Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Department of Computer Science and Engineering (Data Science)

JHANVI PAREKH
60009210033
CSE (Data Science)

Subject: Machine Learning – I (DJ19DSC402)

AY: 2022-23

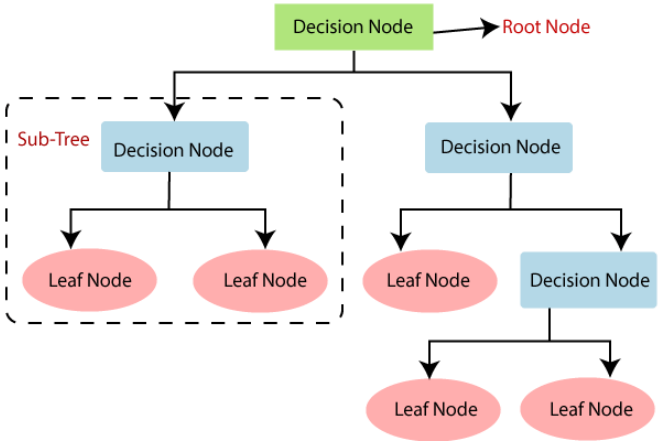
Experiment 2 - 3

(Decision Tree)

Aim: Implement Decision Tree on the given Datasets to build a classifier and Regressor. Apply appropriate pruning method to overcome overfitting.

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.** In a Decision tree, there are two nodes, which are the **Decision Node** and **Leaf Node.**
Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.
The decisions or the test are performed on the basis of features of the given dataset.
It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions. It is called a decision tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure.
A decision tree simply asks a question, and based on the answer (Yes/No), it further split the tree into subtrees. Below diagram explains the general structure of a decision tree:



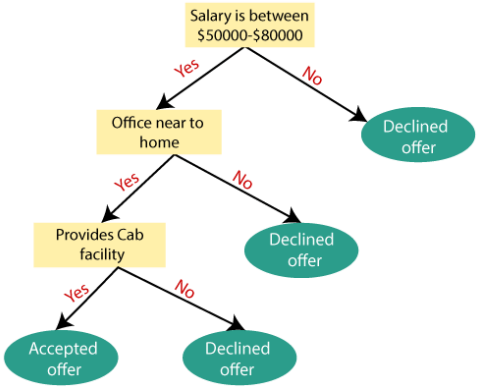
Decision Tree Terminologies

Root Node: Root node is from where the decision tree starts. It represents the entire dataset, which further gets divided into two or more homogeneous sets.

Leaf Node: Leaf nodes are the final output node, and the tree cannot be segregated further after getting a leaf node.
Splitting: Splitting is the process of dividing the decision node/root node into sub-nodes according to the given conditions.
Branch/Sub Tree: A tree formed by splitting the tree.
Pruning: Pruning is the process of removing the unwanted branches from the tree.
Parent/Child node: The root node of the tree is called the parent node, and other nodes are called the child nodes.

Steps in building a Tree
Step-1: Begin the tree with the root node, says S, which contains the complete dataset.
Step-2: Find the best attribute in the dataset using **Attribute Selection Measure (ASM).**
Step-3: Divide the S into subsets that contains possible values for the best attributes.
Step-4: Generate the decision tree node, which contains the best attribute.
Step-5: Recursively make new decision trees using the subsets of the dataset created in step -3. Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node.

Example: Suppose there is a candidate who has a job offer and wants to decide whether he should accept the offer or Not. So, to solve this problem, the decision tree starts with the root node (Salary attribute by ASM). The root node splits further into the next decision node (distance from the office) and one leaf node based on the corresponding labels. The next decision node further gets split into one decision node (Cab facility) and one leaf node. Finally, the decision node splits into two leaf nodes (Accepted offers and Declined offer). Consider the below diagram:



Attribute Selection Measures
While implementing a Decision tree, the main issue arises that how to select the best attribute for the root node and for sub-nodes. So, to solve such problems there is a technique which is called as **Attribute selection measure or ASM.** By this measurement, we can easily select the best attribute for the nodes of the tree. There are two popular techniques for ASM, which are:
1. Information Gain:
Information gain is the measurement of changes in entropy after the segmentation of a dataset based on an attribute. It calculates how much information a feature provides us about a class.

According to the value of information gain, we split the node and build the decision tree.
A decision tree algorithm always tries to maximize the value of information gain, and a node/attribute having the highest information gain is split first. It can be calculated using the below formula:
Information Gain= Entropy(S)- [(Weighted Avg) *Entropy(each feature)
Entropy: Entropy is a metric to measure the impurity in a given attribute. It specifies randomness in data. Entropy can be calculated as:

Entropy(s)= -P(yes)log2 P(yes)- P(no) log2 P(no)

Where,
S= Total number of samples
P(yes)= probability of yes
P(no)= probability of no

2. Gini Index:
Gini index is a measure of impurity or purity used while creating a decision tree in the CART(Classification and Regression Tree) algorithm.
An attribute with the low Gini index should be preferred as compared to the high Gini index.
It only creates binary splits, and the CART algorithm uses the Gini index to create binary splits.
Gini index can be calculated using the below formula:

$$\text{Gini Index} = 1 - \sum_j P_j^2$$

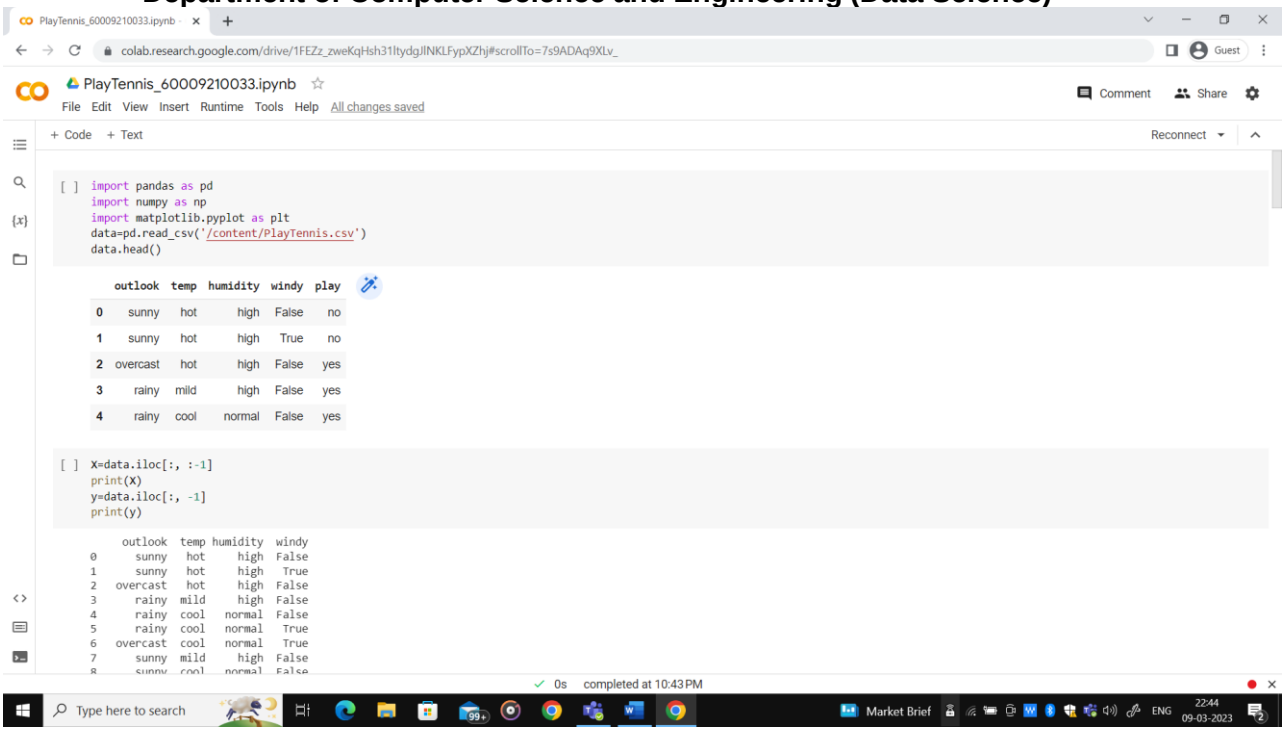
Pruning: Getting an Optimal Decision tree
Pruning is a process of deleting the unnecessary nodes from a tree in order to get the optimal decision tree. A too-large tree increases the risk of overfitting, and a small tree may not capture all the important features of the dataset. Therefore, a technique that decreases the size of the learning tree without reducing accuracy is known as Pruning. There are mainly two types of tree **pruning** technology used:

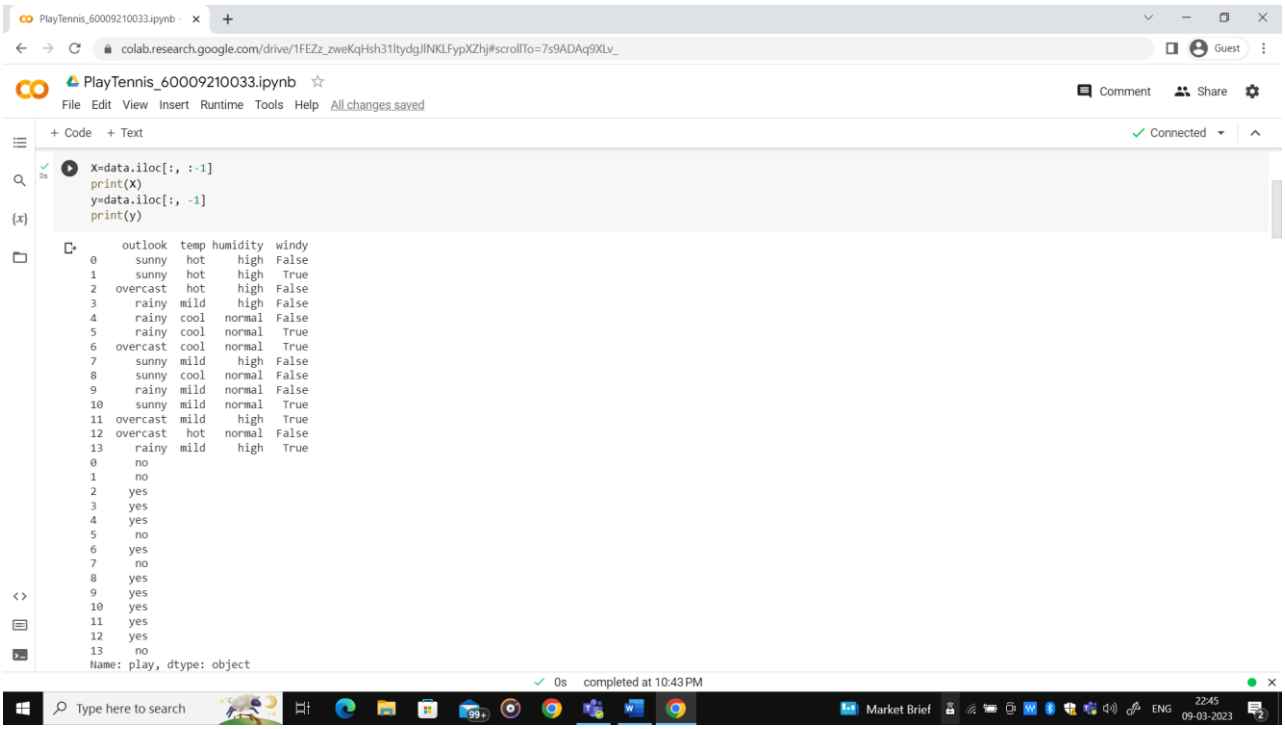
- Cost Complexity Pruning
- Reduced Error Pruning.

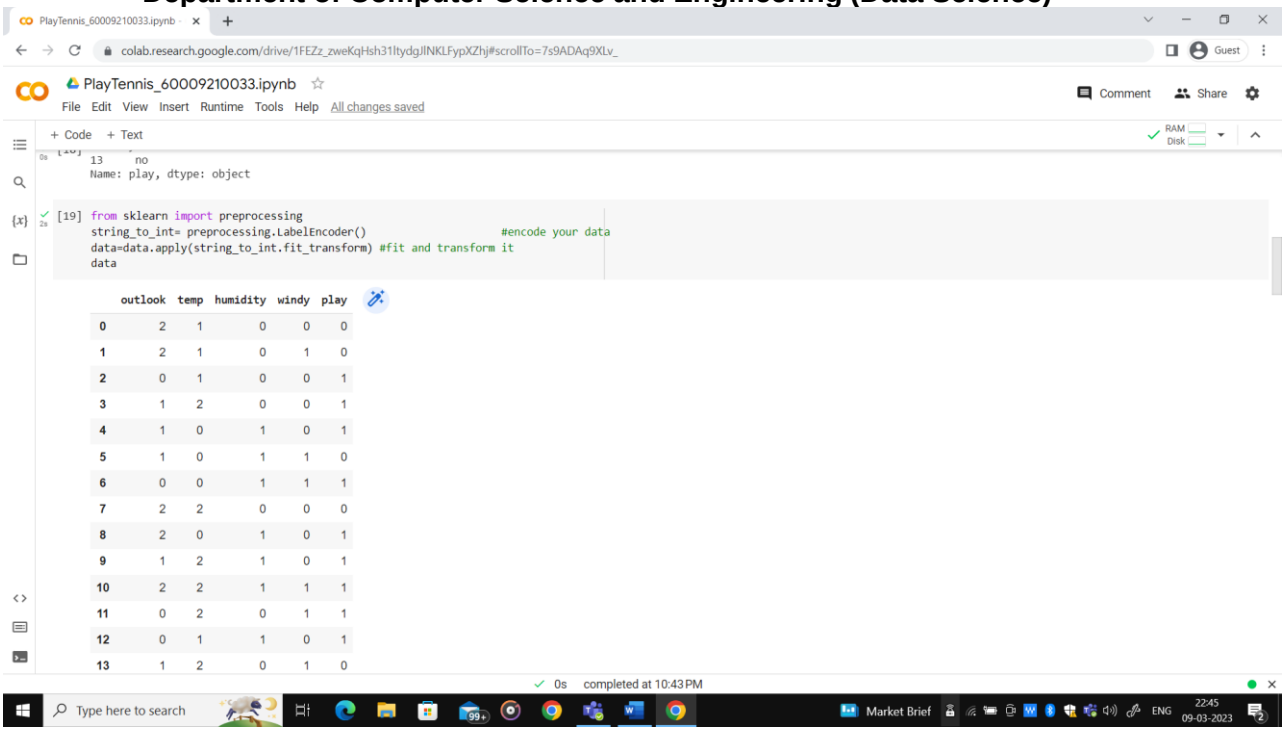
Lab Assignments to complete in this session:

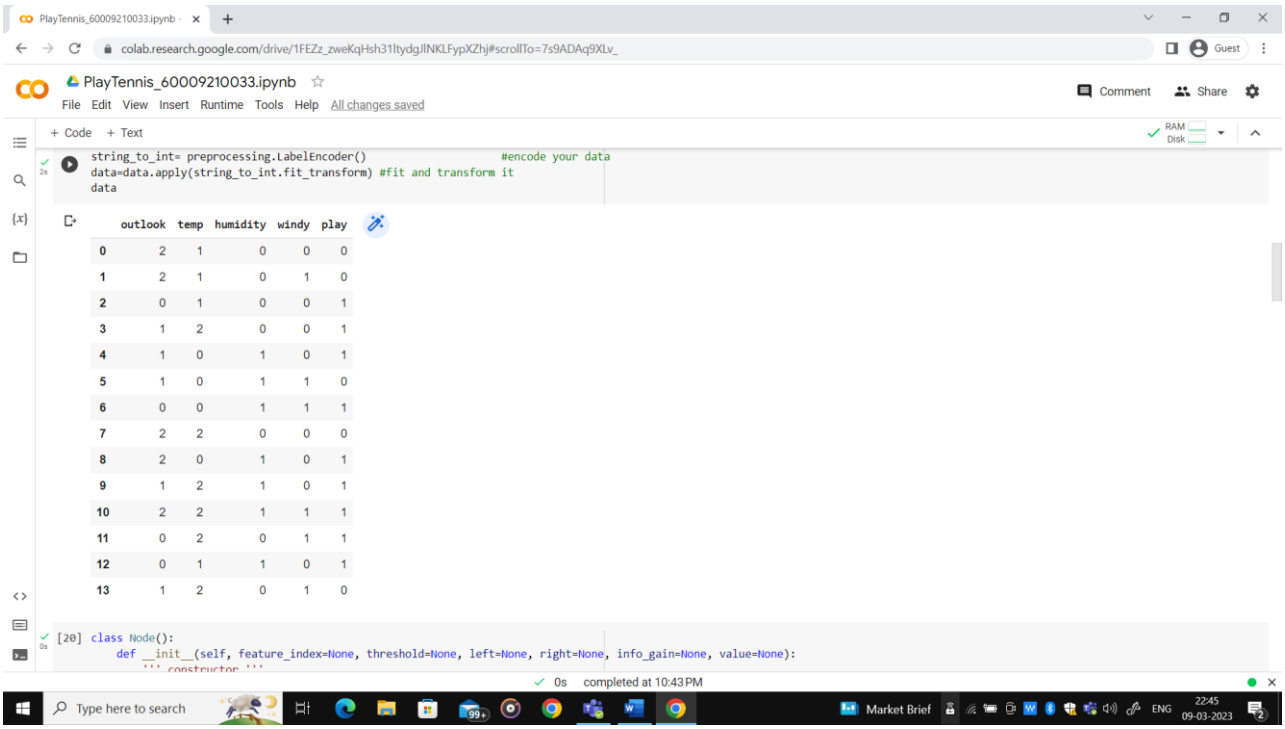
Use the given dataset and perform the following tasks:
Dataset 1: PlayTennis.csv
Dataset 2: Iris.csv
Dataset 3: Breastcancer.csv
Dataset 4: car prediction.csv

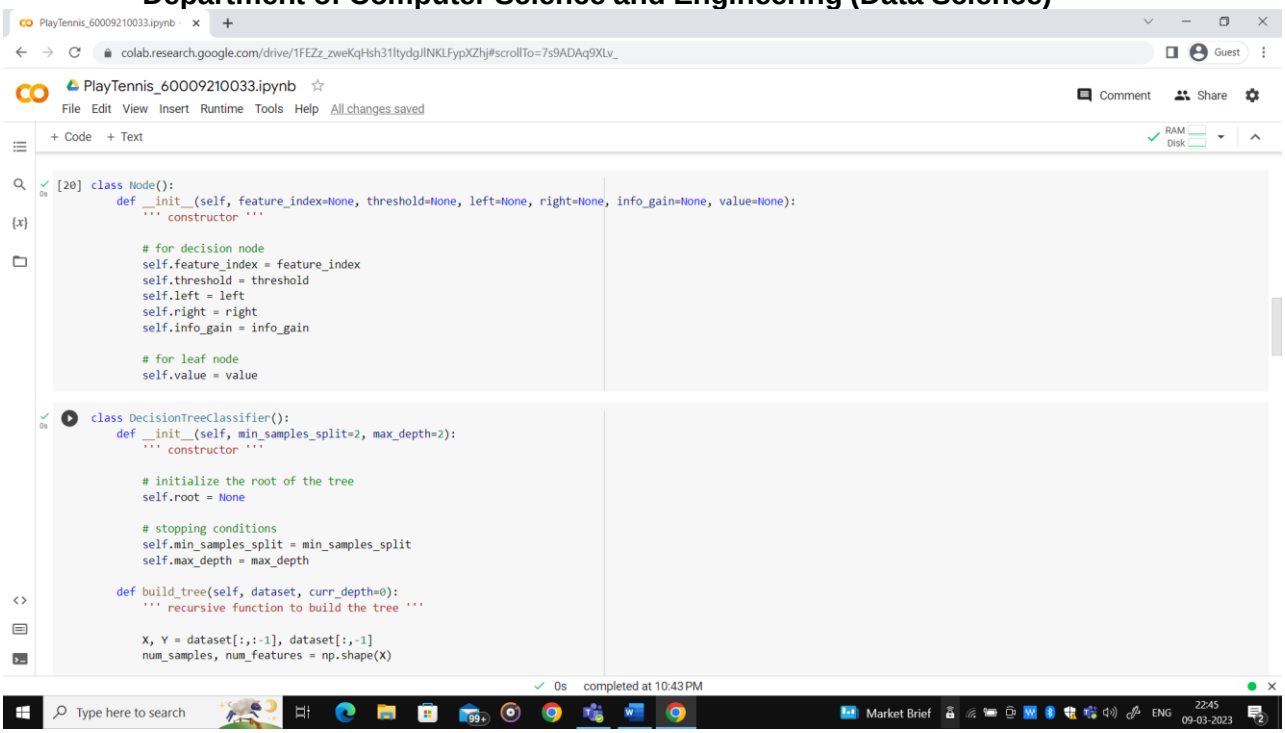
1. Implement Decision tree classifier from scratch using Dataset 1 by defining Node class and Tree class.

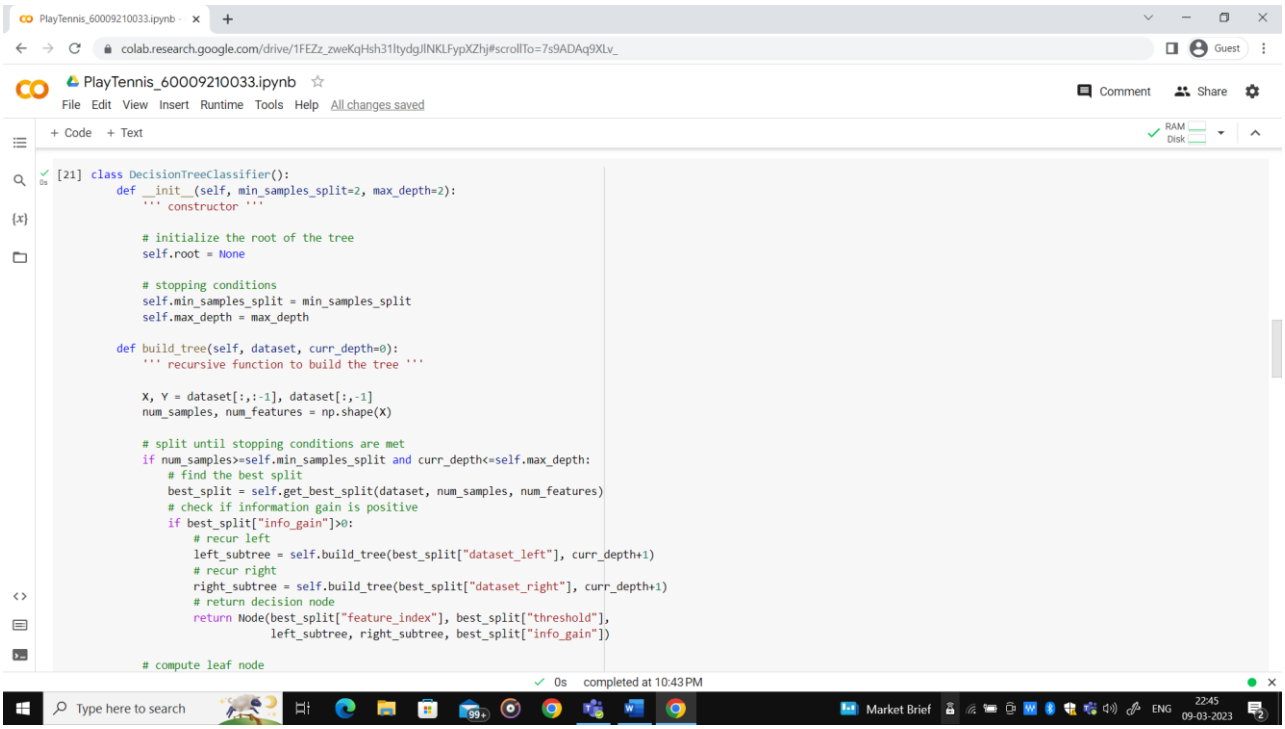

PlayTennis_60009210033.ipynb
colab.research.google.com/drive/1FEZz_zweKqHsh31ltydgJlNKLFypXZhj#scrollTo=7s9ADAq9XLv_
PlayTennis_60009210033.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment Share
+ Code + Text
Reconnect
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
data=pd.read_csv('/content/PlayTennis.csv')
data.head()
outlook temp humidity windy play
0 sunny hot high False no
1 sunny hot high True no
2 overcast hot high False yes
3 rainy mild high False yes
4 rainy cool normal False yes
X=data.iloc[:, :-1]
print(X)
y=data.iloc[:, -1]
print(y)
outlook temp humidity windy
0 sunny hot high False
1 sunny hot high True
2 overcast hot high False
3 rainy mild high False
4 rainy cool normal False
5 rainy cool normal True
6 overcast cool normal True
7 sunny mild high False
0s completed at 10:43 PM
Type here to search
Market Brief
ENG
22:44
09-03-2023

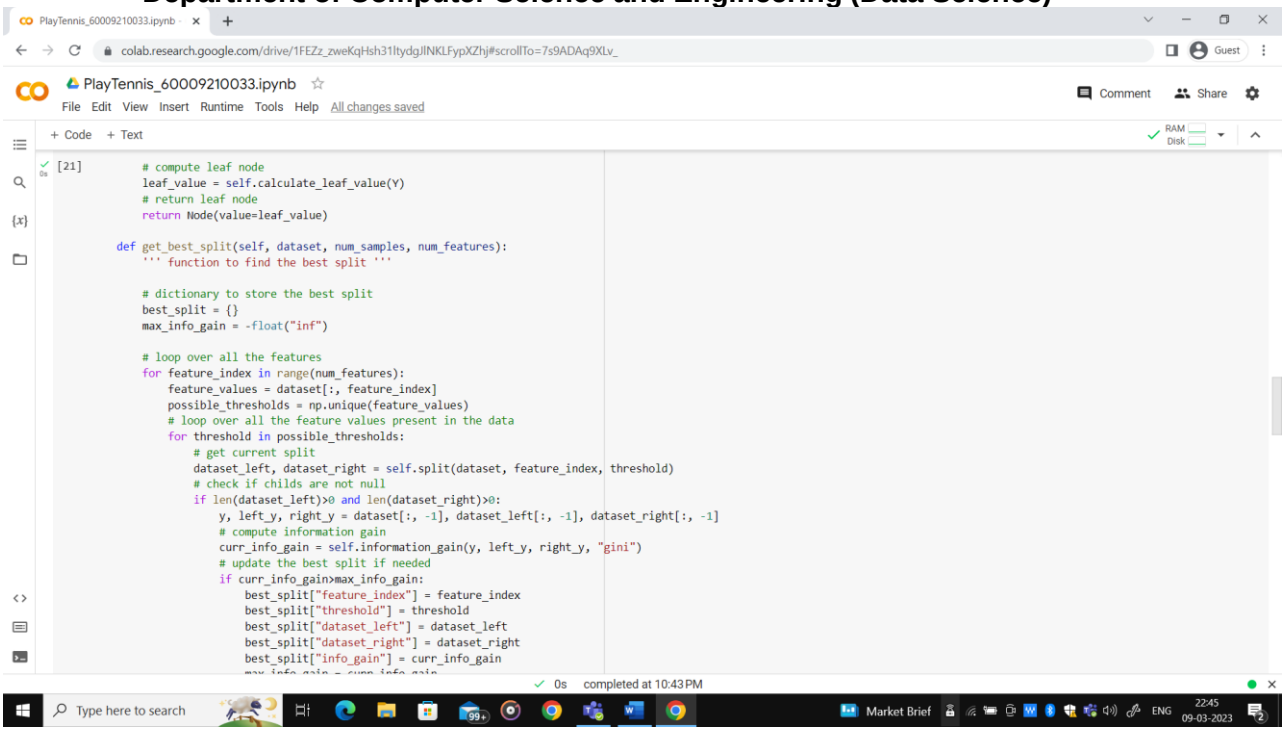

PlayTennis_60009210033.ipynb
colab.research.google.com/drive/1FEZz_zweKqHsh31ltydgJlNKLFypXZhj#scrollTo=7s9ADAq9XLv_
PlayTennis_60009210033.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment Share
+ Code + Text
Connected
X=data.iloc[:, :-1]
print(X)
y=data.iloc[:, -1]
print(y)
outlook temp humidity windy
0 sunny hot high False
1 sunny hot high True
2 overcast hot high False
3 rainy mild high False
4 rainy cool normal False
5 rainy cool normal True
6 overcast cool normal True
7 sunny mild high False
8 sunny cool normal False
9 rainy mild normal False
10 sunny mild normal True
11 overcast mild high True
12 overcast hot normal False
13 rainy mild high True
0 no
1 no
2 yes
3 yes
4 yes
5 no
6 yes
7 no
8 yes
9 yes
10 yes
11 yes
12 yes
13 no
Name: play, dtype: object
0s completed at 10:43 PM
Type here to search
Market Brief
ENG
22:45
09-03-2023


PlayTennis_60009210033.ipynb
colab.research.google.com/drive/1FEZz_zweKqHsh31ltydgJlNKLFypXZhj#scrollTo=7s9ADAq9XLv_
PlayTennis_60009210033.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
13 no
Name: play, dtype: object
[19] from sklearn import preprocessing
string_to_int= preprocessing.LabelEncoder() #encode your data
data=data.apply(string_to_int.fit_transform) #fit and transform it
data
outlook temp humidity windy play
0 2 1 0 0 0
1 2 1 0 1 0
2 0 1 0 0 1
3 1 2 0 0 1
4 1 0 1 0 1
5 1 0 1 1 0
6 0 0 1 1 1
7 2 2 0 0 0
8 2 0 1 0 1
9 1 2 1 0 1
10 2 2 1 1 1
11 0 2 0 1 1
12 0 1 1 0 1
13 1 2 0 1 0
0s completed at 10:43 PM
Type here to search
Market Brief
ENG
22:45
09-03-2023


PlayTennis_60009210033.ipynb
colab.research.google.com/drive/1FEZz_zweKqHsh31ltydgJlNKLFypXZhj#scrollTo=7s9ADAq9XLv_
PlayTennis_60009210033.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
string_to_int= preprocessing.LabelEncoder() #encode your data
data=data.apply(string_to_int.fit_transform) #fit and transform it
data
outlook temp humidity windy play
0 2 1 0 0 0
1 2 1 0 1 0
2 0 1 0 0 1
3 1 2 0 0 1
4 1 0 1 0 1
5 1 0 1 1 0
6 0 0 1 1 1
7 2 2 0 0 0
8 2 0 1 0 1
9 1 2 1 0 1
10 2 2 1 1 1
11 0 2 0 1 1
12 0 1 1 0 1
13 1 2 0 1 0
[20] class Node():
def __init__(self, feature_index=None, threshold=None, left=None, right=None, info_gain=None, value=None):
''' constructor '''
0s completed at 10:43 PM
Type here to search
Market Brief
ENG
22:45
09-03-2023


PlayTennis_60009210033.ipynb
colab.research.google.com/drive/1FEZz_zweKqHsh31ltydgJlNKLFypXZhj#scrollTo=7s9ADAq9XLv_
PlayTennis_60009210033.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
[20] class Node():
def __init__(self, feature_index=None, threshold=None, left=None, right=None, info_gain=None, value=None):
''' constructor '''
# for decision node
self.feature_index = feature_index
self.threshold = threshold
self.left = left
self.right = right
self.info_gain = info_gain
# for leaf node
self.value = value
class DecisionTreeClassifier():
def __init__(self, min_samples_split=2, max_depth=2):
''' constructor '''
# initialize the root of the tree
self.root = None
# stopping conditions
self.min_samples_split = min_samples_split
self.max_depth = max_depth
def build_tree(self, dataset, curr_depth=0):
''' recursive function to build the tree '''
X, Y = dataset[:,:-1], dataset[:,-1]
num_samples, num_features = np.shape(X)
0s completed at 10:43 PM
Type here to search
Market Brief
ENG
22:45
09-03-2023


PlayTennis_60009210033.ipynb
colab.research.google.com/drive/1FEZz_zweKqHsh31ltydgJlNKLFypXZhj#scrollTo=7s9ADAq9XLv_
PlayTennis_60009210033.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
[21] class DecisionTreeClassifier():
def __init__(self, min_samples_split=2, max_depth=2):
''' constructor '''
# initialize the root of the tree
self.root = None
# stopping conditions
self.min_samples_split = min_samples_split
self.max_depth = max_depth
def build_tree(self, dataset, curr_depth=0):
''' recursive function to build the tree '''
X, Y = dataset[:,:-1], dataset[:,-1]
num_samples, num_features = np.shape(X)
# split until stopping conditions are met
if num_samples>=self.min_samples_split and curr_depth<=self.max_depth:
# find the best split
best_split = self.get_best_split(dataset, num_samples, num_features)
# check if information gain is positive
if best_split["info_gain"]>0:
# recur left
left_subtree = self.build_tree(best_split["dataset_left"], curr_depth+1)
# recur right
right_subtree = self.build_tree(best_split["dataset_right"], curr_depth+1)
# return decision node
return Node(best_split["feature_index"], best_split["threshold"],
left_subtree, right_subtree, best_split["info_gain"])
compute leaf node
0s completed at 10:43 PM
Type here to search
Market Brief
ENG
22:45
09-03-2023



PlayTennis_60009210033.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

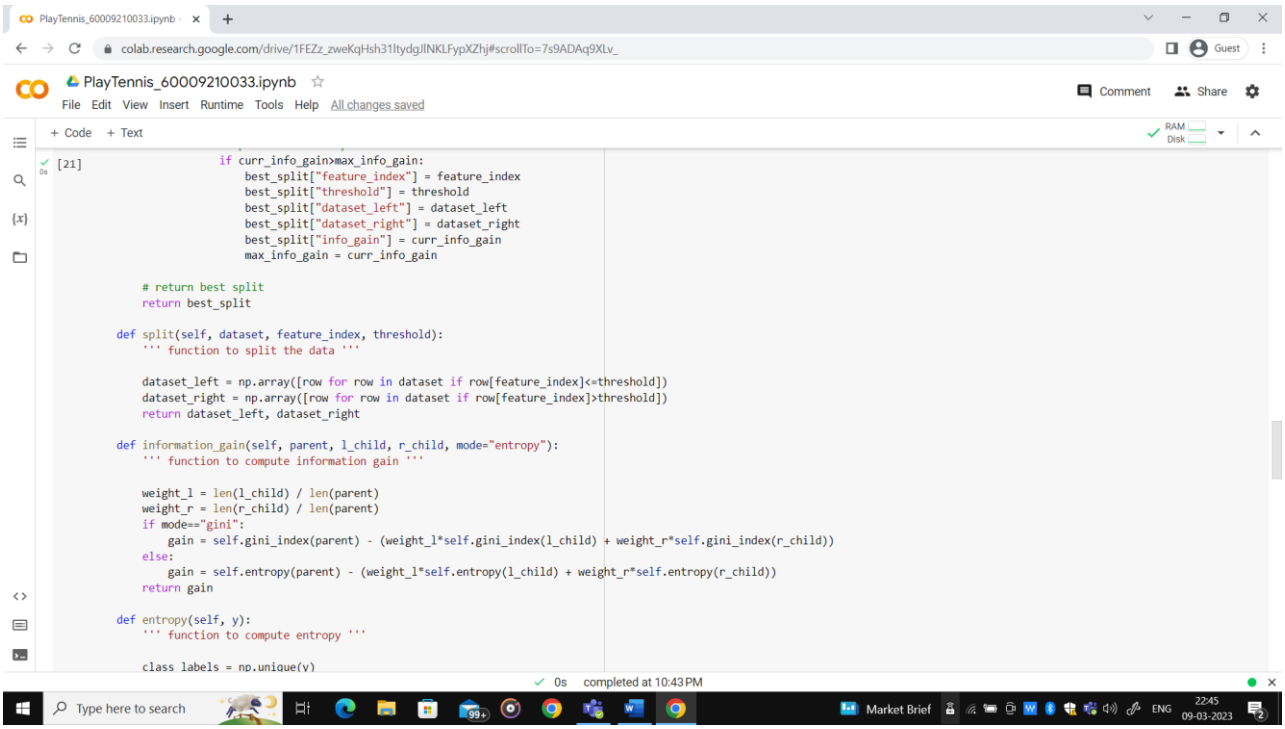
+ Code + Text

RAM Disk

[21]

```
        # compute leaf node
        leaf_value = self.calculate_leaf_value(Y)
        # return leaf node
        return Node(value=leaf_value)

    def get_best_split(self, dataset, num_samples, num_features):
        ''' function to find the best split '''

        # dictionary to store the best split
        best_split = {}
        max_info_gain = -float("inf")

        # loop over all the features
        for feature_index in range(num_features):
            feature_values = dataset[:, feature_index]
            possible_thresholds = np.unique(feature_values)
            # loop over all the feature values present in the data
            for threshold in possible_thresholds:
                # get current split
                dataset_left, dataset_right = self.split(dataset, feature_index, threshold)
                # check if childs are not null
                if len(dataset_left)>0 and len(dataset_right)>0:
                    y, left_y, right_y = dataset[:, -1], dataset_left[:, -1], dataset_right[:, -1]
                    # compute information gain
                    curr_info_gain = self.information_gain(y, left_y, right_y, "gini")
                    # update the best split if needed
                    if curr_info_gain>max_info_gain:
                        best_split["feature_index"] = feature_index
                        best_split["threshold"] = threshold
                        best_split["dataset_left"] = dataset_left
                        best_split["dataset_right"] = dataset_right
                        best_split["info_gain"] = curr_info_gain
```

0s completed at 10:43 PM



PlayTennis_60009210033.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

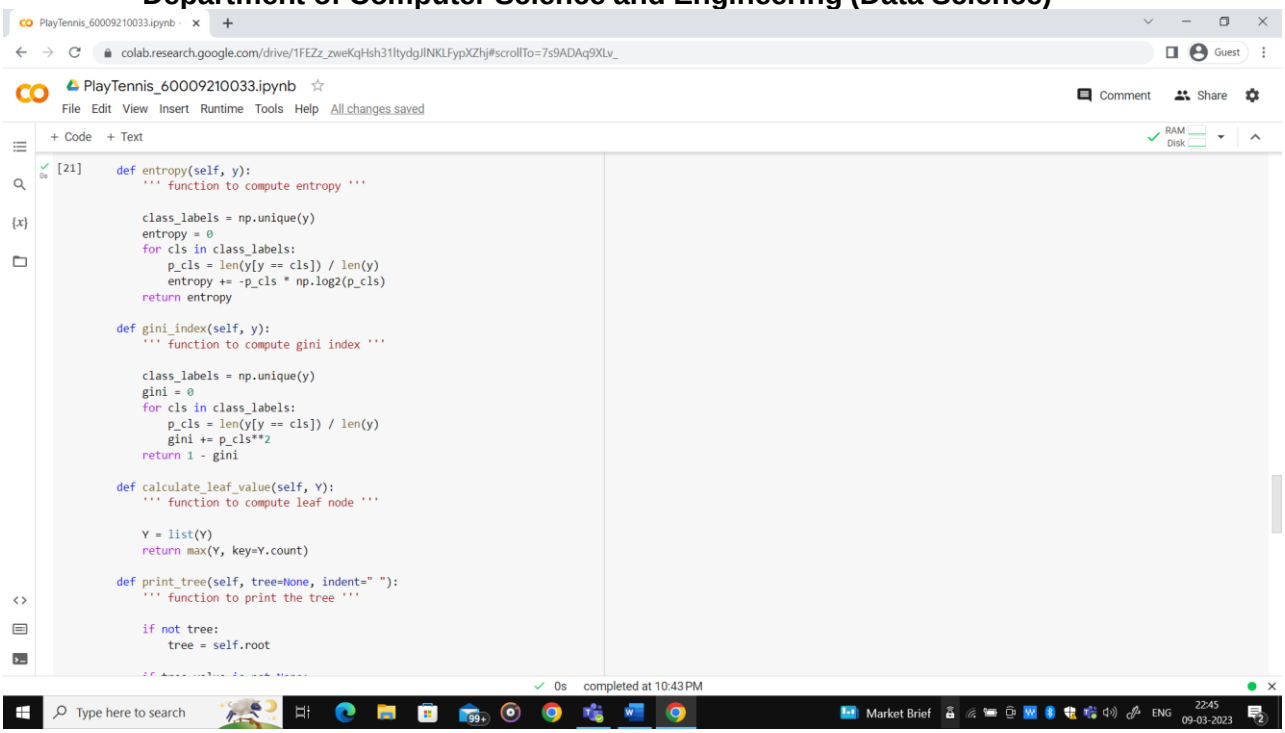
+ Code + Text

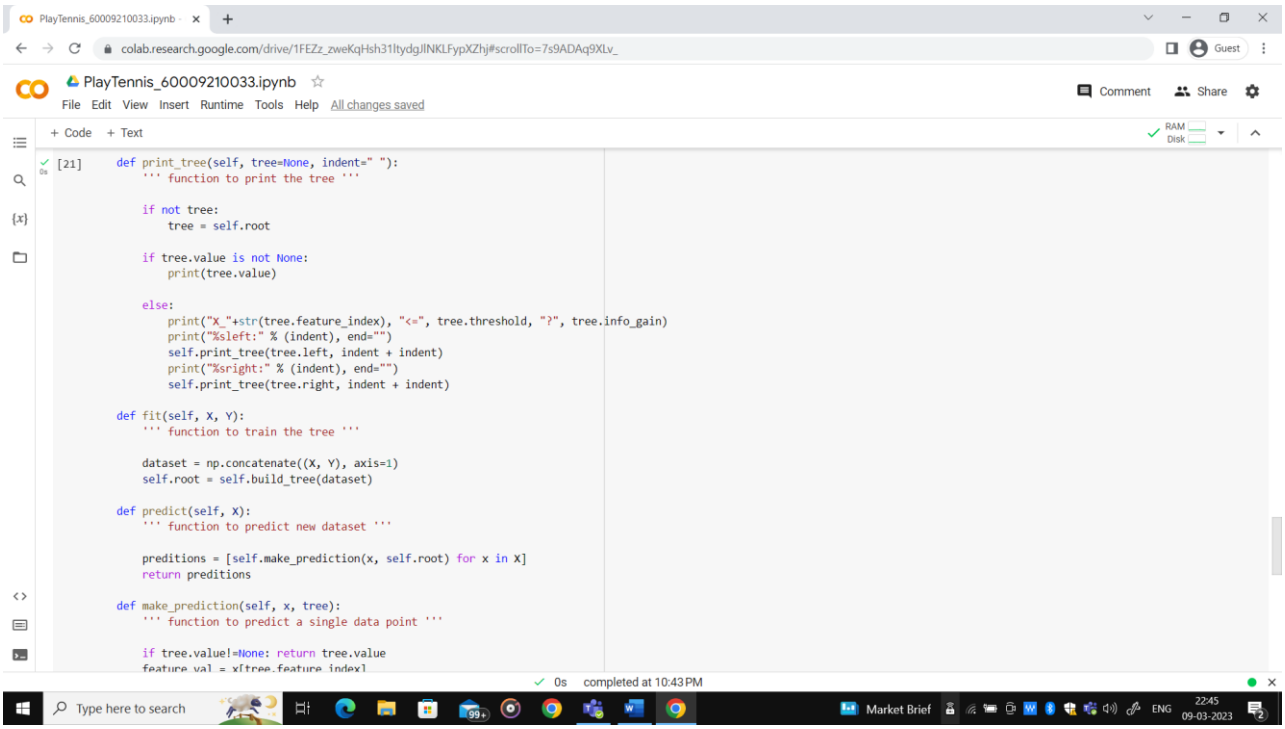
RAM Disk

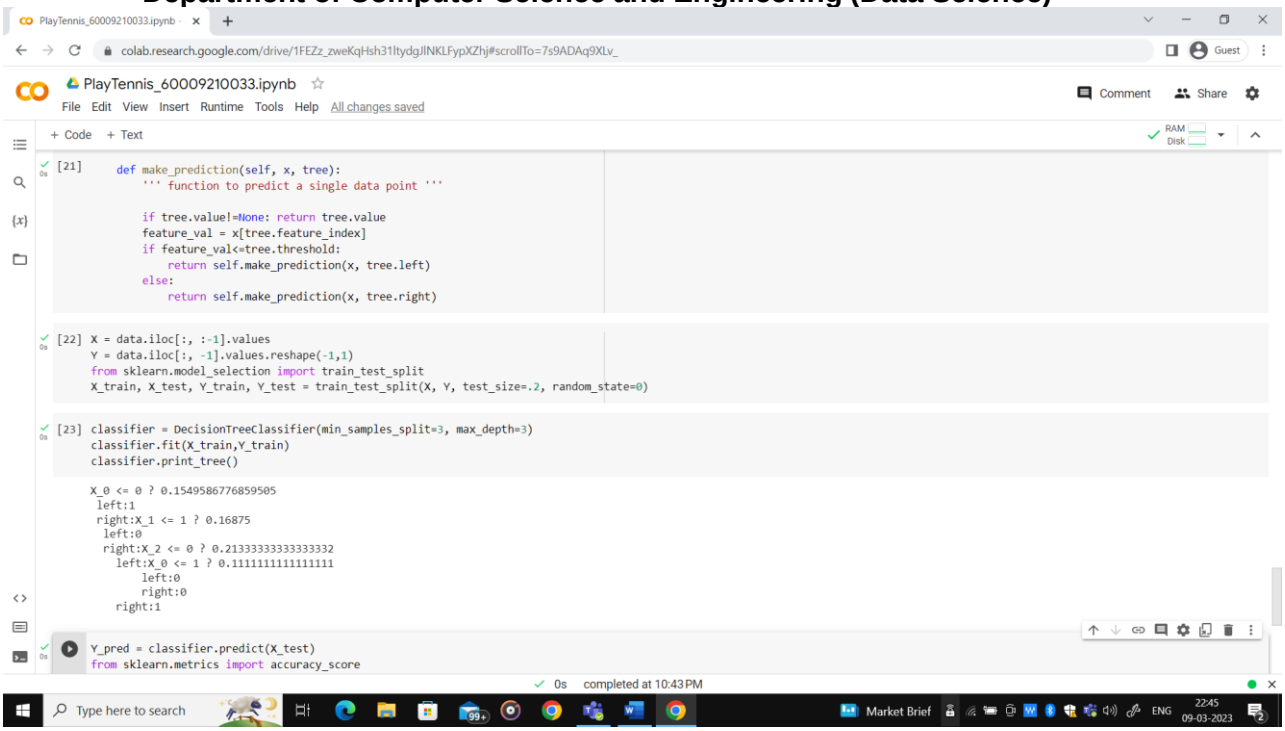
[21]

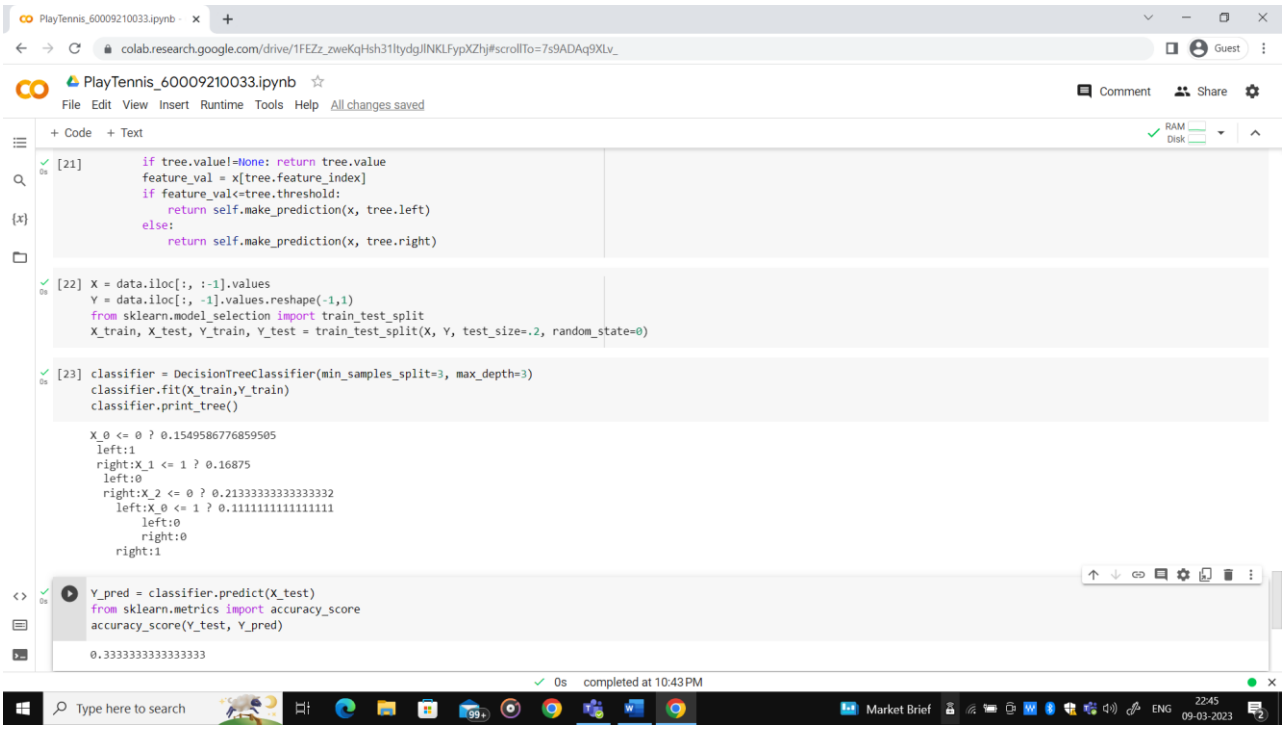
```
                    if curr_info_gain>max_info_gain:
                        best_split["feature_index"] = feature_index
                        best_split["threshold"] = threshold
                        best_split["dataset_left"] = dataset_left
                        best_split["dataset_right"] = dataset_right
                        best_split["info_gain"] = curr_info_gain
                        max_info_gain = curr_info_gain

        # return best split
        return best_split

    def split(self, dataset, feature_index, threshold):
        ''' function to split the data '''

        dataset_left = np.array([row for row in dataset if row[feature_index]<=threshold])
        dataset_right = np.array([row for row in dataset if row[feature_index]>threshold])
        return dataset_left, dataset_right

    def information_gain(self, parent, l_child, r_child, mode="entropy"):
        ''' function to compute information gain '''

        weight_l = len(l_child) / len(parent)
        weight_r = len(r_child) / len(parent)
        if mode=="gini":
            gain = self.gini_index(parent) - (weight_l*self.gini_index(l_child) + weight_r*self.gini_index(r_child))
        else:
            gain = self.entropy(parent) - (weight_l*self.entropy(l_child) + weight_r*self.entropy(r_child))
        return gain

    def entropy(self, y):
        ''' function to compute entropy '''

        class_labels = np.unique(y)
```

0s completed at 10:43 PM


PlayTennis_60009210033.ipynb
colab.research.google.com/drive/1FEZz_zweKqHsh31ltydgJlNKLFypXZhj#scrollTo=7s9ADAq9XLv_
PlayTennis_60009210033.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
[21]
def entropy(self, y):
''' function to compute entropy '''
class_labels = np.unique(y)
entropy = 0
for cls in class_labels:
p_cls = len(y[y == cls]) / len(y)
entropy += -p_cls * np.log2(p_cls)
return entropy
def gini_index(self, y):
''' function to compute gini index '''
class_labels = np.unique(y)
gini = 0
for cls in class_labels:
p_cls = len(y[y == cls]) / len(y)
gini += p_cls**2
return 1 - gini
def calculate_leaf_value(self, Y):
''' function to compute leaf node '''
Y = list(Y)
return max(Y, key=Y.count)
def print_tree(self, tree=None, indent=" "):
''' function to print the tree '''
if not tree:
tree = self.root
0s completed at 10:43 PM
Type here to search
Market Brief
ENG
22:45
09-03-2023

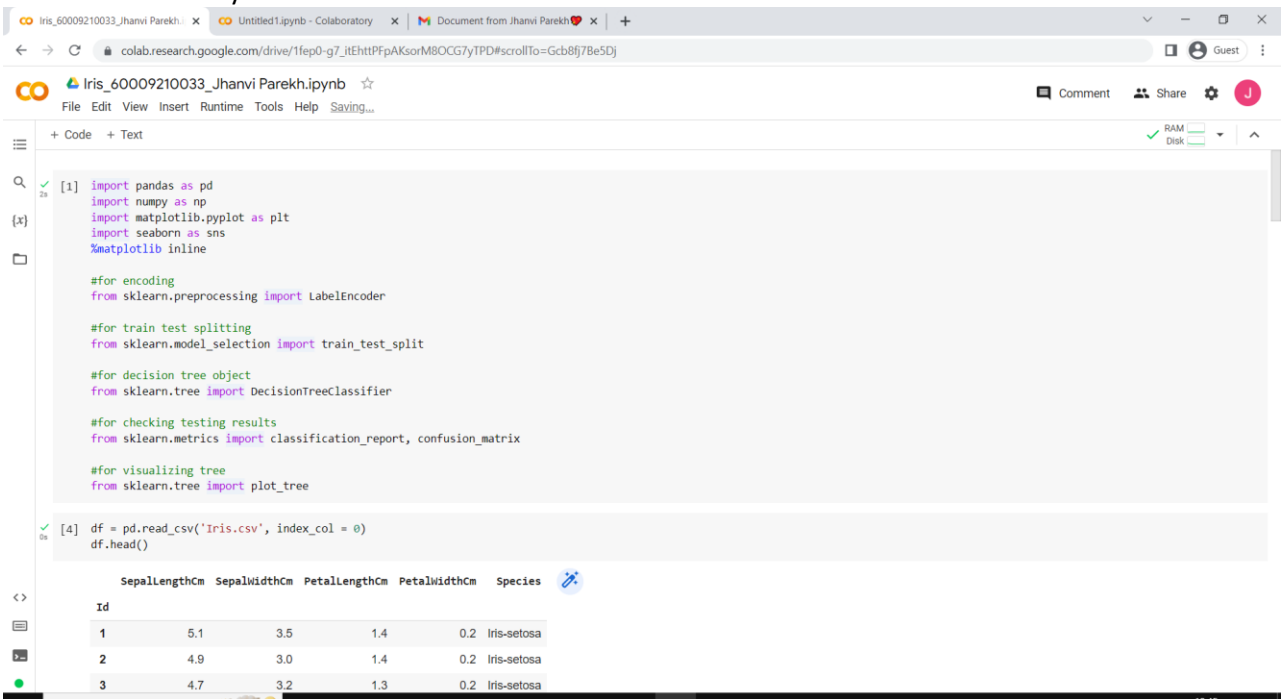

PlayTennis_60009210033.ipynb
colab.research.google.com/drive/1FEZz_zweKqHsh31ltydgJlNKLFypXZhj#scrollTo=7s9ADAq9XLv_
PlayTennis_60009210033.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
[21]
def print_tree(self, tree=None, indent=" "):
''' function to print the tree '''
if not tree:
tree = self.root
if tree.value is not None:
print(tree.value)
else:
print("X_"+str(tree.feature_index), "<=", tree.threshold, "?", tree.info_gain)
print("%sleft:" % (indent), end="")
self.print_tree(tree.left, indent + indent)
print("%sright:" % (indent), end="")
self.print_tree(tree.right, indent + indent)
def fit(self, X, Y):
''' function to train the tree '''
dataset = np.concatenate((X, Y), axis=1)
self.root = self.build_tree(dataset)
def predict(self, X):
''' function to predict new dataset '''
preditions = [self.make_prediction(x, self.root) for x in X]
return preditions
def make_prediction(self, x, tree):
''' function to predict a single data point '''
if tree.value!=None: return tree.value
0s completed at 10:43 PM
Type here to search
Market Brief
ENG
22:45
09-03-2023





https://colab.research.google.com/drive/1FEZz_zweKqHsh31ltydgJlNKLFypXZhj?usp=sharing

2. Use python libraries to build a decision tree classifier on Dataset 2. Analyze the results using confusion matrix and accuracy.



Iris_60009210033_Jhanvi Parekh.ipynb

File Edit View Insert Runtime Tools Help Saving...

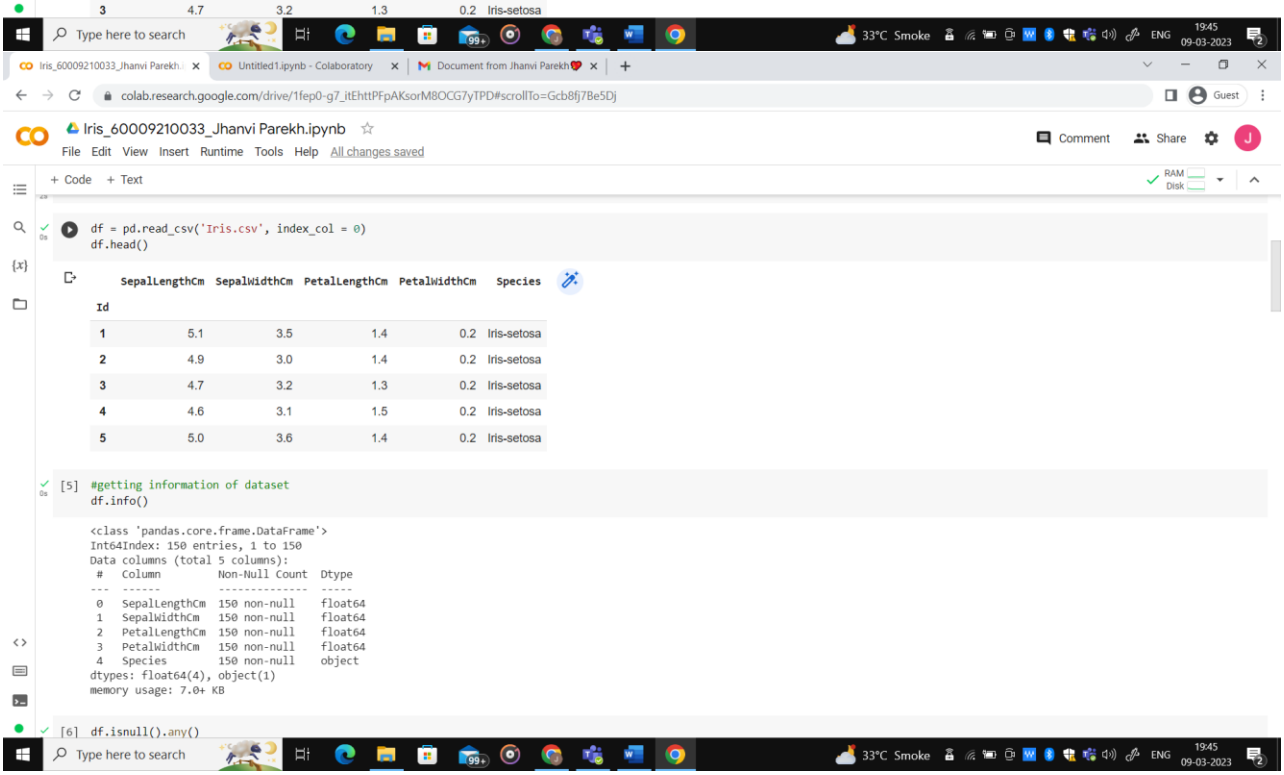
Comment Share

+ Code + Text

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

    #for encoding
    from sklearn.preprocessing import LabelEncoder

    #for train test splitting
    from sklearn.model_selection import train_test_split

    #for decision tree object
    from sklearn.tree import DecisionTreeClassifier

    #for checking testing results
    from sklearn.metrics import classification_report, confusion_matrix

    #for visualizing tree
    from sklearn.tree import plot_tree
```

```
[4] df = pd.read_csv('Iris.csv', index_col = 0)
    df.head()
```

Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
1	5.1	3.5	1.4	0.2	Iris-setosa
2	4.9	3.0	1.4	0.2	Iris-setosa
3	4.7	3.2	1.3	0.2	Iris-setosa



Iris_60009210033_Jhanvi Parekh.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

+ Code + Text

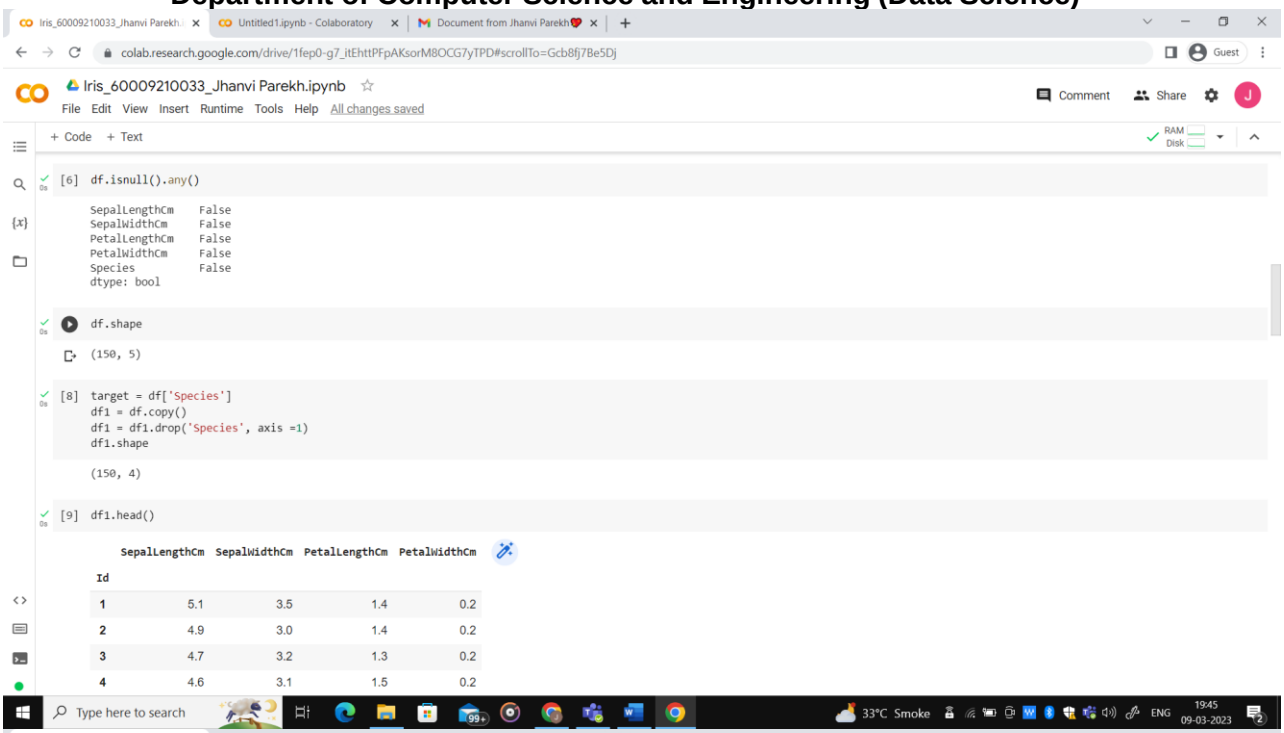
```
df = pd.read_csv('Iris.csv', index_col = 0)
df.head()
```

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

```
[5] #getting information of dataset
    df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 150 entries, 1 to 150
Data columns (total 5 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   SepalLengthCm  150 non-null    float64
 1   SepalWidthCm   150 non-null    float64
 2   PetalLengthCm  150 non-null    float64
 3   PetalWidthCm   150 non-null    float64
 4   Species        150 non-null    object
dtypes: float64(4), object(1)
memory usage: 7.0+ KB
```

```
[6] df.isnull().any()
```



Iris_60009210033_Jhanvi Parekh.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
[6] df.isnull().any()

SepalLengthCm    False
SepalWidthCm     False
PetalLengthCm    False
PetalWidthCm     False
Species          False
dtype: bool
```

```
df.shape
```

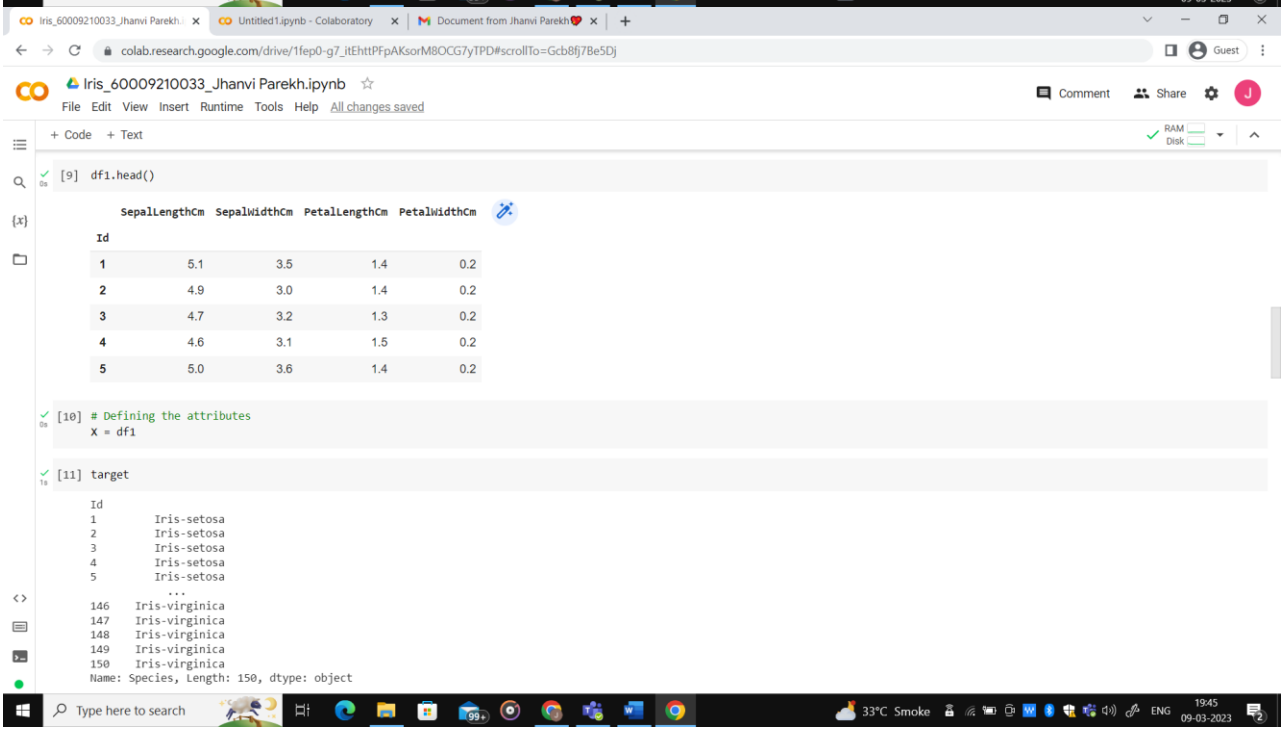
```
(150, 5)
```

```
[8] target = df['Species']
df1 = df.copy()
df1 = df1.drop('Species', axis =1)
df1.shape
```

```
(150, 4)
```

```
[9] df1.head()
```

Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
1	5.1	3.5	1.4	0.2
2	4.9	3.0	1.4	0.2
3	4.7	3.2	1.3	0.2
4	4.6	3.1	1.5	0.2



Iris_60009210033_Jhanvi Parekh.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

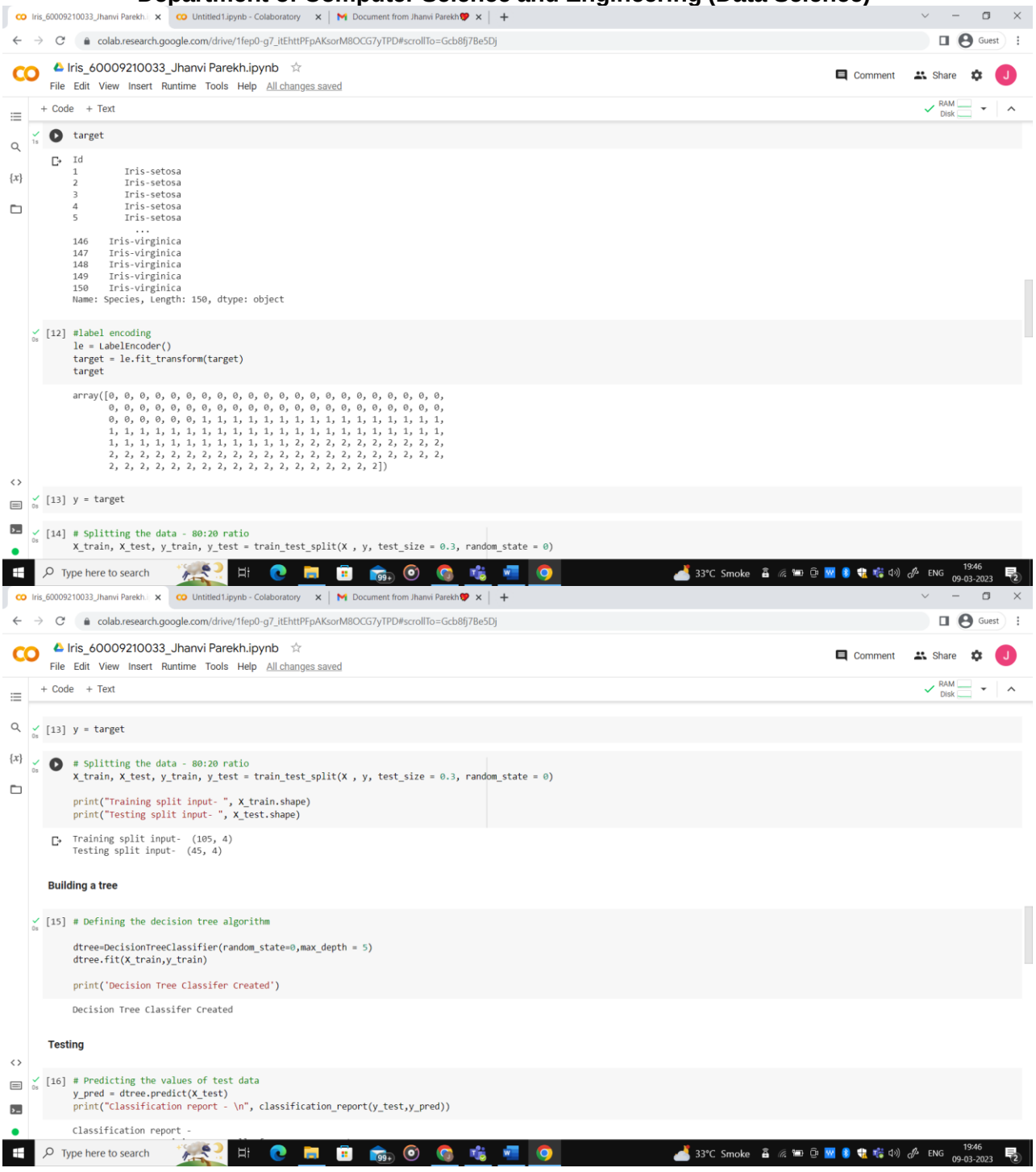
```
[9] df1.head()
```

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

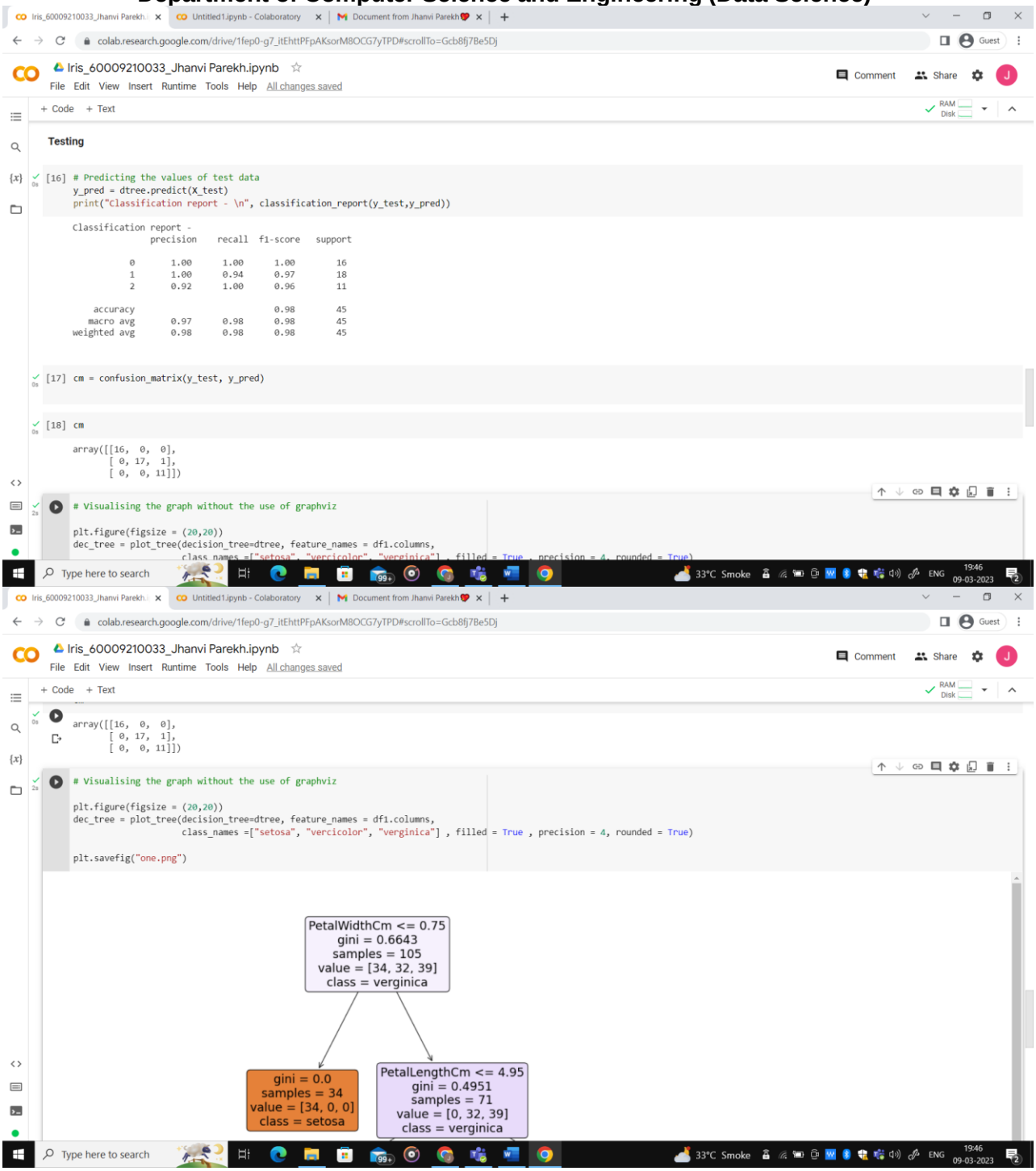
```
[10] # Defining the attributes
X = df1
```

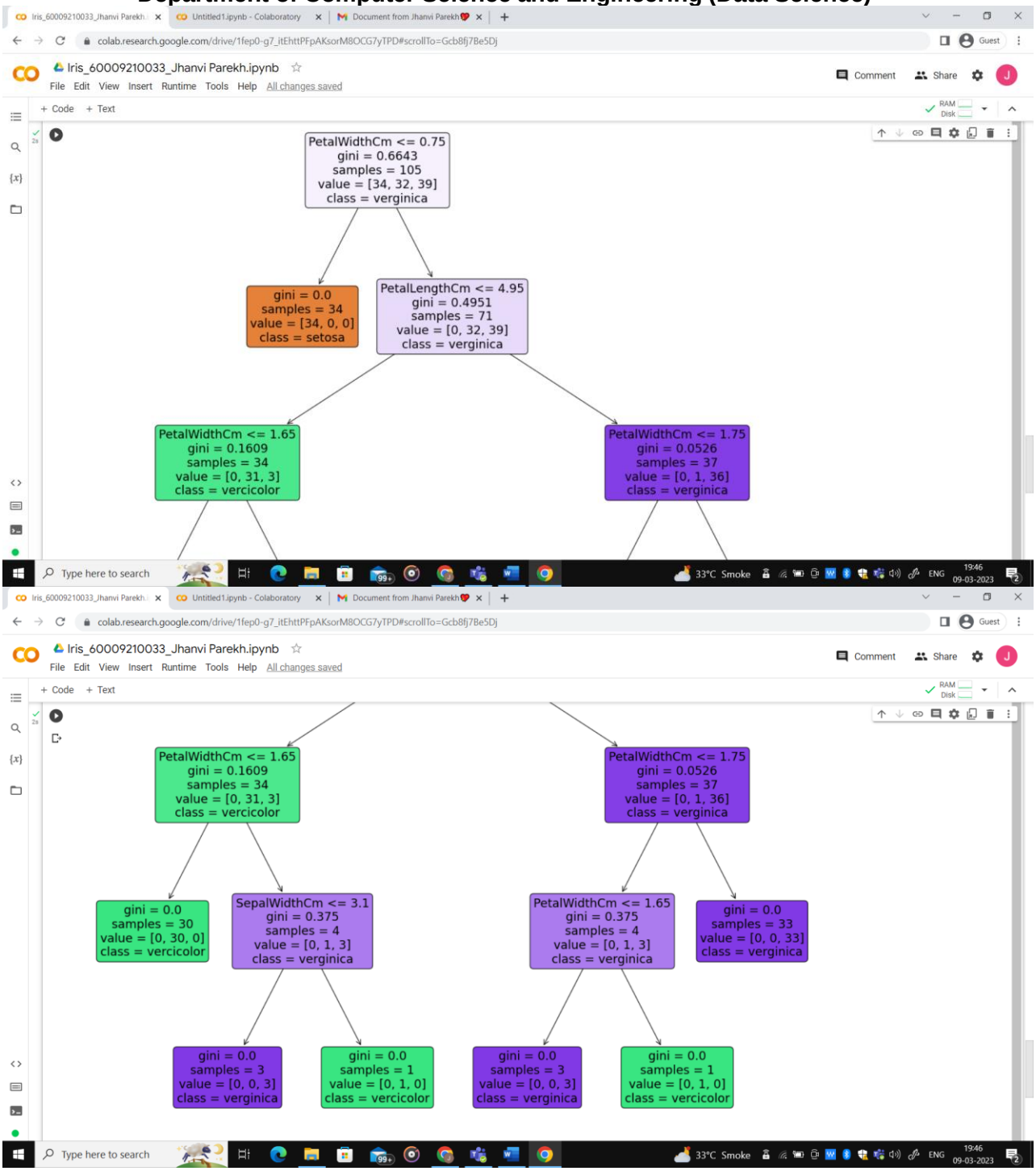
```
[11] target
```

```
Id
1         Iris-setosa
2         Iris-setosa
3         Iris-setosa
4         Iris-setosa
5         Iris-setosa
             ...
146    Iris-virginica
147    Iris-virginica
148    Iris-virginica
149    Iris-virginica
150    Iris-virginica
Name: Species, Length: 150, dtype: object
```


Iris_60009210033_Jhanvi Parekh.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
target
Id
1 Iris-setosa
2 Iris-setosa
3 Iris-setosa
4 Iris-setosa
5 Iris-setosa
...
146 Iris-virginica
147 Iris-virginica
148 Iris-virginica
149 Iris-virginica
150 Iris-virginica
Name: Species, Length: 150, dtype: object
[12] #label encoding
le = LabelEncoder()
target = le.fit_transform(target)
target
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
[13] y = target
[14] # Splitting the data - 80:20 ratio
X_train, X_test, y_train, y_test = train_test_split(X , y, test_size = 0.3, random_state = 0)
Type here to search
33°C Smoke
ENG
19:46
09-03-2023

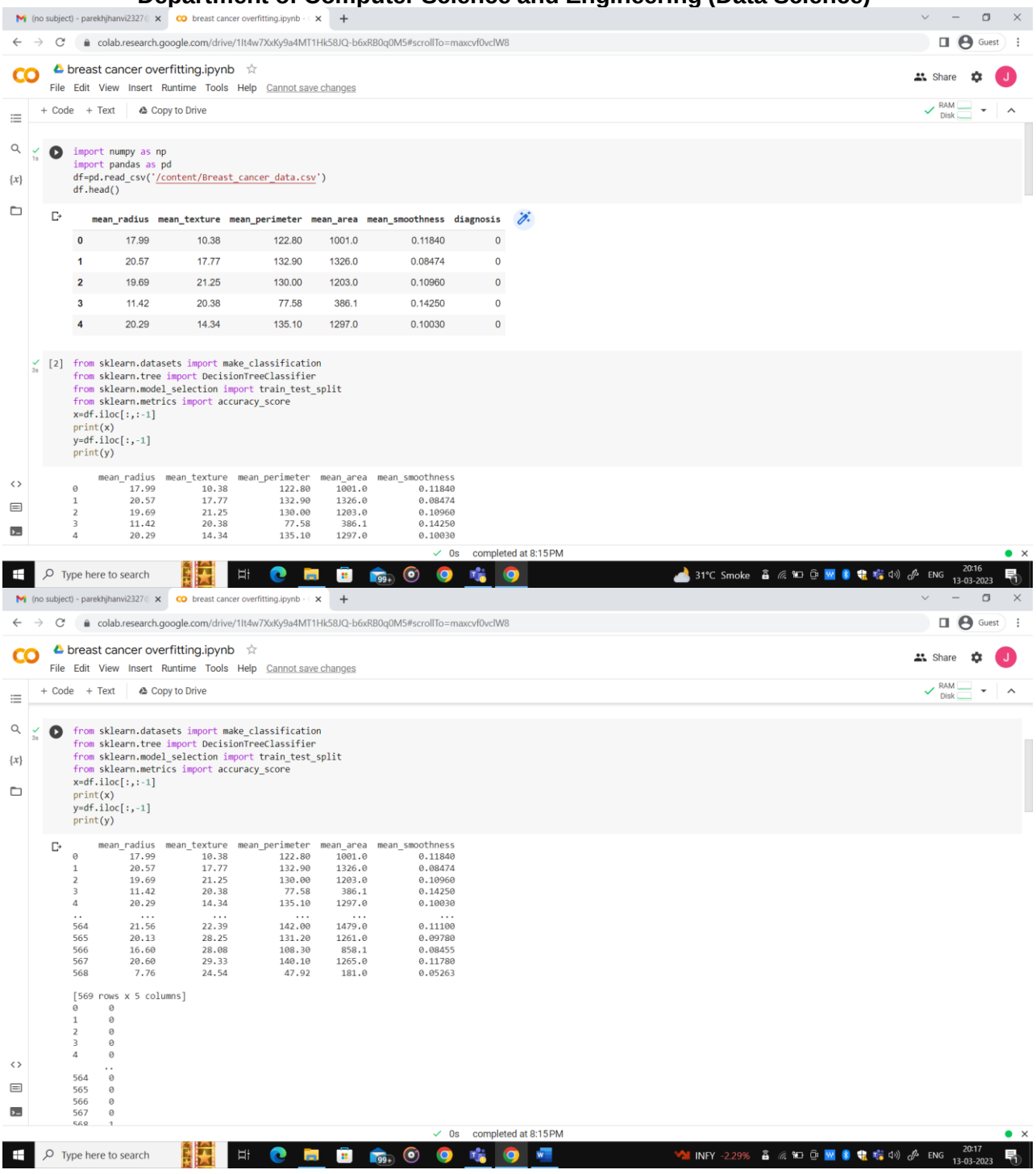
Iris_60009210033_Jhanvi Parekh.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
[13] y = target
Splitting the data - 80:20 ratio
X_train, X_test, y_train, y_test = train_test_split(X , y, test_size = 0.3, random_state = 0)
print("Training split input- ", X_train.shape)
print("Testing split input- ", X_test.shape)
Training split input- (105, 4)
Testing split input- (45, 4)
Building a tree
[15] # Defining the decision tree algorithm
dtree=DecisionTreeClassifier(random_state=0,max_depth = 5)
dtree.fit(X_train,y_train)
print('Decision Tree Classifer Created')
Decision Tree Classifer Created
Testing
[16] # Predicting the values of test data
y_pred = dtree.predict(X_test)
print("Classification report - \n", classification_report(y_test,y_pred))
Classification report -
Type here to search
33°C Smoke
ENG
19:46
09-03-2023


Iris_60009210033_Jhanvi Parekh.ipynb
Testing
[16] # Predicting the values of test data
y_pred = dtree.predict(X_test)
print("Classification report - \n", classification_report(y_test,y_pred))
Classification report -
precision recall f1-score support
0 1.00 1.00 1.00 16
1 1.00 0.94 0.97 18
2 0.92 1.00 0.96 11
accuracy 0.98 45
macro avg 0.97 0.98 0.98 45
weighted avg 0.98 0.98 0.98 45
[17] cm = confusion_matrix(y_test, y_pred)
[18] cm
array([[16, 0, 0],
[0, 17, 1],
[0, 0, 11]])
# Visualising the graph without the use of graphviz
plt.figure(figsize = (20,20))
dec_tree = plot_tree(decision_tree=dtree, feature_names = df1.columns,
class_names =["setosa", "vercicolor", "verginica"] , filled = True , precision = 4, rounded = True)
plt.savefig("one.png")
PetalWidthCm <= 0.75
gini = 0.6643
samples = 105
value = [34, 32, 39]
class = verginica
gini = 0.0
samples = 34
value = [34, 0, 0]
class = setosa
PetalLengthCm <= 4.95
gini = 0.4951
samples = 71
value = [0, 32, 39]
class = verginica

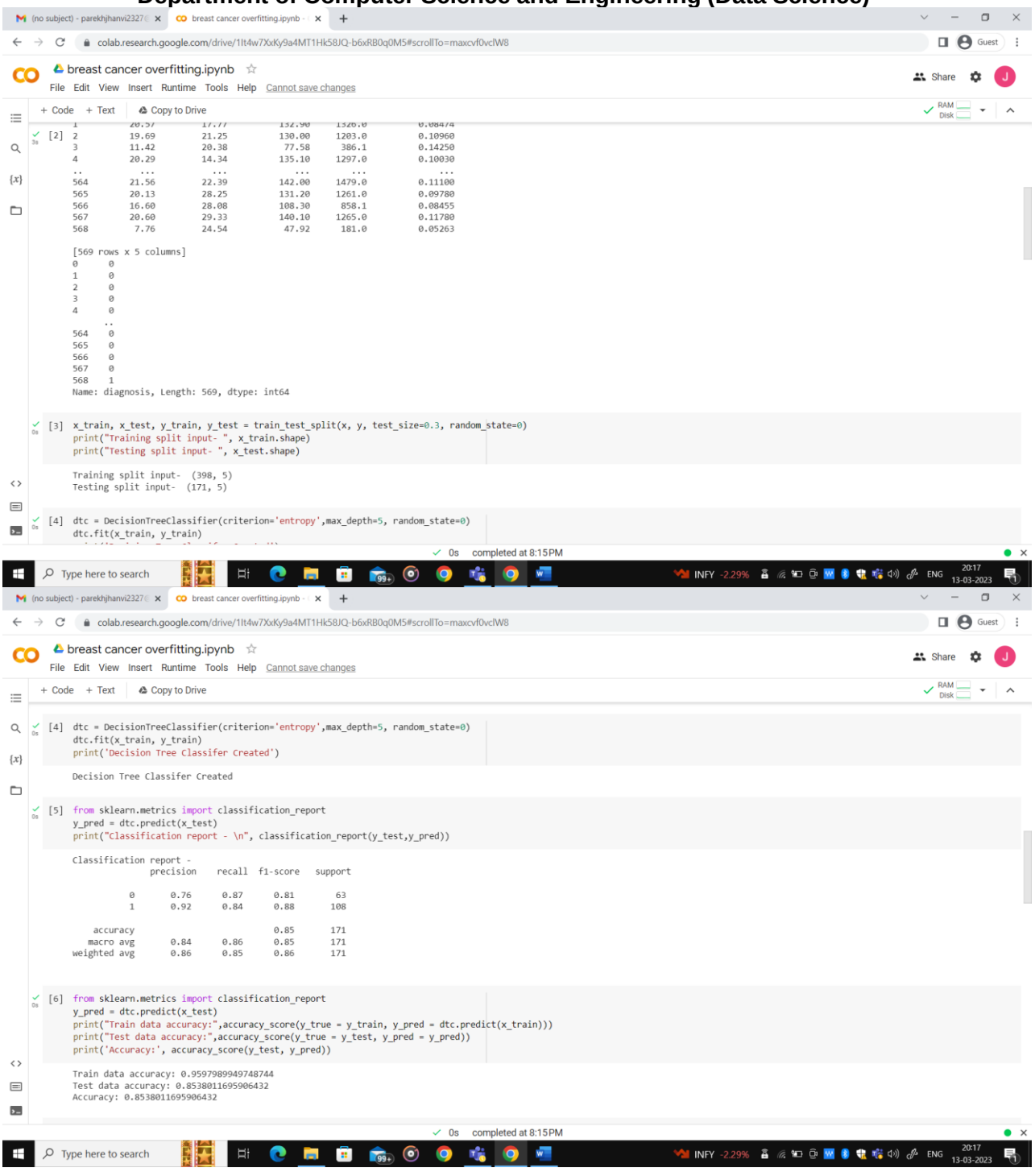


https://colab.research.google.com/drive/1fep0-g7_itEhttPFpAKsorM8OCG7yTPD?usp=sharing

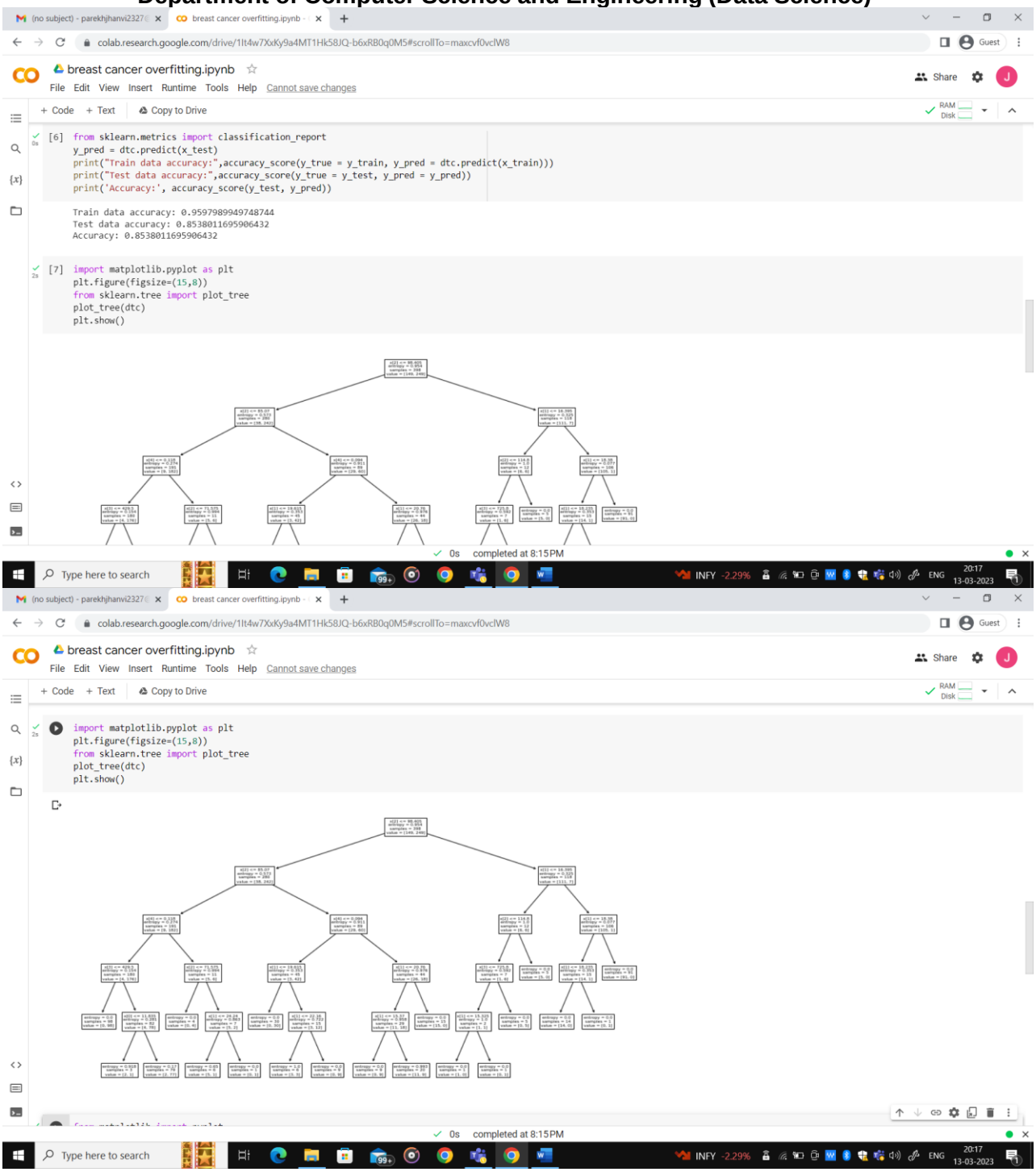
3. Write a code to show overfitting in the decision tree classifier built using Dataset 3. Use sklearn and matplotlib.

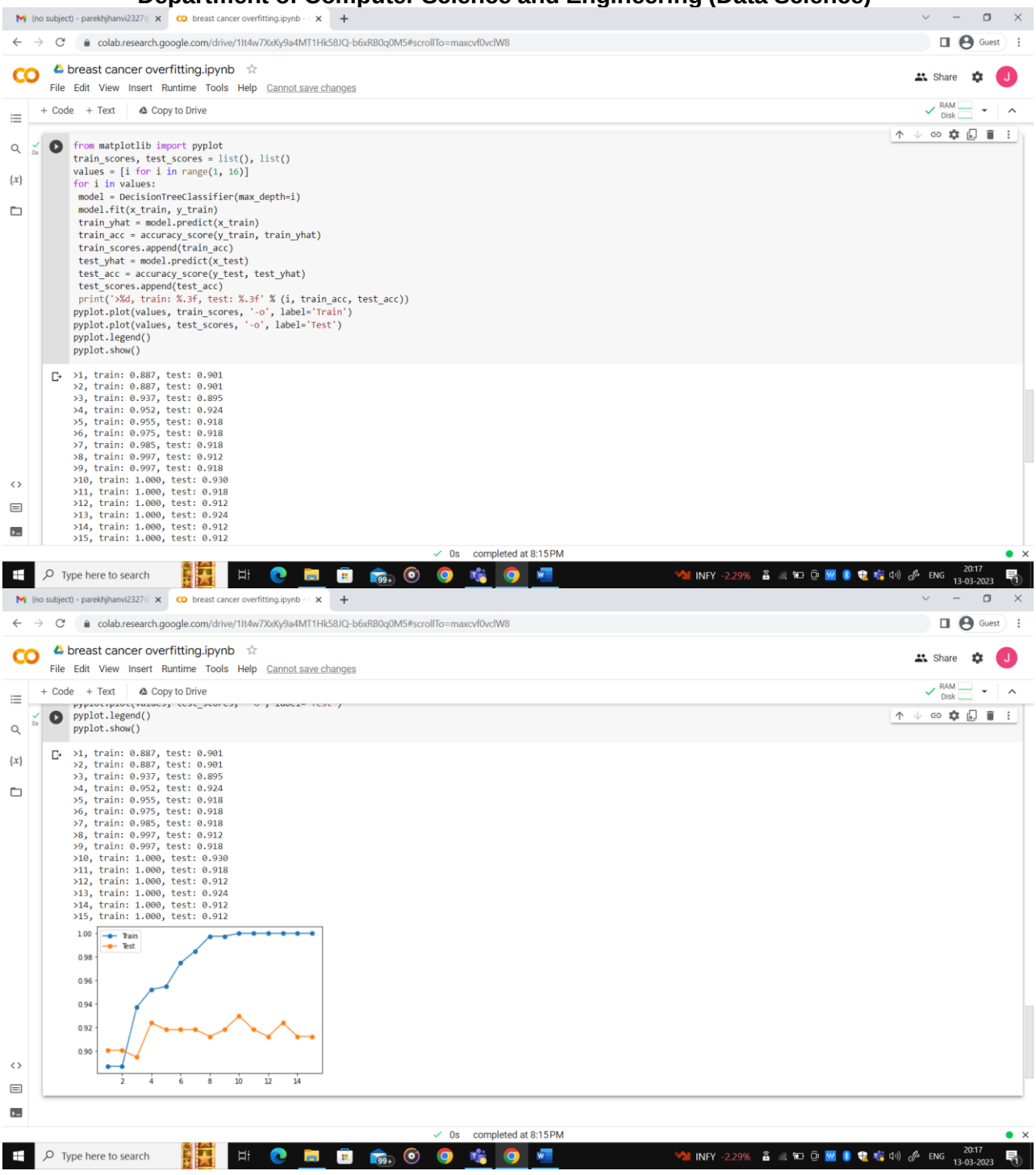

breast cancer overfitting.ipynb
File Edit View Insert Runtime Tools Help Cannot save changes
+ Code + Text Copy to Drive
import numpy as np
import pandas as pd
df=pd.read_csv('/content/Breast_cancer_data.csv')
df.head()
mean_radius mean_texture mean_perimeter mean_area mean_smoothness diagnosis
0 17.99 10.38 122.80 1001.0 0.11840 0
1 20.57 17.77 132.90 1326.0 0.08474 0
2 19.69 21.25 130.00 1203.0 0.10960 0
3 11.42 20.38 77.58 386.1 0.14250 0
4 20.29 14.34 135.10 1297.0 0.10030 0
[2] from sklearn.datasets import make_classification
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
x=df.iloc[:,:-1]
print(x)
y=df.iloc[:,-1]
print(y)
mean_radius mean_texture mean_perimeter mean_area mean_smoothness
0 17.99 10.38 122.80 1001.0 0.11840
1 20.57 17.77 132.90 1326.0 0.08474
2 19.69 21.25 130.00 1203.0 0.10960
3 11.42 20.38 77.58 386.1 0.14250
4 20.29 14.34 135.10 1297.0 0.10030
0s completed at 8:15PM
Type here to search
31°C Smoke
ENG 20:16 13-03-2023

breast cancer overfitting.ipynb
File Edit View Insert Runtime Tools Help Cannot save changes
+ Code + Text Copy to Drive
from sklearn.datasets import make_classification
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
x=df.iloc[:,:-1]
print(x)
y=df.iloc[:,-1]
print(y)
mean_radius mean_texture mean_perimeter mean_area mean_smoothness
0 17.99 10.38 122.80 1001.0 0.11840
1 20.57 17.77 132.90 1326.0 0.08474
2 19.69 21.25 130.00 1203.0 0.10960
3 11.42 20.38 77.58 386.1 0.14250
4 20.29 14.34 135.10 1297.0 0.10030
.. ..
564 21.56 22.39 142.00 1479.0 0.11100
565 20.13 28.25 131.20 1261.0 0.09780
566 16.60 28.08 108.30 858.1 0.08455
567 20.60 29.33 140.10 1265.0 0.11780
568 7.76 24.54 47.92 181.0 0.05263
[569 rows x 5 columns]
0 0
1 0
2 0
3 0
4 0
..
564 0
565 0
566 0
567 0
0s completed at 8:15PM
Type here to search
INFY -2.29%
ENG 20:17 13-03-2023


breast cancer overfitting.ipynb
File Edit View Insert Runtime Tools Help Cannot save changes
+ Code + Text Copy to Drive
[2] 2 19.69 21.25 130.00 1203.0 0.10960
3 11.42 20.38 77.58 386.1 0.14250
4 20.29 14.34 135.10 1297.0 0.10030
564 21.56 22.39 142.00 1479.0 0.11100
565 20.13 28.25 131.20 1261.0 0.09780
566 16.60 28.08 108.30 858.1 0.08455
567 20.60 29.33 140.10 1265.0 0.11780
568 7.76 24.54 47.92 181.0 0.05263
[569 rows x 5 columns]
0 0
1 0
2 0
3 0
4 0
564 0
565 0
566 0
567 0
568 1
Name: diagnosis, Length: 569, dtype: int64
[3] x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=0)
print("Training split input- ", x_train.shape)
print("Testing split input- ", x_test.shape)
Training split input- (398, 5)
Testing split input- (171, 5)
[4] dtc = DecisionTreeClassifier(criterion='entropy',max_depth=5, random_state=0)
dtc.fit(x_train, y_train)
0s completed at 8:15PM
Type here to search
INFY -2.29%
ENG
20:17
13-03-2023

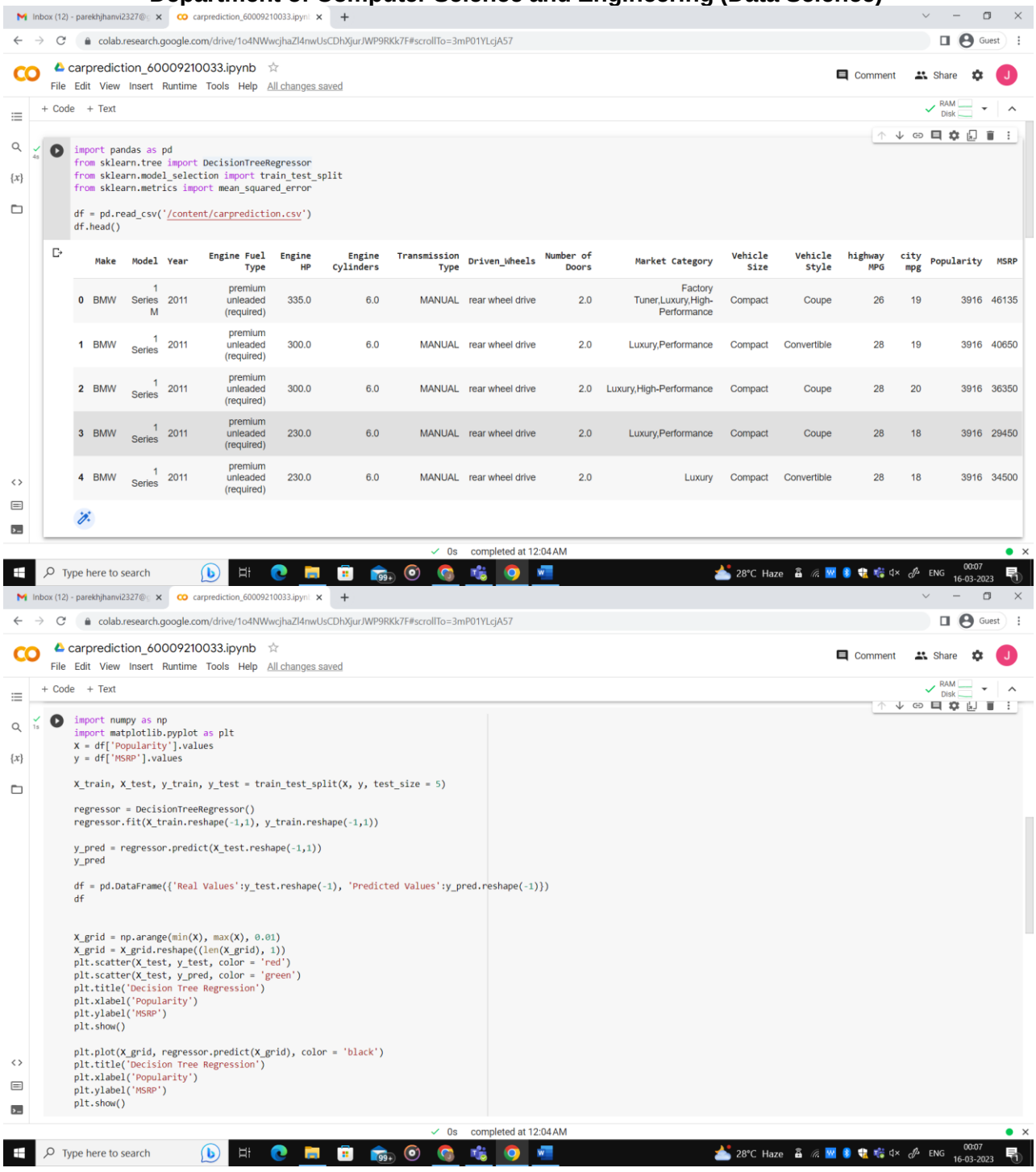
breast cancer overfitting.ipynb
File Edit View Insert Runtime Tools Help Cannot save changes
+ Code + Text Copy to Drive
[4] dtc = DecisionTreeClassifier(criterion='entropy',max_depth=5, random_state=0)
dtc.fit(x_train, y_train)
print('Decision Tree Classifer Created')
Decision Tree Classifer Created
[5] from sklearn.metrics import classification_report
y_pred = dtc.predict(x_test)
print("Classification report - \n", classification_report(y_test,y_pred))
Classification report -
precision recall f1-score support
0 0.76 0.87 0.81 63
1 0.92 0.84 0.88 108
accuracy 0.85 171
macro avg 0.84 0.86 0.85 171
weighted avg 0.86 0.85 0.86 171
[6] from sklearn.metrics import classification_report
y_pred = dtc.predict(x_test)
print("Train data accuracy:",accuracy_score(y_true = y_train, y_pred = dtc.predict(x_train)))
print("Test data accuracy:",accuracy_score(y_true = y_test, y_pred = y_pred))
print('Accuracy:', accuracy_score(y_test, y_pred))
Train data accuracy: 0.9597989949748744
Test data accuracy: 0.8538011695906432
Accuracy: 0.8538011695906432
0s completed at 8:15PM
Type here to search
INFY -2.29%
ENG
20:17
13-03-2023


breast cancer overfitting.ipynb
File Edit View Insert Runtime Tools Help Cannot save changes
+ Code + Text Copy to Drive
[6] from sklearn.metrics import classification_report
y_pred = dtc.predict(x_test)
print("Train data accuracy:",accuracy_score(y_true = y_train, y_pred = dtc.predict(x_train)))
print("Test data accuracy:",accuracy_score(y_true = y_test, y_pred = y_pred))
print('Accuracy:', accuracy_score(y_test, y_pred))
Train data accuracy: 0.9597989949748744
Test data accuracy: 0.8538011695906432
Accuracy: 0.8538011695906432
[7] import matplotlib.pyplot as plt
plt.figure(figsize=(15,8))
from sklearn.tree import plot_tree
plot_tree(dtc)
plt.show()
0s completed at 8:15PM
Type here to search
INFY -2.29%
ENG
20:17
13-03-2023



https://colab.research.google.com/drive/1It4w7XxKy9a4MT1Hk58JQ-b6xRB0q0M5?usp=sharing

4. Implement Decision tree regressor on Dataset 4.



carprediction_60009210033.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
import pandas as pd
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

df = pd.read_csv('/content/carprediction.csv')
df.head()
```

	Make	Model	Year	Engine Fuel Type	Engine HP	Engine Cylinders	Transmission Type	Driven_Wheels	Number of Doors	Market Category	Vehicle Size	Vehicle Style	highway MPG	city mpg	Popularity	MSRP
0	BMW	1 Series M	2011	premium unleaded (required)	335.0	6.0	MANUAL	rear wheel drive	2.0	Factory Tuner,Luxury,High-Performance	Compact	Coupe	26	19	3916	46135
1	BMW	1 Series	2011	premium unleaded (required)	300.0	6.0	MANUAL	rear wheel drive	2.0	Luxury,Performance	Compact	Convertible	28	19	3916	40650
2	BMW	1 Series	2011	premium unleaded (required)	300.0	6.0	MANUAL	rear wheel drive	2.0	Luxury,High-Performance	Compact	Coupe	28	20	3916	36350
3	BMW	1 Series	2011	premium unleaded (required)	230.0	6.0	MANUAL	rear wheel drive	2.0	Luxury,Performance	Compact	Coupe	28	18	3916	29450
4	BMW	1 Series	2011	premium unleaded (required)	230.0	6.0	MANUAL	rear wheel drive	2.0	Luxury	Compact	Convertible	28	18	3916	34500

0s completed at 12:04 AM

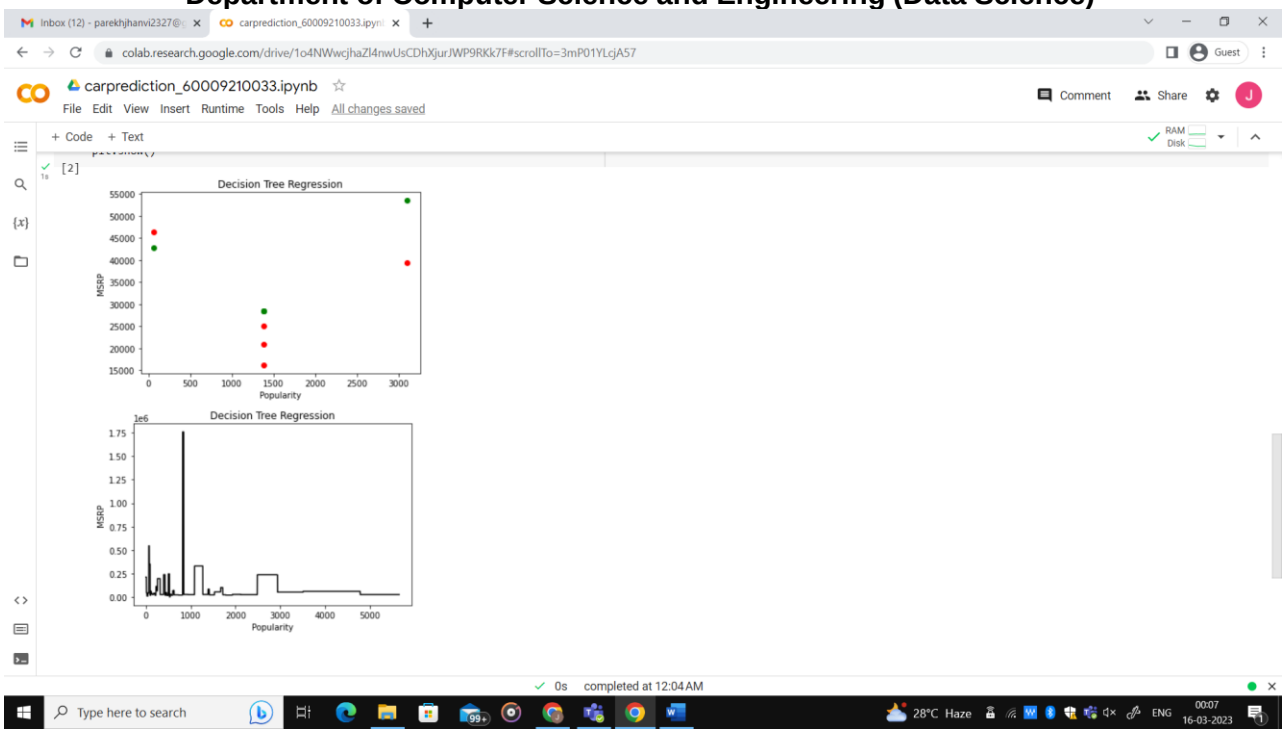
carprediction_60009210033.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
import numpy as np
import matplotlib.pyplot as plt
X = df['Popularity'].values
y = df['MSRP'].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 5)

regressor = DecisionTreeRegressor()
regressor.fit(X_train.reshape(-1,1), y_train.reshape(-1,1))

y_pred = regressor.predict(X_test.reshape(-1,1))
y_pred

df = pd.DataFrame({'Real Values':y_test.reshape(-1), 'Predicted Values':y_pred.reshape(-1)})
df


X_grid = np.arange(min(X), max(X), 0.01)
X_grid = X_grid.reshape((len(X_grid), 1))
plt.scatter(X_test, y_test, color = 'red')
plt.scatter(X_test, y_pred, color = 'green')
plt.title('Decision Tree Regression')
plt.xlabel('Popularity')
plt.ylabel('MSRP')
plt.show()

plt.plot(X_grid, regressor.predict(X_grid), color = 'black')
plt.title('Decision Tree Regression')
plt.xlabel('Popularity')
plt.ylabel('MSRP')
plt.show()
```

0s completed at 12:04 AM



https://colab.research.google.com/drive/1o4NWwcjhaZI4nwUsCDhXjurJWP9RKk7F?usp=sharing