LAB-3

Implementation of ID3 Algorithm

```
import numpy as np
import pandas as pd
from graphviz import Digraph
def entropy(data):
 class_probabilities = data.iloc[:, -1].
 value_counts(normalize = True)
 return -np.sum(class_probabilities * np.
 log2(class_probabilities))

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]
```

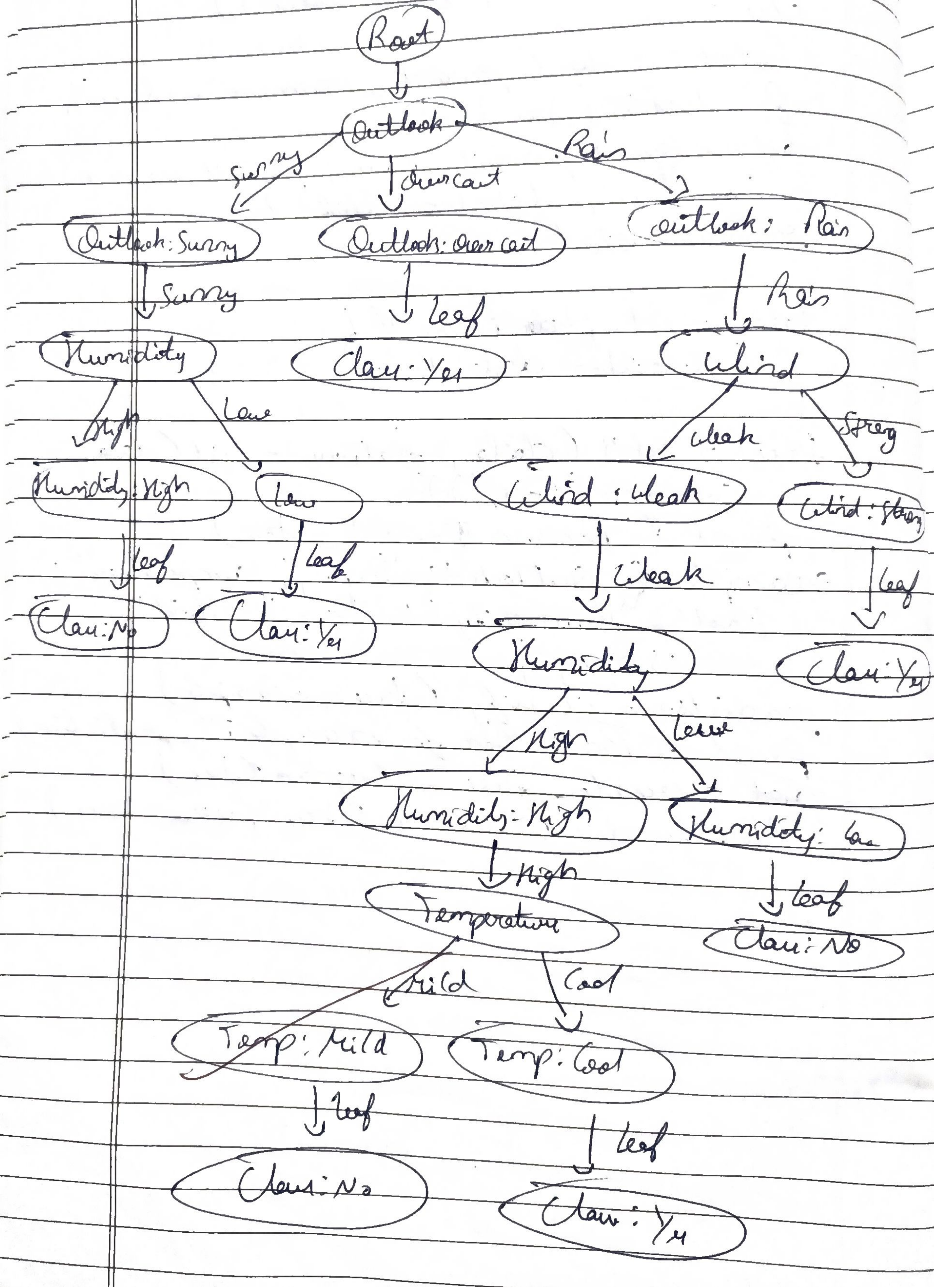
```
    best = best_feature (data)
      tree = { best: {}}
    new_features = features.copy()
    new_features.remove ( best)
    for value in data [best].unique():
     subset = data [data [best] == value]
     tree [best] [value] = id3 (subset,
           new_features)
    return tree
    def classify ( 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", engi=
  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}_{value}"
     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 ([
    ['weather dataset']:

tree = id3 (data, features = list(data.
    columns [:-1]))
print ("Decision Tree: ", tree)
example = { 'Outlook': 'Sunny', 'Temperature':
    'cool', 'Humidity': 'Low', 'wind':
        'strong'}
prediction = classify (tree, example)
print ("Prediction for example:", prediction)
dot = create_tree_diagram (tree)
dot. render ("decision_tree", view=True)
```

Output


Root
Outlook
Sunny
Overcast
Rain
Outlook: Sunny
Outlook: Overcast
outlook: Rain
Sunny
leaf
Rain
Humidity
Class: Yes
Wind
High
Low
Weak
Strong
Humidity: High
Low
Wind: Weak
Wind: Strong
leaf
leaf
Weak
leaf
Class: No
Class: Yes
Humidity
Class: Yes
High
Low
Humidity: High
Humidity: Low
High
leaf
Temperature
Class: No
Mild
Cool
Temp: Mild
Temp: Cool
leaf
leaf
Class: No
Class: Yes

End to end machine learning project working with real data. look at the big picture, visualize the data, Prepare the data, select and train the model and fine .,

1) Get the data

```
import pandas as pd
housing = pd.read_csv("sample_data/
california_housing_train.csv")
```

2) Discover the data

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

3) Visualize the data

```
import matplotlib pyplot as plt
import seaborn as sns
plt.hist(housing['median_income'])
plt.show()
plt.scatter(housing['median_income']
housing['median_house_value'])
plt show()
sns.heatmap(housing corr(). annot = True)
plt.show()
```

4) prepare the data

```
housing.isnull().sum()
```

5) Select and train the model

```
from sklearn model.selection
import train_test_split
from sklearn preprocessing import
```

```
one plot encoder
X= housing.drop('median house value,
    axis=1)
y = housin x median. house value')
x_train, X_test, y_train, y_test =
train_test_split(x, y, test_size = 0.2,
random_state = 42)
from sklearn.linear_model import
  Linear Regression
model = Linear Regression()
model.fit(x_train, y_train)
```

6) Fine tune your model

```
from sklearn.metrics import root_mean
squared error
import numpy as np
y_pred = model.predict(x_test)
rmse = root_mean_squared_error(y_test,
y_pred)
print(f'RMSE: {rmse}')
```