ASSIGNMENT 1

CSCI 6561 – FALL 2020

Submited by:

Abhinav Mandava (B00841453)
Anqi Chen (B00838586)
Ahsan Kamal (B00853723)
Raj Kumar Reddy Gangi (B00849566)
Ram Prasath Meganathan (B00851418)

Contents

1. Finding resources

1.1. The List of Weather Resources

1) The Weather Network [1]
   Pros: has hourly, daily, weekly, monthly data
   Cons: a lot of Ads, API available when signed up for a 30-day trial
   API used: Weather Source [2]

2) Weather Underground [3]
   Cons: weather conditions in details
   Pros: a lot of Ads, hard to download

3) Dark Sky [4]
   Pros: user-friendly UI
   Cons: API is no longer accepting new signups

4) Weather Spark [5]
   Pros: detailed data with statistic visualization graphs
   Cons: hard to download

5) The National Weather Service/NOAA [6]
   Pros: various dataset with detailed weather elements, official site
   Cons: only offers weather data inside the US

6) Weather Canada [7]
   Pros: easy to download (CSV, XML, txt), official hourly data with different weather elements, data from several stations in one city can easily be found
   Cons: only pure data without any visualization

7) NS Weather Station Data [8]
   Pros: official data, easy to download (CSV, XML, APIs), many columns in this dataset
   Cons: period is 2011-07-01 to 2019-05-01 (only enough for our analysis)

1.2. Comparison

We need hourly weather data of Halifax to analyze the possible factors triggering headaches. Considering the data format and accessibility, API is a better format to be used in the web development rather than data analysis. Besides, most of the APIs are not free for download. Also, official websites are more reliable for consistency and stability. As a result, we chose the **Climate Weather dataset** from **Government Canada** (#6) as the definitive data source.

2. Pre-processing Data

Raw data is collected from the weather websites for all locations (five sources) related to Halifax. It is then filtered to remove parameters that will not value training the machine learning

model. The initial dataset count came up to approximately 3000 records for four months since we have considered the training data and testing data on an hourly basis. After filtration, we eliminated a few parameters and identified Temperature, Dew Point Temperature, Relative Humidity, Wind Speed, Visibility, and Station Pressure as input features to be fed to the model. All the collected datasets were combined into a single dataset through a python program. The program took the weather data for all the locations as input and calculated the mean value for all of them. Date and time were kept as the constant value to identify the weather information uniquely for each day. After weather data is consolidated, the student's work and workout data are added as additional features. They are represented as binary values to facilitate the training and prediction of the machine learning model. Since the student's schedule and workout schedule are given hourly, we have consolidated the weather data in hours. Each occurrence of an event is mapped as '0' and non-occurrence as '1'. After transforming the data, the transformed dataset ready to be fed into the machine learning model. The dataset used while making the actual predictions will not have a Headache column since that is the determined output parameter.

Temperature	Dew Point Temp	Rel Hum	Wind Spd	Visibility	Stn Press	Work	Workout	Headache
16.675	11.075	69.5	11	24.1	101.365	0	0	0
16.1	11.375	73.5	10.25	24.1	101.43	0	0	0
16	11.125	73	10.75	24.1	101.4925	0	0	0
15.475	11.5	77.25	12.75	24.1	101.54	0	0	0

Figure 1 Screenshot of the transformed Dataset used as input

2.1. Descriptive analysis

With Halifax's climate data at hand, we need some initial statistics to be able to get a high-level understanding of Halifax's climate in the given duration (Sep1, 2020 - December 30, 2020).

	Temperature	Dew Point Temp	Rel Hum	Wind Spd	Visibility	Stn Press
count	2928.000000	2928.000000	2928.000000	2928.000000	2925.000000	2928.000000
mean	7.030659	3.289981	78.579816	16.259882	20.768205	100.594850
std	7.433787	7.607795	14.129073	8.795470	6.967021	1.048191
min	-11.060000	-16.380000	33.200000	1.000000	0.200000	95.640000
25%	0.920000	-1.820000	69.600000	9.800000	24.100000	99.981500
50%	7.720000	3.770000	80.600000	14.675000	24.100000	100.742500
75%	12.742500	9.220000	90.600000	20.800000	24.100000	101.359000
max	23.750000	20.150000	99.000000	67.000000	32.200000	103.038000

Figure 2 Description of the selected features

For the given duration (Sep1, 2020 - December 30, 2020), the probability density curves for the key climatic factors are as follows:

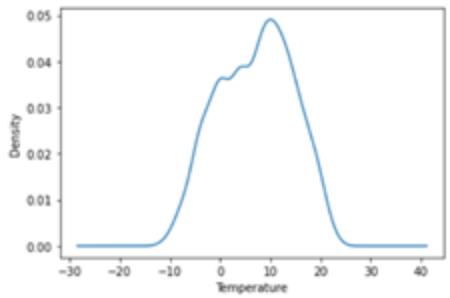

Figure 3 Probability density of 'Temperature'

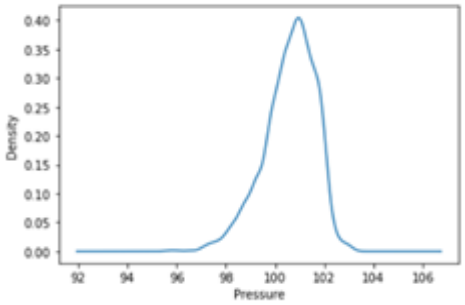

Figure 4 Probability density of 'Pressure'

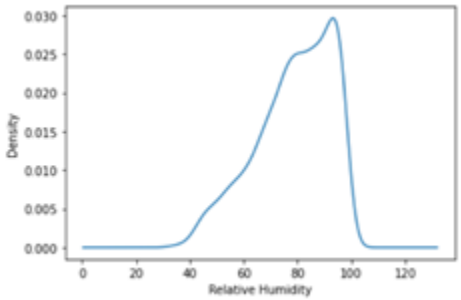

Figure 5 Probability density of 'relative Humidity'

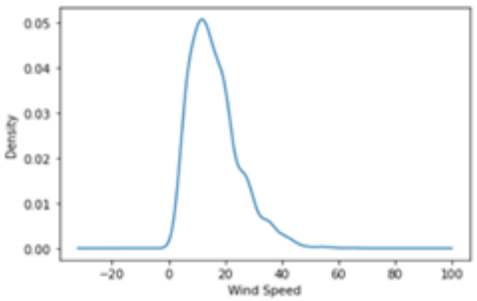

Figure 6 Probability density of "Wind Speed

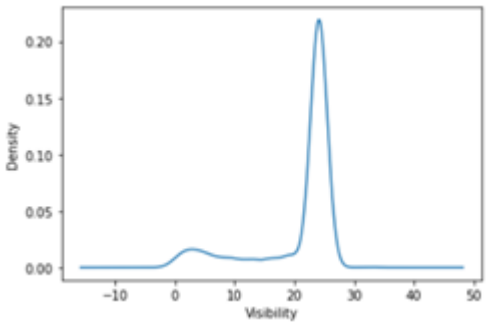

Figure 7 Probability density of 'Visibility'

3. Feature Selection

After researching the possible causes of Headache in online sources, we have selected the necessary features and used it to determine if the feature is necessary for the prediction. Case studies have suggested that the following factors could be the possible cause for the headaches [9]:

- Whether or barometric pressure changes: 73%
- Intense odours: 64%
- Bright or flickering lights: 59%
- Smoke: 53%
- Extreme heat or cold: 38%
- Altitude changes: 31%
- High winds: 18%

We have selected our determining features on relevance to the above factors.

The total number of features selected is eight. Each of these features, either individually or when combined with the others, can predict if the person gets a headache or not. This binary classification can be expressed as the mapping F: A1×A2…×AN→H, where A1, A2,…AN are the features/attributes considered (Temperature, Pressure, Humidity, etc.), and H is the target feature whose value is to be predicted, i.e., Headache. To approximate the function F, the language of the model we chose was a 'Decision Tree.'

While choosing a decision node at each level of the decision tree, we should determine which feature does the best job in splitting/deciding the target feature's values for the number of samples given at that level of the tree. There are two main criteria for choosing the feature as the decision node: *Gini Impurity* and *Entropy.*

Gini impurity of a feature measures the degree of impurity of the split when a particular feature is used for splitting the space. In other words, it measures how much impure the subspaces are after splitting the space with the given feature's values. The feature with the least Gini impurity value gets to be the next deciding node. Gini impurity can be calculated as follows:

$$Gini(E) = 1 - \sum_{j=1}^{c} p_j^2$$

E is the candidate feature for the decision node

pj is the probability of the class-j of the target feature.

Entropy is the degree of randomness in the subspace. Information gain can be calculated for a particular feature by measuring entropy change after splitting the space with the given feature. Entropy can be measured as follows:

$$Entropy: H(E) = -\sum_{j=1}^{c} p_j \log p_j$$

Information gain can be calculated by $Entropy_{before} - Entropy_{split}$

4. Model and Prediction

4.1. Most Important Factor

In our decision tree, we used Gini impurity as the criterion for choosing the decision nodes. The root node for our decision tree is ***Work*** with the least **Gini impurity** value of **0.5**. This implies that 'Work' has been the most important feature in our feature set to decide the status of the person's Headache.

4.2. Building a Predictive Model

We had very few samples of Has Headache class compared to No Headache class; one way to overcome this issue was to generate new samples of the under-represented class. We compared various oversampling techniques such as random oversampling, synthetic minority oversampling, and adaptive synthetic oversampling. Finally, we used the Adaptive synthetic

oversampling technique to generate new samples of minority class from the available samples better than others in the decision tree model evaluation [10].

We have also used **StratifiedKFold** to split the data; this ensures that test data and train data have the same proportion of each class [11]. We have used **the DecisionTreeClassifier from the sklearn; it uses the Gini criterion to measure the split's quality**. We didn't use the max_depth parameter so that the tree is expanded until all leaves are pure [12]. Fit the DecisionTreeClassifier with the training data and calculate the accuracy of the predictions; we got overall accuracy of 90.15%. Figure 8 shows the tree before pruning.

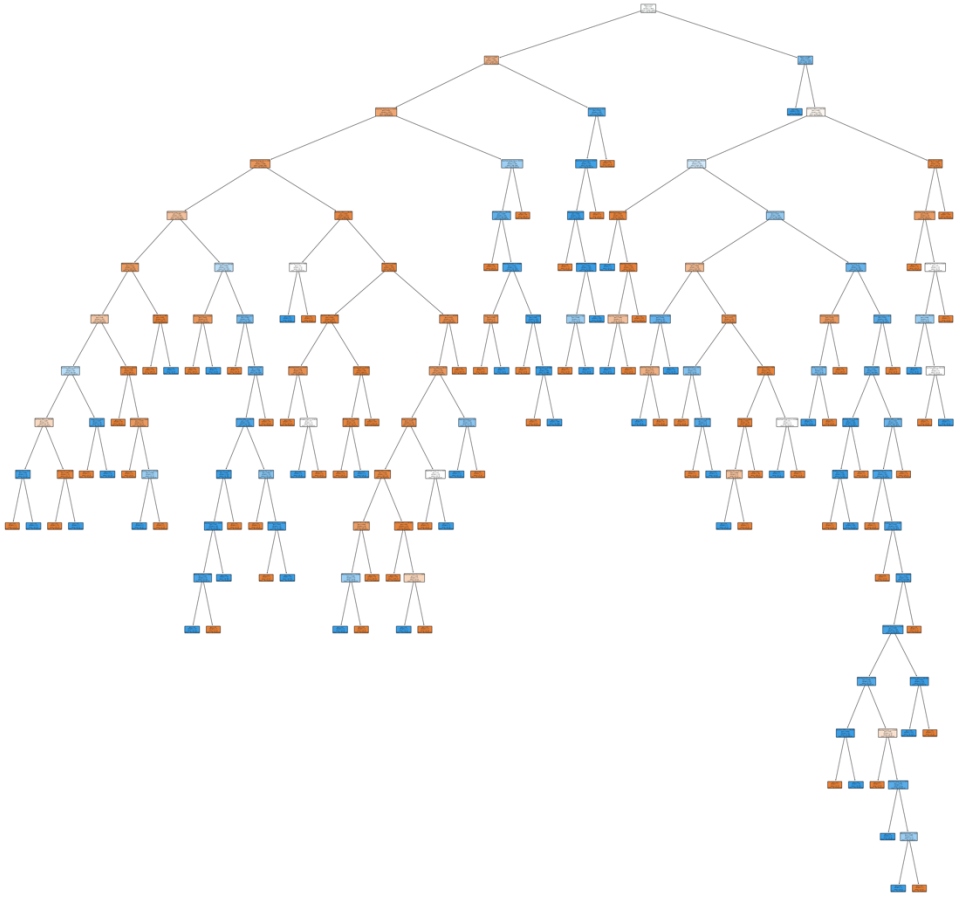

Figure 8 Decision tree before pruning

4.2.1. Pruning the Decision Tree

Cost_complexity_pruning_path is used to control the depth of the tree. This function returns the range of the alpha values. The alpha value is sensitive to the datasets. We used cross-validation to find the optimal value of alpha. For each value of the alpha run, five-fold cross-validation, calculate and store accuracy for each alpha. Then plot the graph of alphas with accuracy [13].

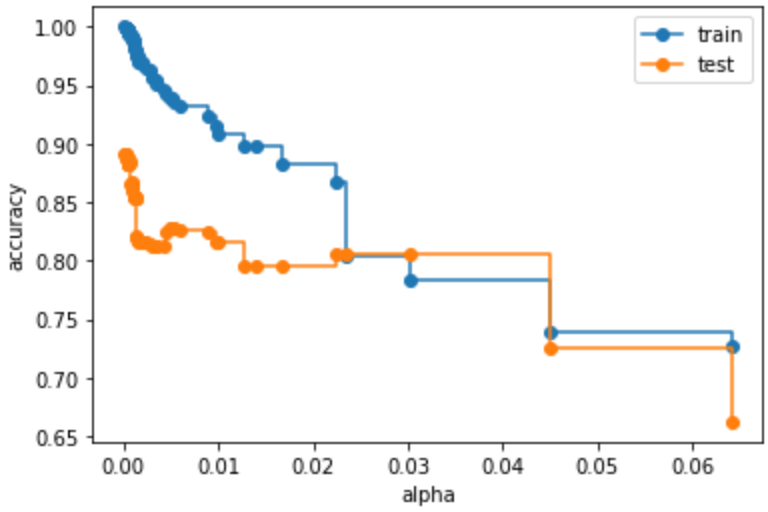

Figure 9 accuracy with different alpha values

From the above figure 9, it is seen that for alpha value 0.0225, we have the highest accuracy where testing and training datasets are performing well. Now we have calculated the model again with the alpha value of 0.0225.

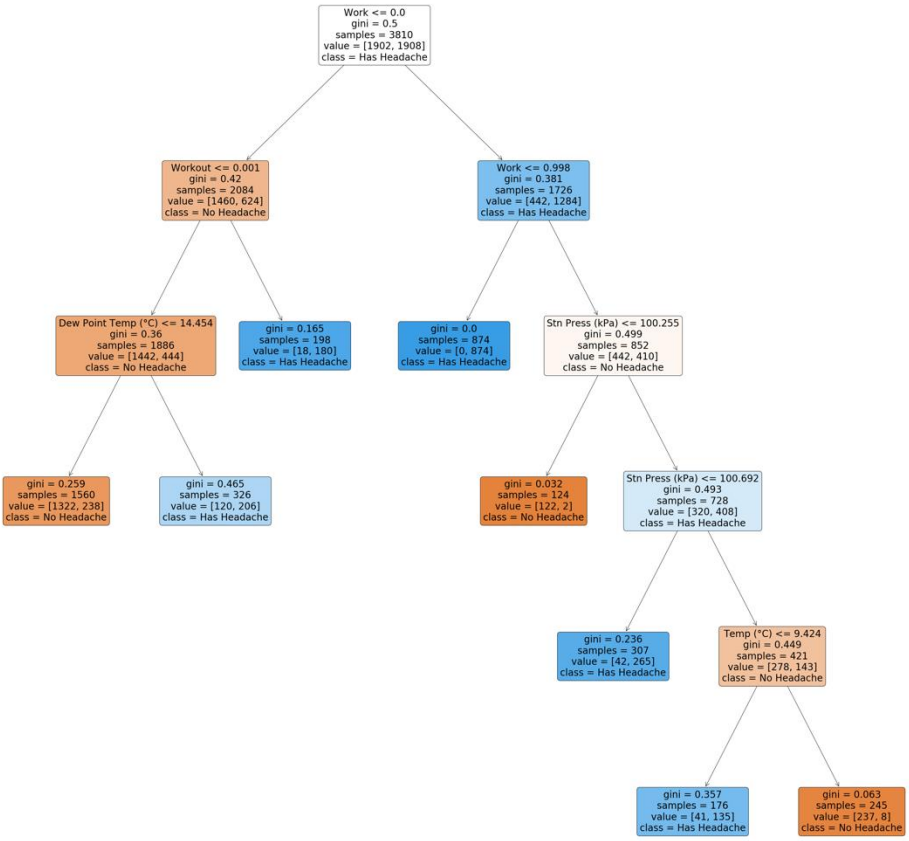

Figure 10 Decision tree after pruning

Figure 10 shows the decision tree after the pruning. This new model has an accuracy of 90%

4.3. Predictions and Suggestions

Based on the model we have built, we used Halifax's weather data from September 1, 2020, to October 2, 2020, to predict whether he will get a headache or not. From our prediction between the periods, the hours when his probability of Headache is more than 70% are shown (Table 1).

Table 1 Predicted time slots of having a headache

Year	Month	Day	Time	Headache probability
2020	9	17	13:00	0.855384615
2020	9	17	14:00	0.855384615
2020	9	17	15:00	0.855384615
2020	9	17	16:00	0.855384615
2020	9	17	17:00	0.855384615
2020	9	18	9:00	0.855384615
2020	9	18	10:00	0.855384615
2020	9	18	11:00	0.855384615
2020	9	18	12:00	0.855384615
2020	9	18	13:00	0.855384615
2020	9	18	14:00	0.855384615
2020	9	18	15:00	0.855384615
2020	9	18	16:00	0.855384615
2020	9	18	17:00	0.855384615
2020	9	22	9:00	0.855384615
2020	9	22	10:00	0.855384615
2020	9	22	11:00	0.855384615
2020	9	30	1:00	0.855384615
2020	9	30	2:00	0.855384615
2020	9	30	3:00	0.855384615
2020	10	1	9:00	0.855384615
2020	10	1	10:00	0.855384615
2020	10	1	11:00	0.855384615
2020	10	1	12:00	0.855384615
2020	10	1	13:00	0.855384615
2020	10	1	14:00	0.855384615
2020	10	1	15:00	0.855384615
2020	10	1	16:00	0.855384615
2020	10	1	17:00	0.855384615
2020	10	2	12:00	0.855384615
2020	10	2	13:00	0.855384615
2020	10	2	14:00	0.855384615
2020	10	2	15:00	0.855384615
2020	10	2	16:00	0.855384615
2020	10	2	17:00	0.855384615

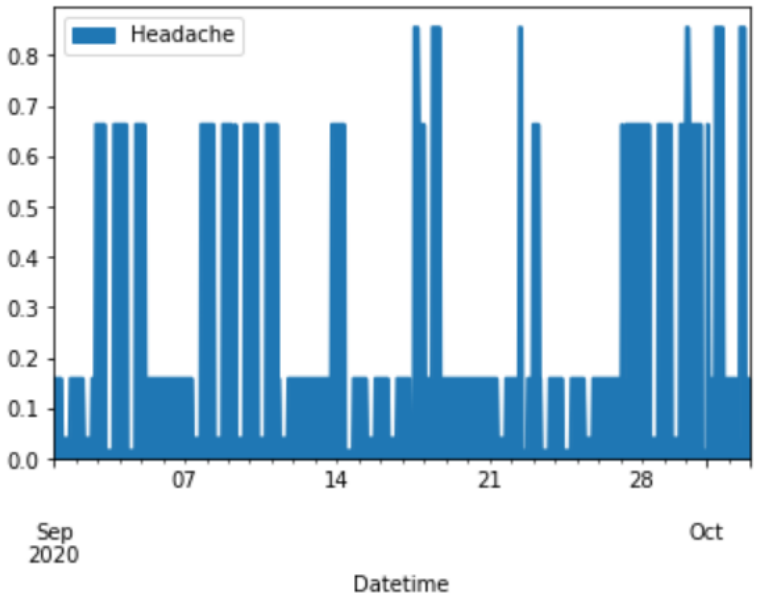

Figure 11 Probabilities of headache w.r.t datetime

The above figure 11 shows the probability of headaches from Sept 1 2020 to Oct 2 2020

For days that Headache's probability is more than 70%, our suggestion is trying not to go to Work at first. If he must go to Work, he also needs to check the weather forecast before going out. After knowing the most important factors triggering his headaches, if the Stn pressure is above 100.255, he'd better not going outside as the possibility of having during that time is very high.

Based on his symptoms and possible factors, we seriously suspect that he suffers from barometer headache and migraine [14], triggered by the weather, especially the barometric pressure. As he has the Headache for such a long time and cannot control the weather, he may need to see the doctor for more advice and further suggestions.

5. Summary

In this project, we have built a decision tree classifier for determining the probability of a person getting a headache on a particular hour of a day, given the weather, workout, and work details. First, we built our dataset after computing average weather details from five different Halifax locations. After doing pre-processing on our weather dataset, we added hourly Work and workout details of a person given to us in our dataset as two new features/columns. Next, we used the Adaptive synthetic oversampling technique to balance our dataset's labels/classes as there were only 72 rows in our dataset with headache 'Yes' label. After that, we used stratified k-fold cross-validation to train and test our decision tree model. Then, we pruned our decision tree after computing the optimal value of the alpha parameter. Finally, we evaluated our model by calculating accuracy, precision, recall, and average_precision and plotting the confusion matrix and ROC curve. Our model got an accuracy of 90 percent, 98 percent precision, and 63 percent recall.

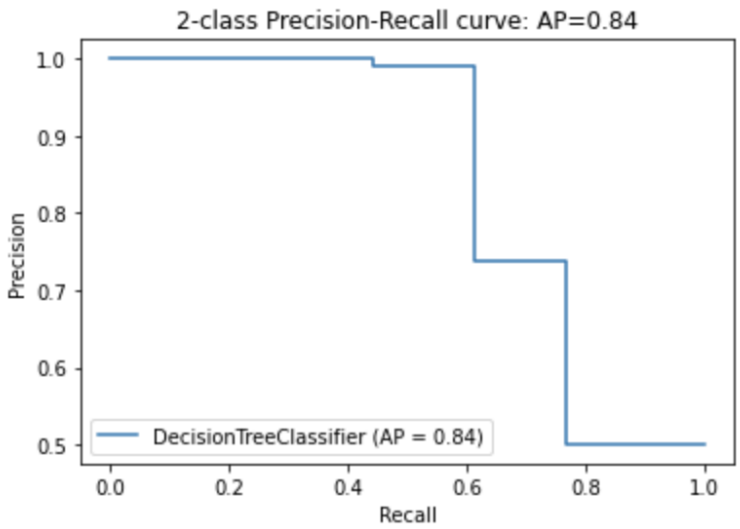

Figure 13 Precision-Recall curve

6. Team Members

Table 2 Team contribution

Abhinav Mandava	Performed descriptive analysis. Analyzed the features and built the initial decision tree. Performed 3-fold stratified cross-validation and reported the deficiencies for further improvement of the model. Plotted the probabilities of headaches w.r.t date and time.
Anqi Chen	I have done some researches on the data source of weather websites and analyzing the possible features. I am also learned how to generate the probabilities by using max_depth and RandomForest to truncate the tree. Discussed how to divide into three data sets: training data, validating data and testing set.
Ahsan Kamal	Wrote the script to build our training and a testing dataset containing hourly weather, workout, Work and Headache. Wrote the code to balance our dataset using the Adaptive synthetic oversampling technique. Wrote the code to evaluate the performance of our model using recall, precision, and ROC curve. Refactored the code by using OOP.
Raj Kumar Reddy Gangi	I analyzed the resources and processed data. I have built the DecisionTreeClassifier using the StratifiedKFold split function. Then prune the tree by calculating the optimal alpha value for the DecisionTreeClassifier. Used the model to create predictions for the Sept 2020 data.
Ram Prasath Meganathan	I have contributed to gathering data for building the datasets from the online source. I have pre-processed them and transformed them in a format required by the machine learning algorithm. I have participated in deciding the features of the machine learning model. I have worked on getting the predictions using the dataset as output and contributed to documentation.

References

[1] P. W. N. Inc, "The Weather Network - Weather forecasts, maps, news and videos," *The Weather Network*. http://www.theweathernetwork.com/ca (accessed October 04, 2020).

[2] "OnPoint® API," *Weather Source*. https://weathersource.com/products/onpoint-api/ (accessed October 04, 2020).

[3] "Goffs, Canada Weather History | Weather Underground." https://www.wunderground.com/history/daily/CYHZ/date/2020-10-3 (accessed October 04, 2020).

[4] "Weather," *Dark Sky*. https://darksky.net/forecast/44.6462,-63.5736/us12/en (accessed October 04, 2020).

[5] "Historical Weather during 2020 at Halifax Stanfield International Airport, Canada - Weather Spark." https://weatherspark.com/h/y/147454/2020/Historical-Weather-during-2020-at-Halifax-Stanfield-International-Airport-Canada (accessed October 04, 2020).

[6] N. US Department of Commerce, "National Weather Service." https://www.weather.gov/ (accessed October 04, 2020).

[7] E. and C. C. Canada, "Historical Climate Data - Climate - Environment and Climate Change Canada," October 31, 2011. https://climate.weather.gc.ca/ (accessed October 04, 2020).

[8] "NS Weather Station Data | Open Data | Nova Scotia." https://data.novascotia.ca/Environment-and-Energy/NS-Weather-Station-Data/kafq-j9u4 (accessed October 04, 2020).

[9] Rebecca Buffum Taylor, "Weather as a Headache and Migraine Trigger," WebMD, 11-Nov-2008. [Online]. Available: https://www.webmd.com/migraines-headaches/headache-and-migraine-trigger-weather. [Accessed: 04-Oct-2020]

[10] "Over-sampling — imbalanced-learn 0.5.0 documentation", Imbalanced-learn.readthedocs.io, 2020. [Online]. Available: https://imbalanced-learn.readthedocs.io/en/stable/over_sampling.html. [Accessed: 04- Oct- 2020].

[11] "sklearn.model_selection.StratifiedKFold — scikit-learn 0.23.2 documentation", Scikit-learn.org, 2020. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.html [Accessed: 04- Oct- 2020].

[12] "sklearn.tree.DecisionTreeClassifier — scikit-learn 0.23.2 documentation", Scikit-learn.org, 2020. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html. [Accessed: 04- Oct- 2020].

[13] "Post pruning decision trees with cost complexity pruning — scikit-learn 0.23.2 documentation", Scikit-learn.org, 2020. [Online]. Available: https://scikit-learn.org/stable/auto_examples/tree/plot_cost_complexity_pruning.html. [Accessed: 04-Oct- 2020].

[14] "Barometric Pressure Headaches: What You Should Know." https://www.healthline.com/health/headache/barometric-pressure-headache (accessed October 04, 2020).

Appendix

Code for pre-processing and dataset preparation

```
import pandas as pd
import datetime

class DatasetPrep:
    avg_weather_data = None
    work_column = None
    workout_column = None
    headache_column = None

    def read_and_prepare_weather_dataset(self):
        # read month-wise climate data containing hourly weather details from 5 different dataset sources
        # sep climate dataset
        HALIFAX_DOCKYARD_Sep =
pd.read_csv('dataset/HALIFAX_DOCKYARD/en_climate_hourly_NS_8202240_09-2019_P1H.csv')
        HALIFAX_KOOTENAY_Sep =
pd.read_csv('dataset/HALIFAX_KOOTENAY/en_climate_hourly_NS_8202252_09-2019_P1H.csv')
        HALIFAX_STANFIELD_INTL_A1_Sep = pd.read_csv(
            'dataset/HALIFAX_STANFIELD_INTL_A1/en_climate_hourly_NS_8202251_09-2019_P1H.csv')
        HALIFAX_STANFIELD_INTL_A2_Sep = pd.read_csv(
            'dataset/HALIFAX_STANFIELD_INTL_A2/en_climate_hourly_NS_8202249_09-2019_P1H.csv')
        HALIFAX_WINDSOR_PARK_Sep = pd.read_csv(
            'dataset/HALIFAX_WINDSOR_PARK/en_climate_hourly_NS_8202255_09-2019_P1H.csv')

        # oct climate dataset
        HALIFAX_DOCKYARD_Oct =
pd.read_csv('dataset/HALIFAX_DOCKYARD/en_climate_hourly_NS_8202240_10-2019_P1H.csv')
        HALIFAX_KOOTENAY_Oct = pd.read_csv('dataset/HALIFAX_KOOTENAY/en_climate_hourly_NS_8202252_10-
2019_P1H.csv')
        HALIFAX_STANFIELD_INTL_A1_Oct = pd.read_csv(
            'dataset/HALIFAX_STANFIELD_INTL_A1/en_climate_hourly_NS_8202251_10-2019_P1H.csv')
        HALIFAX_STANFIELD_INTL_A2_Oct = pd.read_csv(
            'dataset/HALIFAX_STANFIELD_INTL_A2/en_climate_hourly_NS_8202249_10-2019_P1H.csv')
        HALIFAX_WINDSOR_PARK_Oct = pd.read_csv(
            'dataset/HALIFAX_WINDSOR_PARK/en_climate_hourly_NS_8202255_10-2019_P1H.csv')

        # nov climate dataset
        HALIFAX_DOCKYARD_Nov =
pd.read_csv('dataset/HALIFAX_DOCKYARD/en_climate_hourly_NS_8202240_11-2019_P1H.csv')
        HALIFAX_KOOTENAY_Nov =
pd.read_csv('dataset/HALIFAX_KOOTENAY/en_climate_hourly_NS_8202252_11-2019_P1H.csv')
        HALIFAX_STANFIELD_INTL_A1_Nov = pd.read_csv(
            'dataset/HALIFAX_STANFIELD_INTL_A1/en_climate_hourly_NS_8202251_11-2019_P1H.csv')
        HALIFAX_STANFIELD_INTL_A2_Nov = pd.read_csv(
            'dataset/HALIFAX_STANFIELD_INTL_A2/en_climate_hourly_NS_8202249_11-2019_P1H.csv')
        HALIFAX_WINDSOR_PARK_Nov = pd.read_csv(
            'dataset/HALIFAX_WINDSOR_PARK/en_climate_hourly_NS_8202255_11-2019_P1H.csv')

        # dec climate dataset
```

```
    HALIFAX_DOCKYARD_Dec =
pd.read_csv('dataset/HALIFAX_DOCKYARD/en_climate_hourly_NS_8202240_12-2019_P1H.csv')
    HALIFAX_KOOTENAY_Dec =
pd.read_csv('dataset/HALIFAX_KOOTENAY/en_climate_hourly_NS_8202252_12-2019_P1H.csv')
    HALIFAX_STANFIELD_INTL_A1_Dec = pd.read_csv(
        'dataset/HALIFAX_STANFIELD_INTL_A1/en_climate_hourly_NS_8202251_12-2019_P1H.csv')
    HALIFAX_STANFIELD_INTL_A2_Dec = pd.read_csv(
        'dataset/HALIFAX_STANFIELD_INTL_A2/en_climate_hourly_NS_8202249_12-2019_P1H.csv')
    HALIFAX_WINDSOR_PARK_Dec = pd.read_csv(
        'dataset/HALIFAX_WINDSOR_PARK/en_climate_hourly_NS_8202255_12-2019_P1H.csv')

    # read date/time column and concatinate them
    date_time_col_sep = HALIFAX_DOCKYARD_Sep['Date/Time']
    date_time_col_oct = HALIFAX_DOCKYARD_Oct['Date/Time']
    date_time_col_nov = HALIFAX_DOCKYARD_Nov['Date/Time']
    date_time_col_dec = HALIFAX_DOCKYARD_Dec['Date/Time']
    date_time_col = pd.concat([date_time_col_sep, date_time_col_oct, date_time_col_nov, date_time_col_dec])

    # read following columns month-wise:
    # Date/Time, Temp, Dew Point temp, Rel Hum , Wind Spd, Visibility, Stn Press
    # sep
    halifax_dockyard_sep = HALIFAX_DOCKYARD_Sep[
        ['Date/Time', 'Temp (°C)', 'Dew Point Temp (°C)', 'Rel Hum (%)', 'Wind Spd (km/h)', 'Visibility (km)',
         'Stn Press (kPa)']]
    halifax_kootenay_sep = HALIFAX_KOOTENAY_Sep[
        ['Date/Time', 'Temp (°C)', 'Dew Point Temp (°C)', 'Rel Hum (%)', 'Wind Spd (km/h)', 'Visibility (km)',
         'Stn Press (kPa)']]
    halifax_stanfield_intl_a1_sep = HALIFAX_STANFIELD_INTL_A1_Sep[
        ['Date/Time', 'Temp (°C)', 'Dew Point Temp (°C)', 'Rel Hum (%)', 'Wind Spd (km/h)', 'Visibility (km)',
         'Stn Press (kPa)']]
    halifax_stanfield_intl_a2_sep = HALIFAX_STANFIELD_INTL_A2_Sep[
        ['Date/Time', 'Temp (°C)', 'Dew Point Temp (°C)', 'Rel Hum (%)', 'Wind Spd (km/h)', 'Visibility (km)',
         'Stn Press (kPa)']]
    halifax_windsor_sep = HALIFAX_WINDSOR_PARK_Sep[
        ['Date/Time', 'Temp (°C)', 'Dew Point Temp (°C)', 'Rel Hum (%)', 'Wind Spd (km/h)', 'Visibility (km)',
         'Stn Press (kPa)']]

    # oct
    halifax_dockyard_oct = HALIFAX_DOCKYARD_Oct[
        ['Date/Time', 'Temp (°C)', 'Dew Point Temp (°C)', 'Rel Hum (%)', 'Wind Spd (km/h)', 'Visibility (km)',
         'Stn Press (kPa)']]
    halifax_kootenay_oct = HALIFAX_KOOTENAY_Oct[
        ['Date/Time', 'Temp (°C)', 'Dew Point Temp (°C)', 'Rel Hum (%)', 'Wind Spd (km/h)', 'Visibility (km)',
         'Stn Press (kPa)']]
    halifax_stanfield_intl_a1_oct = HALIFAX_STANFIELD_INTL_A1_Oct[
        ['Date/Time', 'Temp (°C)', 'Dew Point Temp (°C)', 'Rel Hum (%)', 'Wind Spd (km/h)', 'Visibility (km)',
         'Stn Press (kPa)']]
    halifax_stanfield_intl_a2_oct = HALIFAX_STANFIELD_INTL_A2_Oct[
        ['Date/Time', 'Temp (°C)', 'Dew Point Temp (°C)', 'Rel Hum (%)', 'Wind Spd (km/h)', 'Visibility (km)',
         'Stn Press (kPa)']]
    halifax_windsor_oct = HALIFAX_WINDSOR_PARK_Oct[
        ['Date/Time', 'Temp (°C)', 'Dew Point Temp (°C)', 'Rel Hum (%)', 'Wind Spd (km/h)', 'Visibility (km)',
         'Stn Press (kPa)']]
```

```
# nov
halifax_dockyard_nov = HALIFAX_DOCKYARD_Nov[
    ['Date/Time', 'Temp (°C)', 'Dew Point Temp (°C)', 'Rel Hum (%)', 'Wind Spd (km/h)', 'Visibility (km)',
     'Stn Press (kPa)']]
halifax_kootenay_nov = HALIFAX_KOOTENAY_Nov[
    ['Date/Time', 'Temp (°C)', 'Dew Point Temp (°C)', 'Rel Hum (%)', 'Wind Spd (km/h)', 'Visibility (km)',
     'Stn Press (kPa)']]
halifax_stanfield_intl_a1_nov = HALIFAX_STANFIELD_INTL_A1_Nov[
    ['Date/Time', 'Temp (°C)', 'Dew Point Temp (°C)', 'Rel Hum (%)', 'Wind Spd (km/h)', 'Visibility (km)',
     'Stn Press (kPa)']]
halifax_stanfield_intl_a2_nov = HALIFAX_STANFIELD_INTL_A2_Nov[
    ['Date/Time', 'Temp (°C)', 'Dew Point Temp (°C)', 'Rel Hum (%)', 'Wind Spd (km/h)', 'Visibility (km)',
     'Stn Press (kPa)']]
halifax_windsor_nov = HALIFAX_WINDSOR_PARK_Nov[
    ['Date/Time', 'Temp (°C)', 'Dew Point Temp (°C)', 'Rel Hum (%)', 'Wind Spd (km/h)', 'Visibility (km)',
     'Stn Press (kPa)']]

# dec
halifax_dockyard_dec = HALIFAX_DOCKYARD_Dec[
    ['Date/Time', 'Temp (°C)', 'Dew Point Temp (°C)', 'Rel Hum (%)', 'Wind Spd (km/h)', 'Visibility (km)',
     'Stn Press (kPa)']]
halifax_kootenay_dec = HALIFAX_KOOTENAY_Dec[
    ['Date/Time', 'Temp (°C)', 'Dew Point Temp (°C)', 'Rel Hum (%)', 'Wind Spd (km/h)', 'Visibility (km)',
     'Stn Press (kPa)']]
halifax_stanfield_intl_a1_dec = HALIFAX_STANFIELD_INTL_A1_Dec[
    ['Date/Time', 'Temp (°C)', 'Dew Point Temp (°C)', 'Rel Hum (%)', 'Wind Spd (km/h)', 'Visibility (km)',
     'Stn Press (kPa)']]
halifax_stanfield_intl_a2_dec = HALIFAX_STANFIELD_INTL_A2_Dec[
    ['Date/Time', 'Temp (°C)', 'Dew Point Temp (°C)', 'Rel Hum (%)', 'Wind Spd (km/h)', 'Visibility (km)',
     'Stn Press (kPa)']]
halifax_windsor_dec = HALIFAX_WINDSOR_PARK_Dec[
    ['Date/Time', 'Temp (°C)', 'Dew Point Temp (°C)', 'Rel Hum (%)', 'Wind Spd (km/h)', 'Visibility (km)',
     'Stn Press (kPa)']]

# compute month-wise average of following columns from five different dataset sources:
# Temp, Dew Point temp, Rel Hum , Wind Spd, Visibility, Stn Press
avg_sep = pd.concat(
    [halifax_dockyard_sep, halifax_kootenay_sep, halifax_stanfield_intl_a1_sep, halifax_stanfield_intl_a2_sep,
     halifax_windsor_sep], ignore_index=True).groupby(['Date/Time']).mean().reset_index()

avg_oct = pd.concat(
    [halifax_dockyard_oct, halifax_kootenay_oct, halifax_stanfield_intl_a1_oct, halifax_stanfield_intl_a2_oct,
     halifax_windsor_oct], ignore_index=True).groupby(['Date/Time']).mean().reset_index()

avg_nov = pd.concat(
    [halifax_dockyard_nov, halifax_kootenay_nov, halifax_stanfield_intl_a1_nov, halifax_stanfield_intl_a2_nov,
     halifax_windsor_nov], ignore_index=True).groupby(['Date/Time']).mean().reset_index()

avg_dec = pd.concat(
    [halifax_dockyard_dec, halifax_kootenay_dec, halifax_stanfield_intl_a1_dec, halifax_stanfield_intl_a2_dec,
     halifax_windsor_dec], ignore_index=True).groupby(['Date/Time']).mean().reset_index()
```

```
    # concatinate month-wise average dataset from Sep-Dec
    avg_weather_data = pd.concat([avg_sep, avg_oct, avg_nov, avg_dec])
    len(avg_weather_data)
    return avg_weather_data


def create_work_column(self, avg_weather_data):
    # create work column (hour-wise) based on the working hours, weekends, and holidays
    work_column = pd.DataFrame(columns=['Work'])

    # Mon-fri: 9Am to 5Pm (working hours)
    # October 27 to November 03: no work (holidays)
    start_date_of_no_work = '2019-10-27 00:00'
    end_date_of_no_work = '2019-11-03 23:00'

    for i, d in avg_weather_data['Date/Time'].iteritems():
        date_time = datetime.datetime.strptime(d, '%Y-%m-%d %H:%M')
        if date_time.weekday() in (5, 6):
            # if weekend
            work_column = work_column.append({'Work': 0}, ignore_index=True)
        elif (start_date_of_no_work <= d) and (end_date_of_no_work >= d):
            # holidays
            work_column = work_column.append({'Work': 0}, ignore_index=True)
        else:
            work_hours = d.split()[-1]
            # working hours: 9AM to 5PM
            if work_hours >= '09:00' and work_hours <= '17:00':
                work_column = work_column.append({'Work': 1}, ignore_index=True)
            else:
                work_column = work_column.append({'Work': 0}, ignore_index=True)

    avg_weather_data['Date/Time'].iloc[0:5], work_column[0:5]
    len(work_column)
    # add Work col to our weather dataset
    avg_weather_data['Work'] = work_column['Work']


def create_headache_column(self, avg_weather_data):
    # store given date and time of headaches in list
    headache_dates = [['2019-09-05', '07:00', '10:00', 8], ['2019-09-15', '09:00', '15:00', 7],
                      ['2019-09-27', '15:00', '21:00', 8],
                      ['2019-10-15', '14:00', '18:00', 7], ['2019-10-16', '04:00', '18:00', 10],
                      ['2019-10-28', '15:00', '21:00', 6],
                      ['2019-11-28', '17:00', '18:00', 6], ['2019-12-10', '05:00', '10:00', 4],
                      ['2019-12-15', '02:00', '10:00', 7],
                      ['2019-12-20', '13:00', '18:00', 9]]

    # create headache column based on headache date/time
    headache_column = pd.DataFrame(columns=['Headache'])
    headache_dict = {}
    for headache_date in headache_dates:
        start_time = int(headache_date[1].split(':')[0])
        end_time = int(headache_date[2].split(':')[0])
```

```
            for hour in range(start_time, end_time + 1):
                headache_dict[f"{headache_date[0]} {hour:02}:00"] = headache_date[3]

        len(headache_dict.keys())

        for i, d in avg_weather_data['Date/Time'].iteritems():
            if headache_dict.get(d) != None:
                print(d, headache_dict.get(d))
                headache_column = headache_column.append({'Headache': 1}, ignore_index=True)
            else:
                headache_column = headache_column.append({'Headache': 0}, ignore_index=True)

        # add Headache column to our weather dataset
        avg_weather_data['Headache'] = headache_column['Headache']

    def create_workout_column(self, avg_weather_data):
        # store given workout/gym date and time in a list
        workout_dates = [['2019-09-02', '18:00', '19:00'], ['2019-09-04', '18:00', '19:00'],
                         ['2019-09-06', '18:00', '19:00'],
                         ['2019-09-27', '18:00', '19:00'],
                         ['2019-10-08', '18:00', '19:00'], ['2019-10-20', '20:00', '21:00'],
                         ['2019-10-27', '06:00', '07:00'],
                         ['2019-10-28', '18:00', '19:00'],
                         ['2019-11-02', '18:00', '19:00'], ['2019-11-07', '18:00', '19:00'],
                         ['2019-11-09', '18:00', '19:00'],
                         ['2019-11-12', '18:00', '19:00'], ['2019-11-20', '18:00', '19:00'],
                         ['2019-12-15', '18:00', '19:00'], ['2019-12-16', '18:00', '19:00']]

        # create workout/gym column (hour-wise) based on given workout/gym date/time.
        workout_column = pd.DataFrame(columns=['Workout'])
        workout_dict = {}
        for workout_date in workout_dates:
            start_time = int(workout_date[1].split(':')[0])
            end_time = int(workout_date[2].split(':')[0])
            print(start_time, end_time)
            for hour in range(start_time, end_time + 1):
                workout_dict[f"{workout_date[0]} {hour:02}:00"] = 1

        for i, d in avg_weather_data['Date/Time'].iteritems():
            if workout_dict.get(d) != None:
                print(d, workout_dict.get(d))
                workout_column = workout_column.append({'Workout': workout_dict.get(d)}, ignore_index=True)
            else:
                workout_column = workout_column.append({'Workout': 0}, ignore_index=True)

        # add workout to our weather dataset
        avg_weather_data['Workout'] = workout_column['Workout']

    def save_dataset_to_csv(self, avg_weather_data):
        # save dataset as a csv file
        avg_weather_data.to_csv("full_dataset.csv", index=False)
```

```
if __name__ == "__main__":
    dataset_prep = DatasetPrep()
    avg_weather_data = dataset_prep.read_and_prepare_weather_dataset()
    dataset_prep.create_work_column(avg_weather_data)
    dataset_prep.create_workout_column(avg_weather_data)
    dataset_prep.create_headache_column(avg_weather_data)
    dataset_prep.save_dataset_to_csv(avg_weather_data)
    print("ok")
```

Code for Model building

```
import pandas as pd
import numpy as np #to calculate mean and standard deviation
import matplotlib.pyplot as plt # to draw graphs
from sklearn.tree import DecisionTreeClassifier # to build classification tree
from sklearn.tree import plot_tree #to draw the classification tree
from sklearn.model_selection import train_test_split #split data into Training and Testing datasets
from sklearn.model_selection import cross_val_score #cross validation
from sklearn.metrics import confusion_matrix #to create a confusion matrix
from sklearn.metrics import plot_confusion_matrix # to plot the confunsion matrix
from sklearn.model_selection import StratifiedKFold # Stratification K-fold
from sklearn.metrics import recall_score as RecallScore
from sklearn.metrics import average_precision_score as AveragePrecisionScore
from sklearn import metrics
from imblearn.over_sampling import RandomOverSampler
from imblearn.over_sampling import SMOTE, ADASYN
from sklearn.metrics import precision_score

from sklearn.metrics import plot_precision_recall_curve
import matplotlib.pyplot as plt

class TrainTestDecisionTree:

    def __init__(self):
        self.clf_dts = []
        self.X_train_arr = []
        self.Y_train_arr = []
        self.X_test_arr = []
        self.Y_test_arr = []
        self.probs_arr = []
        self.X = None
        # features (columns)
        self.X_val = None
        # labels vector
        self.Y = None
        # final_decision_tree
        self.clf_dt = None
        # test data
        self.X_test = None
        self.Y_test = None

    def read_and_clean_dataset(self):
        # Read data form dataset columns-[Temp (Â°C),Dew Point Temp (Â°C),Rel Hum (%),Wind Spd (km/h),Visibility
```

```
(km),Stn Press (kPa),Work,Workout,Headache]
        df = pd.read_csv("dataset_without_datetime.csv");

        # check the datatypes of the data
        print(df.dtypes)

        # check for the missing data
        print(df['Temp (°C)'].unique())
        print(df['Dew Point Temp (°C)'].unique())
        print(df['Rel Hum (%)'].unique())
        print(df['Wind Spd (km/h)'].unique())
        print(df['Visibility (km)'].unique())
        print(df['Work'].unique())
        print(df['Workout'].unique())
        print(df['Headache'].unique())

        # Visibility has nan of 3 rows
        df = df.dropna()

        # Features for predection, copy all columns(X) except Heacache col(Y)
        self.X = df.drop('Headache', axis=1).copy()
        self.X_val = df.drop('Headache', axis=1).values

        # Feature to predict->Headache(Y)
        self.Y = df['Headache'].values
        return df

    def balance_dataset_using_oversampling(self):
        # randomly duplicating examples from the minority class (has Headache) and adding them to the training
dataset to create more balanced training dataset
        # Naive random over-sampling
        ros = RandomOverSampler(random_state=0)
        # X_resampled, Y_resampled = ros.fit_resample(X_val, Y)
        # Synthetic Minority Oversampling Technique
        # X_resampled, Y_resampled = SMOTE().fit_resample(X_val, Y)
        #  Adaptive Synthetic  Oversampling Technique
        X_resampled, Y_resampled = ADASYN().fit_resample(self.X_val, self.Y)
        from collections import Counter
        print(sorted(Counter(Y_resampled).items()))
        self.X_val = X_resampled
        self.Y = Y_resampled

    def train_dataset_using_k_fold_stratification(self):
        # The folds are made by preserving the percentage of samples for each class.
        # Iterate for 3 times
        skf = StratifiedKFold(n_splits=3)
        for train_index, test_index in skf.split(self.X_val, self.Y):
            # get testing and training data
            X_train, X_test = self.X_val[train_index], self.X_val[test_index]
            Y_train, Y_test = self.Y[train_index], self.Y[test_index]

            # preserve the values in the arrays
            self.X_train_arr.append(X_train)
```

```
            self.Y_train_arr.append(Y_train)
            self.X_test_arr.append(X_test)
            self.Y_test_arr.append(Y_test)

            # Use Decision Tree and fit the training data to the model
            clf_dt = DecisionTreeClassifier(random_state=30)
            clf_dt = clf_dt.fit(X_train, Y_train)

            # preserve all the
            self.clf_dts.append(clf_dt)

            # get the predections for the test data
            y_pred = clf_dt.predict(X_test)

            # plot the confusion matrix for eact fold and print the accuracy.
            plot_confusion_matrix(clf_dt, X_test, Y_test, display_labels=["No Headache", "Has Headache"])
            print("Accuracy:", metrics.accuracy_score(Y_test, y_pred))
            # plt.figure(figsize=(50,50))
            # plot_tree(clf_dt,filled=True,rounded=True,class_names=["No Headache","Has
Headache"],feature_names=X.columns)

            # Accuracy: 0.7938461538461539
            # Accuracy: 0.9507692307692308
            # Accuracy: 0.96

        plot_tree(clf_dt, filled=True, rounded=True, class_names=["No Headache", "Has Headache"],
                  feature_names=self.X.columns)


    def prune_decision_tree(self):
        # choosing 3rd item since accuracy has 96%
        i = 2
        X_train = self.X_train_arr[i]
        Y_train = self.Y_train_arr[i]
        self.X_test = self.X_test_arr[i]
        self.Y_test = self.Y_test_arr[i]

        # cost complexity pruning for calculating alpha valuse
        path = self.clf_dts[i].cost_complexity_pruning_path(X_train, Y_train)
        ccp_alphas = path.ccp_alphas
        # do not consider the alpha with highest value
        ccp_alphas = ccp_alphas[:-1]

        clf_dts = []

        # clculating alpha PART-1

        # for each alpha build the decision tree
        for ccp_alpha in ccp_alphas:
            clf_dt = DecisionTreeClassifier(random_state=0, ccp_alpha=ccp_alpha)
            clf_dt = clf_dt.fit(X_train, Y_train)
            clf_dts.append(clf_dt)
```

```
        # claculate the scores of each decision tree
        train_scores = [clf_dt.score(X_train, Y_train) for clf_dt in clf_dts]
        test_scores = [clf_dt.score(self.X_test, self.Y_test) for clf_dt in clf_dts]

        # plot the accuracy across differenr alpha values
        fig, ax = plt.subplots()
        ax.set_xlabel("alpha")
        ax.set_ylabel("accuracy")
        ax.plot(ccp_alphas, train_scores, marker='o', label="train", drawstyle='steps-post')
        ax.plot(ccp_alphas, test_scores, marker='o', label="test", drawstyle='steps-post')
        ax.legend()
        plt.show()

        alpha_loop_values = []

        # clculating alpha PART-2

        # calculate decision tree for different alphas
        # run 5 fold cross validation
        for ccp_alpha in ccp_alphas:
            clf_dt = DecisionTreeClassifier(random_state=0, ccp_alpha=ccp_alpha)
            scores = cross_val_score(clf_dt, X_train, Y_train, cv=5)
            alpha_loop_values.append([ccp_alpha, np.mean(scores), np.std(scores)])

        # plot the graph of alphas with mean accuracy
        alpha_results = pd.DataFrame(alpha_loop_values, columns=['alpha', 'mean_accuracy', 'std'])
        alpha_results.plot(x='alpha', y='mean_accuracy', yerr='std', marker='o', linestyle='--')

        # from the figure the best fit alpha is 0.0009

        self.clf_dt = DecisionTreeClassifier(random_state=0, ccp_alpha=0.024)

        self.clf_dt = self.clf_dt.fit(X_train, Y_train)

        plot_confusion_matrix(self.clf_dt, self.X_test, self.Y_test, display_labels=["No Headache", "Has Headache"])
        plt.figure(figsize=(50, 50))
        plot_tree(self.clf_dt, filled=True, rounded=True, class_names=["No Headache", "Has Headache"],
                  feature_names=self.X.columns)

    def model_evaluation(self):
        # calculate precision, recall, and average_precision_score. Plot ROC curve.
        precision = precision_score(self.Y_test, self.clf_dt.predict(self.X_test), average='binary')

        recall_score = RecallScore(self.Y_test, self.clf_dt.predict(self.X_test), average='binary')
        print(precision, recall_score)

        average_precision = AveragePrecisionScore(self.Y_test, self.clf_dt.predict_proba(self.X_test).T[1])

        disp = plot_precision_recall_curve(self.clf_dt, self.X_test, self.Y_test)
        disp.ax_.set_title('2-class Precision-Recall curve: '
                           'AP={0:0.2f}'.format(average_precision))
```

```
if __name__ == "__main__":
    trainTestDecisionTree = TrainTestDecisionTree()
    trainTestDecisionTree.read_and_clean_dataset()
    trainTestDecisionTree.balance_dataset_using_oversampling()
    trainTestDecisionTree.train_dataset_using_k_fold_stratification()
    trainTestDecisionTree.prune_decision_tree()
    trainTestDecisionTree.model_evaluation()
    print("ok")
```