Experiment No. 5
Implement classification algorithm (Decision Tree)
Date of Performance:
Date of Submission:

Aim: To implement Decision Tree classification

Objective: Develop a program to implement decision tree classification

Theory:

Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.

A decision tree is a tree where each -

- Node - a feature(attribute)
- Branch - a decision(rule)

Leaf - an outcome(categorical or continuous

Basic algorithm (a greedy algorithm)

- Tree is constructed in a top-down recursive divide-and-conquer manner
- At start, all the training examples are at the root
- Attributes are categorical (if continuous-valued, they are discretized in advance)
- Examples are partitioned recursively based on selected attributes
- Test attributes are selected on the basis of a heuristic or statistical measure (e.g., information gain)

Conditions for stopping partitioning

- All samples for a given node belong to the same class
- There are no remaining attributes for further partitioning – majority voting is employed for classifying the leaf
- There are no samples left

Entropy

- In ID3, entropy is calculated for each remaining attribute.
- The attribute with the smallest entropy is used to split the set S on that particular iteration.

- Entropy = 0 implies it is of pure class, that means all are of same category

Information Gain

- Information Gain is calculated for a split by subtracting the weighted entropies of each branch from the original entropy
- Information Gain IG(A) tells us how much uncertainty in S was reduced after splitting set S on attribute A.

ID3 stands for Iterative Dichotomiser 3

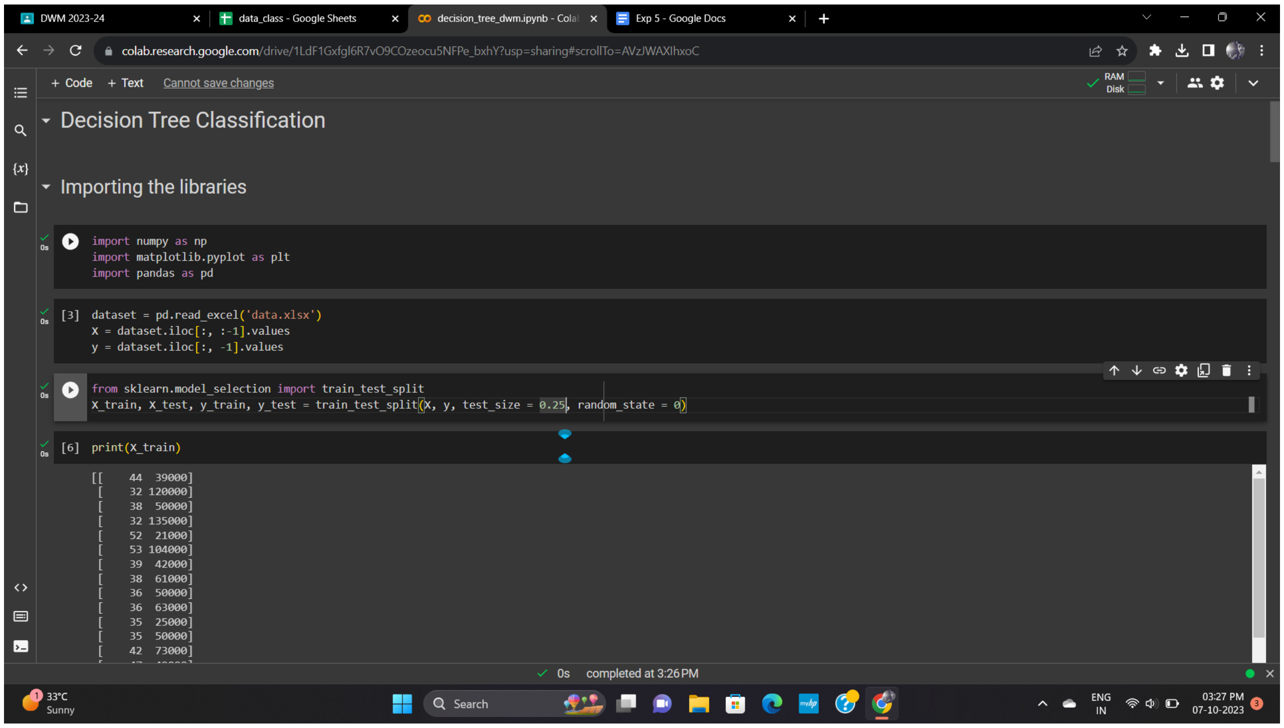
- It is a classification algorithm that follows a greedy approach by selecting a best attribute that yields maximum Information Gain(IG) or minimum Entropy(H).

Steps:

1. Calculate entropy for dataset.
2. For each attribute/feature.
    1. Calculate entropy for all its categorical values.
    2. Calculate information gain for the feature.
3. Find the feature with maximum information gain.

Repeat it until we get the desired tree

Code and output:



Decision Tree Classification

Importing the libraries

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

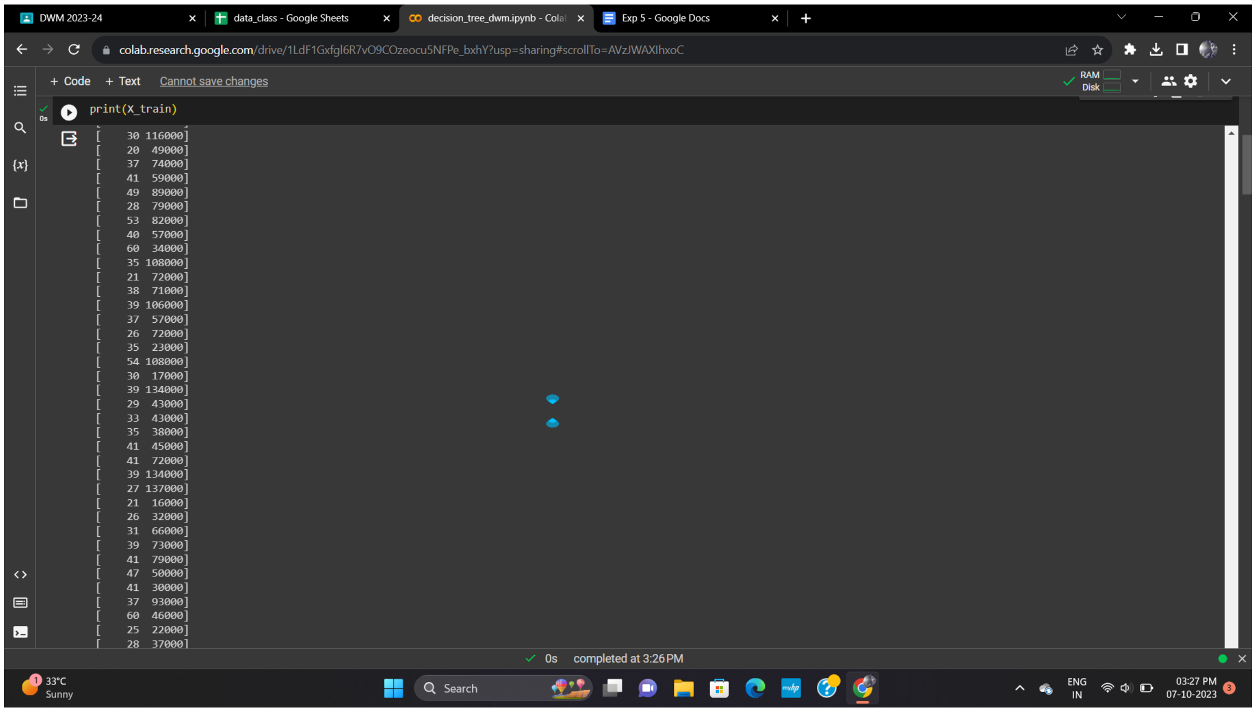
```
[3] dataset = pd.read_excel('data.xlsx')
    X = dataset.iloc[:, :-1].values
    y = dataset.iloc[:, -1].values
```

```
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, random_state = 0)
```

```
[6] print(X_train)
```

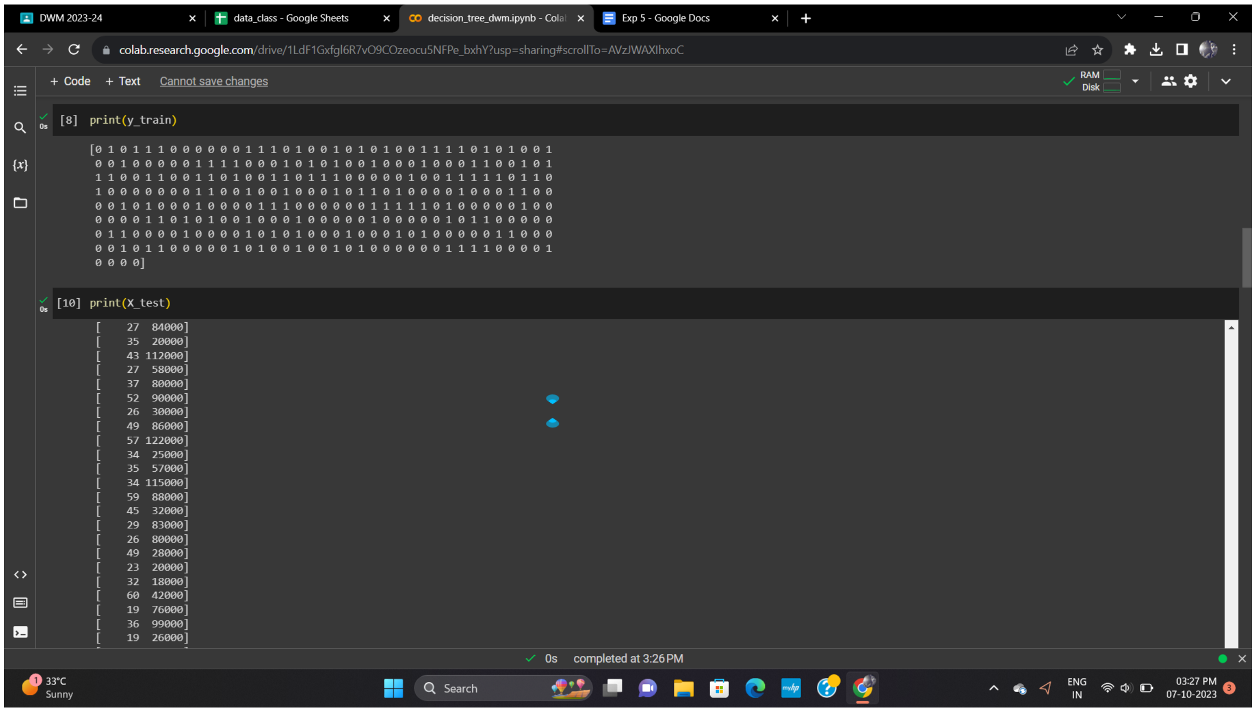
```
[[    44  39000]
 [    32 120000]
 [    38  50000]
 [    32 135000]
 [    52  21000]
 [    53 104000]
 [    39  42000]
 [    38  61000]
 [    36  50000]
 [    36  63000]
 [    35  25000]
 [    35  50000]
 [    42  73000]
```

✓ 0s completed at 3:26 PM



```
print(X_train)

 [    30 116000]
 [    20  49000]
 [    37  74000]
 [    41  59000]
 [    49  89000]
 [    28  79000]
 [    53  82000]
 [    40  57000]
 [    60  34000]
 [    35 108000]
 [    21  72000]
 [    38  71000]
 [    39 106000]
 [    37  57000]
 [    26  72000]
 [    35  23000]
 [    54 108000]
 [    30  17000]
 [    39 134000]
 [    29  43000]
 [    33  43000]
 [    35  38000]
 [    41  45000]
 [    41  72000]
 [    39 134000]
 [    27 137000]
 [    21  16000]
 [    26  32000]
 [    31  66000]
 [    39  73000]
 [    41  79000]
 [    47  50000]
 [    41  30000]
 [    37  93000]
 [    60  46000]
 [    25  22000]
 [    28  37000]
```

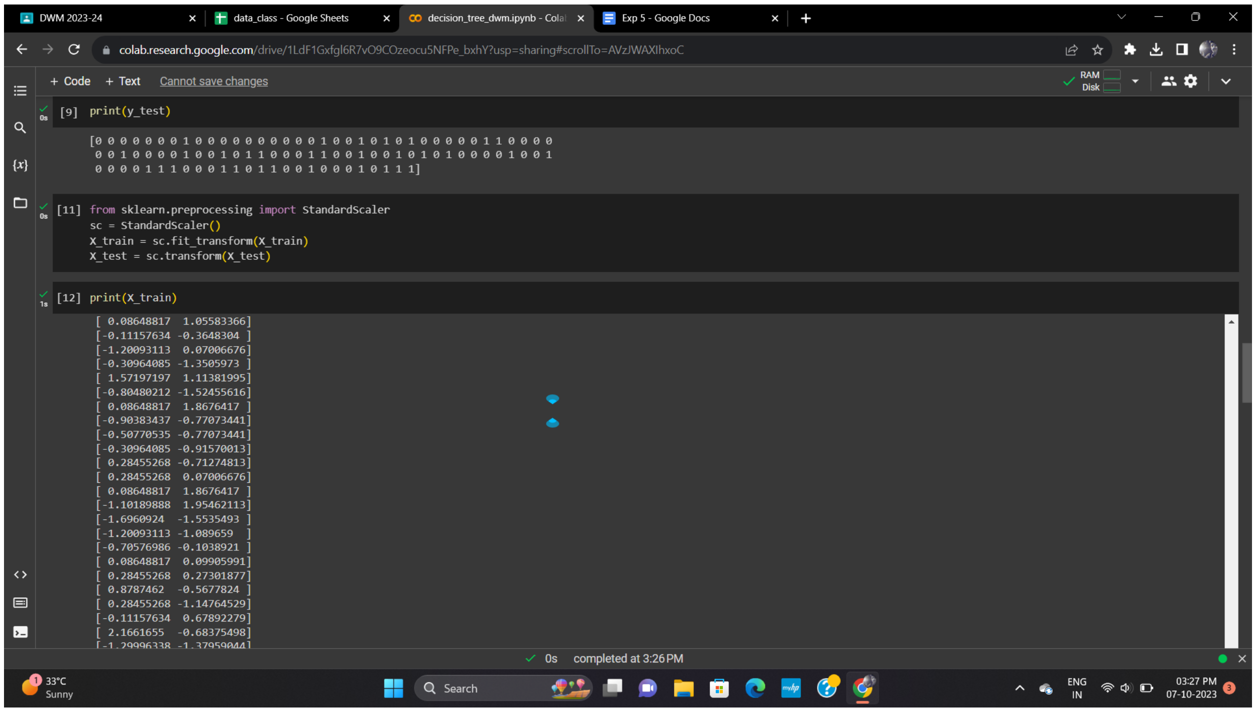


```
[8] print(y_train)

[0 1 0 1 1 1 0 0 0 0 0 0 1 1 1 0 1 0 0 1 0 1 0 1 0 0 1 1 1 1 0 1 0 1 0 0 1
 0 0 1 0 0 0 0 0 1 1 1 1 0 0 0 1 0 1 0 1 0 0 1 0 0 0 1 0 0 0 1 1 0 0 1 0 1
 1 1 0 0 1 1 0 0 1 1 0 1 0 0 1 1 0 1 1 1 0 0 0 0 0 1 0 0 1 1 1 1 1 0 1 1 0
 1 0 0 0 0 0 0 0 1 1 0 0 1 0 0 1 0 0 0 1 0 1 1 0 1 0 0 0 0 1 0 0 0 1 1 0 0
 0 0 1 0 1 0 0 0 1 0 0 0 0 1 1 1 0 0 0 0 0 0 1 1 1 1 1 0 1 0 0 0 0 0 1 0 0
 0 0 0 0 1 1 0 1 0 1 0 0 1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 1 1 0 0 0 0 0
 0 1 1 0 0 0 0 1 0 0 0 0 1 0 1 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 0 0 1 1 0 0 0
 0 0 1 0 1 1 0 0 0 0 0 1 0 1 0 0 1 0 0 1 0 1 0 0 0 0 0 0 1 1 1 1 0 0 0 0 1
 0 0 0 0]

[10] print(X_test)

 [    27  84000]
 [    35  20000]
 [    43 112000]
 [    27  58000]
 [    37  80000]
 [    52  90000]
 [    26  30000]
 [    49  86000]
 [    57 122000]
 [    34  25000]
 [    35  57000]
 [    34 115000]
 [    59  88000]
 [    45  32000]
 [    29  83000]
 [    26  80000]
 [    49  28000]
 [    23  20000]
 [    32  18000]
 [    60  42000]
 [    19  76000]
 [    36  99000]
 [    19  26000]
```



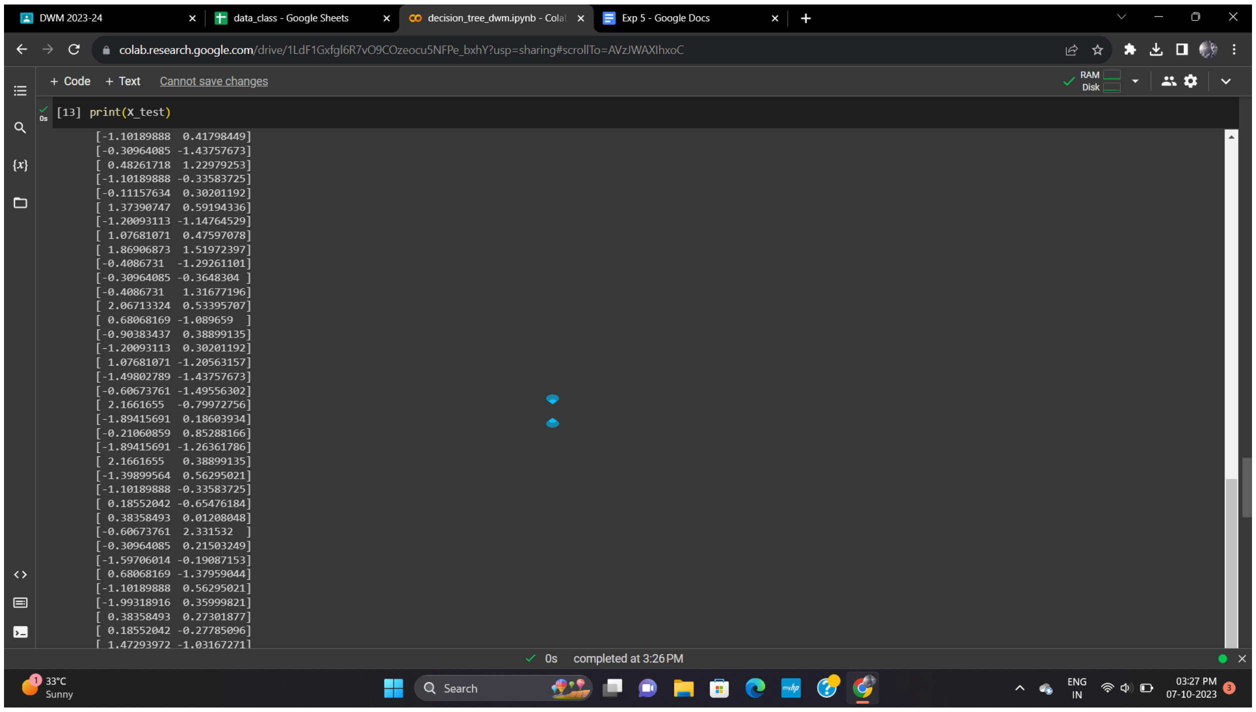
```
[9] print(y_test)

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

```
[11] from sklearn.preprocessing import StandardScaler
     sc = StandardScaler()
     X_train = sc.fit_transform(X_train)
     X_test = sc.transform(X_test)
```

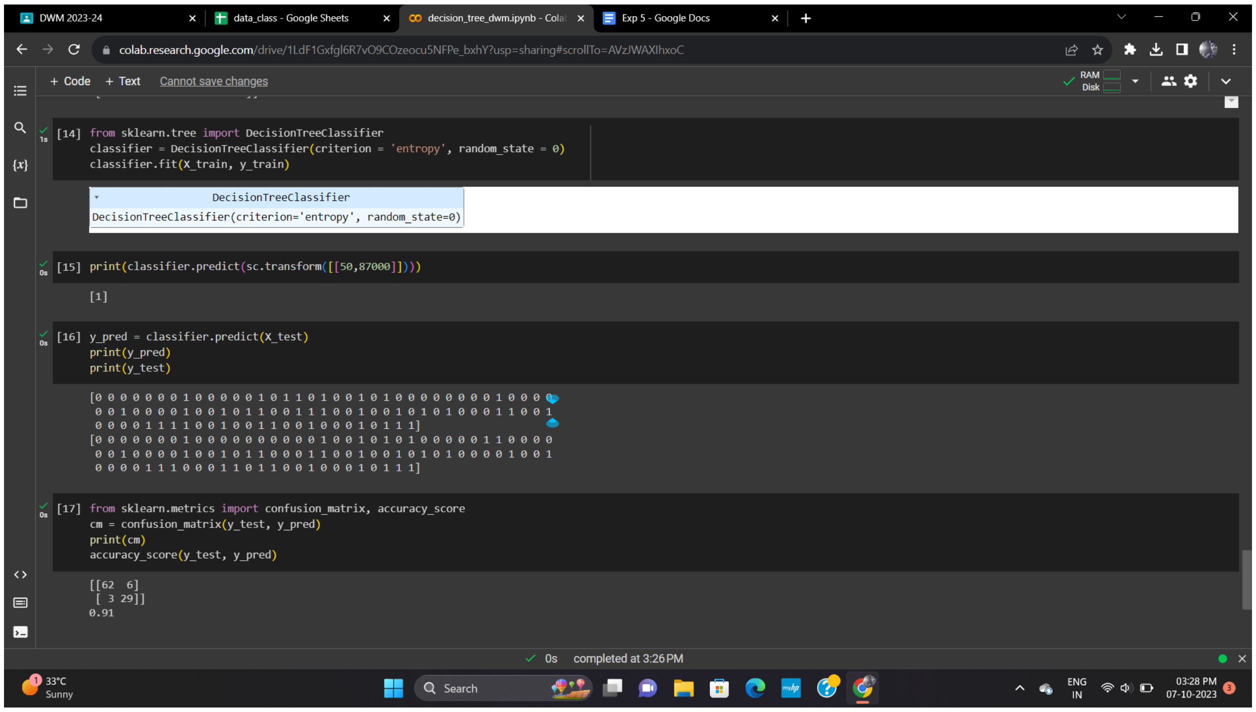
```
[12] print(X_train)
 [ 0.08648817  1.05583366]
 [-0.11157634 -0.3648304 ]
 [-1.20093113  0.07006676]
 [-0.30964085 -1.3505973 ]
 [ 1.57197197  1.11381995]
 [-0.80480212 -1.52455616]
 [ 0.08648817  1.8676417 ]
 [-0.90383437 -0.77073441]
 [-0.50770535 -0.77073441]
 [-0.30964085 -0.91570013]
 [ 0.28455268 -0.71274813]
 [ 0.28455268  0.07006676]
 [ 0.08648817  1.8676417 ]
 [-1.10189888  1.95462113]
 [-1.6960924  -1.5535493 ]
 [-1.20093113 -1.089659  ]
 [-0.70576986 -0.1038921 ]
 [ 0.08648817  0.09905991]
 [ 0.28455268  0.27301877]
 [ 0.8787462  -0.5677824 ]
 [ 0.28455268 -1.14764529]
 [-0.11157634  0.67892279]
 [ 2.1661655  -0.68375498]
 [-1.29996338 -1.37959044]
```

0s completed at 3:26 PM



```
[13] print(X_test)
 [-1.10189888  0.41798449]
 [-0.30964085 -1.43757673]
 [ 0.48261718  1.22979253]
 [-1.10189888 -0.33583725]
 [-0.11157634  0.30201192]
 [ 1.37390747  0.59194336]
 [-1.20093113 -1.14764529]
 [ 1.07681071  0.47597078]
 [ 1.86906873  1.51972397]
 [-0.4086731  -1.29261101]
 [-0.30964085 -0.3648304 ]
 [-0.4086731   1.31677196]
 [ 2.06713324  0.53395707]
 [ 0.68068169 -1.089659  ]
 [-0.90383437  0.38899135]
 [-1.20093113  0.30201192]
 [ 1.07681071 -1.20563157]
 [-1.49802789 -1.43757673]
 [-0.60673761 -1.49556302]
 [ 2.1661655  -0.79972756]
 [-1.89415691  0.18603934]
 [-0.21060859  0.85288166]
 [-1.89415691 -1.26361786]
 [ 2.1661655   0.38899135]
 [-1.39899564  0.56295021]
 [-1.10189888 -0.33583725]
 [ 0.18552042 -0.65476184]
 [ 0.38358493  0.01208048]
 [-0.60673761  2.331532  ]
 [-0.30964085  0.21503249]
 [-1.59706014 -0.19087153]
 [ 0.68068169 -1.37959044]
 [-1.10189888  0.56295021]
 [-1.99318916  0.35999821]
 [ 0.38358493  0.27301877]
 [ 0.18552042 -0.27785096]
 [ 1.47293972 -1.03167271]
```

0s completed at 3:26 PM


DWM 2023-24
data_class - Google Sheets
decision_tree_dwm.ipynb - Colab
Exp 5 - Google Docs
colab.research.google.com/drive/1LdF1Gxfgl6R7vO9COzeocu5NFPe_bxhY?usp=sharing#scrollTo=AVzJWAXlhxoC
+ Code
+ Text
Cannot save changes
RAM
Disk

```
[14] from sklearn.tree import DecisionTreeClassifier
     classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
     classifier.fit(X_train, y_train)

     DecisionTreeClassifier
     DecisionTreeClassifier(criterion='entropy', random_state=0)

[15] print(classifier.predict(sc.transform([[50,87000]])))

     [1]

[16] y_pred = classifier.predict(X_test)
     print(y_pred)
     print(y_test)

     [0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 1 1 0 1 0 0 1 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0
      0 0 1 0 0 0 0 1 0 0 1 0 1 1 0 0 1 1 1 0 0 1 0 0 1 0 1 0 1 0 0 0 1 1 0 0 1
      0 0 0 0 1 1 1 1 0 0 1 0 0 1 1 0 0 1 0 0 0 1 0 1 1 1]
     [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 1 0 0 0 0 0 1 1 0 0 0 0
      0 0 1 0 0 0 0 1 0 0 1 0 1 1 0 0 0 1 1 0 0 1 0 0 1 0 1 0 1 0 0 0 0 1 0 0 1
      0 0 0 0 1 1 1 0 0 0 1 1 0 1 1 0 0 1 0 0 0 1 0 1 1 1]

[17] from sklearn.metrics import confusion_matrix, accuracy_score
     cm = confusion_matrix(y_test, y_pred)
     print(cm)
     accuracy_score(y_test, y_pred)

     [[62  6]
      [ 3 29]]
     0.91
```

Conclusion: The goal of using a Decision Tree is to create a training model that is used to predict the class or value of the target variable by learning simple decision rules inferred from prior data.The accuracy of the trained data is 0.91.