dataset) in total. The **output** shows:

	Days	Outlook	Temprature	Routine	Wear Jacket?
0	1	Sunny	Cold	Indoor	No
1	2	Cloudy	Cold	Indoor	Yes
2	3	Cloudy	Warm	Outdoor	No
3	4	Sunny	Cold	Outdoor	Yes
4	5	Cloudy	Cold	Outdoor	Yes
5	6	Sunny	Warm	Outdoor	No
6	7	Cloudy	Warm	Indoor	No
7	8	Sunny	Warm	Indoor	No

Explanation:

Let's go through the code step by step and explain each part:

import pandas as pd

-It makes working with structured data (like CSV files or database tables) easier. You can filter, transform, and analyze data easily using pandas.

import numpy as np

-Mathematical library. It helps with working on large sets of numbers efficiently, performing mathematical operations, and handling multi-dimensional arrays (like tables of numbers).

import sklearn

-It provides machine learning tools and algorithms for tasks like classification, regression, clustering, and more. It helps you build and train machine learning models.

from pandas.core.dtypes.common import is_numeric_dtype

- It imports the is_numeric_dtype function from the pandas.core.dtypes.common module. This function is used to check if a given pandas data type is numeric or not.

from sklearn.preprocessing import LabelEncoder

- This is an encoding technique used to convert categorical data (textual data) into numerical format. In your code, you are using it to encode categorical features so that they can be used as input to the machine learning model.

from sklearn.tree import DecisionTreeClassifier

- This is a class provided by scikit-learn for building decision tree-based classification models. It is used to create and train a Decision Tree Classifier.

from sklearn import tree

- This module within scikit-learn provides functions for visualizing decision trees and exporting them in textual formats.

import matplotlib.pyplot as plt

-It helps create visualizations such as line plots, scatter plots, and histograms. You can use it to represent your data visually.

import seaborn as sns

-It builds on top of matplotlib and makes it simpler to create statistical visualizations with better aesthetics. It's useful for creating attractive plots like heatmaps, violin plots, and more.

import warnings as wr
wr.filterwarnings('ignore')

-Ignores the warnings.

df=pd.read_csv("game data.csv")

This line reads the CSV file named 'game data.csv' and loads its contents into a pandas DataFrame called `df`. A DataFrame is a two-dimensional labeled data structure with rows and columns, similar to a table in a database or a spreadsheet.

df

This line prints the DataFrame.

1.7.4 Fetching data from google drive

Before you can access files from Google Drive, you need to mount your Google Drive in the Colab environment. This allows you to link your Google Drive account to the Colab notebook. To do this, use the following code:

```
from google.colab import drive
drive.mount('/content/drive')
```

When you run this code, it will provide a link to an authentication page where you need to grant permission to access your Google Drive. After granting permission, you will receive a code that you should enter in the Colab notebook. Once successfully mounted, your Google Drive will be accessible.

After mounting Google Drive, you can define the file path to your dataset on Google Drive. You should specify the path to your dataset file on your Google Drive folder. For example:

```
file_path = "/content/drive/My Drive/your_folder/game data.csv"
```

Replace "your_folder" with the actual folder path where your dataset file is located on Google Drive.

Now, you can use Pandas to read the dataset from Google Drive. Here's how you can do it:

```
import pandas as pd

df = pd.read_csv(file_path)
```

This code reads the CSV file from the specified file path and loads it into a Pandas DataFrame named df. You can then proceed to work with the dataset.

1.7.5 Data Pre-processing

Prepare the data for modeling.

1.7.5.1 Checking the NULL values

Check for and handle NULL values.

df.drop('Days', axis=1, inplace=True)

-df will be automatically updated after dropping the Days column as it has inplace = True.

Output:

	Outlook	Temprature	Routine	Wear Jacket?
0	Sunny	Cold	Indoor	No
1	Cloudy	Cold	Indoor	Yes
2	Cloudy	Warm	Outdoor	No
3	Sunny	Cold	Outdoor	Yes
4	Cloudy	Cold	Outdoor	Yes
5	Sunny	Warm	Outdoor	No
6	Cloudy	Warm	Indoor	No
7	Sunny	Warm	Indoor	No

1.7.5.2 Define dependent (target) and independent (predictor) features

You are preparing your data for machine learning by splitting it into two parts: the target variable y and the feature variables x. This is a common preprocessing step in machine learning. Let me explain these lines of code:

y=df['Wear Jacket?']

Here, you are creating a new variable y that will store the target variable you want to predict. In your dataset, the target variable is 'Wear Jacket?'. So, y will contain the values of whether a person wears a jacket or not.

x=df.drop('Wear Jacket?', axis=1)

In this line, you are creating a new DataFrame x that will store the feature variables used to make predictions.

df.drop('Wear Jacket?', axis=1) is used to drop the 'Wear Jacket?' column from the original DataFrame df. As a result, x will contain all the columns in df except for 'Wear Jacket?'.

Now, you have separated your data into x (features) and y (target), which is a common practice in machine learning. You can use x to train your machine learning model to predict the 'Wear Jacket?' values stored in y.

Feature Encoding:

```
for col in x.columns:
  if is_numeric_dtype(x[col]):
    continue
  else:
    x[col]=LabelEncoder().fit_transform(x[col])
```

#if numeric, then skip. If not numeric, then encode with Label Encoder.

It is a loop that iterates through the columns of the feature DataFrame x and encodes non-numeric (categorical) columns using the LabelEncoder. Here's a breakdown of the code:

for col in x.columns:: This line starts a loop that iterates through each column in the DataFrame x.

if is_numeric_dtype(x[col]):: Within the loop, this conditional statement checks whether the current column (x[col]) contains numeric data. It uses the is_numeric_dtype function to perform this check.

continue: If the current column is numeric, the loop continues to the next column without taking any action on the current column.

else:: If the current column is not numeric (i.e., it's categorical), the code inside the else block is executed.

x[col]=LabelEncoder().fit_transform(x[col]): In this line, the LabelEncoder is applied to the non-numeric (categorical) column to transform its values into numerical labels. Here's what this line does:

LabelEncoder(): Creates an instance of the LabelEncoder class.
.fit_transform(x[col]): Applies the fit_transform method of the LabelEncoder to the values in the current column (x[col]). This method fits the encoder to the unique values in the column and transforms those values into numerical labels.

By the end of this loop, all non-numeric (categorical) columns in the feature DataFrame x will be encoded with numerical labels, making them suitable for use in machine learning models that require numerical input data. This is a common preprocessing step when working with categorical data in machine learning.

Before:

	Outlook	Temprature	Routine
0	Sunny	Cold	Indoor
1	Cloudy	Cold	Indoor
2	Cloudy	Warm	Outdoor
3	Sunny	Cold	Outdoor
4	Cloudy	Cold	Outdoor
5	Sunny	Warm	Outdoor
6	Cloudy	Warm	Indoor
7	Sunny	Warm	Indoor

After:

	Outlook	Temprature	Routine
0	1	0	0
1	0	0	0
2	0	1	1
3	1	0	1
4	0	0	1
5	1	1	1
6	0	1	0
7	1	1	0

x.columns
This line of code will output a list of the column names in the DataFrame x, and it will look something like this:

```
Index(['Outlook', 'Temprature', 'Routine'], dtype='object')
```

- Outlook (sunny -> 1, cloudy -> 0)
- Temperature (cold -> 0, warm -> 1)
- Routine (indoor -> 0, outdoor -> 1)

1.7.5.3 Training Decision Tree Classifier

clf = DecisionTreeClassifier()
clf.fit(x,y)

Explanation:

You are creating an instance of the DecisionTreeClassifier class and then fitting (training) the classifier using your feature variables x and target variable y. Let's break down these two lines of code:

clf = DecisionTreeClassifier()

Here, you create an instance of the DecisionTreeClassifier class and assign it to the variable clf. This instance represents your decision tree classifier model.

clf.fit(x,y)

This line of code trains the decision tree classifier using your feature variables (x) as the input and the target variable (y) as the output or labels.

The fit method is a common method in scikit-learn classifiers that takes the training data and performs the training process. In this case, it constructs a decision tree based on the patterns in the training data.

After this step, your clf variable holds a trained decision tree classifier that can be used to make predictions on new data.

With these two lines of code, you have successfully created and trained a Decision Tree Classifier using your feature variables (x) and target variable (y). This classifier is now ready to make predictions based on the patterns it learned during training.

1.7.5.4 Prediction

clf.predict([[1,0,0]]) #Sunny, Cold, Indoor = ( ) ? (1,0,0) #No
clf.predict([[1,0,1]]) #Sunny, Cold, Outdoor = ( ) ? (1,0,1) #Yes

Explanation:

you are using the trained Decision Tree Classifier (clf) to make predictions based on specific input feature values. Let's break down these prediction lines:

Predicting for Input [1, 0, 0] (Sunny, Cold, Indoor):

clf.predict([[1,0,0]])

Here, you are using the predict method of the trained classifier (clf) to make a prediction.
The input [1, 0, 0] represents the feature values Sunny, Cold, and Indoor, respectively.
The model will analyze these feature values and predict the corresponding target value.
The comment #Sunny, Cold, Indoor = ( ) ? (1,0,0) #No suggests that you expect the prediction to be "No."

clf.predict([[1,0,1]])

Similarly, this line predicts the outcome for the input [1, 0, 1], representing Sunny, Cold, and Outdoor.
The comment #Sunny, Cold, Outdoor = ( ) ? (1,0,1) #Yes suggests that you expect the prediction to be "Yes."
These lines demonstrate how to use a trained decision tree classifier to make predictions for specific combinations of feature values. The classifier will evaluate the input features and provide predictions based on the decision tree's rules, which it learned during training.

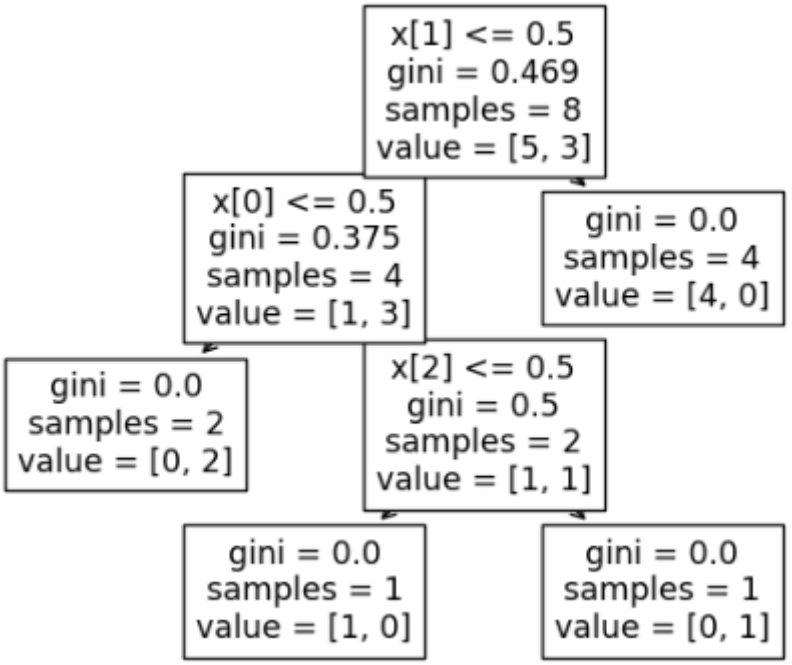
1.7.6 Tree Represntation

You are using the tree.plot_tree function to visualize the trained Decision Tree Classifier (clf) in different ways. Let's explain each part of the code:

Visualizing the Decision Tree:

tree.plot_tree(clf)

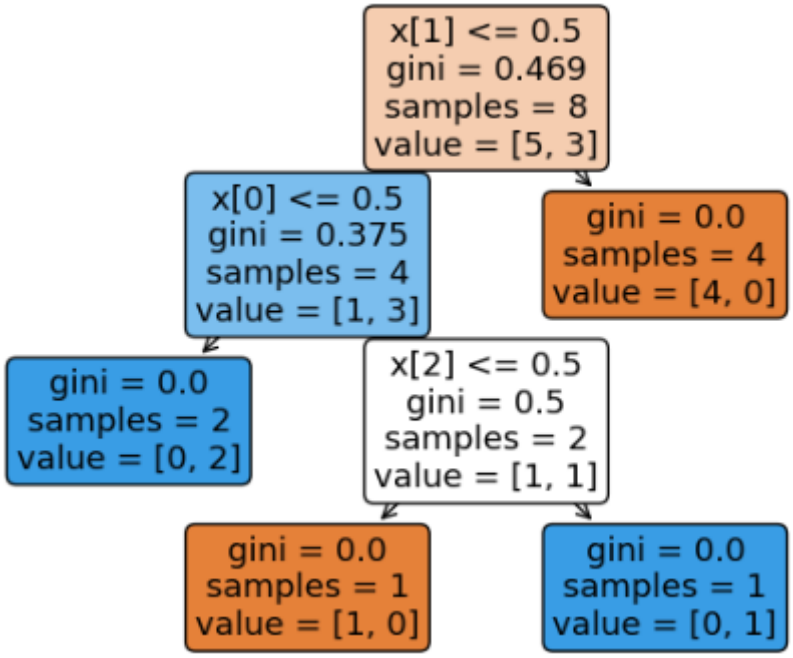
This line of code generates a basic visual representation of the decision tree, but it doesn't include any additional styling or feature names.



Visualizing the Decision Tree with Styling:

tree.plot_tree(clf, rounded=True, filled=True)

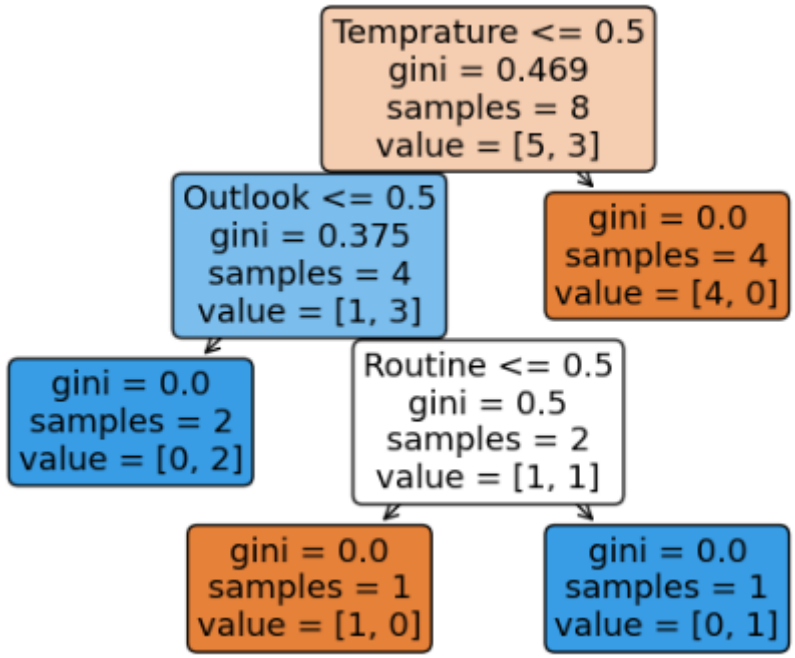
In this line, you are using the rounded=True and filled=True arguments to style the decision tree plot. Rounded nodes and filled boxes make the tree plot more visually appealing.



Visualizing the Decision Tree with Feature Names:

tree.plot_tree(clf, rounded=True, filled=True, feature_names=x.columns)

Here, you are enhancing the tree plot by adding feature names to the nodes. feature_names=x.columns provides the names of the features (Outlook, Temperature, Routine) associated with each node in the tree.



Setting Figure Size:

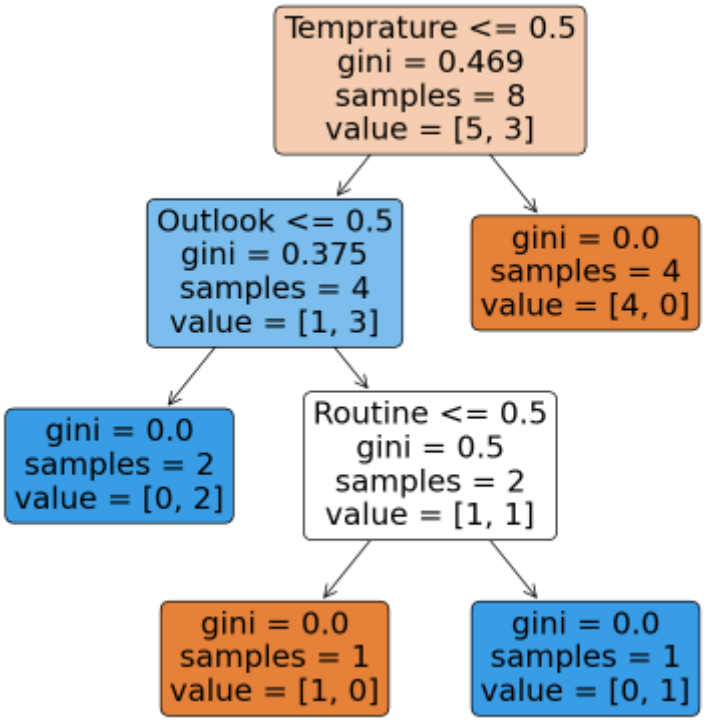
plt.figure(figsize=(10,10))

Before generating the final tree plot, you use plt.figure(figsize=(10,10)) to specify the size of the figure. This line of code sets the figure size to 10 inches in width and 10 inches in height, making the plot larger and more readable.

Generating the Final Tree Plot:

tree.plot_tree(clf, rounded=True, filled=True, feature_names=x.columns)

Finally, you generate the tree plot with rounded nodes, filled boxes, and feature names. This plot will be larger due to the specified figure size.



By combining these lines of code, you can visualize the decision tree in different styles and with feature names, which can help you better understand how the tree makes decisions based on the provided features.

1.7.7 Text Representation

You are exporting the trained Decision Tree Classifier clf as a text representation and then printing it. Let's break down these lines:

Exporting the Decision Tree as Text:

text_rep = tree.export_text(clf)

In this line, you use the tree.export_text function to export the decision tree clf as a text representation. This representation will show the structure of the decision tree along with its decision rules.

Printing the Text Representation:

print(text_rep)

It properly prints the text representation of the decision tree.

```
|--- feature_1 <= 0.50
|   |--- feature_0 <= 0.50
|   |   |--- class: Yes
|   |--- feature_0 >  0.50
|   |   |--- feature_2 <= 0.50
|   |   |   |--- class: No
|   |   |--- feature_2 >  0.50
|   |   |   |--- class: Yes
|--- feature_1 >  0.50
|   |--- class: No
```

Converting Feature Names to a List of Strings:

x_updated = x.columns.tolist()

Here, you convert the column names of the feature DataFrame x into a list of strings and store it in the variable x_updated. This will be used to specify feature names in the next step.

Exporting the Decision Tree with Feature Names as Text:

text_rep = tree.export_text(clf, feature_names=x_updated)

In this line, you export the decision tree clf as a text representation again, but this time, you specify the feature_names parameter as x_updated. This means that the feature names will be included in the text representation to provide context for the decision rules.

Printing the Final Text Representation:

print(text_rep)

Finally, you print the text representation of the decision tree, which now includes feature names.

```
|--- Temprature <= 0.50
|   |--- Outlook <= 0.50
|   |   |--- class: Yes
|   |--- Outlook >  0.50
|   |   |--- Routine <= 0.50
|   |   |   |--- class: No
|   |   |--- Routine >  0.50
|   |   |   |--- class: Yes
|--- Temprature >  0.50
|   |--- class: No
```

By executing these lines of code, you can view a textual representation of your decision tree along with the feature names associated with each decision rule. This can be helpful for understanding the logic of the decision tree model.

1.8 RESULT/COMMENTS:

Run all the codes sequentially, observe all the outputs, and write comments in the report form.

Notes: The dataset is small in size and it does not contain real data. Moreover, the Training and Test data selection are random. So, the simulated results could be different from the results shown in this document.

1.9 INSTRUCTIONS:

1. Run all the codes/models and show them to the invigilator/instructor.
1. Observe each of the results and provide your comments in the form.
3. Fill up the report form properly, take a signature from the instructor, and submit.

REFERENCE:

[1] https://github.com/TITHI-KHAN/Decision-Tree/tree/main

EXPERIMENT NO.: 2.1
TITLE: MULTIPLE LINEAR REGRESSION USING PYTHON WITH SCIKIT-LEARN LIBRARY

DATE:
Batch & Section:

Student ID: _____ Student Name:

_____

Objectives of the experiment:

Some Outputs/Graphs with appropriate title:

Comments (overall understanding):

To fill by the instructor:

Marks from the instructor: ☐ X ☐ Y ☐ Z

Instructor's Name and Signature:

_____