1. Given the dataset which represents the mood of a student to go to class depending on the four features such as “Temperature”, “Time” , “Friend_Attending” , “Windy, we are required to train a classification model and make predictions. Various steps which we performed in the process are:
    - Preprocessing
    - Training
    - Cross Validation
    - Making Predictions
    - Plotting the decision tree and decision surface

Preprocessing : In the dataset, we found that we had categorical features like “Time” which could be either morning, afternoon or evening . Similarly all other features were also categorical . While preprocessing, we converted categorical features to numeric values using ordinal encoder. Ordinal encoder encoded the categories to floating numbers . As the dataset was small, preprocessing only needed the data to be converted into numbers from categories. The codes given to categories under each feature are:

```
Time: {“Morning”: 0 , “Afternoon: 1 , “Evening”: 2},
Temperature: {“Cool”: 0, “Warm”: 1, “Hot”: 2},
Friend Attending: {“No”: 0, “Yes”: 1},
Windy: {“Low”: 0, “High”: 1},
Attend: {“No”: 0, “Yes”: 1}
```

Training : Before training , we need to divide the dataset into training and testing . We train the model on training data and use the rest of the data to test the model in order to get an idea of how better the model will work on unseen data . In this lab , we used 12 data points for training and 2 for testing. This splitting is done using `train_test_split` Function from sklearn . We made two decision tree models, `model_1` which used entropy as criteria and `model_2` which used gini impurity as the criteria .

Cross Validation : Cross validation is used to evaluate the machine learning models. Cross validation is used to avoid overfitting . In this lab we used 5 fold cross validation .We split the training data `X_train` into 5 parts. 4 parts are used for training and the 5th part is used for testing . This process is repeated 5 times. Thus 5 models are formed .We used the function `cross_val_score(model_1, X_train, y_train, cv=5)` to find the accuracy of every such model. Similarly `cross_val_score(model_2, X_train,y_train, cv=5)` Is used to validate the `model_2` (gini based)

- We get 2 accuracy arrays corresponding to `model_1` and `model_2`. However , we notice that we also get warning as `UserWarning: The least populated class in`

`y has only 4 members, which is less than n_splits=5.% (min_groups, self.n_splits)), UserWarning)` which is due to the reason that the classes in y are less populated.

- We also found the depth and number of leaves of the decision tree for both `model_1` and `model_2` using functions `model_1.get_depth()` and `model_2.get_depth()` respectively. On running the notebook many times , we realised that we get different depth and number of leaves . This is due to the reason that each time we run the notebook, the decision tree trains from scratch and gives priority to different features accordingly.

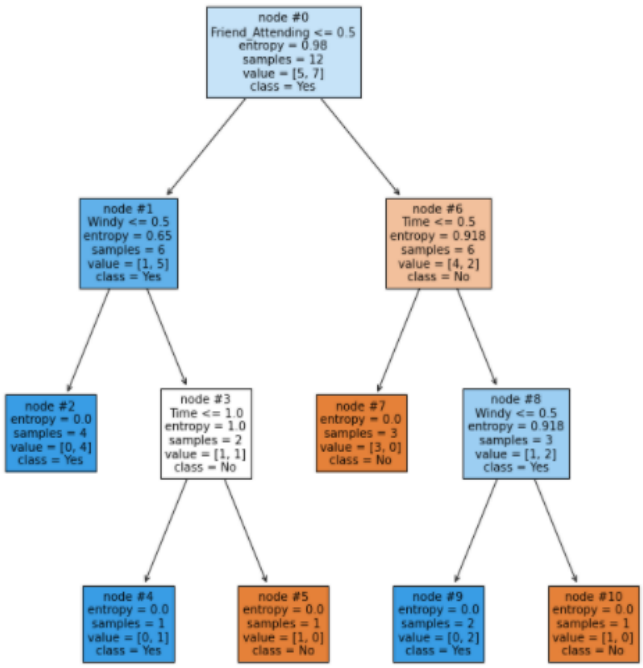
Performing classification and calculating prediction accuracy:

- We use `model_1.apply(input)` and `model_2.apply(input)` function to get the index of the leaf each sample is predicted as for both models . input refers to the feature space of the dataset. The function gives the node ids of leaf for each sample which gives an idea of the path followed while classifying. (Note : Node ids can be seen where the tree is plotted).
- To get the decision path , we use `model_1.decision_path(X_test).toarray()` and `model_2.decision_path(X_test).toarray()`. Decision path gives a 2-D array where the number of rows are equal to the number of samples in X_test and number of columns are equal to the number of nodes where for each row , each column represent whether the particular node was travelled or not (1/0) while predicting for the particular sample. We realise that the decision path may or may not be the same for `model_1` and `model_2` .
- Gini impurity for `model_2` is calculated using `model_2.feature_importances_` which gives gini impurity for each feature.
- ```
  y_pred = model_1.predict(X_train)
  score = accuracy_score(y_train, y_pred)
  print(score)
  ```
  The code snippet above gives the accuracy score on training data which is obviously 1 because the model has learnt from training data itself .
  ```
  y_pred = model_1.predict(X_test)
  score = accuracy_score(y_test, y_pred)
  print(score)
  ```
  This code snippet just above will give the testing accuracy score . We analysed that on running the notebook several times we either get accuracy as 1 or 0.5 which is due to the reason that the dataset is very small and prone to overfitting.

Plotting decision tree and decision surface :

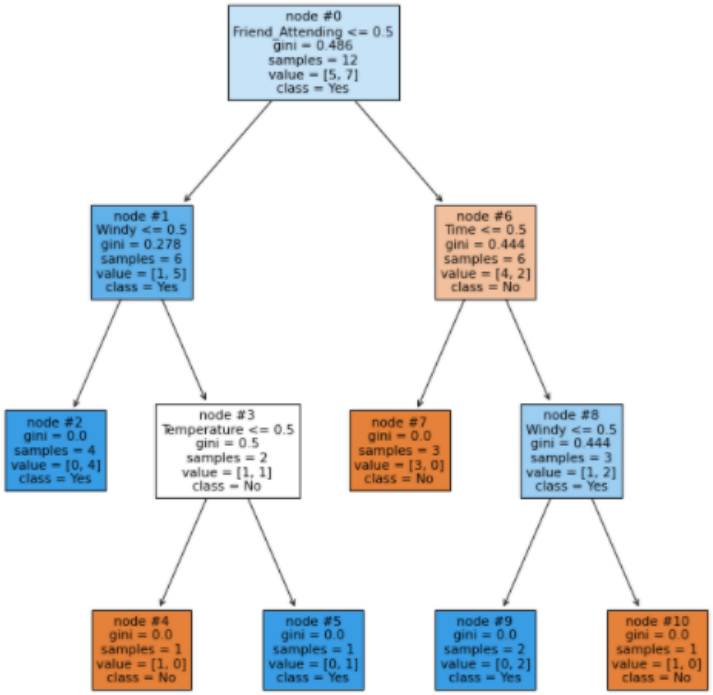
For `model_1` :
  ```

```
plt.figure(figsize=(12,12))
tree.plot_tree(model_1, feature_names=["Time", "Temperature", "Friend_Attending", "Windy"], class_names= ["No", "Yes"], filled=True, node_ids= True)
plt.show()
```



For `model_2` :

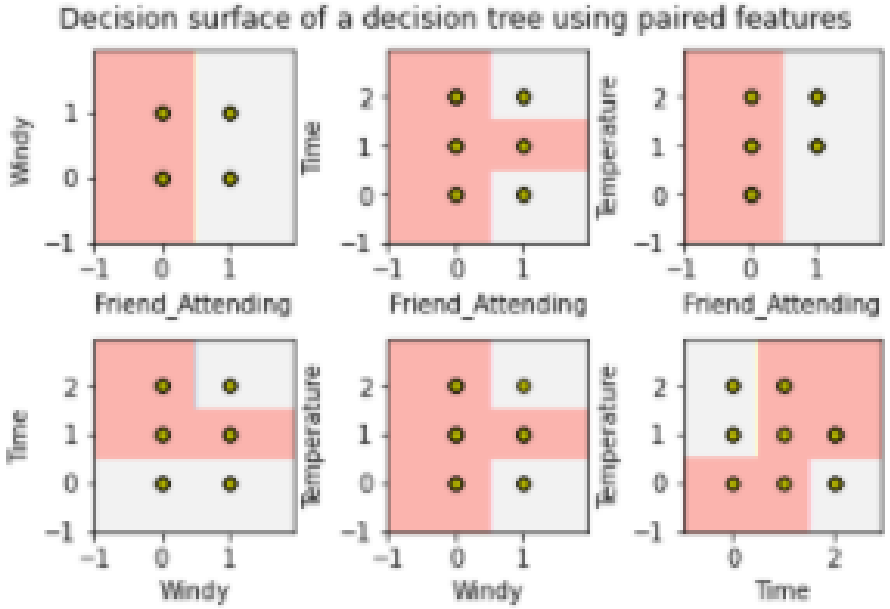
```
tree.plot_tree(model_2, feature_names=["Time", "Temperature", "Friend_Attending", "Windy"], class_names= ["No", "Yes"], filled=True, node_ids= True)
plt.show()
```



On running the notebook several times , we noticed that the decision tree for both models may or may not be the same .

Decision surfaces :

```
(-1.0, 2.9600000000000035, -1.0, 2.9600000000000035)
```



As the number of features are 4, we plotted the decision surface using features in a pairwise manner . For each graph, we can see the decision boundary separating the samples.

2. Given the dataset with `3141` rows and `11` columns we train a decision tree regressor model to predict the `Upper 95% Confidence Interval for Trend.`

Preprocessing :

- The data consists of many rows having '*' '**' characters which are meaningless, so using the below code, we first replaced these characters with NaN and then dropped the rows containing NaN values .

```
deaths.replace('**', np.nan, inplace=True)
deaths.replace('*',np.nan, inplace=True)
deaths.dropna(axis=0,inplace=True)
```

- One of the values in the column "Average Deaths per Year" was "1,57,376". The below code replaces the ',' characters and convert the string into float

```
deaths["Average Deaths per Year"] = deaths["Average Deaths per Year"].str.replace(",","").astype(float)
```

- The values in various columns such as "**Age-Adjusted Death Rate**" are strings . so we convert them to numeric values using the below code.

```
for column in deaths.columns:
  if deaths[column].dtype==object and column!="County" and column
!="Met Objective of 45.5? (1)" and column !="Recent Trend (2)"  :
    deaths[column]= deaths[column].astype(float)
  print(deaths[column].dtype)
```

- After that we convert the categorical data column categories into numeric values using ordinal encoder like in 1st question ,

```
encod = OrdinalEncoder()
deaths[["Met Objective of 45.5? (1)","Recent Trend (2)"]]=
encod.fit_transform(deaths[["Met Objective of 45.5?
(1)","Recent Trend (2)"]])
```

We drop the redundant features column .In this case , it is "County" which has all distinct values, so it doesn't give any information for learning patterns.

```
deaths = deaths.drop("County", axis="columns")
```

Training : We split the data into training and testing like we did in the previous question and fit a decision tree regressor using the following code

```
X_train, X_test, y_train, y_test = train_test_split(input_deaths,
target_deaths, test_size=0.2, random_state=0)

regressor = DecisionTreeRegressor(random_state=0)
regressor.fit(X_train, y_train)
```

Cross Validation : For this dataset, I used 10 fold cross validation . The below code gives an array of length 10 which represents the accuracy of 10 models . I realised that the accuracy for each of the 10 models is much more as compared to the accuracy of each of 5 models if I used 5 fold cross validation .

```
cross_val_score(regressor, X_train, y_train, cv=10)
```

Perform decision tree regression and calculating the squared error between the predicted and the ground-truth values for the test data :

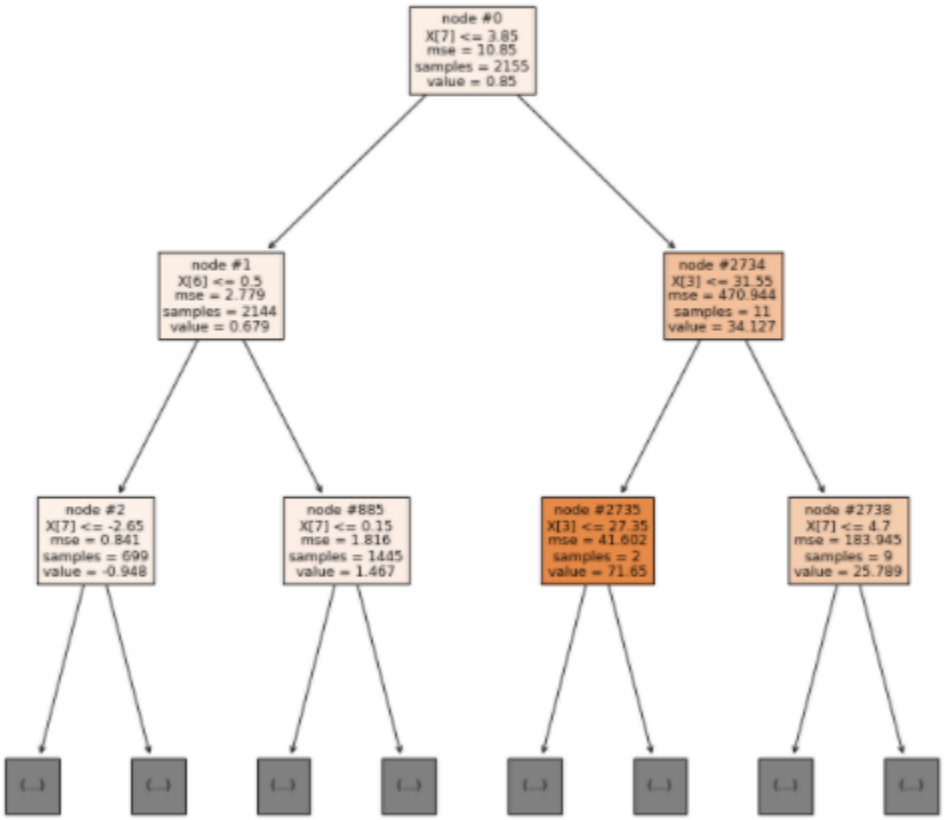
The following code gives an array of length the same as X_test consisting of predicted values for each test data point.

```
y_pred= regressor.predict(X_test)
print(y_pred)
```

While calculating the mean squared error , I gave multioutput parameter as “uniform average” because it defines aggregating of multiple output values.

```
mean_squared_error(y_test, y_pred, multioutput='uniform_average')
```

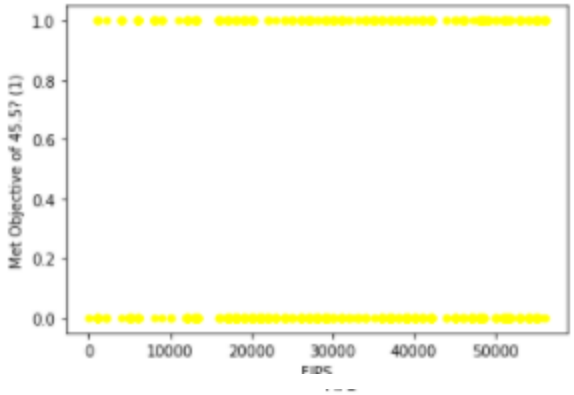
Plot the decision tree and other graphs :

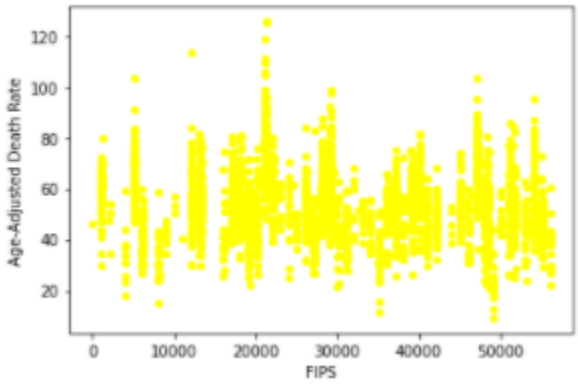


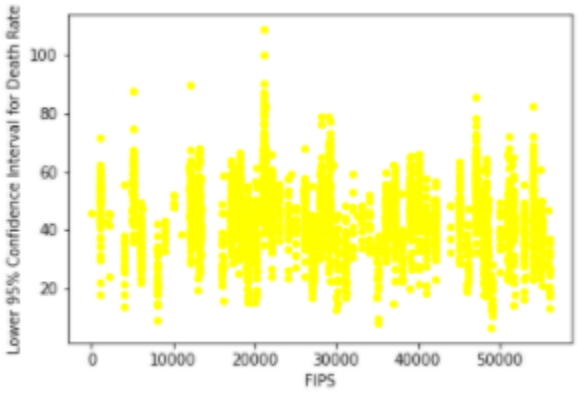
The tree shown above is not completely printed due to the reason that the number of leaves are high so we used a parameter `max_depth=2` to limit the maximum depth of the tree in the code below

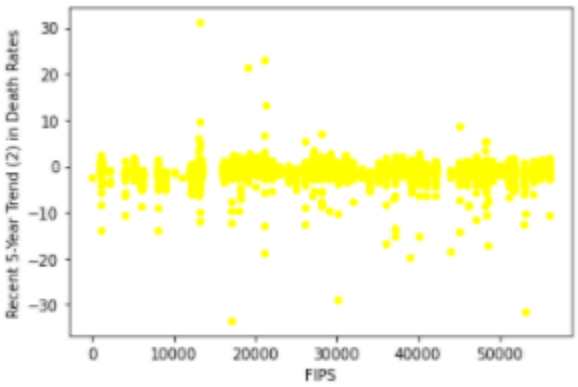
```
tree.plot_tree(regressor, filled=True, node_ids= True, max_depth=2)
```

Some scattered plots created between features in pair-wise manner are as follows:


Met Objective of 45.5? (1)
1.0
0.8
0.6
0.4
0.2
0.0
0
10000
20000
30000
40000
50000
FIPS


Age-Adjusted Death Rate
120
100
80
60
40
20
0
10000
20000
30000
40000
50000
FIPS


Lower 95% Confidence Interval for Death Rate
100
80
60
40
20
0
10000
20000
30000
40000
50000
FIPS


Recent 5-Year Trend (2) in Death Rates
30
20
10
0
−10
−20
−30
0
10000
20000
30000
40000
50000
FIPS