```
# -*- coding: utf-8 -*-


import pandas as pd
import sklearn
from sklearn import tree
from sklearn.model_selection import \
  train_test_split
import pydotplus
import collections
import xlsxwriter
from sklearn.preprocessing import MinMaxScaler

#read dataset-define model-predict on
# test data-graph

class decision_tree():

  def __init__(self):
      self.scaler = MinMaxScaler()

  def dataset_next_day(self,path):
    # data processing for part 1
    df = pd.read_csv(path)
    y = df['NEXTDAYPRECIPFLAG']
    x = df[df.columns[1: -2]]
    X_train, X_test, y_train, y_test = \
      train_test_split(
      x, y, test_size=0.3, random_state=42)
    self.scaler.fit(X_train)
    x_train = self.scaler.transform(X_train)
    x_test = self.scaler.transform(X_test)
    return X_train, X_test, y_train, y_test

  def dataset_day_after_tomorrow(self,path):
    # data processing for part 2
    df = pd.read_csv(path)
    df.drop(df.tail(1).index,inplace=True)
    y = df['day_after_tomorrow']
    x = df[df.columns[1: -1]]
    x.drop(['NEXTDAYPRECIPAMT'],
           axis = 1, inplace = True)
    X_train, X_test, y_train, y_test = \
      train_test_split(
      x, y, test_size=0.3, random_state=42)
    self.scaler.fit(X_train)
    x_train = self.scaler.transform(X_train)
    x_test = self.scaler.transform(X_test)
    return X_train, X_test, y_train, y_test

  def model_tree(self,depth,crit,
                 X_train, y_train):
    #define decision tree model
    clf = tree.DecisionTreeClassifier(
      criterion=crit,max_depth=depth)
    clf = clf.fit(X_train, y_train)
    return clf

  def pred(self,model,X_test):
```

```
    #make prediction on test dataset
    prediction=model.predict(X_test)
    return prediction

  def t_p_r(self,y_test,prediction):
    # calcualte true positive rate
    y_t=list(y_test)
    type(y_t)
    count_tp=0
    p=prediction
    for i in range (len(prediction)):
        if y_t[i]==p[i]:
            count_tp+=1
    trp=count_tp/len(p)
    return trp*100

  def writegraphtofile(self,classifier,
                       features, classnames,
                       pathname):
      # write tree to image
      dot_data = tree.export_graphviz(
        classifier, out_file=None,
        class_names=classnames,
        feature_names=features,
        filled=True, rounded=True,
        special_characters=True)
      graph = pydotplus.graph_from_dot_data(
        dot_data)
      colors = ('lightblue', 'pink')
      edges = collections.defaultdict(list)
      for edge in graph.get_edge_list():
          edges[edge.get_source()].append(
            int(edge.get_destination()))
      for edge in edges:
          edges[edge].sort()
          for i in range(2):
              dest = graph.get_node(
                str(edges[edge][i]))[0]
              dest.set_fillcolor(
                colors[i])
      graph.write_png(pathname)

# predict precipitation for part 1 and 2

class prediction():
  #prediction class
  def __init__(self) :
      self.t=decision_tree()

  def pred_next_day(self,path,
                    depth,crit,
                    png_path):
    # predicting next day preciptation
    X_train, X_test, y_train, y_test=\
      self.t.dataset_next_day(path)
    model= self.t.model_tree(
      depth,crit,X_train, y_train)
    prediction=self.t.pred(model,X_test)
    tpr=self.t.t_p_r(y_test,prediction)
```

```
    features=X_train.columns
    self.t.writegraphtofile(model,
                            features,
                            ('F','T'),
                            png_path)
    return tpr

  def pred_day_after_tomorrow(self,path,
                              depth,crit,
                              png_path):
    # predicting for part 2
    X_train, X_test, \
    y_train, y_test=\
      self.dataset_day_after_tomorrow(path)
    model= self.t.model_tree(
      depth,crit, X_train, y_train)
    prediction=self.t.pred(model,X_test)
    tpr=self.t.t_p_r(y_test,prediction)
    features=X_train.columns
    self.t.writegraphtofile(model,
                            features ,
                            ('F','T'),
                            png_path)
    return tpr

#fetch and run

class run():
  #last class -> fetch file, display
  # and save results
  def __init__(self):
    self.t=prediction()
  def next_day(self):
    # for next day prediction with
    # criterion entropy
    path='/content/file.csv'
    p="/content/entropy/depth"
    crit='entropy'
    print('*'*10,'entropy','*'*10)
    for depth in range(4,20):
      png_path=p+str(depth)+'.png'
      tpr=self.t.pred_next_day(
        path,depth,crit,png_path)
      print('depth of the model=',depth,
            "and true positive rate=",
            tpr)

    # for next day prediction
    # with criterion gini
    p="/content/gini/depth"
    crit='gini'
    print('*'*10,'gini','*'*10)
    for depth in range(4,20):
      png_path=p+str(depth)+'.png'
      tpr=self.t.pred_next_day(path,depth,
                               crit,
                               png_path)
      print('depth of the model=',depth,
            "and true positive rate=",tpr)
```

```
  def day_after_tomorrow(self):
    # for day after tomorrow prediction
    # with criterion entropy
    path='/content/file.csv'
    p="/content/entropy/depth"
    crit='entropy'
    print('*'*10,'entropy','*'*10)
    for depth in range(4,20):
      png_path=p+str(depth)+'.png'
      tpr=self.t.pred_day_after_tomorrow(path,
                                         depth,
                                         crit,
                                         png_path)
      print('depth of the model=',depth,
            "and true positive rate=",tpr)

    # for day after tomorrow prediction
    # with criterion gini
    p="/content/gini/depth"
    crit='gini'
    print('*'*10,'gini','*'*10)
    for depth in range(4,20):
      png_path=p+str(depth)+'.png'
      tpr=self.t.pred_day_after_tomorrow(path,
                                         depth,crit,
                                         png_path)
      print('depth of the model=',depth,
            "and true positive rate=",tpr)

df = pd.read_pickle('Cleaned_Dataset.pkl')
df.to_csv('DatasetCleaned.csv')
path='DataCleaned.csv'
t=run()
t.next_day()
t.day_after_tomorrow()
```