—: LAB: 03 :—

* Implementing the ID3 Algorithm :—

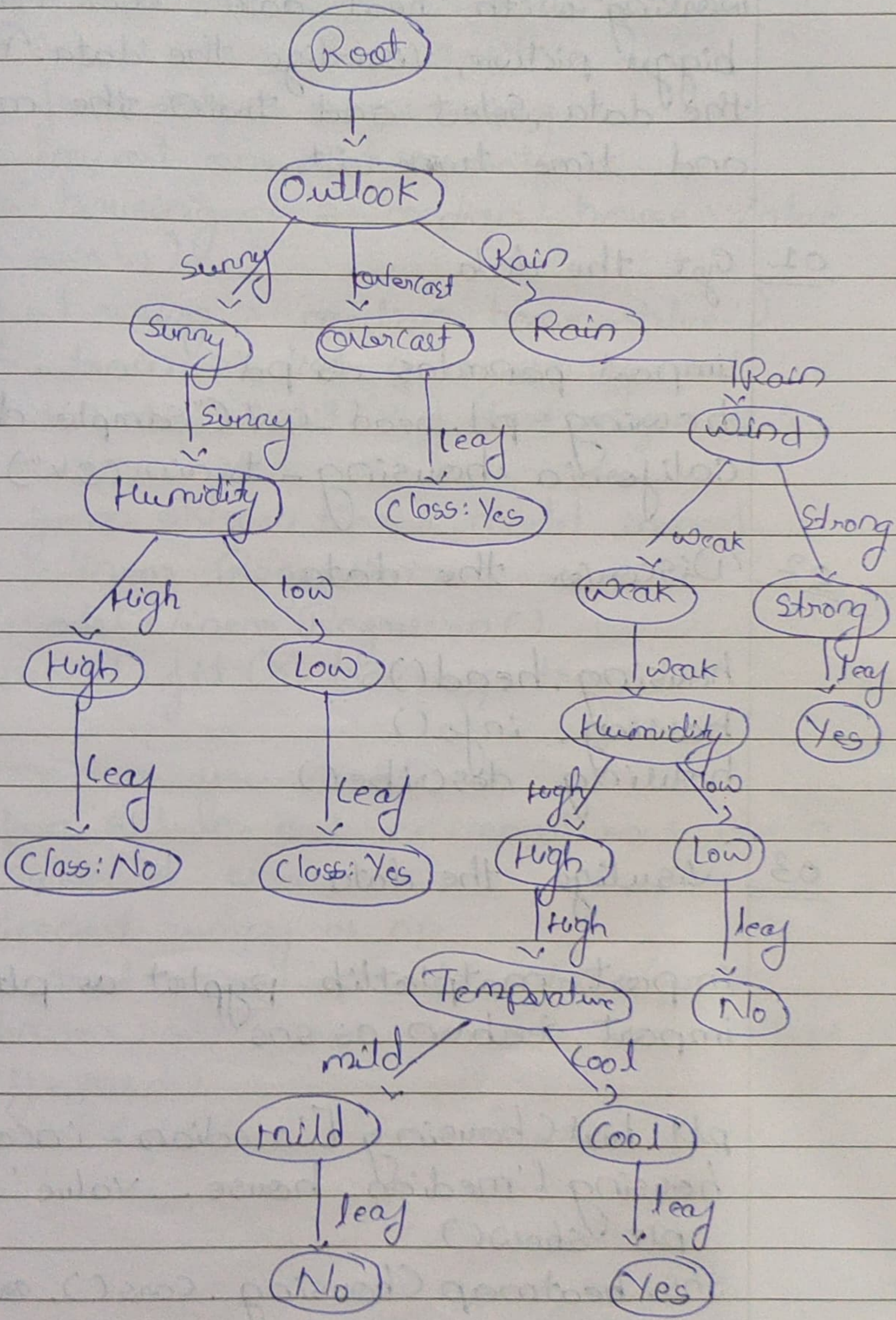
#Code

```
import pandas as pd
import numpy as np
from graphviz import Digraph
#Now calculating Entropy

  def entropy (data):
    Class-probabilities = data .iloc [:, -1].
 Value_counts(normaliz=True)
    return -np.sum(class-probabilities *np.log2
 (class probabilities))

 # Calculate IG,
   def information gain (data, feature):
   total-entropy = entropy (data)
   feature-values = data [feature]. unique()
   Weighted-entropy =0

   for value in feature-values:
       Subset = data [data [feature]== value]
       Weighted- entropy += (len (subset) /
   len(data)) *entropy (subset)
 return total-entropy - Weighted_entropy

   def best-feature (data):
     features = data. columns [:-1]
     gains ={feature : information_gain(data,
          feature ) for feature in features}
   return max(gains, key =gains.get)
```

```
def id3 (data, features = None):
    if len (data. iloc[:, -1] . unique()) ==1:
    return  data. iloc [:, -1]. iloc [0]

    if len (features) = =0:
        return data . iloc [:, -1]. mode () [0]

  best = best_ feature (data)
  tree = {best : {}}

  return tree

def classify (tree, example):
    if not  isinstance (tree, dict):
        return tree
    feature = list (tree. Keys ()) [0]
    value = example [feature]
    return classify (tree [feature] [value ], example)

def create_ tree_ diagram (tree, dot=None,
parent_ name= "Root", parent_ value = " ");
    if dot is None:
        dot= Digraph (format= "png", engine="do

    if isinstance (tree, dict):
        for feature, branches in tree . items():
            feature _name = f"{parent_ name}
                _{feature}"
    dot. node (feature_ name, feature)
    dot. edge( parent_name, feature_ name,
    label = parent_ value)
```

```
for value, subtree in branches.items():
    value_name = f"{feature_name}_{feature}"
    dot.node(value_name, f"{feature}:
        {value}")
    dot.edge(feature_name, value_name,
     label=str(value))

     create_tree_diagram(subtree, dot,
    value_name, str(value))

  else:
    dot.node(parent_name+"_class", f"Class:
    {tree}")
    dot.edge(parent_name, parent_name+"_class",
  label="Leaf")

    return dot

      data = pd.DataFrame({
        "Dataset discussed in classroom"
        })

  tree = id3(data, features = list(data.columns
        [:-1]))
  print("Decision Tree:", tree)

  example = {"outlook": }
  prediction = classify(tree, example)
  print("Prediction for the example:",
     prediction)

   dot = create_tree_diagram(tree)
   dot.render("decision_tree", view = True)
```

⇒ Output :-


Root
Outlook
Sunny
Overcast
Rain
Sunny
OverCast
Rain
Rain
Sunny
leaf
Humidity
Class: Yes
Wind
High
low
Weak
Strong
High
Low
Weak
Strong
Leaf
Leaf
weak
leaf
Class: No
Class: Yes
Humidity
Yes
High
low
High
Low
High
leaf
Temperature
No
mild
Cool
mild
Cool
leaf
leaf
No
Yes

→ End to end machine learning project working with real data. Look at the biggur picture, visualize the data. Prepare the data, Select and train the model and time tume it.

01 Got the data

```
import panadas as pd
housing=pd.read_csv("sample data/
california:housing_train.csv")
```

02 Discover the data

```
housing.head()
housing. info()
housing, describe()
```

03 Visulize the data

```
import matplotlib pyplot as plt
import seaborn as sns

plt.hist(housing['median_income']
housing['median_house_value'])
  plt show()
sns.heatmap(housing.cors(). annot=True)
  plt.show()
```

04 Prepare the data

```
  housing.Snull().sum()
```

Q5 Select and train the model

```
from sklearn model. selection
import train, text, split
from sklearn preprocessing
import one plot encounter
x = housing. drop ('median: house: value,
axis = 1)
y = housing x median. house value')
x_train, x_list, y_train, y_list
train_test: split (x, y, test, size = 0, 2,
random_state = +2)
from sklearn linear. model import
linear Regression
model = Linear Regression ()
model. fit (x_train, y_train)
```

Q6. Fine tune your model

```
from sklearn. matrices import rout_mean
squarred error
import numpy as np
y_pred = model_predit (x_test)
rmse = root. mean_squared_error (y_test,
y_pred)
print (s' RMSE = {rmse}").
```