EXPERIMENT 1

AIM: Introduction to Google collab and libraries Panda and Numpy

THEORY:

1. Google Colab:

Google Colaboratory, commonly called Google Colab, is a free, cloud-based platform provided by Google that allows users to write and execute Python code in a web browser. It is widely used for machine learning, data analysis, and education because it provides free access to powerful computing resources, including GPUs and TPUs. One of its main advantages is that no local setup is needed — everything runs in the cloud, and notebooks can be easily shared via Google Drive, just like Google Docs.

2. Pandas:

Pandas is a popular open-source Python library used for data analysis and manipulation. It provides powerful data structures like:

Series (one-dimensional data)

DataFrame (two-dimensional, tabular data like spreadsheets or SQL tables)

Pandas makes it easy to clean, organize, filter, and analyze datasets, with built-in functions for handling missing values, grouping data, merging datasets, and performing descriptive statistics. It is one of the most essential tools in data science.

3. NumPy:

NumPy (Numerical Python) is a fundamental Python library for scientific computing. It provides support for large, multi-dimensional arrays and matrices, along with a wide collection of mathematical functions to operate on them efficiently.
Key features include:

Fast array computations

Linear algebra operations

Fourier transforms

Random number generation

NumPy forms the backbone of many other Python libraries, including Pandas, SciPy, and TensorFlow.

CODE:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
from sklearn.model_selection import train_test_split
```

Double-click (or enter) to edit

```
diabetes=pd.read_csv('diabetes.csv')
diabetes.shape
```

```
(768, 9)
```

```
diabetes.head()
```

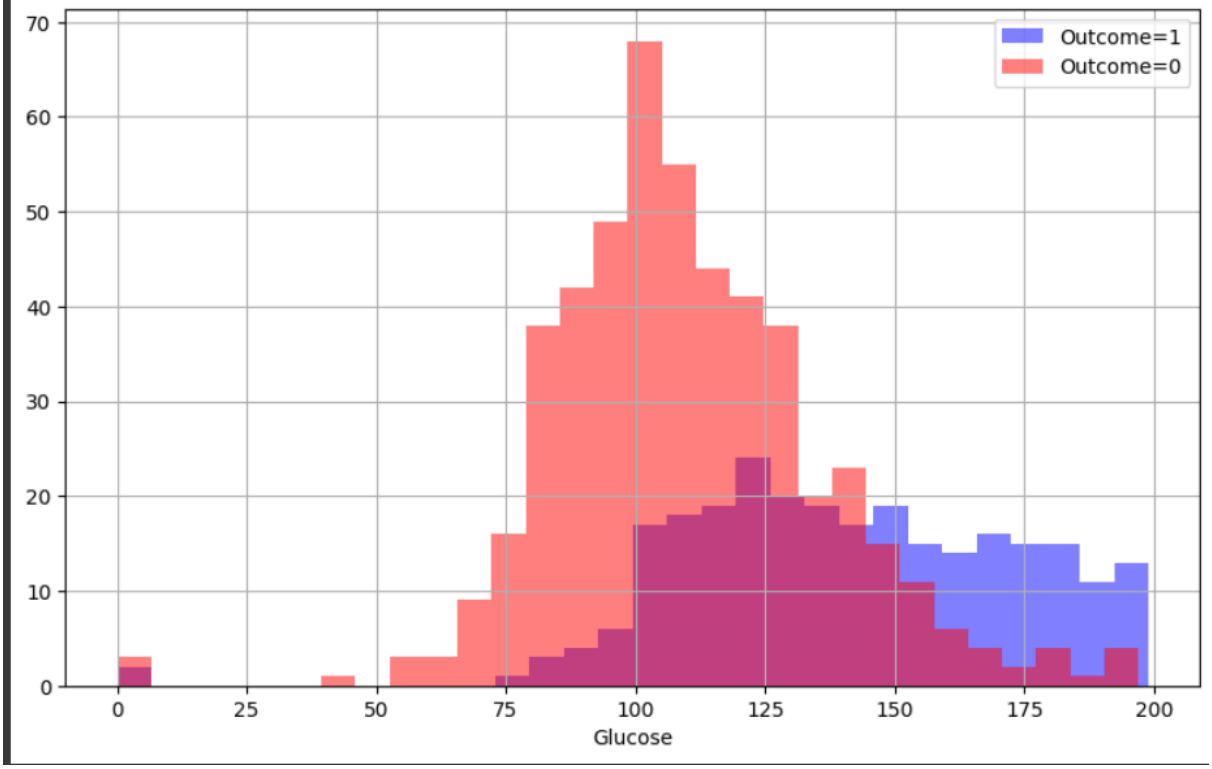
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

```
diabetes.describe()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

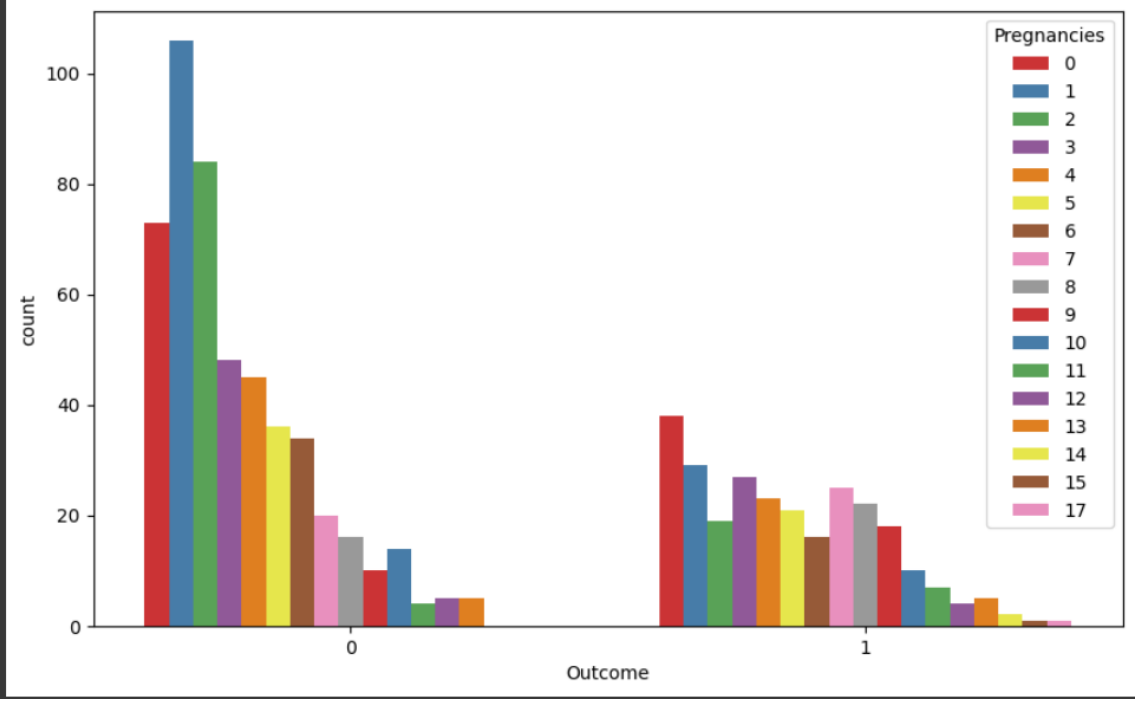
```
plt.figure(figsize=(10,6))
diabetes[diabetes['Outcome']==1]['Glucose'].hist(alpha=0.5,color='blue',bins=30,label='Outcome=1')
diabetes[diabetes['Outcome']==0]['Glucose'].hist(alpha=0.5,color='red',bins=30,label='Outcome=0')
plt.legend()
plt.xlabel('Glucose')
```

```
Text(0.5, 0, 'Glucose')
```




```
plt.figure(figsize=(10,6))
sns.countplot(x='Outcome',hue='Pregnancies',data=diabetes,palette='Set1')
```

```
<Axes: xlabel='Outcome', ylabel='count'>
```



LEARNING OUTCOMES:

_____

_____

_____

_____

EXPERIMENT 2

AIM: Program to demonstrate simple Linear Regression

THEORY:

Linear regression is one of the simplest and most widely used **statistical and machine learning techniques** for modeling the relationship between a **dependent variable (target)** and one or more **independent variables (predictors/features)**.

At its core, it tries to fit a **straight line (linear function)** to the data that best explains the relationship between the input(s) and the output.

Key Idea:

- If you have **one independent variable**, the model is called **Simple Linear Regression**. The equation looks like:

$$y=\beta 0 +\beta 1\ x+\epsilon$$

where:

  - y = dependent variable (output)
  - x = independent variable (input)
  - β0 = intercept (value of y when x=0)
  - β1 = slope (change in y for a unit change in x)
  - ϵ = error term (captures randomness/ unexplained part)
- If you have **multiple independent variables**, it's **Multiple Linear Regression**:

$$y=\beta 0 +\beta 1\ x1 +\beta 2\ x2 +\cdots+\beta n\ xn +\epsilon$$

Assumptions of Linear Regression:

1. Linearity – Relationship between predictors and target is linear.
2. Independence – Observations are independent of each other.
3. Homoscedasticity – Constant variance of errors.
4. Normality – Errors are normally distributed (for inference).
1. No/low multicollinearity – Predictors should not be highly correlated

CODE:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
from sklearn.model_selection import train_test_split
```

```
from collections.abc import ValuesView
ds=pd.read_csv('house.csv')
X=ds.iloc[:,:-1].values
Y=ds.iloc[:,1].values
```

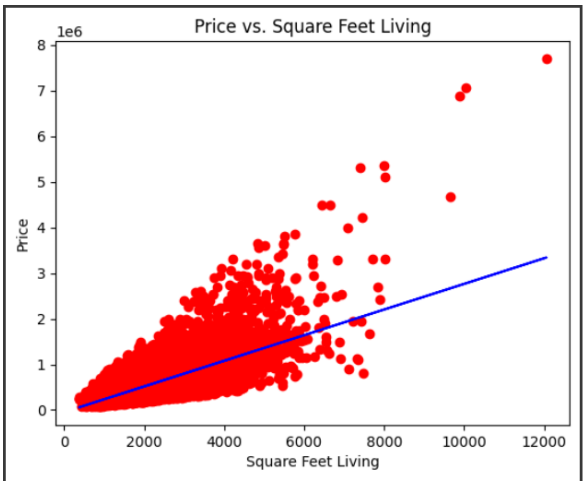
```
[ ] ds.head()
```

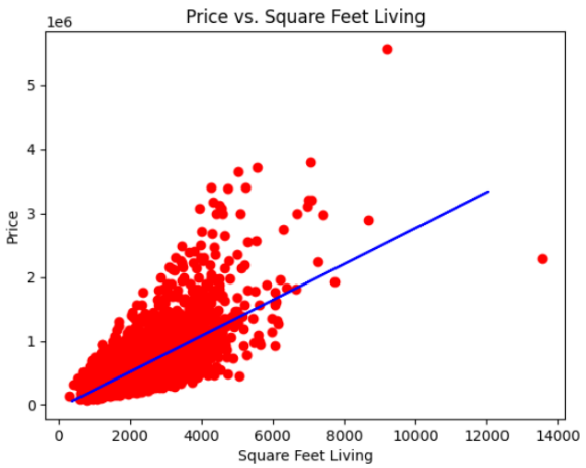
	sqft_living	price
0	1180	221900
1	2570	538000
2	770	180000
3	1960	604000
4	1680	510000

```
[ ] from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.25)


from sklearn.linear_model import LinearRegression
regressor=LinearRegression()
regressor.fit(X_train,Y_train)
```

```
LinearRegression
LinearRegression()
```

```
Y_pred=regressor.predict(X_test)
plt.scatter(X_train,Y_train,color='red')
plt.plot(X_train,regressor.predict(X_train),color='blue')
plt.title('Price vs. Square Feet Living')
plt.xlabel('Square Feet Living')
plt.ylabel('Price')
plt.show()
plt.scatter(X_test,Y_test,color='red')
plt.plot(X_train,regressor.predict(X_train),color='blue')
plt.title('Price vs. Square Feet Living')
plt.xlabel('Square Feet Living')
plt.ylabel('Price')
plt.show()
```





LEARNING OUTCOMES:

EXPERIMENT 3

AIM: Program to demonstrate simple Logistic Regression

THEORY:

Logistic Regression is a supervised machine learning algorithm mainly used for **classification problems**. Unlike linear regression, which predicts continuous values, logistic regression predicts the **probability of belonging to a class**. It first computes a linear combination of inputs:

$$z=w0 +w1\ x1 +w2\ x2 +\cdots+wn\ xn$$

and then applies the **sigmoid (logistic) function** to map the result between 0 and 1:

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

Based on a threshold (commonly 0.5), the model decides the class, e.g., if σ(z)>0.5\sigma(z) > 0.5σ(z)>0.5, predict class 1; otherwise, class 0.

There are different types of logistic regression: **binary** (two classes), **multinomial** (more than two categories), and **ordinal** (ordered categories). Logistic regression is widely applied in domains like **medical diagnosis** (disease/no disease), **finance** (loan default prediction), **marketing** (customer churn), and **NLP** (sentiment analysis). In essence, it is a simple yet powerful algorithm for handling classification tasks by estimating probabilities and converting them into class labels.

CODE:

```
[ ] import pandas as pd
    col_names = ['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bmi', 'pedigree', 'age', 'label']
    # load dataset
    pima = pd.read_csv("pima-indians-diabetes.csv", header=None, names=col_names)
```

```
pima.head()
```

	pregnant	glucose	bp	skin	insulin	bmi	pedigree	age	label
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

+ Code + Text

```
[ ] #split dataset in features and target variable
    feature_cols = ['pregnant', 'insulin', 'bmi', 'age','glucose','bp','pedigree']
    X = pima[feature_cols] # Features
    y = pima.label # Target variable
```

```
[ ] # split X and y into training and testing sets
    from sklearn.model_selection import train_test_split
    X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.35,random_state=0)
```

```
[ ] from sklearn.linear_model import LogisticRegression
    logreg = LogisticRegression()
    logreg.fit(X_train,y_train)
    y_pred=logreg.predict(X_test)
```

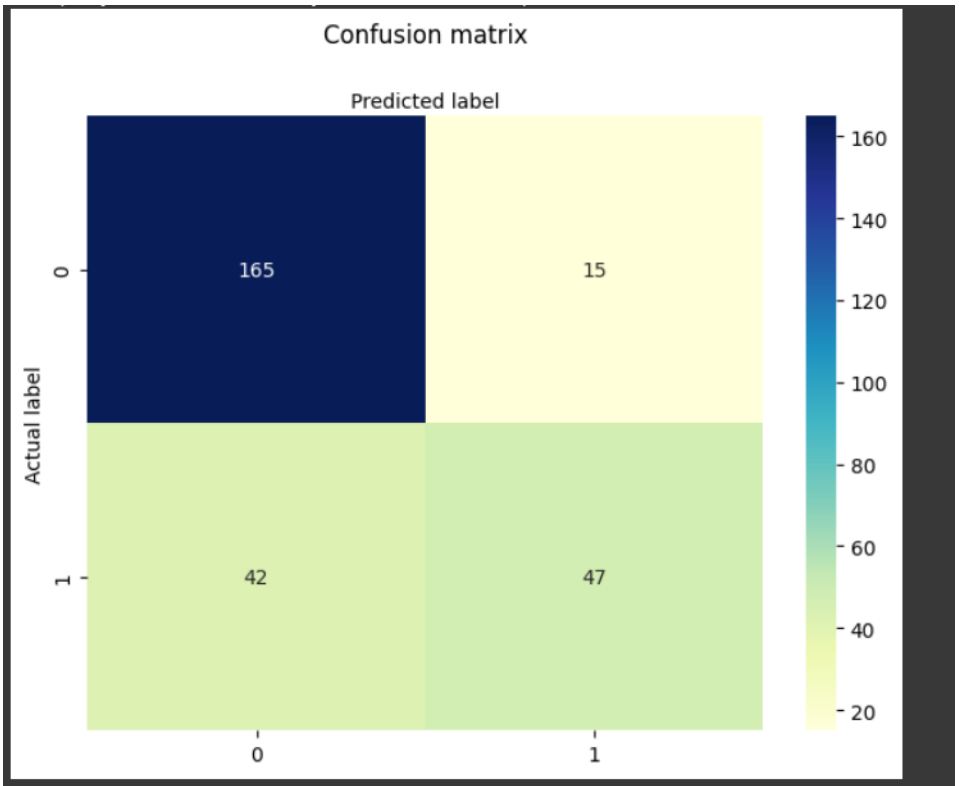
```
/usr/local/lib/python3.12/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning:
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
[ ] from sklearn import metrics
    cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
    cnf_matrix
```

```
array([[165,  15],
       [ 42,  47]])
```

```
[ ] import numpy as np
    import matplotlib.pyplot as plt
    import seaborn as sns
    %matplotlib inline
```

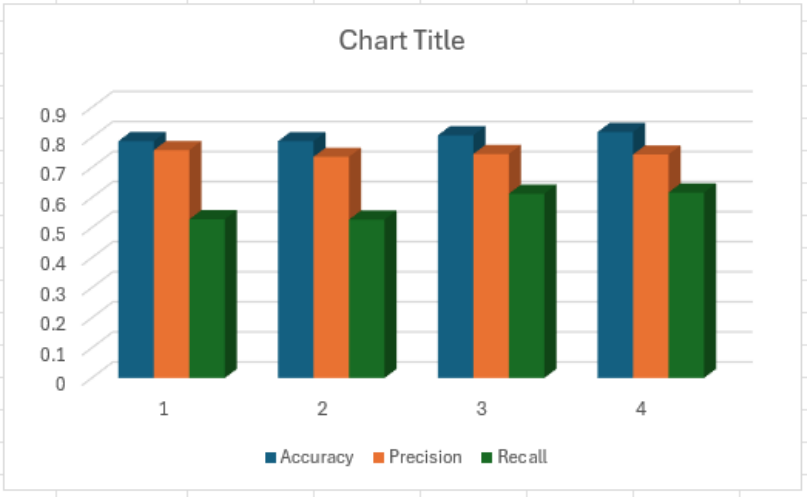
```
class_names=[0,1] # name  of classes
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
# create heatmap
sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="YlGnBu" ,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```




```
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("Precision:",metrics.precision_score(y_test, y_pred))
print("Recall:",metrics.recall_score(y_test, y_pred))
```

```
Accuracy: 0.7881040892193308
Precision: 0.7580645161290323
Recall: 0.5280898876404494
```

	Accuracy	Precision	Recall	Ratio
2	0.788104089	0.758064516	0.528089888	0.35
3	0.787878788	0.735849057	0.527027027	0.3
4	0.807291667	0.745098039	0.612903226	0.25
5	0.818181818	0.743589744	0.617021277	0.2



LEARNING OUTCOMES:

_____

_____

_____

_____

EXPERIMENT 4

Aim: Program to demonstrate Decision Tree- ID3 Algorithm

Theory: A Decision Tree is a supervised learning method used for classification. It splits data into branches based on attributes until a decision (class label) is reached. The ID3 (Iterative Dichotomiser 3) algorithm, proposed by Ross Quinlan, builds trees using Entropy and Information Gain.

Key Concepts:

- Entropy (measure of impurity):

$$Entropy(S) = -\sum p_i \log_2(p_i)$$

where *pi* is the probability of class *ii*

- Information Gain (attribute selection):

$$Gain(S, A) = Entropy(S) - \sum_v \frac{|S_v|}{|S|} \times Entropy(S_v)$$

Attribute with maximum gain is chosen for

splitting. Steps of ID3:

1. Calculate entropy of dataset.
2. Compute information gain for each attribute.
3. Select attribute with highest gain as decision node.
4. Split dataset into subsets and repeat recursively.
5. Stop when all records belong to one class or no attributes remain.

Advantages:

- Simple and easy to interpret.
- Works well with categorical data.

Limitations:

- Biased towards attributes with many values.
- Sensitive to noise and overfitting.

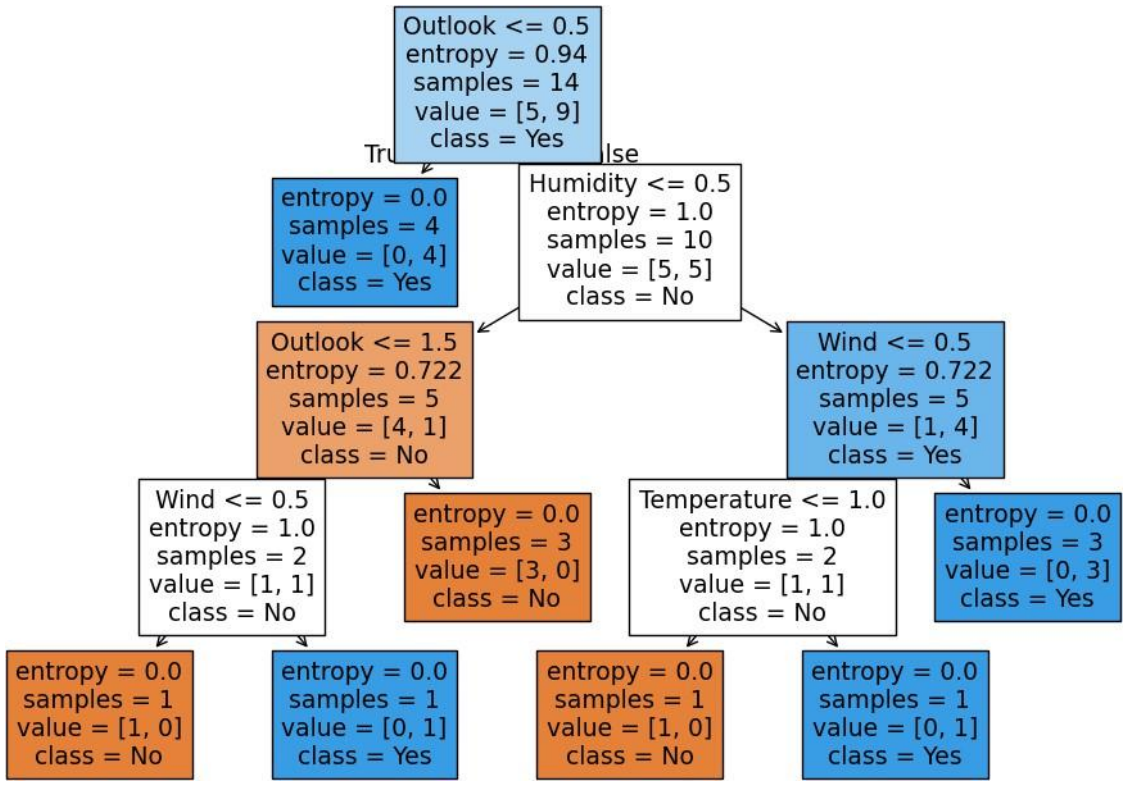
CODE:

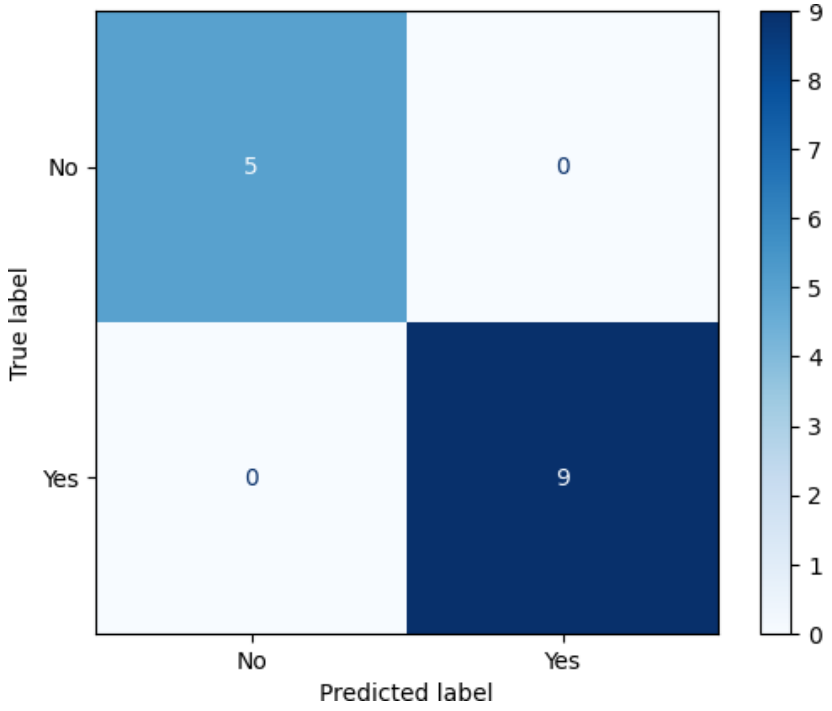
```
import pandas
as pd import
numpy as np
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import confusion_matrix,
ConfusionMatrixDisplay import matplotlib.pyplot as plt


#
#
Da
ta
se
t
#
data = {
    'Outlook': ['Sunny', 'Sunny', 'Overcast', 'Rain', 'Rain', 'Rain',
    'Overcast',
               'Sunny', 'Sunny', 'Rain', 'Sunny', 'Overcast',
               'Overcast',
'Rain'],
    'Temperature': ['Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool',
    'Cool',
                    'Mild', 'Cool', 'Mild', 'Mild', 'Mild', 'Hot',
                    'Mild'],
    'Humidity': ['High', 'High', 'High', 'High', 'Normal', 'Normal',
    'Normal',
                 'High', 'Normal', 'Normal', 'Normal', 'High',
                 'Normal', 'High'],
    'Wind': ['Weak', 'Strong', 'Weak', 'Weak', 'Weak', 'Strong',
    'Strong',
            'Weak', 'Weak', 'Weak', 'Strong', 'Strong', 'Weak',
            'Strong'],
    'PlayTennis': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes',
                   'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No']
}


df = pd.DataFrame(data)


#
# Label
Encoding
#
encoder =
LabelEncoder()
for col in
df.columns:
```

```
    df[col] = encoder.fit_transform(df[col])

X =
df.drop('PlayTennis',
axis=1) y =
df['PlayTennis']

#
#
Decisio
n Tree
#
clf = DecisionTreeClassifier(criterion='entropy',
random_state=0) clf.fit(X, y)


#
# Plot
Decision
Tree #
plt.figure(figsize=(12,8))
plot_tree(clf, feature_names=X.columns, class_names=['No', 'Yes'],
filled=True) plt.show()

#
#
Prediction
Example #
sample = [[0, 1, 0, 1]]
print("Predicted:",
clf.predict(sample))

#
#
Confusion
Matrix #
y_pred = clf.predict(X)
cm = confusion_matrix(y, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=['No', 'Yes']) disp.plot(cmap="Blues")
plt.show()

print("\n--- Confusion Matrix Values
---") print("Confusion Matrix:\n",
cm)
tn, fp, fn, tp = cm.ravel()
print(f"True Negative: {tn}, False Positive: {fp}, False Negative:
{fn}, True Positive: {tp}")
```

```
#
# Entropy &
Information Gain #
def entropy(labels):
    """Calculate entropy of a list of labels"""
    values, counts = np.unique(labels,
    return_counts=True) probs = counts /
    counts.sum()
    return -np.sum(probs * np.log2(probs))

def information_gain(data, feature, target):
    """Calculate Information Gain of a feature relative to the
    target""" total_entropy = entropy(data[target])
    values, counts = np.unique(data[feature], return_counts=True)
    weighted_entropy = 0
    for v, c in zip(values, counts):
            subset = data[data[feature] == v][target]
      weighted_entropy += (c/len(data)) *
      entropy(subset)
      return total_entropy - weighted_entropy


# Dataset before encoding for readability
df_original = pd.DataFrame(data)

print("\n--- Entropy & Information Gain ---")
print(f"Entropy of PlayTennis:
{entropy(df_original['PlayTennis']):.4f}")

   for feature in ['Outlook', 'Temperature',
   'Humidity', 'Wind']: ig =
   information_gain(df_original, feature, 'PlayTennis')
```

OUTPUT:


Outlook <= 0.5
entropy = 0.94
samples = 14
value = [5, 9]
class = Yes
Tru
lse
entropy = 0.0
samples = 4
value = [0, 4]
class = Yes
Humidity <= 0.5
entropy = 1.0
samples = 10
value = [5, 5]
class = No
Outlook <= 1.5
entropy = 0.722
samples = 5
value = [4, 1]
class = No
Wind <= 0.5
entropy = 0.722
samples = 5
value = [1, 4]
class = Yes
Wind <= 0.5
entropy = 1.0
samples = 2
value = [1, 1]
class = No
entropy = 0.0
samples = 3
value = [3, 0]
class = No
Temperature <= 1.0
entropy = 1.0
samples = 2
value = [1, 1]
class = No
entropy = 0.0
samples = 3
value = [0, 3]
class = Yes
entropy = 0.0
samples = 1
value = [1, 0]
class = No
entropy = 0.0
samples = 1
value = [0, 1]
class = Yes
entropy = 0.0
samples = 1
value = [1, 0]
class = No
entropy = 0.0
samples = 1
value = [0, 1]
class = Yes


No
Yes
True label
5
0
0
9
No
Yes
Predicted label
9
8
7
6
5
4
3
2
1
0

```
--- Confusion Matrix Values ---
Confusion Matrix:
 [[5 0]
 [0 9]]
True Negative: 5, False Positive: 0, False Negative: 0, True Positive: 9

--- Entropy & Information Gain ---
Entropy of PlayTennis: 0.9403
Information Gain(Outlook): 0.2467
Information Gain(Temperature): 0.0292
Information Gain(Humidity): 0.1518
Information Gain(Wind): 0.0481
```

Learning Outcomes: