Assignment – 1

Task description (including task 1, 2 and 3)

Steps for pre-processing of data

1. Imported Traffic data
2. Imported CO data and grouped CO data for each day and did average per day for CO data.
   Reference : Code.pdf cell[4]

3. Merged data for both Tables as shown in snapshot below.
   Reference : Code.pdf cell[5]

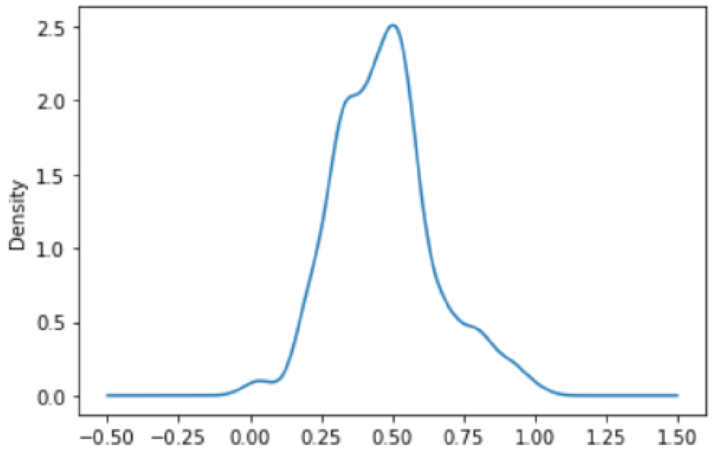
4. Normalized data for columns ADT, AADT and Average as shown below.
   Reference : Code.pdf cell[6]

5. After normalization, performed discretization of data based on CO level and added column COIndex consisting Low if Average is < 0.5 and High if Average is > 0.5.
   Reference : Code.pdf cell[7]

Descriptive analysis of data

1. Below shows normal distribution for column Average, which shows after normalization data lies between 0 and 1.

```
In [8]:  MergedData['Average'].plot.kde()
Out[8]: <AxesSubplot:ylabel='Density'>
```



2. Below I summed up data for each month rather than days and plotted data for columns ADT, AADT and Average which shows in most of the months with increase in Average Daily Traffic (ADT) CO level also increase but in winters i.e. in month of November and December ADT decreases but CO Level increases which shows in those month it doesn’t depend much on ADT but on some other factors.

In [9]:
```python
MonthlyMergedData = MergedData
MonthlyMergedData['Month']= pd.DatetimeIndex(MergedData['Date']).month
MonthlyMergedData = MonthlyMergedData.groupby('Month').mean()
cols_to_norm = ['Average','ADT','AADT']
print(cols_to_norm)
MonthlyMergedData[cols_to_norm] = MonthlyMergedData[cols_to_norm].apply(lambda x: (x - x.min()) / (x.max() - x.min()))
MonthlyMergedData
```

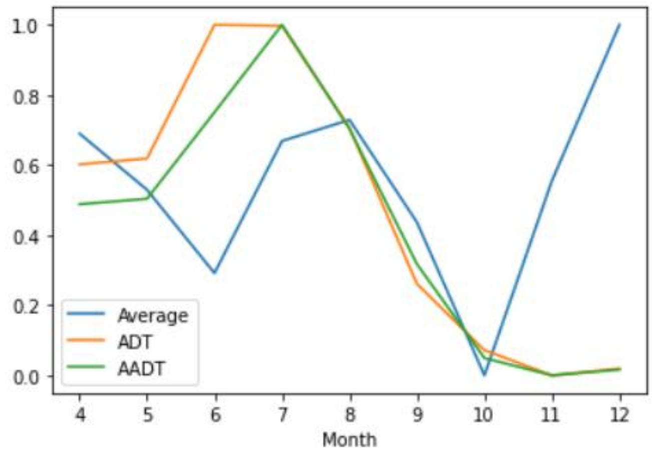
```
['Average', 'ADT', 'AADT']
```

Out[9]:

Month	HIGHWAY	SECTION	SECTION LENGTH	ADT	AADT	Average
4	213.900000	36.100000	5.921000	0.601755	0.488117	0.689652
5	120.608108	66.918919	7.076723	0.618436	0.503683	0.529338
6	141.291667	56.937500	8.790854	1.000000	0.750396	0.291653
7	190.831858	32.734513	8.732389	0.996919	1.000000	0.668474
8	189.225352	49.070423	8.116507	0.705706	0.703298	0.728707
9	99.615385	70.169231	6.644262	0.260777	0.317203	0.436449
10	142.204082	29.306122	4.890633	0.071712	0.049255	0.000000
11	141.250000	90.166667	4.101750	0.000000	0.000000	0.555140
12	103.083333	47.916667	3.670000	0.019251	0.016470	1.000000

In [10]:
```python
MonthlyMergedData[['Average','ADT','AADT']].plot()
```

Out[10]: <AxesSubplot:xlabel='Month'>

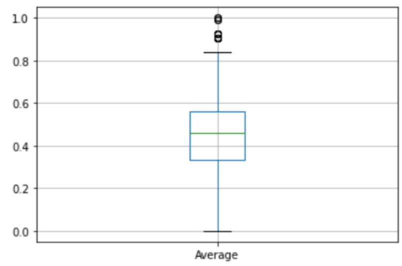


3. Below I analysed box plots for Average, ADT and AADT and found like for
    - For Average column distribution is symmetric with few outliers.

Average distribution is symmetric with few outlier

In [11]:
```python
MergedData.boxplot(column=['Average'])
```

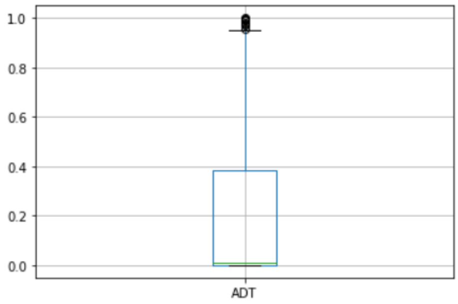
Out[11]: <AxesSubplot:>



- For ADT column distribution is positively skewed with few outliers.

ADT distribution is positively skewed with few outliers

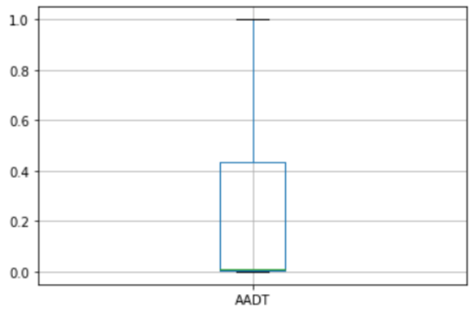
In [12]:
```
MergedData.boxplot(column=['ADT'])
```
Out[12]: `<AxesSubplot:>`



- For AADT column data is positively skewed.

AADT distribution is positively skewed

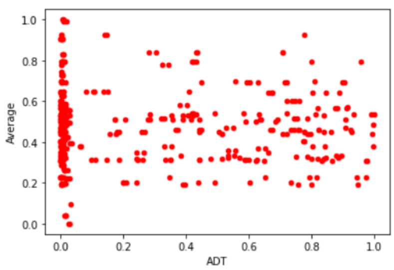
In [13]:
```
MergedData.boxplot(column=['AADT'])
```
Out[13]: `<AxesSubplot:>`



4. Next, I visualized scatter plot between ADT vs Average and AADT vs Average which shows data is non-linear and because of that decision tree is the best model to classify non-linear data.

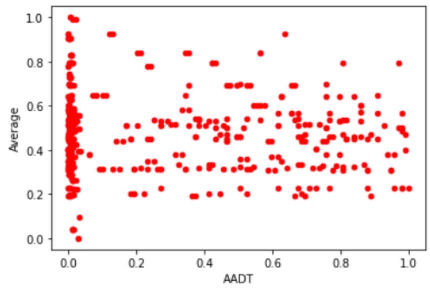
ADT and Average on compare is non-linear

In [14]:
```
MergedData.plot.scatter(x='ADT',
                        y='Average',
                        c='Red')
```
Out[14]: `<AxesSubplot:xlabel='ADT', ylabel='Average'>`



AADT and Average on compare is non-linear

In [15]:
```
MergedData.plot.scatter(x='AADT',
                        y='Average',
                        c='Red')
```
Out[15]: `<AxesSubplot:xlabel='AADT', ylabel='Average'>`

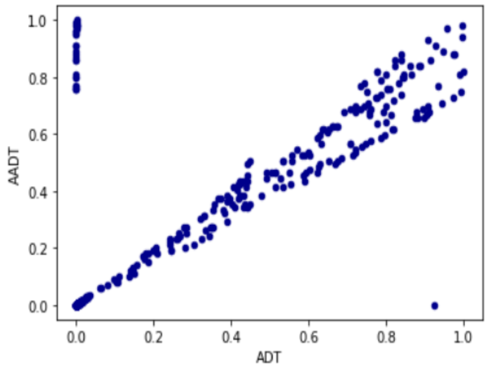


5. Visualized scatter plot for ADT and AADT which shoed and interesting result that both are linear and using both as features will not improve Model accuracy hence, we can skip AADT from features list.

ADT and AADT on compare is linear, so we will consider only ADT into feature and will remove AADT as both are linear and taking both will not improve our model

In [16]:
```
MergedData.plot.scatter(x='ADT',
                        y='AADT',
                        c='DarkBlue')
```

Out[16]: <AxesSubplot:xlabel='ADT', ylabel='AADT'>

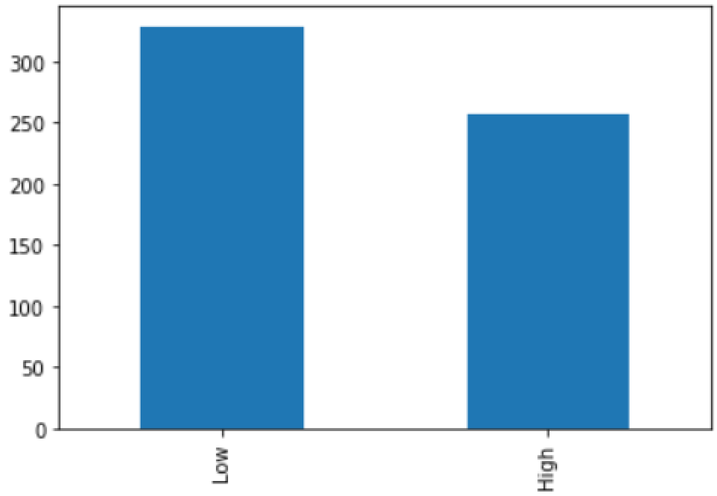


6. Visualized Bar graph for our Target column i.e. COIndex and found that our data is imbalanced as we have more Lows than Highs.

Below is the bar graph for our Target attribute of model i.e. COIndex

In [17]:
```
MergedData['COIndex'].value_counts().plot(kind='bar')
```

Out[17]: <AxesSubplot:>



Summary visualization of our final Merged data

So, the feature that we will consider in our model are Highway, Section, Section Length, ADT and our target will be COIndex.

Below shows Summary Visualization of Data

In [19]:
```
MergedData.describe(include = 'all')
```
Out[19]:

	HIGHWAY	SECTION	SECTION LENGTH	ADT	COIndex
count	586.000000	586.000000	586.000000	586.000000	586
unique	NaN	NaN	NaN	NaN	2
top	NaN	NaN	NaN	NaN	Low
freq	NaN	NaN	NaN	NaN	329
mean	148.576792	52.779863	7.401891	0.197231	NaN
std	125.552938	56.718809	3.981491	0.306120	NaN
min	1.000000	1.000000	0.200000	0.000000	NaN
25%	7.000000	17.000000	4.150000	0.001846	NaN
50%	104.000000	30.000000	7.253500	0.007201	NaN
75%	245.000000	60.000000	10.040000	0.382794	NaN
max	376.000000	270.000000	20.720000	1.000000	NaN

Reasons for feature selection and dropping columns

Reason for removing columns from our Final Data

1. Date & Time- removed because our model cannot understand date and time values.
2. Pollutant- removed because in data it is unique as shown below.
3. Unit- removed because in data it is unique as shown below.

In [21]:
```
AllCOData.describe(include='all')
```
Out[21]:

	Date & Time	Pollutant	Unit	Station	Average
count	236687	236687	236687	236687	205269.000000
unique	236687	1	1	2	NaN
top	01/01/2019 12:00:00 AM	CO	ppm	Halifax	NaN
freq	1	236687	236687	210384	NaN
mean	NaN	NaN	NaN	NaN	0.358628
std	NaN	NaN	NaN	NaN	0.322583
min	NaN	NaN	NaN	NaN	0.000000
25%	NaN	NaN	NaN	NaN	0.130000
50%	NaN	NaN	NaN	NaN	0.270000
75%	NaN	NaN	NaN	NaN	0.500000
max	NaN	NaN	NaN	NaN	11.020000

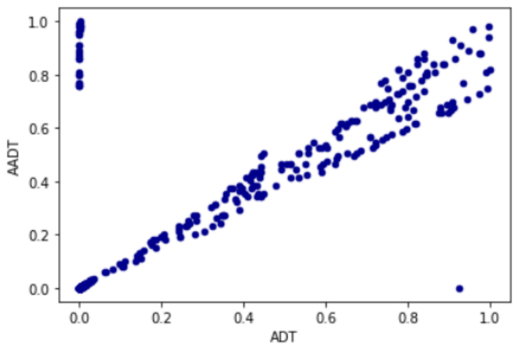
4. We removed station from final data because for year 2019 its unique as shown below.

We removed station from final data because for year 2019 its unique as shown below

```
In [23]:  lCOData["Date & Time"]>='01/01/2019 12:00:00 AM' )& (AllCOData["Date & Time"]<'01/01/2020 12:00:00 AM')]['Station'].nunique()

Out[23]: 1
```

5. We removed AADT from the feature list because ADT and ADDT shows linear relationship as shown in below graph so taking both will not improve accuracy of model hence, we can drop AADT.

```
In [16]:  MergedData.plot.scatter(x='ADT',
                                 y='AADT',
                                 c='DarkBlue')

Out[16]: <AxesSubplot:xlabel='ADT', ylabel='AADT'>
```

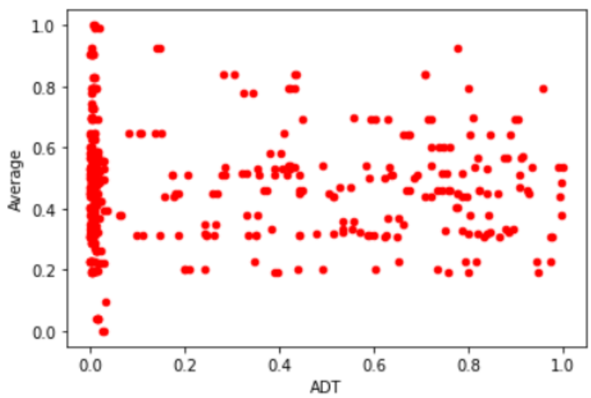


Why decision tree is reasonable model to try this data?

The reason that decision tree is a reasonable model for this data is because as we can see from the snapshot of scatter plot below between ADT and Average which shows that the data between both the columns in non-linear. And we know that for non-linear data decision trees is the best model to classify.

ADT and Average on compare is non-linear

```
In [14]:  MergedData.plot.scatter(x='ADT',
                                 y='Average',
                                 c='Red')

Out[14]: <AxesSubplot:xlabel='ADT', ylabel='Average'>
```



Question i (Task 4)

After creating the model, we found out that ADT is the most influential factor for CO level. As we can see from the snapshot below that using clf_tree.feature_importances_ that ADT returns the highest importance value as compared to other features.

What is the most influential factor for COlevel? Why?

```
In [54]:  demo = MergedData.drop('COIndex',axis = 1)
          pd.Series(clf_tree.feature_importances_ , index = demo.columns)

Out[54]:  HIGHWAY           0.294408
          SECTION           0.167913
          SECTION LENGTH    0.171078
          ADT               0.366600
          dtype: float64
```

Question ii (Task 4)
Formula to calculate Entropy

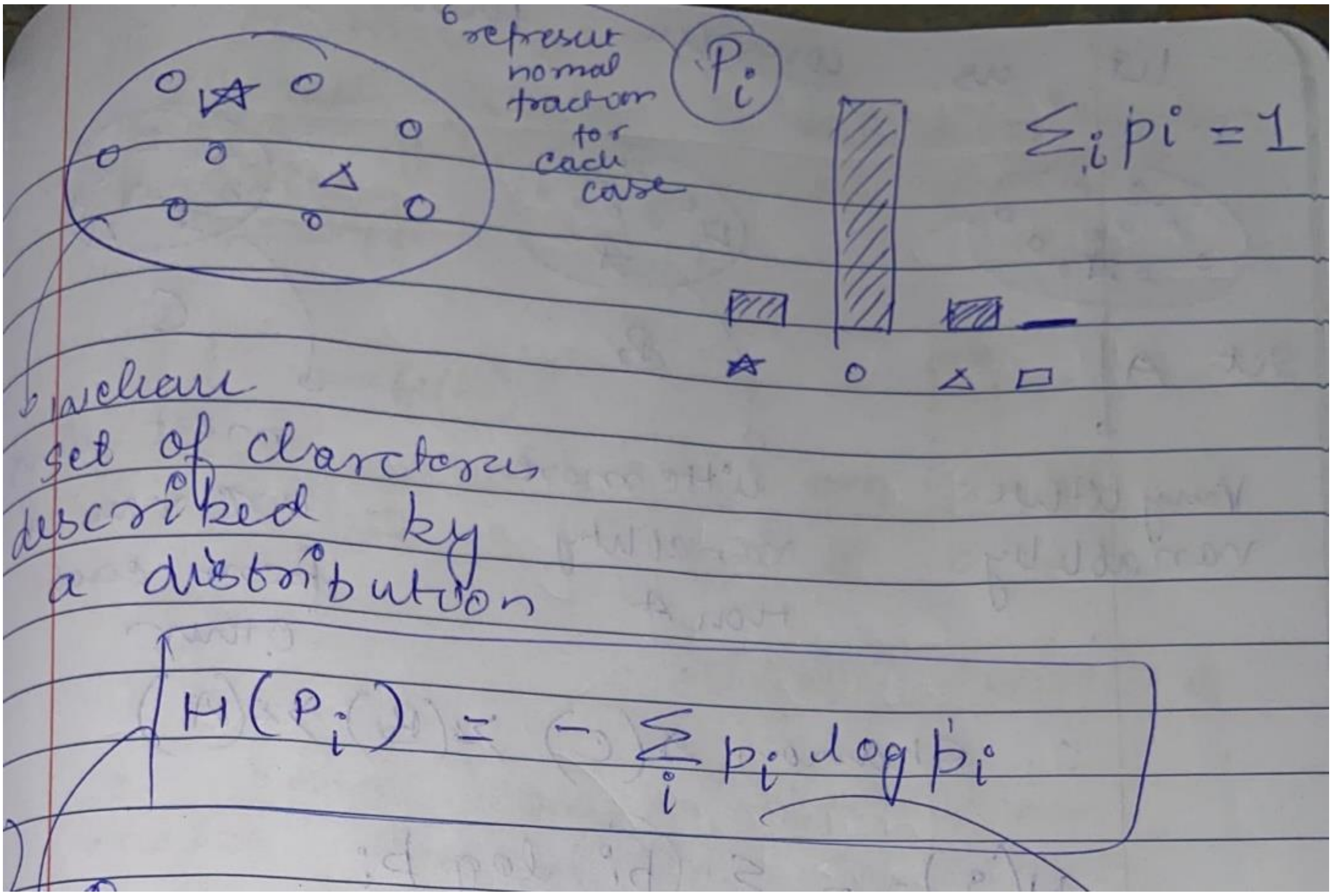


Figure 1 Formula to calculate Entropy

Calculating IG for our features to find root node

$$\text{Entropy} = -\sum_i p_i \log p_i$$

CO Level — Low (329)

CO Level — High (257)

$$\text{Entropy}(329;257) = -\frac{(329)}{(586)} \log_2 (329/586) - \frac{(257)}{(586)} \log_2 (257/586)$$

$$= -0.56 \times (-0.83) - (0.43) \times (-1.18)$$

$$= +0.4648 - (-0.5074)$$

$$= 0.4648 + 0.5074$$

$$\boxed{\text{Entropy}(S) = 0.9722} \text{ —— } (1)$$

Now to Calculate IG

$$\text{Gain}(S, A) = \text{Entropy}(S) - \sum_{v \to \text{values of } A} \frac{|S_v|}{|S|} \text{Entropy}(S_v)$$

Set of all possible values for attribute such as highway $= 63$

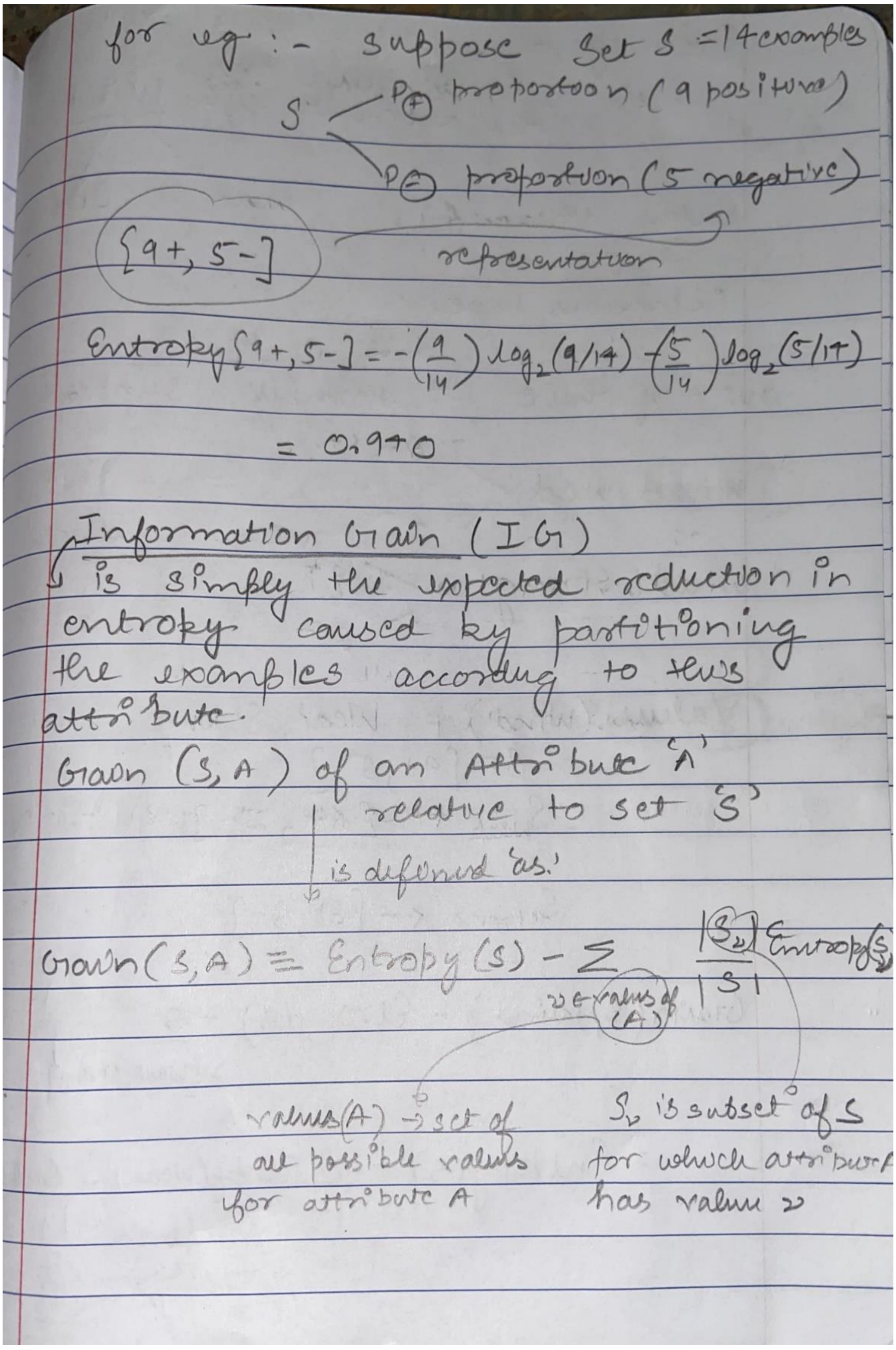
S_v is Subset of S for which attribute has values v.

So, as we have alot of discreet possible values for our features hence it will be very difficult to calculate IG for all features

Figure 2 Calculating Entropy an IG for our CO Model

Calculating Information Gain for Baseball player example as explained in lecture.

Taking the baseball player example where we have dataset of only 14 rows and out of which 9 days he goes to play and 5 days he doesn't.



for eg :- suppose Set S = 14 examples

S → $P_{\oplus}$ proportion (9 positive)

S → $P_{\ominus}$ proportion (5 negative)

[9+, 5−] representation

$$\text{Entropy}[9+, 5-] = -\left(\frac{9}{14}\right)\log_2(9/14) - \left(\frac{5}{14}\right)\log_2(5/14)$$

$$= 0.940$$

Information Gain (IG)

is simply the expected reduction in entropy caused by partitioning the examples according to this attribute.

Gain (S, A) of an Attribute 'A' relative to set 'S' is defined 'as.'

$$\text{Gain}(S,A) \equiv \text{Entropy}(S) - \sum_{v \in \text{values}(A)} \frac{|S_v|}{|S|}\,\text{Entropy}(S_v)$$

values(A) → set of all possible values for attribute A

S_v is subset of S for which attribute A has value v

Figure 3 Formula for Information Gain

For eg:- suppose 'S' is a collection in which one of the attribute is Wind

Wind → weak, Strong

containg 14 exampl [9+, 5−] as in previous page.

out of these 14 examples suppose

Wind weak: $6+$, $2-$

wind strong: $3+$, $3-$

Values (Wind) = Weak, Strong

$$S = [9+, 5-]$$

$$S_{Weak} \leftarrow [6+, 2-]$$

$$S_{Strong} \leftarrow [3+, 3-]$$

$$\text{Gain}(S, \text{Wind}) = \text{Entropy}(S) - \sum_{v\{\text{Weak},\, \text{Strong}\}} \frac{|S_v|}{|S|} \text{Entropy}(S_v)$$

$$= \text{Entropy}(S) - \frac{8}{14}\,\text{Entropy}(\text{weak}) - \frac{6}{14}\,\text{Entro}$$

3

Figure 4 Showing how to calculate IG for Gain(S,Wind)

Entropy (weak)

$$\text{Entropy}(6+, 2-) = -\frac{6}{8}\log_2\left(\frac{6}{8}\right) - \frac{2}{8}\log_2\left(\frac{2}{8}\right)$$

$$= -\left(\frac{6}{8} - (0.415)\right) - \left(\frac{2}{8} \times (-2)\right)$$

$$= -(0.75 \times -0.415) -$$

$$= -(-0.311) - (0.25 \times -2)$$

$$= 0.311 + 0.50$$

$$\text{Entropy(weak)} = 0.811 \quad (1)$$

$$\text{Entropy(strong)} = \text{Entropy}(3+, 3-)$$

$$= 1 \quad (2)$$

Substituting (1) and (2) in (3) we get

$$= 0.940 - \frac{8}{14} \times 0.811 - \frac{6}{14} \times 1$$

$$= 0.940 - \frac{2}{14}(3.244 + 3)$$

$$= 0.940 - \frac{1}{7}(6.244)$$

$$= \frac{6.580 - 6.244}{7}$$

(strong)

$$\text{Gain}(S, \text{wind}) = 0.048$$

Figure 5 Substituting values in IG equation for feature wind

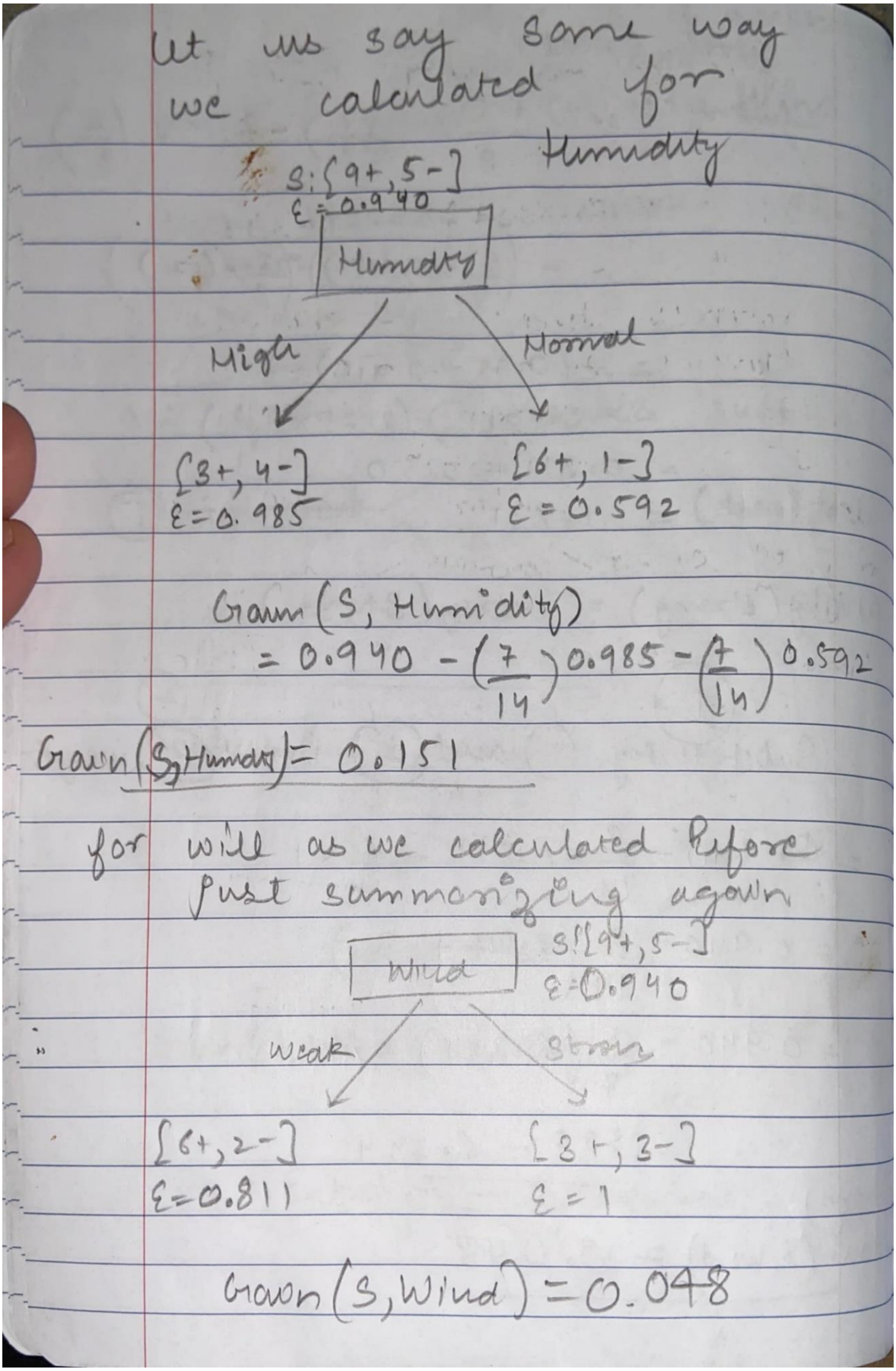


Figure 6 Same way we calculated for feature Humidity

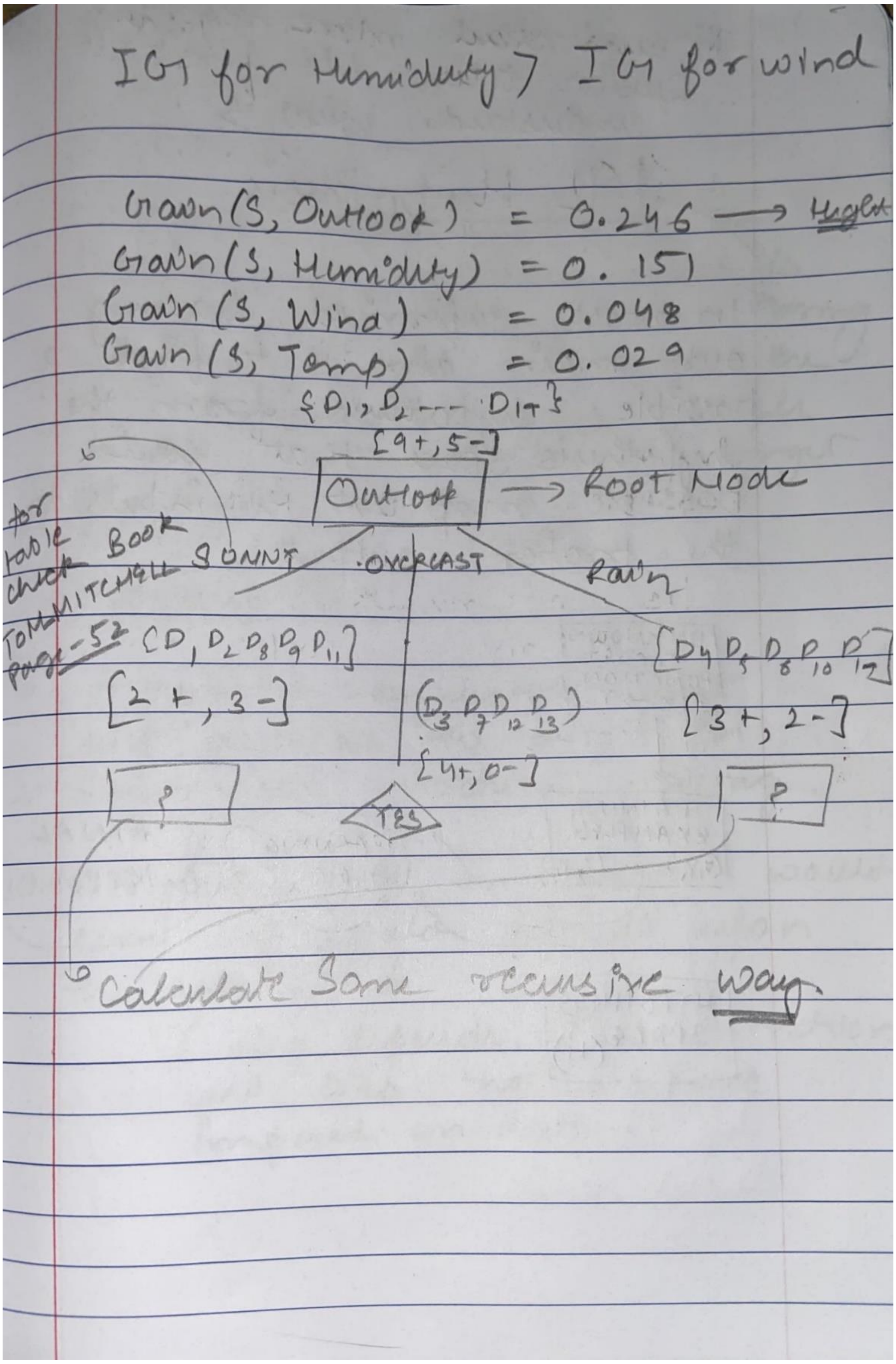


Figure 7 Same we calculated for all features and found out Outlook have highest IG value.

Question iii (Task 4)

a) Classification report for decision tree with 50% train and 50% test data

```
In [40]:  print("Report : \n",
             classification_report(y_test, y_pred))

Report :
               precision    recall  f1-score   support

        High       0.63      0.61      0.62       125
         Low       0.72      0.74      0.73       168

    accuracy                           0.68       293
   macro avg       0.68      0.67      0.67       293
weighted avg       0.68      0.68      0.68       293
```

b) Classification report and confusion matrix after performing 10-fold cross validation

mean of all 10 confusion matrices

```
In [48]:  mean_of_conf_matrix_arrays = np.mean(conf_matrix_list, axis=0)
          mean_of_conf_matrix_arrays

Out[48]:  array([[ 9.6, 16.1],
                 [16.6, 16.3]])
```

Finding True positive(TP), False Negative(FN), False Positive(FP), True Negative(TN)

```
In [49]:  TP = mean_of_conf_matrix_arrays[0][0]
          FN = mean_of_conf_matrix_arrays[0][1]
          FP = mean_of_conf_matrix_arrays[1][0]
          TN = mean_of_conf_matrix_arrays[1][1]
```

Finding of Accuracy based on mean of Confusion matrix

```
In [50]:  Accuracy = (TP + TN)/(TP+FN+FP+TN)
          Accuracy

Out[50]:  0.44197952218430026
```

Finding of Precision based on mean of Confusion matrix

```
In [51]:  Precision = TP/(TP+FP)
          Precision

Out[51]:  0.36641221374045796
```

Finding of Recall based on mean of Confusion matrix

```
In [52]:  Recall = TP/(TP+FN)
          Recall

Out[52]:  0.37354085603112835
```

Finding of F1-measure based on Recall and Precision

```
In [53]:  F1_measure = (2*(Recall)*(Precision))/(Recall+Precision)
          F1_measure

Out[53]:  0.36994219653179183
```

c) Does the model make sense and are there any leaf nodes that are very small?
- Yes, the model makes sense as we can see while performing model with 50% training data the accuracy was 100% but with the left out 50% test data accuracy is 68% which shows that the model is doing overfitting because of which it is working well for trained data but on unseen test data it fails to predict.
- Secondly, dataset is very less because of which model will show overfitting.
- Yes, we do have a lot of small leaf nodes in our model as we can see model is overfitted which we can be control by hyperparameter tuning of min_samples_leaf.

d) which evaluation metric works well for your data and model and why?
- F-1 measure works well for our model because as we can see F1-measure score for Model with 50% training and 50% test data is 0.62 whereas F-1 measure after doing 10-fold cross validation is only 0.369 which shows that the model is not performing well for unseen data i.e., model is simply overfitting (which means model completely fits the training data but fails to generalize testing or unseen data).
- Why we are not considering accuracy because if we blindly fit our data with all the CO level as high, it will in any case will give around 60% accuracy. Moreover, in F1-measure we don't consider True Negative that is why it will work well for our model.

Question iv (Task-4)

After doing hyperparameter tuning based on parameters such as max_depth, min_samples_split, min_samples_leaf we found that:

max_depth:
- Sklearn default value is None.
- In general, the deeper we allow our tree to grow, the more complex our model will become we will have more splits and it will capture more information about the training data that is one of the root causes of overfitting in this model.
- So as our model is overfitted and reducing max_depth will help to combat overfitting **(Reference: cell [55][56][57])**.
- But if we have very low max_depth our model will go underfitted (**Reference: cell [58][59][60]**), so decrease the max_depth based on degree of overfitting we have in our model.

min_samples_split
- Sklearn default value for this is 2.
- min_samples_split also used to control overfitting in model, Higher the values prevent the model in learning relations **(Reference: cell [61][62][63])**.

- And too high values can lead to underfitting of model **(Reference: cell [64][65][66])**, so depending on level of overfitting in model we have to tune our model.

min_samples_leaf

- Sklearn default value for this is 1.
- Similar to min_sample_split, min_sample_leaf is also used to control overfitting by making sure each leaf has more than one element **(Reference: cell [67][68][69]).**
- Having small value will mean that tree will overfit, whereas large value will prevent tree from learning data and will lead to underfitting **(Reference: cell [70][71][72]).**

Summary

- After performing decision tree classification of data, we found that our default classifier model is overfitted and generalizes training data with 100% accuracy but struggles to generalize unseen test data.
- As we have very less amount of data which also contributes as one of the factors for overfitting.
- As our tree is overfitted because of that we have a quite a few leaf nodes in our tree.
- We summarized that; F-1 measure can be the best evaluation metric for our model.
- We found that our model struggles generalizing unseen data as we saw that in 10-fold cross validation technique F-1 score was very low.
- We found that how our model can be improved using hyperparameter tuning.
- We found that how max_depth parameter can combat overfitting, and how very low max_depth value can lead to underfitting.
- Same way we found like how min_samples_split and min_samples_leaf can be used to control overfitting in our model.

References

1. https://pandas.pydata.org/docs/
2. https://scikit-learn.org/stable/
3. https://numpy.org/doc/

Assignment1_Code

October 6, 2021

[1]:
```
import pandas as pd
import numpy as np
import sklearn as sk
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import KFold
from sklearn import tree
```

[2]:
```
TraffData = pd.read_csv("cleaned_traffic_data.csv",sep=',',parse_dates=['Date'])
TraffData['Date'] = TraffData['Date'].dt.strftime('%m/%d/%Y')
TraffData
```

[2]:
```
           Date  HIGHWAY  SECTION  SECTION LENGTH      ADT     AADT
0    09/09/2019        1       47            4.50    2.566    2.430
1    06/17/2019        1       50            7.60    4.266    3.840
2    06/17/2019        1       50            7.60    3.934    3.545
3    06/17/2019        1       50            7.60    2.924    2.640
4    09/09/2019        1       50            7.60    6.164    5.520
..          …        …        …               …        …       …
581  06/27/2019      374       28            6.83  241.000  220.000
582  06/27/2019      374       30           11.04  488.000  440.000
583  06/04/2019      376       10            5.68    1.409    1.320
584  05/28/2019      376       20            5.96    2.215    2.080
585  05/28/2019      376       30            4.76    4.876    4.580

[586 rows x 6 columns]
```

[3]:
```
AllCOData = pd.
 ↪read_csv("Nova_Scotia_Provincial_Ambient_Carbon_Monoxide__CO__Hourly_Data_Halifax_Johnston.
 ↪csv", sep =',')
AllCOData.head()
```

[3]:

```
                Date & Time Pollutant Unit           Station  Average
0  01/01/2019 12:00:00 AM        CO  ppm  Halifax Johnston     0.25
1  01/01/2019 01:00:00 AM        CO  ppm  Halifax Johnston     0.26
2  01/01/2019 02:00:00 AM        CO  ppm  Halifax Johnston     0.20
3  01/01/2019 03:00:00 AM        CO  ppm  Halifax Johnston     0.17
4  01/01/2019 04:00:00 AM        CO  ppm  Halifax Johnston     0.15
```

[4]:

```python
COData = AllCOData[['Date & Time','Average']]
COData.rename(columns=({'Date & Time':'Date'}),inplace=True,)
COData[['Date','Time','AP']] =  COData.Date.str.split(" ", expand = True)
COData = COData[['Date','Average']]
COData = COData[COData['Date'].str.contains('[\d/]2019')]
COData = COData.groupby('Date').mean()
COData
```

```
C:\Users\Mayank\AppData\Local\Programs\Python\Python38\lib\site-
packages\pandas\core\frame.py:5039: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  return super().rename(
C:\Users\Mayank\AppData\Local\Programs\Python\Python38\lib\site-
packages\pandas\core\frame.py:3641: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self[k1] = value[k2]
```

[4]:

```
             Average
Date
01/01/2019  0.146250
01/02/2019  0.152917
01/03/2019  0.198333
01/04/2019  0.178333
01/05/2019  0.197083
…                …
12/27/2019  0.127083
12/28/2019  0.116250
12/29/2019  0.106667
12/30/2019  0.096667
12/31/2019  0.071250

[365 rows x 1 columns]
```

[5]:
```python
MergedData = pd.merge(TraffData,COData,on='Date',how='left')
MergedData
```

[5]:
```
           Date  HIGHWAY  SECTION  SECTION LENGTH      ADT     AADT   Average
0    09/09/2019        1       47            4.50    2.566    2.430  0.122174
1    06/17/2019        1       50            7.60    4.266    3.840  0.144167
2    06/17/2019        1       50            7.60    3.934    3.545  0.144167
3    06/17/2019        1       50            7.60    2.924    2.640  0.144167
4    09/09/2019        1       50            7.60    6.164    5.520  0.122174
..          …      …      …             …      …      …       …
581  06/27/2019      374       28            6.83  241.000  220.000  0.073913
582  06/27/2019      374       30           11.04  488.000  440.000  0.073913
583  06/04/2019      376       10            5.68    1.409    1.320  0.122083
584  05/28/2019      376       20            5.96    2.215    2.080  0.097391
585  05/28/2019      376       30            4.76    4.876    4.580  0.097391

[586 rows x 7 columns]
```

[6]:
```python
cols_to_norm = ['Average','ADT','AADT']
print(cols_to_norm)
MergedData[cols_to_norm] = MergedData[cols_to_norm].apply(lambda x: (x - x.
 ↪min()) / (x.max() - x.min()))
MergedData
```

```
['Average', 'ADT', 'AADT']
```

[6]:
```
           Date  HIGHWAY  SECTION  SECTION LENGTH       ADT      AADT  \
0    09/09/2019        1       47            4.50  0.001571  0.001446
1    06/17/2019        1       50            7.60  0.003282  0.002872
2    06/17/2019        1       50            7.60  0.002948  0.002573
3    06/17/2019        1       50            7.60  0.001932  0.001658
4    09/09/2019        1       50            7.60  0.005191  0.004570
..          …      …      …             …       …       …
581  06/27/2019      374       28            6.83  0.241446  0.221436
582  06/27/2019      374       30           11.04  0.489938  0.443883
583  06/04/2019      376       10            5.68  0.000407  0.000324
584  05/28/2019      376       20            5.96  0.001218  0.001092
585  05/28/2019      376       30            4.76  0.003895  0.003620

      Average
0    0.542614
1    0.698085
2    0.698085
3    0.698085
4    0.542614
..        …
581  0.201447
582  0.201447
```

```
583   0.541973
584   0.367420
585   0.367420

[586 rows x 7 columns]
```

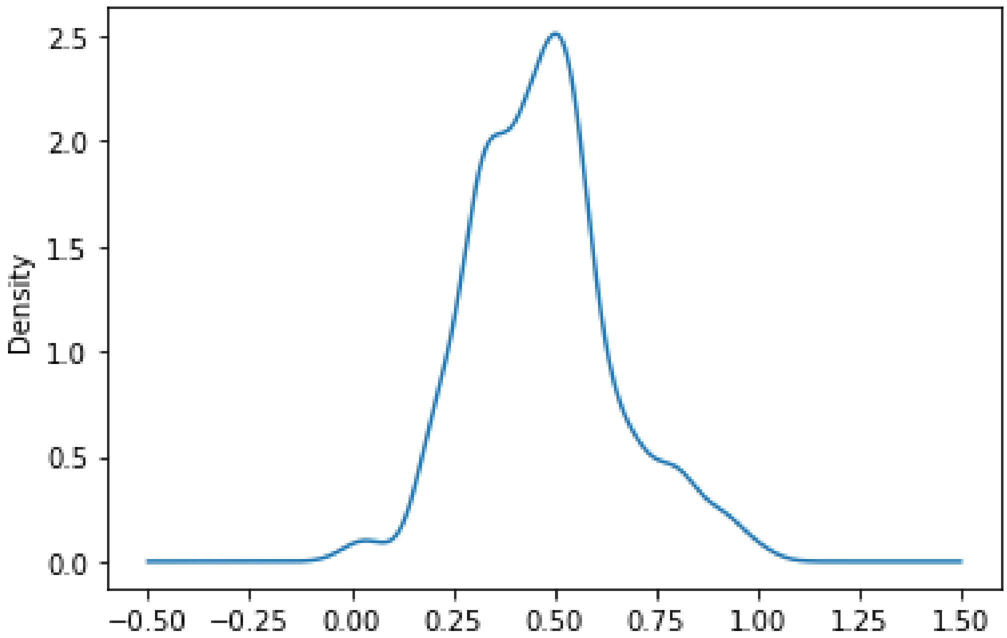
[7]:
```
MergedData['COIndex'] = np.where(MergedData['Average']<0.5,'Low','High')
MergedData
```

[7]:
```
           Date  HIGHWAY  SECTION  SECTION LENGTH       ADT      AADT  \
0    09/09/2019        1       47            4.50  0.001571  0.001446
1    06/17/2019        1       50            7.60  0.003282  0.002872
2    06/17/2019        1       50            7.60  0.002948  0.002573
3    06/17/2019        1       50            7.60  0.001932  0.001658
4    09/09/2019        1       50            7.60  0.005191  0.004570
..          …        …        …               …         …         …
581  06/27/2019      374       28            6.83  0.241446  0.221436
582  06/27/2019      374       30           11.04  0.489938  0.443883
583  06/04/2019      376       10            5.68  0.000407  0.000324
584  05/28/2019      376       20            5.96  0.001218  0.001092
585  05/28/2019      376       30            4.76  0.003895  0.003620

      Average COIndex
0    0.542614    High
1    0.698085    High
2    0.698085    High
3    0.698085    High
4    0.542614    High
..        …       …
581  0.201447     Low
582  0.201447     Low
583  0.541973    High
584  0.367420     Low
585  0.367420     Low

[586 rows x 8 columns]
```

Below shows Normal-Distribution of column Average after normalization

[8]:
```
MergedData['Average'].plot.kde()
```

[8]:
```
<AxesSubplot:ylabel='Density'>
```



Below shows data visulaization monthly and plot for Average, ADT, AADT

[9]:
```
MonthlyMergedData = MergedData
MonthlyMergedData['Month']= pd.DatetimeIndex(MergedData['Date']).month
MonthlyMergedData = MonthlyMergedData.groupby('Month').mean()
cols_to_norm = ['Average','ADT','AADT']
print(cols_to_norm)
MonthlyMergedData[cols_to_norm] = MonthlyMergedData[cols_to_norm].apply(lambda␣
↪x: (x - x.min()) / (x.max() - x.min()))
MonthlyMergedData
```

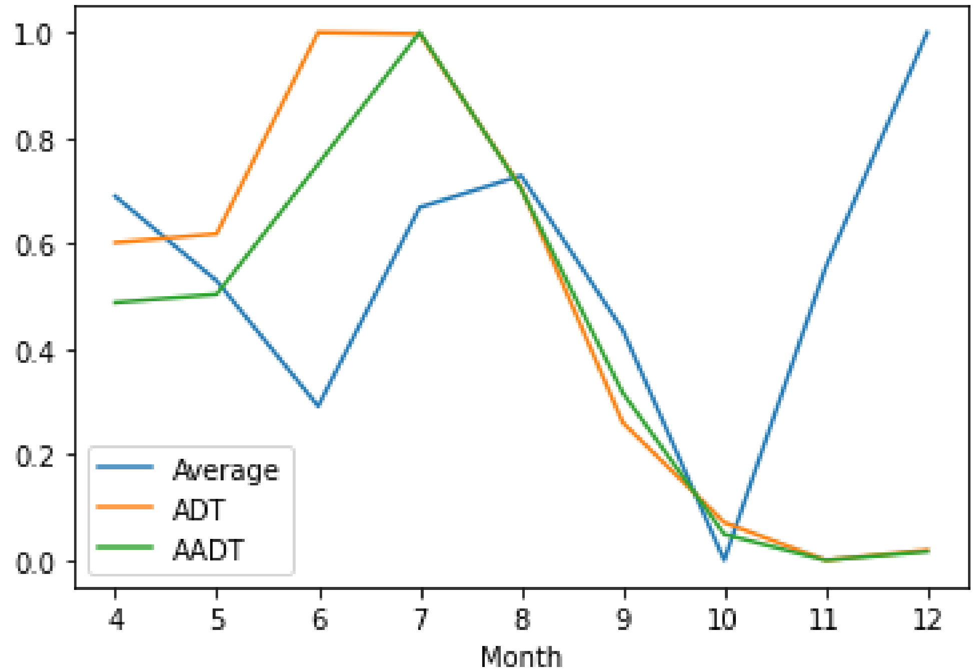
```
['Average', 'ADT', 'AADT']
```

[9]:

Month	HIGHWAY	SECTION	SECTION LENGTH	ADT	AADT	Average
4	213.900000	36.100000	5.921000	0.601755	0.488117	0.689652
5	120.608108	66.918919	7.076723	0.618436	0.503683	0.529338
6	141.291667	56.937500	8.790854	1.000000	0.750396	0.291653
7	190.831858	32.734513	8.732389	0.996919	1.000000	0.668474
8	189.225352	49.070423	8.116507	0.705706	0.703298	0.728707
9	99.615385	70.169231	6.644262	0.260777	0.317203	0.436449
10	142.204082	29.306122	4.890633	0.071712	0.049255	0.000000
11	141.250000	90.166667	4.101750	0.000000	0.000000	0.555140
12	103.083333	47.916667	3.670000	0.019251	0.016470	1.000000

[10]:
```
MonthlyMergedData[['Average','ADT','AADT']].plot()
```

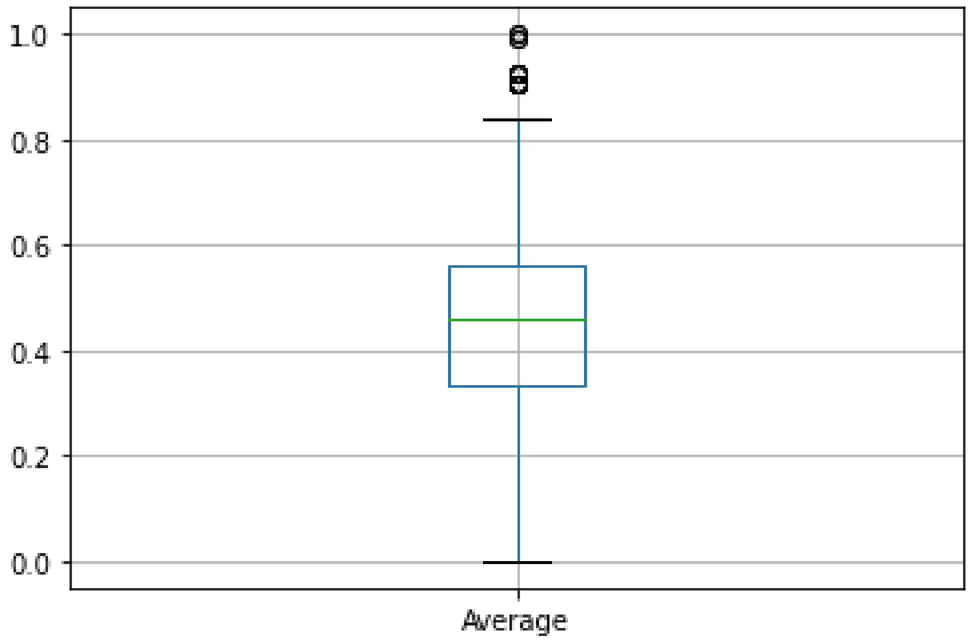
[10]: <AxesSubplot:xlabel='Month'>



Average distribution is symmetric with few outlier

[11]:
```
MergedData.boxplot(column=['Average'])
```

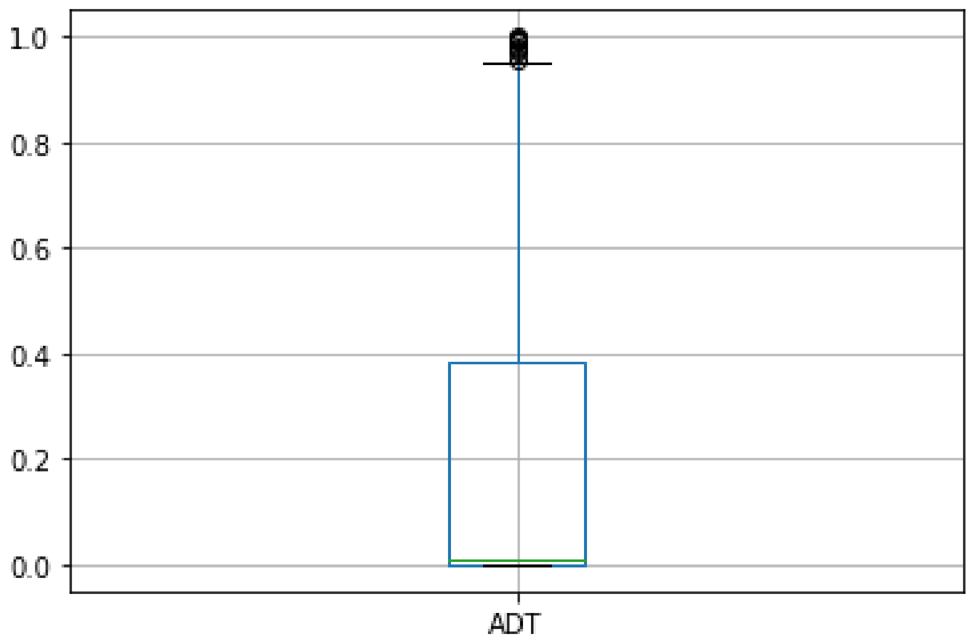
[11]: <AxesSubplot:>



ADT distribution is positively skewed with few outliers

[12]:
```
MergedData.boxplot(column=['ADT'])
```

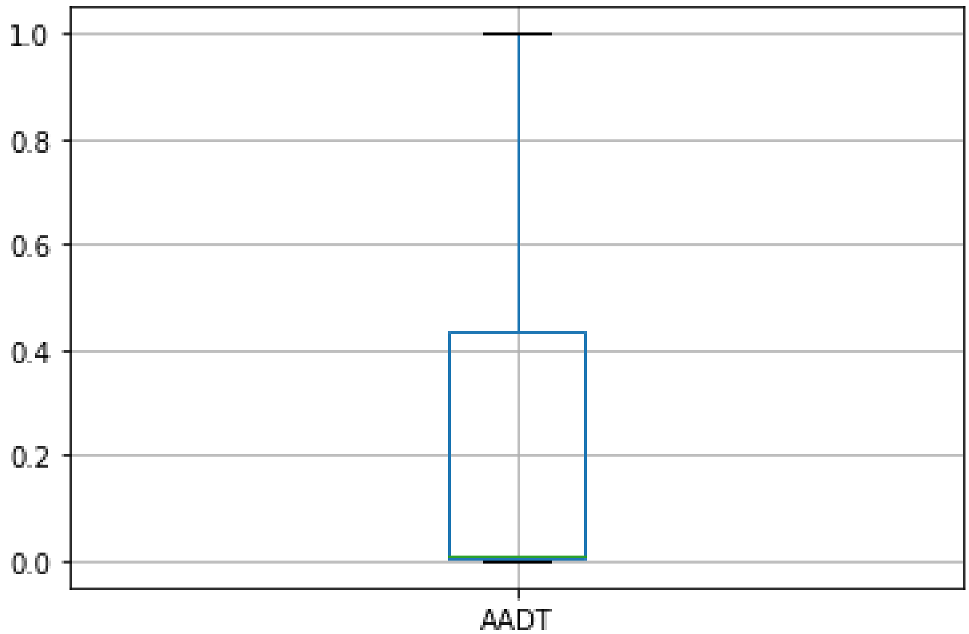
[12]: <AxesSubplot:>



AADT distribution is positively skewed

[13]:
```
MergedData.boxplot(column=['AADT'])
```

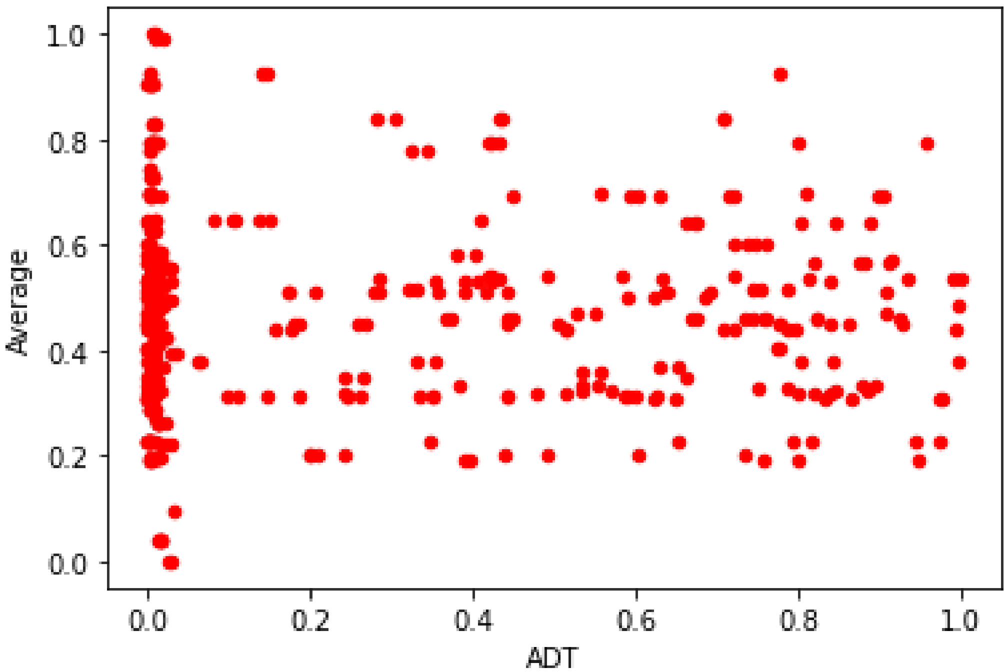
[13]: <AxesSubplot:>



ADT and Average on compare is non-linear

[14]:
```
MergedData.plot.scatter(x='ADT',
                y='Average',
                c='Red')
```

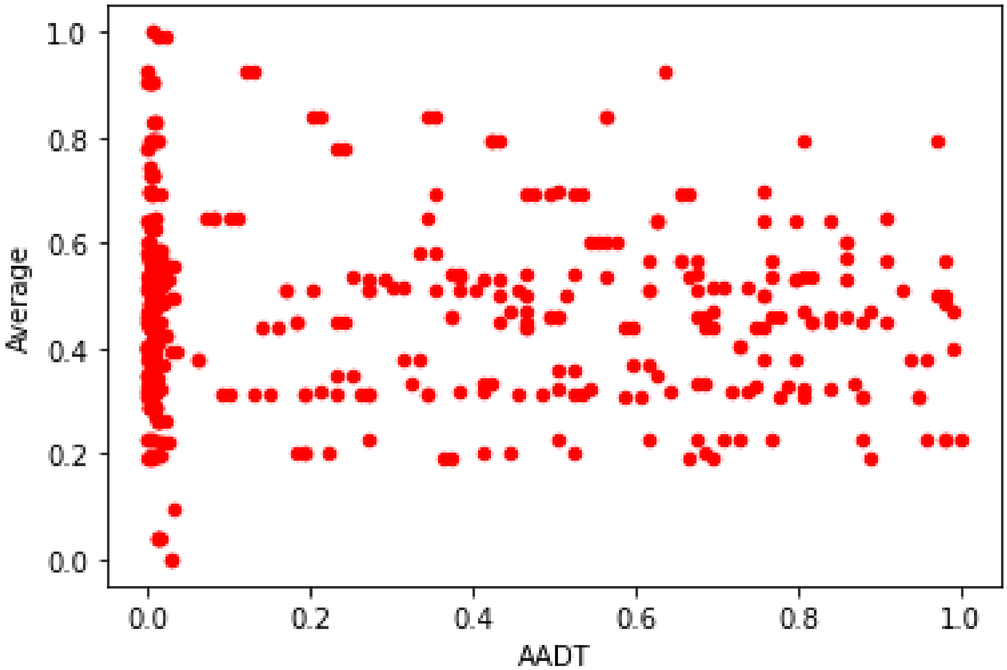
[14]: <AxesSubplot:xlabel='ADT', ylabel='Average'>



AADT and Average on compare is non-linear

[15]:
```
MergedData.plot.scatter(x='AADT',
                        y='Average',
                        c='Red')
```

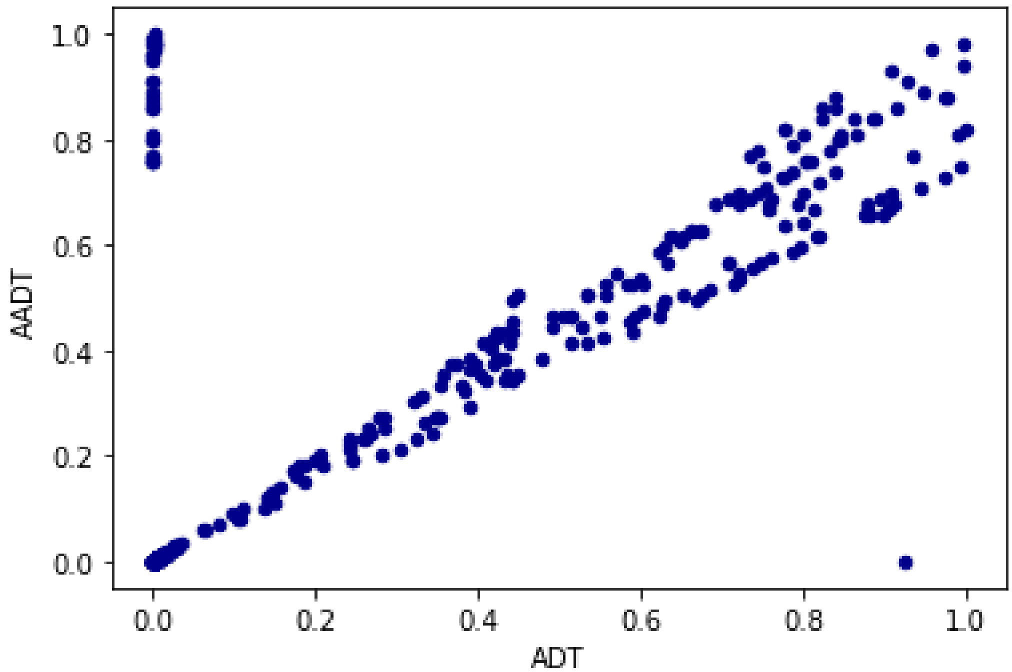
[15]: <AxesSubplot:xlabel='AADT', ylabel='Average'>



ADT and AADT on compare is linear, so we will consider only ADT into feature and will remove AADT as both are linear and taking both will not improve our model

[16]:
```
MergedData.plot.scatter(x='ADT',
                y='AADT',
                c='DarkBlue')
```

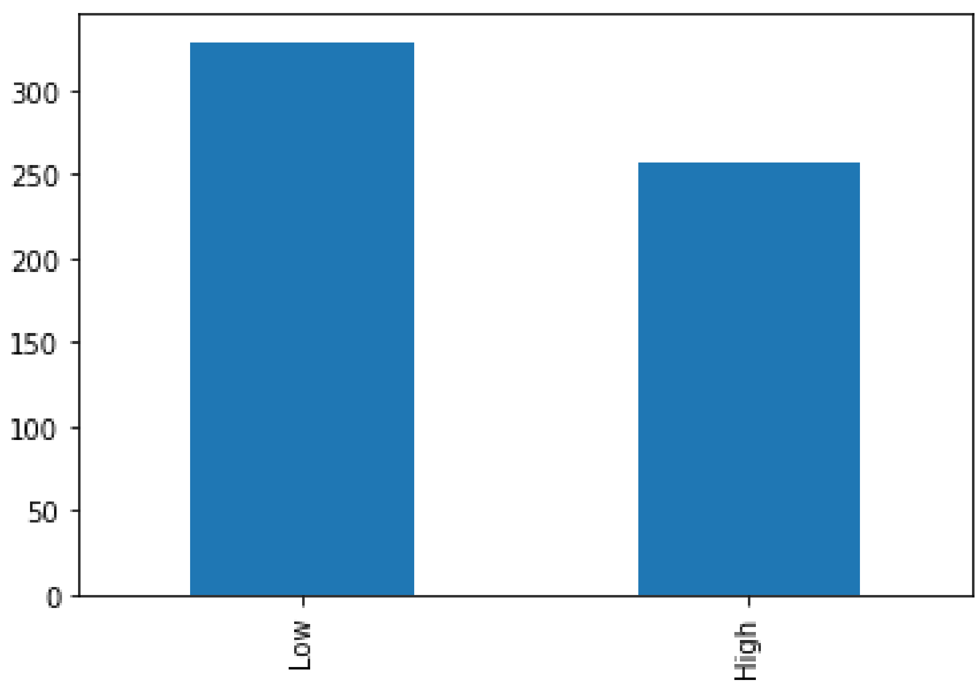
[16]: <AxesSubplot:xlabel='ADT', ylabel='AADT'>



Below is the bar graph for our Target attribute of model i.e. COIndex

[17]:
```
MergedData['COIndex'].value_counts().plot(kind='bar')
```

[17]: <AxesSubplot:>



[18]:
```
MergedData = MergedData[['HIGHWAY','SECTION','SECTION LENGTH','ADT','COIndex']]
```

Below shows Summary Visualization of Data

[19]:
```
MergedData.describe(include = 'all')
```

[19]:

	HIGHWAY	SECTION	SECTION LENGTH	ADT	COIndex
count	586.000000	586.000000	586.000000	586.000000	586
unique	NaN	NaN	NaN	NaN	2
top	NaN	NaN	NaN	NaN	Low
freq	NaN	NaN	NaN	NaN	329
mean	148.576792	52.779863	7.401891	0.197231	NaN
std	125.552938	56.718809	3.981491	0.306120	NaN
min	1.000000	1.000000	0.200000	0.000000	NaN
25%	7.000000	17.000000	4.150000	0.001846	NaN
50%	104.000000	30.000000	7.253500	0.007201	NaN
75%	245.000000	60.000000	10.040000	0.382794	NaN
max	376.000000	270.000000	20.720000	1.000000	NaN

[20]:
```
MergedData['COIndex'].value_counts(normalize=True)
```

[20]:
```
Low     0.561433
High    0.438567
Name: COIndex, dtype: float64
```

Reason for removing colums from our Final Data Date & Time- removed because our model cannot understand date and time values. **Pollutant-** removed because in data it is unique as shown below. **Unit-** removed because in data it is unique as shown below.

[21]:
```
AllCOData.describe(include='all')
```

[21]:

	Date & Time	Pollutant	Unit	Station	Average
count	236687	236687	236687	236687	205269.000000
unique	236687	1	1	2	NaN
top	01/01/2019 12:00:00 AM	CO	ppm	Halifax	NaN
freq	1	236687	236687	210384	NaN
mean	NaN	NaN	NaN	NaN	0.358628
std	NaN	NaN	NaN	NaN	0.322583
min	NaN	NaN	NaN	NaN	0.000000
25%	NaN	NaN	NaN	NaN	0.130000
50%	NaN	NaN	NaN	NaN	0.270000
75%	NaN	NaN	NaN	NaN	0.500000
max	NaN	NaN	NaN	NaN	11.020000

We removed station from final data because for year 2019 its unique as shown below

[22]:
```
AllCOData[(AllCOData["Date & Time"]>='01/01/2019 12:00:00 AM' )&␣
↪(AllCOData["Date & Time"]<'01/01/2020 12:00:00 AM')]['Station'].nunique()
```

[22]: 1

We removed AADT from the feature list because ADT and ADDT shows linear relationship as shown in below graph so taking both will not improve accuracy of model hence we can drop AADT

please refer scatter plot bewtween ADT and AADT plotted above which shoes both are linear

Creating model after splitting data 50% train and 50% test

[23]:
```
MergedData
```

[23]:

```
     HIGHWAY  SECTION  SECTION LENGTH       ADT COIndex
0          1       47            4.50  0.001571    High
1          1       50            7.60  0.003282    High
2          1       50            7.60  0.002948    High
3          1       50            7.60  0.001932    High
4          1       50            7.60  0.005191    High
..       …        …              …       …       …
581      374       28            6.83  0.241446     Low
582      374       30           11.04  0.489938     Low
583      376       10            5.68  0.000407    High
584      376       20            5.96  0.001218     Low
585      376       30            4.76  0.003895     Low

[586 rows x 5 columns]
```

[24]:
```
MergedData.dtypes
```

[24]:
```
HIGHWAY             int64
SECTION             int64
SECTION LENGTH    float64
ADT               float64
COIndex            object
dtype: object
```

[25]:
```
X = MergedData.values[:,0:4]
Y = MergedData.values[:,-1]
```

[26]:
```
X_train, X_test, y_train, y_test = train_test_split(
X,Y, test_size = 0.5, random_state = 100)
```

[27]:
```
clf_tree = DecisionTreeClassifier()
clf_tree.fit(X_train,y_train)
```

[27]: DecisionTreeClassifier()

[28]:
```
y_trainpred = clf_tree.predict(X_train)
```

[29]:
```
print("Confusion Matrix fot raining data: \n",␣
↪confusion_matrix(y_train,y_trainpred))
```

```
Confusion Matrix fot raining data:
 [[132   0]
 [  0 161]]
```

[30]:
```
print ("Accuracy of training data: \n",
accuracy_score(y_train,y_trainpred)*100)
```

```
Accuracy of training data:
 100.0
```

[31]:
```
y_pred = clf_tree.predict(X_test)
```

[32]:
```
print("Confusion Matrix: \n", confusion_matrix(y_test,y_pred))
```

```
Confusion Matrix:
 [[ 76  49]
 [ 44 124]]
```

[33]:
```
conmat = confusion_matrix(y_test,y_pred)
```

[34]:
```
TP = conmat[0][0]
FN = conmat[0][1]
FP = conmat[1][0]
TN = conmat[1][1]
```

[35]:
```
Accuracy = (TP + TN)/(TP+FN+FP+TN)
Accuracy
```

[35]: 0.6825938566552902

[36]:
```
Precision = TP/(TP+FP)
Precision
```

[36]: 0.6333333333333333

[37]:
```
Recall = TP/(TP+FN)
Recall
```

[37]: 0.608

[38]:
```
F1_measure = (2*(Recall)*(Precision))/(Recall+Precision)
F1_measure
```

[38]: 0.620408163265306

[39]:
```
print ("Accuracy of testdata: \n",
accuracy_score(y_test,y_pred)*100)
```

```
Accuracy of testdata:
 68.25938566552901
```

[40]:
```
print("Report : \n",
   classification_report(y_test, y_pred))
```

```
Report :
               precision    recall  f1-score   support

        High       0.63      0.61      0.62       125
         Low       0.72      0.74      0.73       168

    accuracy                           0.68       293
   macro avg       0.68      0.67      0.67       293
weighted avg       0.68      0.68      0.68       293
```

[41]:
```
clf_tree.get_n_leaves()
```

[41]: 78

Creating model using 10-fold cross-validation technique

Creating simple 10-fold cross validation without shuffling

[42]:
```
# 10 - fold
model = DecisionTreeClassifier()
scores = cross_val_score(model, X, Y,cv=10,scoring='accuracy')
scores
```

[42]:
```
array([0.44067797, 0.55932203, 0.42372881, 0.23728814, 0.61016949,
       0.59322034, 0.70689655, 0.43103448, 0.53448276, 0.5862069 ])
```

[43]:
```
print("%0.2f accuracy with a standard deviation of %0.2f" % (scores.mean(),␣
 ↪scores.std()))
```

```
0.51 accuracy with a standard deviation of 0.13
```

Creating simple 10-fold cross validation with shuffling

[44]:
```
#shuffled 10-fold
model = DecisionTreeClassifier()
cv = ShuffleSplit(n_splits=10, random_state=0)
scores = cross_val_score(model, X, Y,cv=cv, scoring='accuracy')
scores
```

[44]:
```
array([0.66101695, 0.69491525, 0.81355932, 0.74576271, 0.71186441,
       0.76271186, 0.86440678, 0.81355932, 0.79661017, 0.77966102])
```

[45]:
```
print("%0.2f accuracy with a standard deviation of %0.2f" % (scores.mean(),␣
↪scores.std()))
```

```
0.76 accuracy with a standard deviation of 0.06
```

Creating confusion matrix and classification report for each fold in 10-fold cross validation

[46]:
```
conf_matrix_list = []
kf = KFold(n_splits=10)
for train_index, test_index in kf.split(X):
    Xtrain, Xtest = X[train_index], X[test_index]
    ytrain, ytest = Y[train_index], Y[test_index]

    clf_tree.fit(Xtrain, ytrain)

    ypred = clf_tree.predict(Xtest)
    conf_matrix = confusion_matrix(ytest, ypred)
    conf_matrix_list .append(conf_matrix)
    print("Report : \n",classification_report(ytest, ypred))
```

```
Report :
               precision    recall  f1-score   support

        High       0.39      0.23      0.29        31
         Low       0.41      0.61      0.49        28

    accuracy                           0.41        59
   macro avg       0.40      0.42      0.39        59
weighted avg       0.40      0.41      0.38        59

Report :
               precision    recall  f1-score   support

        High       0.24      0.38      0.29        21
         Low       0.48      0.32      0.38        38

    accuracy                           0.34        59
   macro avg       0.36      0.35      0.34        59
weighted avg       0.39      0.34      0.35        59

Report :
               precision    recall  f1-score   support

        High       0.14      0.07      0.10        28
         Low       0.42      0.61      0.50        31

    accuracy                           0.36        59
   macro avg       0.28      0.34      0.30        59
```

```
weighted avg       0.29      0.36      0.31        59

Report :
               precision    recall  f1-score   support

        High       0.33      0.31      0.32        26
         Low       0.49      0.52      0.50        33

    accuracy                           0.42        59
   macro avg       0.41      0.41      0.41        59
weighted avg       0.42      0.42      0.42        59

Report :
               precision    recall  f1-score   support

        High       0.43      0.50      0.46        26
         Low       0.55      0.48      0.52        33

    accuracy                           0.49        59
   macro avg       0.49      0.49      0.49        59
weighted avg       0.50      0.49      0.49        59

Report :
               precision    recall  f1-score   support

        High       0.21      0.35      0.27        17
         Low       0.65      0.48      0.55        42

    accuracy                           0.44        59
   macro avg       0.43      0.41      0.41        59
weighted avg       0.52      0.44      0.47        59

Report :
               precision    recall  f1-score   support

        High       0.67      0.36      0.47        28
         Low       0.58      0.83      0.68        30

    accuracy                           0.60        58
   macro avg       0.62      0.60      0.58        58
weighted avg       0.62      0.60      0.58        58

Report :
               precision    recall  f1-score   support

        High       0.54      0.38      0.44        37
         Low       0.28      0.43      0.34        21
```

```
    accuracy                           0.40        58
   macro avg       0.41      0.40      0.39        58
weighted avg       0.45      0.40      0.41        58

Report :
               precision    recall  f1-score   support

        High       0.43      1.00      0.60        25
         Low       0.00      0.00      0.00        33

    accuracy                           0.43        58
   macro avg       0.22      0.50      0.30        58
weighted avg       0.19      0.43      0.26        58

Report :
               precision    recall  f1-score   support

        High       0.20      0.17      0.18        18
         Low       0.65      0.70      0.67        40

    accuracy                           0.53        58
   macro avg       0.43      0.43      0.43        58
weighted avg       0.51      0.53      0.52        58

C:\Users\Mayank\AppData\Local\Programs\Python\Python38\lib\site-
packages\sklearn\metrics\_classification.py:1308: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Mayank\AppData\Local\Programs\Python\Python38\lib\site-
packages\sklearn\metrics\_classification.py:1308: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Mayank\AppData\Local\Programs\Python\Python38\lib\site-
packages\sklearn\metrics\_classification.py:1308: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

[47]:
```
conf_matrix_list
```

[47]:
```
[array([[ 7, 24],
        [11, 17]], dtype=int64),
 array([[ 8, 13],
        [26, 12]], dtype=int64),
 array([[ 2, 26],
```

```
        [12, 19]], dtype=int64),
 array([[ 8, 18],
        [16, 17]], dtype=int64),
 array([[13, 13],
        [17, 16]], dtype=int64),
 array([[ 6, 11],
        [22, 20]], dtype=int64),
 array([[10, 18],
        [ 5, 25]], dtype=int64),
 array([[14, 23],
        [12,  9]], dtype=int64),
 array([[25,  0],
        [33,  0]], dtype=int64),
 array([[ 3, 15],
        [12, 28]], dtype=int64)]
```

mean of all 10 confusion matrices

[48]:
```
mean_of_conf_matrix_arrays = np.mean(conf_matrix_list, axis=0)
mean_of_conf_matrix_arrays
```

[48]:
```
array([[ 9.6, 16.1],
       [16.6, 16.3]])
```

Finding True positive(TP), False Negative(FN), False Positive(FP), True Negative(TN)

[49]:
```
TP = mean_of_conf_matrix_arrays[0][0]
FN = mean_of_conf_matrix_arrays[0][1]
FP = mean_of_conf_matrix_arrays[1][0]
TN = mean_of_conf_matrix_arrays[1][1]
```

Finding of Accuracy based on mean of Confusion matrix

[50]:
```
Accuracy = (TP + TN)/(TP+FN+FP+TN)
Accuracy
```

[50]:
```
0.44197952218430026
```

Finding of Precision based on mean of Confusion matrix

[51]:
```
Precision = TP/(TP+FP)
Precision
```

[51]:
```
0.36641221374045796
```

Finding of Recall based on mean of Confusion matrix

[52]:
```
Recall = TP/(TP+FN)
Recall
```

[52]: 0.37354085603112835

Finding of F1-measure based on Recall and Precision

[53]:
```
F1_measure = (2*(Recall)*(Precision))/(Recall+Precision)
F1_measure
```

[53]: 0.36994219653179183

What is the most influential factor for COlevel? Why?

[54]:
```
demo = MergedData.drop('COIndex',axis = 1)
pd.Series(clf_tree.feature_importances_ , index = demo.columns)
```

[54]:
```
HIGHWAY           0.294408
SECTION           0.167913
SECTION LENGTH    0.171078
ADT               0.366600
dtype: float64
```

Hyperparameter tuning

max_depth

decresing the max_depth will help tackling overfitting of data as we faced with default decision_tree_classifier

[55]:
```
clf_exptree1 = DecisionTreeClassifier(max_depth=8)
clf_exptree1.fit(X_train,y_train)
```

[55]: DecisionTreeClassifier(max_depth=8)

[56]:
```
y_exppred = clf_exptree1.predict(X_train)
print("Confusion Matrix: \n", confusion_matrix(y_train,y_exppred))
print("Report : \n",
   classification_report(y_train, y_exppred))
```

```
Confusion Matrix:
 [[ 85  47]
 [ 10 151]]
Report :
               precision    recall  f1-score   support

        High       0.89      0.64      0.75       132
         Low       0.76      0.94      0.84       161

    accuracy                           0.81       293
   macro avg       0.83      0.79      0.80       293
weighted avg       0.82      0.81      0.80       293
```

[57]:
```
y_exppred = clf_exptree1.predict(X_test)
print("Confusion Matrix: \n", confusion_matrix(y_test,y_exppred))
print("Report : \n",
   classification_report(y_test, y_exppred))
```

```
Confusion Matrix:
 [[ 51  74]
 [ 30 138]]
Report :
               precision    recall  f1-score   support

        High       0.63      0.41      0.50       125
         Low       0.65      0.82      0.73       168

    accuracy                           0.65       293
   macro avg       0.64      0.61      0.61       293
weighted avg       0.64      0.65      0.63       293
```

But if we give max_depth very low our model will go into underfitting as shown below

[58]:
```
clf_exptree1 = DecisionTreeClassifier(max_depth=2)
clf_exptree1.fit(X_train,y_train)
```

[58]: DecisionTreeClassifier(max_depth=2)

[59]:
```
y_exppred = clf_exptree1.predict(X_train)
print("Confusion Matrix: \n", confusion_matrix(y_train,y_exppred))
print("Report : \n",
   classification_report(y_train, y_exppred))
```

```
Confusion Matrix:
 [[ 10 122]
 [  0 161]]
Report :
               precision    recall  f1-score   support

        High       1.00      0.08      0.14       132
         Low       0.57      1.00      0.73       161

    accuracy                           0.58       293
   macro avg       0.78      0.54      0.43       293
weighted avg       0.76      0.58      0.46       293
```

[60]:
```
y_exppred = clf_exptree1.predict(X_test)
print("Confusion Matrix: \n", confusion_matrix(y_test,y_exppred))
print("Report : \n",
   classification_report(y_test, y_exppred))
```

```
Confusion Matrix:
 [[  5 120]
 [  1 167]]
Report :
               precision    recall  f1-score   support

        High       0.83      0.04      0.08       125
         Low       0.58      0.99      0.73       168

    accuracy                           0.59       293
   macro avg       0.71      0.52      0.41       293
weighted avg       0.69      0.59      0.45       293
```

min_samples_split

Increasing min_samples_split can help reducing overfitting in model as shown below

[61]:
```
clf_exptree2 = DecisionTreeClassifier(min_samples_split=9)
clf_exptree2.fit(X_train,y_train)
```

[61]:
```
DecisionTreeClassifier(min_samples_split=9)
```

[62]:
```
y_exppred = clf_exptree2.predict(X_train)
print("Confusion Matrix: \n", confusion_matrix(y_train,y_exppred))
print("Report : \n",
   classification_report(y_train, y_exppred))
```

```
Confusion Matrix:
 [[119  13]
 [ 20 141]]
Report :
               precision    recall  f1-score   support

        High       0.86      0.90      0.88       132
         Low       0.92      0.88      0.90       161

    accuracy                           0.89       293
   macro avg       0.89      0.89      0.89       293
weighted avg       0.89      0.89      0.89       293
```

[63]:
```
y_exppred = clf_exptree2.predict(X_test)
print("Confusion Matrix: \n", confusion_matrix(y_test,y_exppred))
print("Report : \n",
   classification_report(y_test, y_exppred))
```

```
Confusion Matrix:
 [[ 74  51]
 [ 55 113]]
```

```
Report :
               precision    recall  f1-score   support

        High       0.57      0.59      0.58       125
         Low       0.69      0.67      0.68       168

    accuracy                           0.64       293
   macro avg       0.63      0.63      0.63       293
weighted avg       0.64      0.64      0.64       293
```

But too high values for min_samples_split can lead to underfitting as shown below

[64]:
```
clf_exptree2 = DecisionTreeClassifier(min_samples_split=50)
clf_exptree2.fit(X_train,y_train)
```

[64]:
```
DecisionTreeClassifier(min_samples_split=50)
```

[65]:
```
y_exppred = clf_exptree2.predict(X_train)
print("Confusion Matrix: \n", confusion_matrix(y_train,y_exppred))
print("Report : \n",
   classification_report(y_train, y_exppred))
```

```
Confusion Matrix:
 [[113  19]
 [ 57 104]]
Report :
               precision    recall  f1-score   support

        High       0.66      0.86      0.75       132
         Low       0.85      0.65      0.73       161

    accuracy                           0.74       293
   macro avg       0.76      0.75      0.74       293
weighted avg       0.76      0.74      0.74       293
```

[66]:
```
y_exppred = clf_exptree2.predict(X_test)
print("Confusion Matrix: \n", confusion_matrix(y_test,y_exppred))
print("Report : \n",
   classification_report(y_test, y_exppred))
```

```
Confusion Matrix:
 [[83 42]
 [74 94]]
Report :
               precision    recall  f1-score   support

        High       0.53      0.66      0.59       125
```

```
         Low       0.69      0.56      0.62       168

    accuracy                           0.60       293
   macro avg       0.61      0.61      0.60       293
weighted avg       0.62      0.60      0.61       293
```

min_samples_leaf

by increasing min_samples_leaf we can overcome overfitting of model as shown below

[67]:
```
clf_exptree3 = DecisionTreeClassifier(min_samples_leaf=10)
clf_exptree3.fit(X_train,y_train)
```

[67]:
```
DecisionTreeClassifier(min_samples_leaf=10)
```

[68]:
```
y_exppred = clf_exptree3.predict(X_train)
print("Confusion Matrix: \n", confusion_matrix(y_train,y_exppred))
print("Report : \n",
   classification_report(y_train, y_exppred))
```

```
Confusion Matrix:
 [[ 90  42]
 [ 21 140]]
Report :
               precision    recall  f1-score   support

        High       0.81      0.68      0.74       132
         Low       0.77      0.87      0.82       161

    accuracy                           0.78       293
   macro avg       0.79      0.78      0.78       293
weighted avg       0.79      0.78      0.78       293
```

[69]:
```
y_exppred = clf_exptree3.predict(X_test)
print("Confusion Matrix: \n", confusion_matrix(y_test,y_exppred))
print("Report : \n",
   classification_report(y_test, y_exppred))
```

```
Confusion Matrix:
 [[ 59  66]
 [ 40 128]]
Report :
               precision    recall  f1-score   support

        High       0.60      0.47      0.53       125
         Low       0.66      0.76      0.71       168

    accuracy                           0.64       293
```

```
   macro avg       0.63      0.62      0.62       293
weighted avg       0.63      0.64      0.63       293
```

But too high values for min_samples_leaf can lead to underfitting as shown below

[70]:
```
clf_exptree3 = DecisionTreeClassifier(min_samples_leaf=50)
clf_exptree3.fit(X_train,y_train)
```

[70]:
```
DecisionTreeClassifier(min_samples_leaf=50)
```

[71]:
```
y_exppred = clf_exptree3.predict(X_train)
print("Confusion Matrix: \n", confusion_matrix(y_train,y_exppred))
print("Report : \n",
   classification_report(y_train, y_exppred))
```

```
Confusion Matrix:
 [[ 43  89]
 [ 27 134]]
Report :
               precision    recall  f1-score   support

        High       0.61      0.33      0.43       132
         Low       0.60      0.83      0.70       161

    accuracy                           0.60       293
   macro avg       0.61      0.58      0.56       293
weighted avg       0.61      0.60      0.58       293
```

[72]:
```
y_exppred = clf_exptree3.predict(X_test)
print("Confusion Matrix: \n", confusion_matrix(y_test,y_exppred))
print("Report : \n",
   classification_report(y_test, y_exppred))
```

```
Confusion Matrix:
 [[ 33  92]
 [ 38 130]]
Report :
               precision    recall  f1-score   support

        High       0.46      0.26      0.34       125
         Low       0.59      0.77      0.67       168

    accuracy                           0.56       293
   macro avg       0.53      0.52      0.50       293
weighted avg       0.53      0.56      0.53       293
```