A Comparative Study of ML Models on Alcohol Consumption Pattern

Tileshwar Narayan

4/1/24

Statistical Machine Learning II

Abstract

The study is all about exploring the Alcohol consumption classifying factors which helps to understand the important variables which drive alcohol abuse by utilizing different kinds of machine learning models on the data of NSDUH 2020. In this study, the model building has been divided into three different levels. At the first level, Binary classification has been concluded to observe the usage of alcohol and non-usage of alcohol. In this binary classification we have Gradient Boosting Classifier model which has provided an accuracy of 77.77% and some important variables under models are Education Level, Friend's Feedback on Alcohol use, Religion Influence, Youth Selling Drugs, Parents helped to do HW last year.

The study moves forward to the second level, with multi class classification which mainly concentrates on the frequency of the usage of alcohol in a year which expands to multiclass classification to categorize consumption levels in greater detail. In this classification we have the best model as Gradient Boosting model with a score of 79.23% with some important factors like Education Level, Friend's Feedback, Parents Limit TV watching, Peer Drinking and Parent talks to Alcohol use on daily basis.

At the final stage, we have Regression models. The findings contribute valuable insights into the classification landscape, particularly in understanding and finding out some of the best variables which affect alcohol use. In this exercise we have the best model as Bagging with Random Forest ensemble method with a minimum test MSE of 1.764 with some of the important variables as Youth Selling Drugs, Parent talks to Alcohol use on daily basis, youth having Individual Mother, Teacher Feedback and Health Condition.

We can observe some common important factors in all the three models, like Education Level, Youth Selling Drugs, Parent talks to Alcohol use on daily basis and Friend's Feedback along with important factors like on Alcohol use, Religion Influence, Parents help to do HW last year, Parents Limit TV watching, Peer Drinking feeling, youth having Individual Mother, Teacher Feedback and Health Condition.

Introduction

Alcohol consumption patterns among youth are a critical area of study due to their implications for public health and addictions issues. In this study, our goal is to explore and analyze various classification modeling techniques to understand and check the important factors related to ongoing alcohol addictions. Our primary goal is to interrogate the behavior of alcohol consumption through different methodologies, ranging from binary and multiclass classification to regression models. These models would help us to check on the effectiveness of these techniques in capturing the nature of alcohol consumption behaviors.

Overview

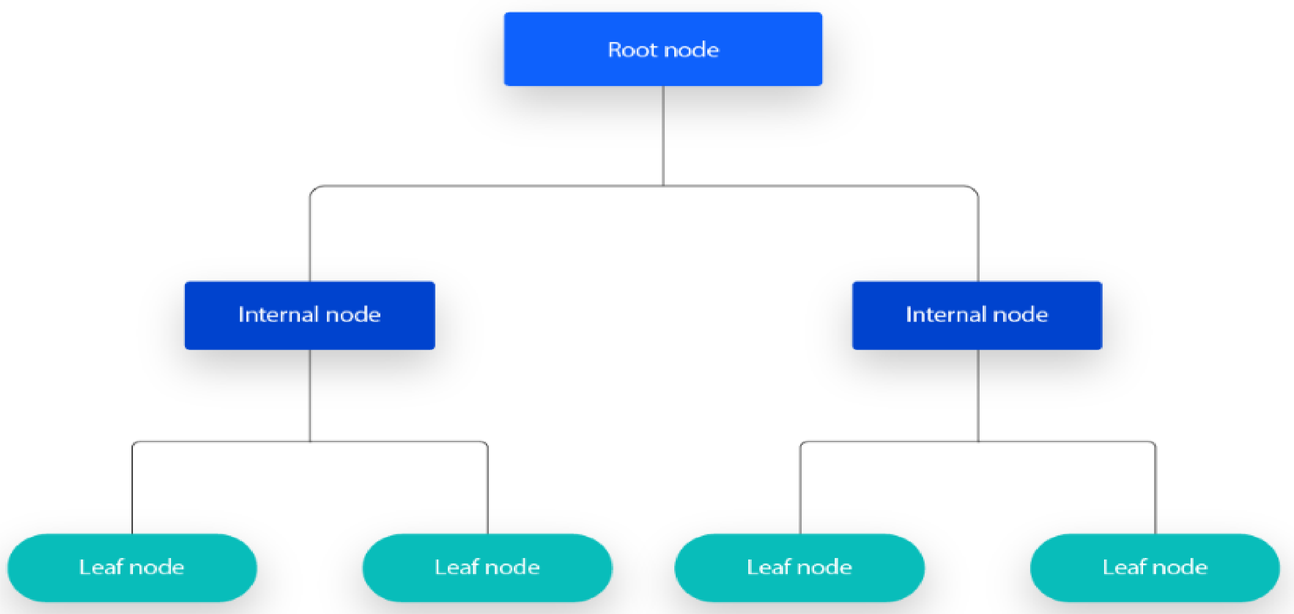
Our study starts by explaining what we aimed to achieve and how we did it. We focused on binary classification models that can split people into two groups: drinkers and non-drinkers. This helps us build a basic understanding of how to use models to classify alcohol use in young people. We then take things further by using multi-class models that can categorize people based on frequency of how frequently they drink. On the final part, we use regression models (regression) to predict how often someone drinks based on things like their demographics and youth experiences. By using these different approaches, we are trying to show how well each model works and how they can help us understand and even predict drinking patterns among young people.

Description of the Dataset

The dataset under study contains a comprehensive array of substances used, demographic data, and youth experience (social) variables related to youth alcohol consumption. It encompasses factors such as parental influence, peer interactions, academic performance, and personal attitudes towards alcohol and many different aspects. This data has great information and encompasses the actual experience of the youth they have faced, providing a great mirror to our current society and the menace caused by substance use.

Theoretical Background:

A decision tree is a non-parametric supervised learning algorithm, which is utilized for both classification and regression tasks. It has a hierarchical tree structure, which consists of a root node, branches, internal nodes, and leaf nodes. A decision tree starts with a root node, which does not have any incoming branches. The outgoing branches from the root node then feed into the internal nodes, also known as decision nodes. Based on the available features, both node types conduct evaluations to form homogenous subsets, which are denoted by leaf nodes, or terminal nodes. The leaf nodes represent all the possible outcomes within the dataset. Below is one such example figure of decision tree.



We have used different models under Decision tree as it can be used in both classification and regression models. For Binary and Multi-class classification we have decision tree classifier, random forest classifiers, bagging with random forest ensemble methods and boosting models. These models can be used in both binary and multi-class classifications.

Decision Tree Classifiers create tree-like model where each internal node represents a decision based on a feature, leading to leaf nodes that correspond to class labels. While Random Forest Classifier (ensemble method) creates multiple decision trees during training and combines their predictions through voting or averaging, resulting in a robust and accurate model.

To make most of the decision tree we do some parameter tunings some of them are **max_depth** which help to control the depth of the tree which helps reduce complexity and overfitting by restricting the number of splits, **min_samples_split** determines the minimum number of samples required to split a node further which help restricting the creation of nodes with too few samples, **min_samples_leaf** helps to determine the minimum number of samples required to be at a leaf node, which in turn makes sure each leaf has a sufficient number of samples to avoid overfitting and **max_features** helps taking the maximum number of features considered for splitting at each node controlling the randomness in feature selection. These features can be used in models in decision tree classifier models but for Random forest ensemble method we must use **n_estimators** which refers to the number of decision trees to be used in an ensemble method which determines the number of individual models that will be trained and combined to make predictions and **max_samples** which refers to the maximum number or proportion of samples to draw from the dataset to train each base estimator in the bagging ensemble

Though we have many advantages of the Decision tree models like its easy to interpret because of the Boolean logic and visual representations of decision trees make them easier to understand and consume, it requires minimal data preparation and can handle various data types, including discrete and continuous values and its more flexible as it can be used both for classification and regression. But we also have some limitation with Decision tree models like, its prone to overfitting as complex decision trees tend to overfit and do not generalize well to new data, secondly it has high variance estimator as small variations within data can produce a very different decision tree and its expensive as decision trees take a greedy search approach during construction, they can be more expensive to train compared to other algorithms.

Methodologies:

We have first started with cleaning of the data and processing further. First, we removed any missing pieces to make sure our dataset is clean and tidy. Then, we focus on the alcohol-related data and columns which are not related to alcohol were set aside i.e., the details about tobacco, marijuana, and cigarettes. After this step we renamed our dataset columns so that it would be easy to understand the proper definition of what the columns are telling us. We have also reassigned value names for values like 991, 993, 85, 94, 97, 98 and 99.

In the binary classification phase, we have created a separate dataset called `binary_df`, where the target variable represents alcohol use, categorized as "Never Used" (0) or "Ever Used" (1). We have employed four models: Decision Tree (DT) classification, Random Forest (RF) with Bagging, RF, and Gradient Boosting. These models utilize 50 estimators and a random state of 42. We evaluate model performance using accuracy metrics, including Confusion Matrix and Classification Report, and visualize decision trees and derive the variable importance along with its top performer graph. During binary classification, the data has been split into 70:30 ratio, meaning 70% train and 30% test data. We have used cross validation methods with GridSearchCV with different parameters like 'max_depth': [3, 5, 7, 10], 'min_samples_split': [2, 5, 10], and 'min_samples_leaf': [1, 2, 4] for classifier models and 'n_estimators': [30, 40, 50], and 'max_samples': [0.5, 0.7, 1.0] for ensemble models.

For multiclass classification, we have utilized the target variable "Alcohol used year," categorized into five parts: "Not used," "1-90 days," "91-180 days," "181-270 days," and "270+ days.", under a different dataset named as multi_df which has been used to employ DT Classifier, RF with Bagging, and Gradient Boosting Classifier, and we assess accuracy through Confusion Matrix and Classification Report. Along with variable importance and graphs. In this data, we have used the same data split used in the binary classification method i.e., 70:30 ratio, meaning 70% train and 30% test data. We have used cross validation methods with GridSearchCV with different parameters as 'max_depth': [3, 5, 7, 10], 'min_samples_split': [2, 5, 10], and 'min_samples_leaf': [1, 2, 4] for classifier models and 'n_estimators': [30, 40, 50], and 'max_samples': [0.5, 0.7, 1.0] for ensemble models.

Moving to regression analysis, we predict "Alcohol Used Last Month". In this target variable, I have converted the 991 and 993 data as 0 which is never used, and rest has not been modified as its for regression model. A separate dataframe named `reg_df` is created to accommodate the target variable "Alcohol used last month." And we have used DT Regressor, RF Regressor with Bagging, and Gradient Boosting Regressor, we calculate Test Mean Squared Error (MSE) for each model, visualize decision trees, and determine variable importance. For this regression model, we have divided the data into 70:30 ratio. Here also the parameter tuning has been the same way as we have done earlier. The only difference between classification and regression model is the way we have searched for the best model. In classification we have used scoring='accuracy' which specifies the metric for evaluation, cv=5 which is number of cross-validation folds, verbose=1 which controls the verbosity of the output and n_jobs=-1 which utilizes all available CPU cores for computations. But for regression models, have used the scoring as 'r2' specifies the metric for evaluation as the coefficient of determination (R^2).

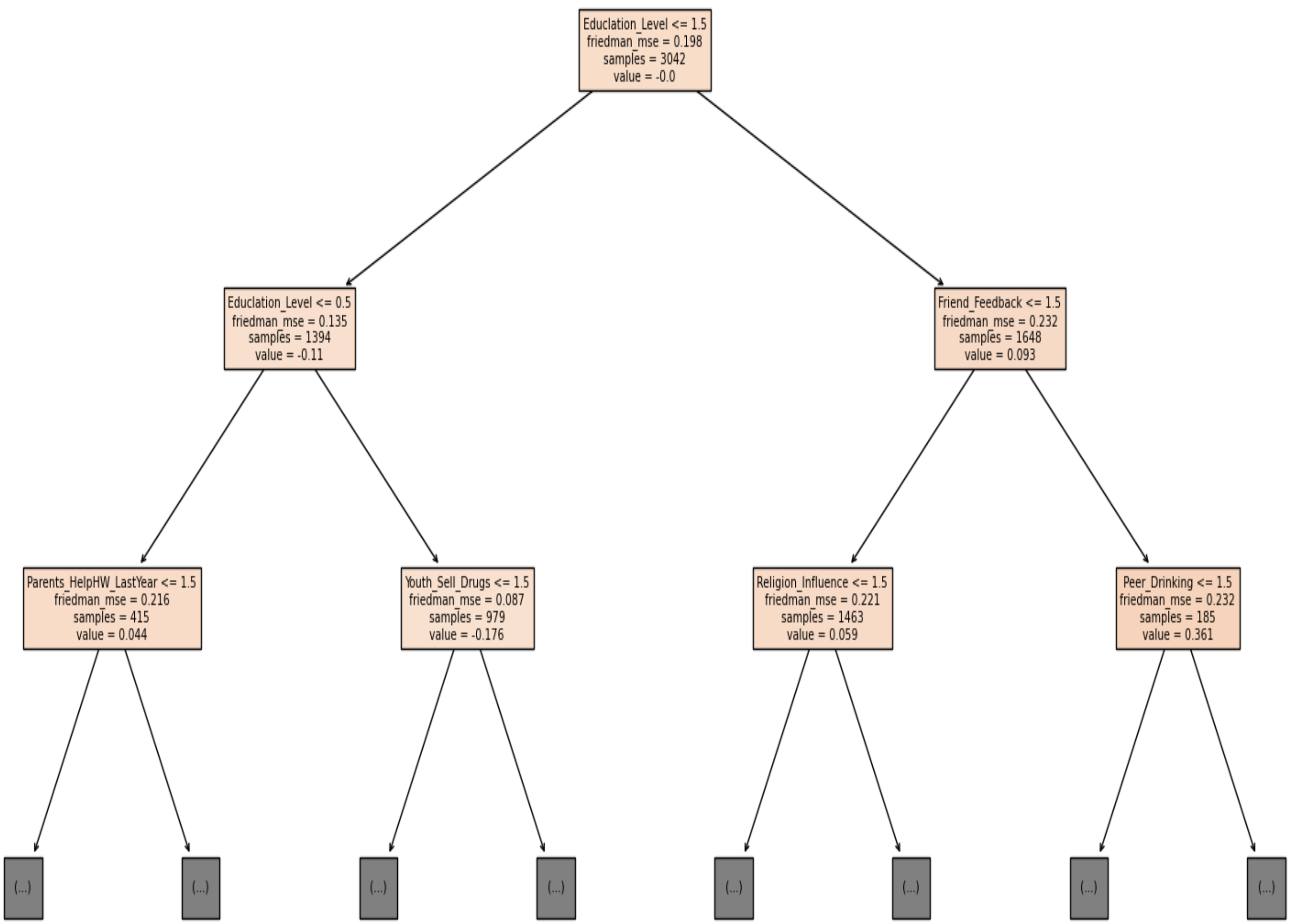
Computational Results:

We will start with Binary Classification Models. Though we have computed Accuracy, Confusion Matrix, Classification reports, Tree, and variable importance for all the 4 models used under Binary, we will describe the best model over here that we have derived i.e., Gradient Boosting.

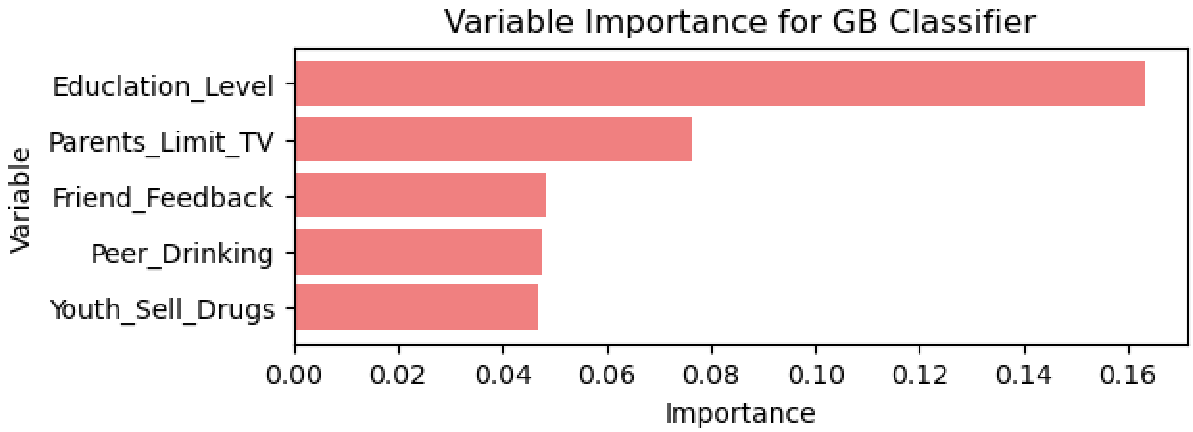
Accuracy Score, Confusion Matrix, and Classification Report:

Gradient Boosting shows a test accuracy of 77.70%. On the confusion matrix, the classifier correctly recognized 889 non-drinkers (category 0) and 125 drinkers (category 1) on the matrix. On the other hand, it yields 66 misclassified drinkers and 225 misclassified non-drinkers. Besides accuracy, recall and F1-score would tease out other aspects of the model's effectiveness. Non-drinking participants (class 0) had a precision of 0.80 while drinkers (class 1) had a precision of 0.65. The generalization rate for non-drinkers was 0.93, on the other hand drinkers rating was 0.36. The balanced precision and recall in the F1-scores were 0.86 and 0.46, accordingly for non-drinkers and drinkers.

Tree Graph: This tree has pruned to max_depth as 2 as the actual tree was too maced up and unable to understand.



Variable Importance for Top 5 variables:



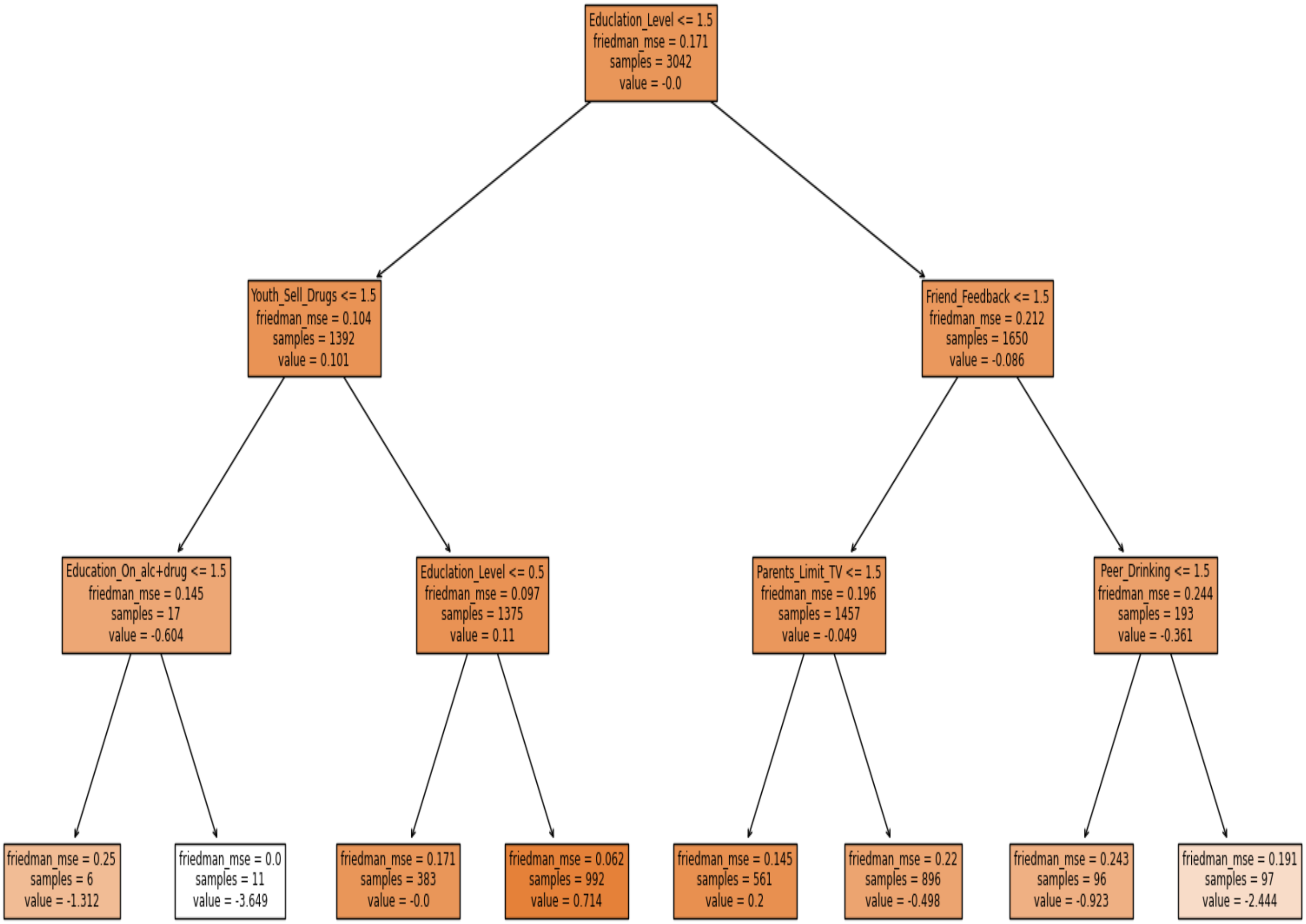
The graph above shows the top 5 variables that are most important as predicted by the model. The values Education Level, Parents limit on television, Friends feedback in alcohol use, Youths thought on Peer drinking and youth selling drugs provides us the most important variables to be used for the classification.

Now we move towards the Multi-class Classification where we will choose the best model which has the highest score in this case its Random Forest with Bagging Classifier.

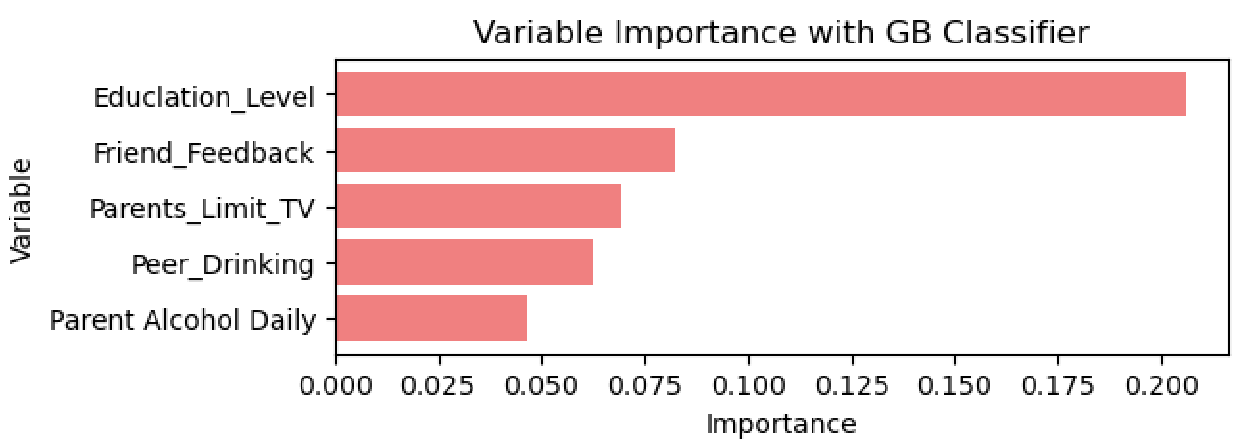
Now let's Review Accuracy, Confusion Matrix, and Classification Report.

Gradient Boosting provided an accuracy of about 79.23% on the data set that overall is moderate. In the confusion matrix the main classification being that 977 predictions were correctly classified either as class 0 (no alcohol consumption) or class 1 (low frequency of alcohol consumption) while 57 were correctly classified as class 1. Nevertheless, the model was inconsistent with the classifications of instances in classes 2, 3 and 4. For instance, class 2 was not correctly classified 14 times as class 0, 4 times as class 1 and none as class 2. Given that classes 3 and 4 were all wrongfully classified as class 0 in class 3 and class 4, it can be inferred that the model is unable to predict the outcome of these classes. Besides precision, recall, and F1-score, these metrics give you the ability for model's performance examination on each class. For class 0, the precision was 0.81, the recall was 0.96, and the F-score was 0.88. The precision for class 1 was 0.58, the recall was 0.21, and hence the F1-score was 0.31. Nevertheless, the class values of classes 2, 3, and 4 were all 0, meaning the model was struggling to make predictions for these classes.

Let's check the tree diagram:



Now let’s check the variable importance graph.



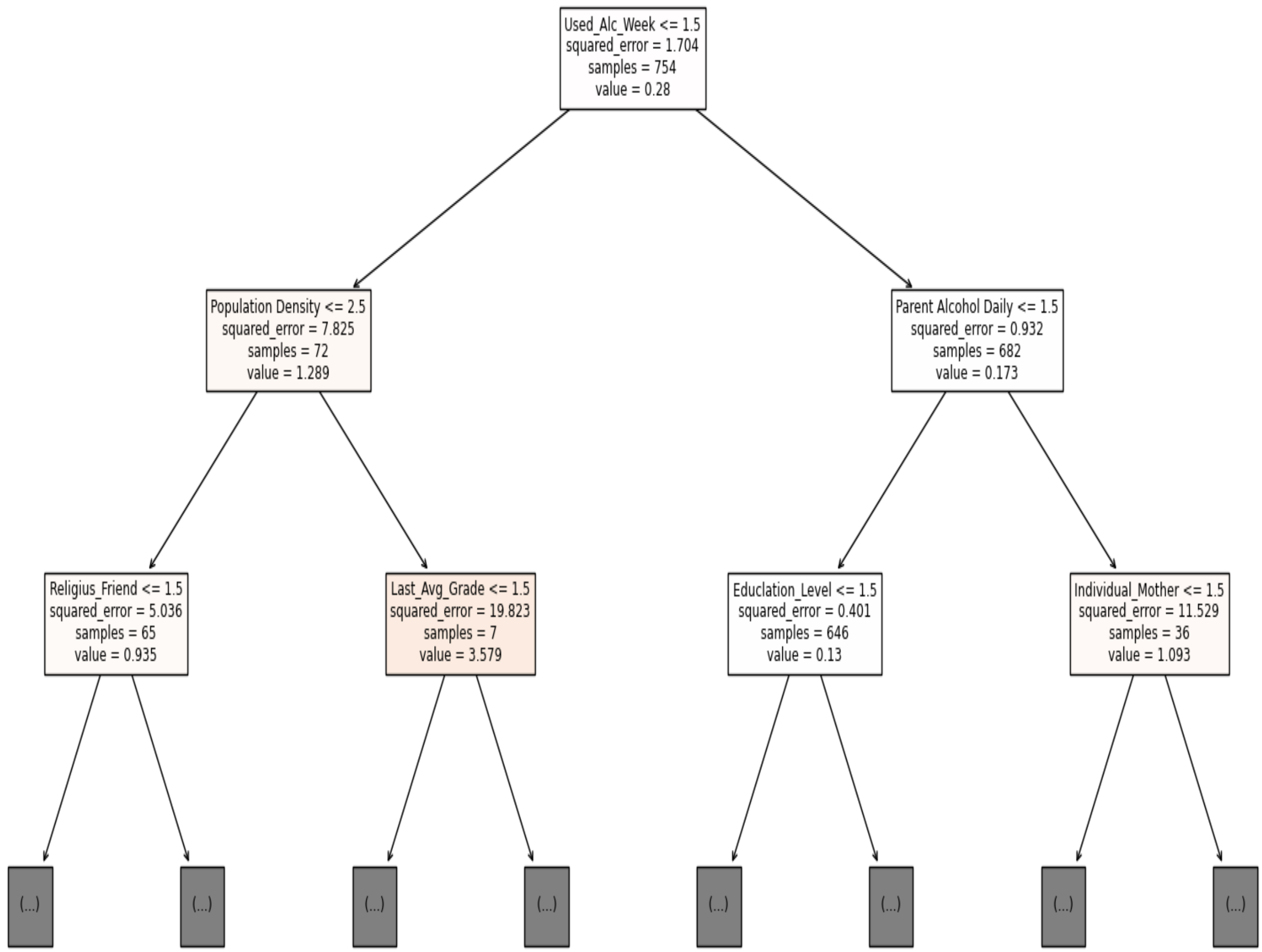
In this graph we can observe that again Education Level is there with Friends feedback, Parents Limit on TV and youths thinking on peer drinking are the same as the binary classification model we have.

Now we will check the Regression model performance through some data, tree diagram and variable importance graph. The best performing model in regression is the Bagging with Random Forest model.

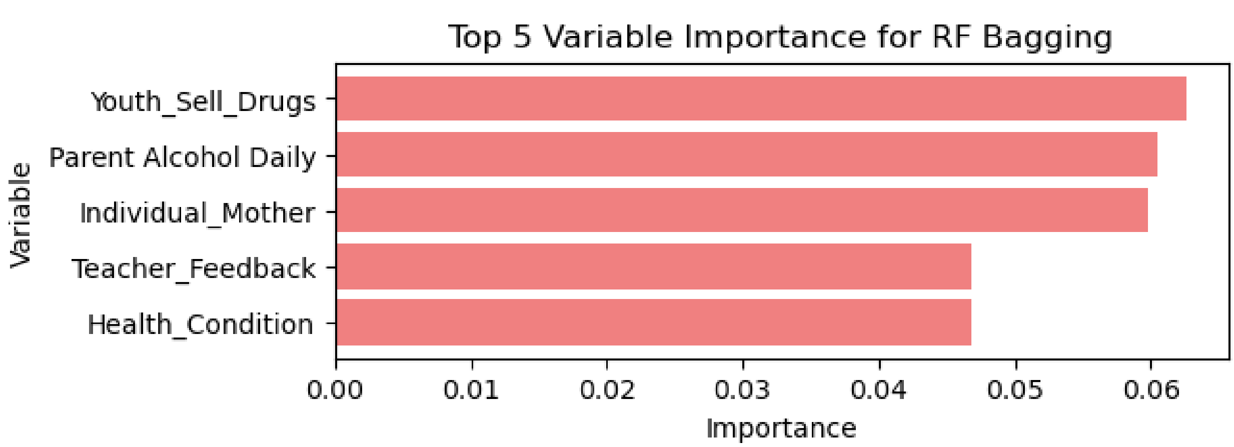
We have the Test MSE (Mean Squared Error) for Gradient Boosting Regressor:

```
RF Regressor Bagging MSE: 1.7647723669859512
```

Now let's check the tree for this model.



Let's check the model variable importance graph and what predictors seem important for this model.



In this model, we have youth selling drugs, parents checking on youth on daily alcohol use, youth with single parent (mother) along with teacher's feedback and health condition. The regression model has provided these top variables which can be linked to alcohol use. Use of alcohol can be dependent on many other factors but as per our model some the best to classify between alcohol use.

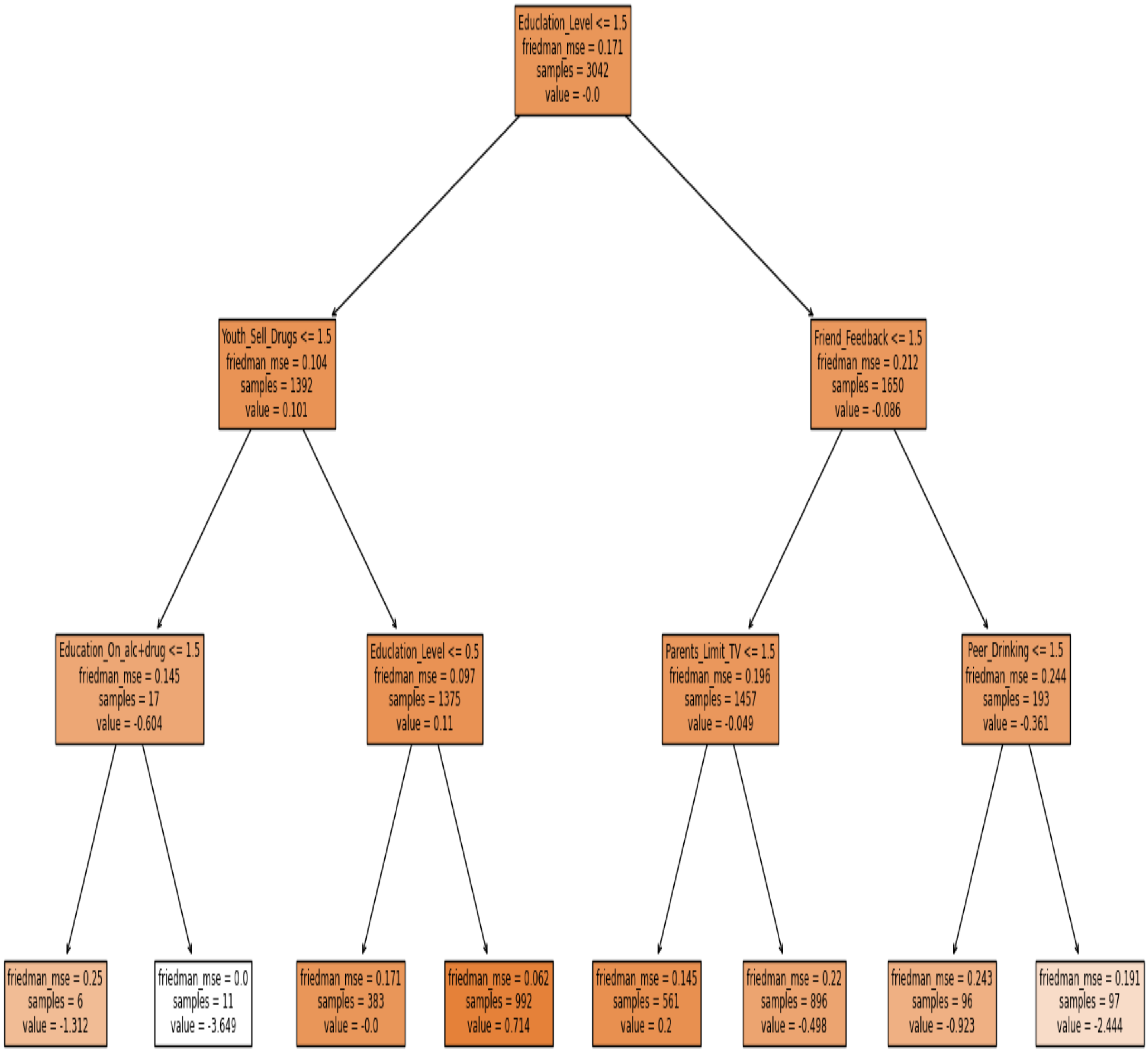
We have our models' computed values of the best models.

```
Binary Classification Results:
Decision Tree Accuracy: 0.7448275862068966
Bagging with RF Accuracy: 0.7639846743295019
Random Forest Accuracy: 0.7647509578544062
Gradient Boosting Accuracy: 0.7770114942528735

Multi-Class Classification Results:
Decision Tree Accuracy: 0.7770114942528735
Bagging with Random Forest Accuracy: 0.7862068965517242
Gradient Boosting Accuracy: 0.7923371647509578

Regression Results:
Decision Tree MSE: 1.9294968496848945
Bagging with Random Forest MSE: 1.7647723669859512
Gradient Boosting MSE: 2.082048518298126
```

The overall results states that in Binary Classification we have the Gradient Boosting as our best model with 77.77% accuracy. For the Multi-class Classification, we have the Gradient Boosting Classification as our best with an Accuracy of 79.23%. In the Regression analysis we have Bagging with Random Forest model which has the lowest MSE as 1.7647

Discussions:

A tree generally starts with a single node, which branches into possible outcomes. Each of those outcomes leads to additional nodes, which branch off into other possibilities. This gives it a treelike shape. We will now go through a tree diagram and let's go through its decision-making effects and its meaning. The tree we are considering for now is the Gradient Boosting Classifier tree.



Our tree will start at the root node, which is Education Level. Since the education level has been classified under 4 categories 0 as Uneducated, 1 which has studied less than 5th standard, 2 those who studied from 6 to 12 and 3 college students or have studied. Since we observe that the root node specifies whether Education level is less than equal to 1.5 basically corresponds to the youth had an education which is more than 5th standard or less than 5th standard education. A split at 1.5 would effectively mean that individuals with a value of 0 and 1 are grouped on one side of the split, while those with a value of 2 and above are on the other side. Now if the answer is Yes it will move towards the Friend's Feedback and if its No it will move towards whether the youth have sold any drugs. The value <= 1.5 suggest either 1 or 2, meaning 1 has been considered as he has sold drugs, and 2 meaning youth has not sold any drugs. Again, the node will check whether the condition is depicting whether its Yes or No. If Yes, then again it will check whether the youths Education level is less than equal to 0.5 which can be interpreted as the model is now checking whether he is educated or uneducated. While if you move towards No, it will check it will check the whether the youth has taken any education on drug and alcohol usage. The divide of the graph with education level = or < 0.5 will result to 2 different end nodes. At the "No" node of the Friedman Mean Squared Error, the mse is 0.171 for the main part (383 samples) of them. In this part, the value for alcohol usage is predicted as zero by the following model, which signifies less odds for alcohol consumption by people with low level of education. On the other side, "Yes" node with less Friedman mse 0.062 than the previous shows a better prediction in determining the alcohol consumption of this subgroup of 992 samples. The estimate of 0.714 implies that people with less education, who have been informed about the dangers of alcohol, are more susceptible to having a drinking habit.

Our dataset is a combination of binary and numerical data which has been used in different forms which has been used to determine and predict the factors that are influencing the use of alcohol in youths. Binary variables, like our target variables alcohol use and other variables from youth experiences, offer a clear distinction between positive and negative of a factor. The dataset does not have Ordinal data directly, it has been transformed into distinct numerical values which make it easier to interpret and can be modified according for general people to understand. Numerical variables, like age or academic performance, provide continuous insights into their relationship with alcohol use frequency. The predictions derived from the binary data sets are straight forward which simplify the model complexity which may be proved to be good in some complex issues. While ordinal data provides a clear interpretation of the problems, it may lack precision. The best is the numerical values which will best in every case, but we should be clear what numerical value say and what they represent. Selecting a numerical value provides great interpretability but we should try to focus on the usage and where to use it.

The findings highlighted the fact that several variables, the most significant among which being the classification models, significantly affect the classification of alcohol consumption. Considering the binary case, the most significant forecasters among them are these five factors: Education level, Parents who set TV time limit, Friends' reaction in case of alcohol use, Youth's perception of peer drinking, and youth who sell drugs. Education Level function as a line for socioeconomic status and educational value among people, thus those who are highly educated may drink less because they are more educated and better able to make decisions. The parents’ restrictions on television will show parental involvement and providing guidance. Therefore, parental supervision as a mean will help to manage their children's alcohol use behavior by setting rules and developing good habit.

Additionally, the Friends' input in alcohol use provides evidence of the effect of peer relationships on youth behavior, with peers reinforcing one another likely to draw the youth into alcohol consumption. The Youth’s Perspectives of Peer drinking defines the social perception of Peers who drink and if observing peers consuming alcohol influences the use of alcohol among youths. First, drug sold by youth is one of the factors which may affect either drug use or participation in such drug-related events that can later lead to alcoholism and addiction. In multiclass classification also similar patterns are found, in which Education level, Friends' feedback, Parents' Limit on TV and Youth's perception of peer drinking are already in the line. In regression analysis, youth selling drugs, parental monitoring of youth alcohol use, youth from single-parent households, teacher's feedback, and health condition are important predictors.

Interpretation of Findings:

Finally, we are left with the fundamental causes of alcohol abuse that can be pulled out from the different dynamics of alcohol addiction. Some determining factors include the high level of education, parental influence, peer influence and drug activities, education is the main factor in the binary classification. Unlike multiclass classification these factors are related ones e.g. education, social feedback, and parental influence. Other additional factors such as drug abuse, parenting supervision inadequacy, family influence, and sufferings of health are also explained by this intricate regression analysis. Such observations, therefore, indicate the degree of complexity associated with the use of alcohol among the youth and prove that these factors indeed help to counter the use alcohol among youths.

Conclusion:

This analysis has shown that there are factors that lead to alcohol consumption by young people. These factors including individual behaviors, demographic status, friendship and peer interaction, parental involvement and environmental factors were found to be working together to affect the drinking experience. The binary classification yielded highly significant predictors which are the intensity of education, limiting of television use, feedback by peers on alcohol use, thoughts on peer drinking, and interfere in drug related activities. Multiclass classification results emphasized that education holds a great position and social feedback was more reliable. Parents also had significant influence on children. These parameters were further shown to give out the most accurate information during regression analysis which include substance abuse, parental monitoring, family structure, and health condition, which revels these factors are the signs showing the importance of targeting underlying causes and implementation of relevant policy by government. Overall, the models we used have provided some major factors which are important to check and classify the alcohol use cases.

Bibliography:

1. Lisa Mueller and Marisa Bjorland: Interpreting a Tree Diagram (8)

   https://study.com/skill/learn/interpreting-a-tree-diagram-explanation.html

2. How Machine Learning Can Be Used for Multiclass Classification in Python (5-6) (Codes)
   https://www.turing.com/kb/how-machine-learning-can-be-used-for-multiclass-classification-in-python

3. Ilija Eftimov: Predict heart attack outcomes using decision tree classifier from scratch and scikit-learn (8-9) (Code)
   https://ieftimov.com/posts/classifier-decision-trees/

4. What is Decision Tree? (Theoretical Background)
   https://www.ibm.com/topics/decision-trees

Codes and Outputs

a45xvbiqr

April 27, 2024

[1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import statsmodels.api as sm
import warnings
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import RandomForestRegressor, BaggingRegressor
from sklearn.tree import DecisionTreeRegressor, plot_tree
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_squared_error
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import confusion_matrix, accuracy_score,␣
↪classification_report
from sklearn.ensemble import BaggingClassifier
from sklearn.model_selection import GridSearchCV
warnings.filterwarnings("ignore")
```

[2]:
```
# Reading the Data
df = pd.read_csv(r"C:\Users\tiles\Downloads\youth_data.csv")
```

[3]:
```
# Checking the first 5 Rows
df.head(5)
```

[3]:
```
   iralcfy  irmjfy  ircigfm  IRSMKLSS30N  iralcfm  irmjfm  ircigage  \
0      993     991       91           91     93.0    91.0       991
1      991     991       91           91     91.0    91.0       991
2      993     993       93           91     93.0    93.0        13
3      991     991       91           91     91.0    91.0       991
4      991     991       91           91     91.0    91.0       991

   irsmklsstry  iralcage  irmjage  …  eduschlgo  EDUSCHGRD2  eduskpcom  \
0          991        12      991  …          1           5          0
```

```
1          991       991       991  …          1           5           0
2          991        13        13  …          1           4           0
3          991       991       991  …          1           7           0
4          991       991       991  …          1           3           0

   imother  ifather  income  govtprog  POVERTY3  PDEN10  COUTYP4
0        1        1       4         2         3       2        2
1        1        1       4         2         3       1        1
2        1        1       4         1         3       1        1
3        1        1       2         2         1       2        2
4        1        1       4         2         3       2        2

[5 rows x 79 columns]
```

[4]:
```python
# Checking Column Names:
df.columns
```

[4]:
```
Index(['iralcfy', 'irmjfy', 'ircigfm', 'IRSMKLSS30N', 'iralcfm', 'irmjfm',
       'ircigage', 'irsmklsstry', 'iralcage', 'irmjage', 'mrjflag', 'alcflag',
       'tobflag', 'alcydays', 'mrjydays', 'alcmdays', 'mrjmdays', 'cigmdays',
       'smklsmdays', 'schfelt', 'tchgjob', 'avggrade', 'stndscig', 'stndsmj',
       'stndalc', 'stnddnk', 'parchkhw', 'parhlphw', 'PRCHORE2', 'PRLMTTV2',
       'parlmtsn', 'PRGDJOB2', 'PRPROUD2', 'argupar', 'YOFIGHT2', 'YOGRPFT2',
       'YOHGUN2', 'YOSELL2', 'YOSTOLE2', 'YOATTAK2', 'PRPKCIG2', 'PRMJEVR2',
       'prmjmo', 'PRALDLY2', 'YFLPKCG2', 'YFLTMRJ2', 'yflmjmo', 'YFLADLY2',
       'FRDPCIG2', 'FRDMEVR2', 'frdmjmon', 'FRDADLY2', 'talkprob', 'PRTALK3',
       'PRBSOLV2', 'PREVIOL2', 'PRVDRGO2', 'GRPCNSL2', 'PREGPGM2', 'YTHACT2',
       'DRPRVME3', 'ANYEDUC3', 'rlgattd', 'rlgimpt', 'rlgdcsn', 'rlgfrnd',
       'irsex', 'NEWRACE2', 'HEALTH2', 'eduschlgo', 'EDUSCHGRD2', 'eduskpcom',
       'imother', 'ifather', 'income', 'govtprog', 'POVERTY3', 'PDEN10',
       'COUTYP4'],
      dtype='object')
```

We can observe that the columns are not in same format, so lets make all the column names in lowercase, so that it would be easy to understand

[5]:
```python
df.columns = df.columns.str.lower()
```

Since I am working for Alcohol abuse, i will remove the columns related to Tobacco, Marijuana and Cigrattes. So that it becomes a little more clear to work on the rest of the dataset.

[6]:
```python
col = ['irmjfy', 'ircigfm', 'irsmklss30n', 'irmjfm', 'ircigage',␣
 ↪'irsmklsstry','irmjage', 'mrjflag', 'tobflag', 'mrjydays', 'mrjmdays',
       'cigmdays',␣
 ↪'smklsmdays','stndscig','stndsmj','prpkcig2','prmjevr2','prmjmo',␣
 ↪'yflpkcg2','yfltmrj2','yflmjmo', 'frdpcig2','frdmevr2',
      'frdmjmon']
```

```
df = df.drop(columns=col)
df.columns
```

[6]:
```
Index(['iralcfy', 'iralcfm', 'iralcage', 'alcflag', 'alcydays', 'alcmdays',
       'schfelt', 'tchgjob', 'avggrade', 'stndalc', 'stnddnk', 'parchkhw',
       'parhlphw', 'prchore2', 'prlmttv2', 'parlmtsn', 'prgdjob2', 'prproud2',
       'argupar', 'yofight2', 'yogrpft2', 'yohgun2', 'yosell2', 'yostole2',
       'yoattak2', 'praldly2', 'yfladly2', 'frdadly2', 'talkprob', 'prtalk3',
       'prbsolv2', 'previol2', 'prvdrgo2', 'grpcnsl2', 'pregpgm2', 'ythact2',
       'drprvme3', 'anyeduc3', 'rlgattd', 'rlgimpt', 'rlgdcsn', 'rlgfrnd',
       'irsex', 'newrace2', 'health2', 'eduschlgo', 'eduschgrd2', 'eduskpcom',
       'imother', 'ifather', 'income', 'govtprog', 'poverty3', 'pden10',
       'coutyp4'],
      dtype='object')
```

0.0.1 EDA

[7]:
```
# Shape of the dataset
df.shape
```

[7]:
```
(5500, 55)
```

[8]:
```
# Checking for NA values
NA_values = df.isna().sum()
NA_columns = NA_values[NA_values > 0]
NA_columns
```

[8]:
```
tchgjob      16
avggrade    353
stndalc     242
stnddnk     298
parchkhw     19
parhlphw     38
prchore2     18
prlmttv2     42
parlmtsn    132
prgdjob2     22
prproud2     21
argupar      87
yofight2     23
yogrpft2     29
yohgun2      23
yosell2      15
yostole2     13
yoattak2     13
praldly2     55
yfladly2     52
```

```
frdadly2     80
talkprob    146
prtalk3      90
prbsolv2    166
previol2     79
prvdrgo2     67
grpcnsl2     66
pregpgm2     51
ythact2      26
drprvme3     89
anyeduc3     78
rlgattd     135
rlgimpt     120
rlgdcsn     131
rlgfrnd     153
health2       1
dtype: int64
```

[9]:
```
# We can observe that we have many columns which has missing or NA values in␣
↪their dataset, so we will remove these NA values
df = df.dropna()
```

[10]:
```
NA_values = df.isna().sum()
NA_columns = NA_values[NA_values > 0]
NA_columns
```

[10]:
```
Series([], dtype: int64)
```

[11]:
```
# Now we can observe that there are no more NA values in our dataset and now␣
↪lets check the shape of the data.
df.shape
```

[11]:
```
(4347, 55)
```

Lets rename all the columns so that it would be easy to understand the dataset.

[12]:
```
col_names = {
    'iralcfy': 'Alc_Frq_Year',
    'iralcfm': 'Alc_Frq_Month',
    'iralcage': 'Alc_use_Age',
    'alcflag': 'Alc_Use',
    'alcydays': 'Alc_Last_Year',
    'alcmdays': 'Alc_Last_Month',
    'schfelt': 'Exp_of_School',
    'tchgjob': 'Teacher_Feedback',
    'avggrade': 'Last_Avg_Grade',
    'stndalc': 'Used_Alc',
```

```
    'stnddnk': 'Used_Alc_Week',
    'parchkhw': 'Parents_CheckHW_LastYear',
    'parhlphw': 'Parents_HelpHW_LastYear',
    'prchore2': 'Youth_doing_HChores',
    'prlmttv2': 'Parents_Limit_TV',
    'parlmtsn': 'Parents_Limit_Snacks',
    'prgdjob2': 'Parents_Appreciation',
    'prproud2': 'Proud_Parents',
    'argupar': 'Argument_Parents',
    'yofight2': 'Youth_Fight',
    'yogrpft2': 'Youth_Group_Fight',
    'yohgun2': 'Youth_have_Gun',
    'yosell2': 'Youth_Sell_Drugs',
    'yostole2': 'Youth_Steals',
    'yoattak2': 'Youth_Attacked',
    'praldly2': 'Parent Alcohol Daily',
    'yfladly2': 'Peer_Drinking',
    'frdadly2': 'Friend_Feedback',
    'talkprob': 'Share_Problems',
    'prtalk3': 'Talked_with_Parents',
    'prbsolv2': 'Part_Extracurricular',
    'previol2': 'Part_Violence_Prevention',
    'prvdrgo2': 'Part_Substance_Prevention',
    'grpcnsl2': 'Part_Help_Substance_Use',
    'pregpgm2': 'Part_Preg/STD_Prevention',
    'ythact2': 'Part_Youth_Act',
    'drprvme3': 'Yth_seen_alc+drug_prevention_ad',
    'anyeduc3': 'Education_On_alc+drug',
    'rlgattd': 'Number_Religion_Attend',
    'rlgimpt': 'Yth_Believe_Religion Imp',
    'rlgdcsn': 'Religion_Influence',
    'rlgfrnd': 'Religius_Friend',
    'irsex': 'Gender',
    'newrace2': 'Race',
    'health2': 'Health_Condition',
    'eduschlgo': 'Attending_School',
    'eduschgrd2': 'Educlation_Level',
    'eduskpcom': 'School_Skipped',
    'imother': 'Individual_Mother',
    'ifather': 'Individual_Father',
    'income': 'Income',
    'govtprog': 'Part_Gov_Program',
    'poverty3': 'Poverty_Level',
    'pden10': 'Population Density',
    'coutyp4': 'Metro_Size'
}
df = df.rename(columns=col_names)
```

[13]:
```
df.columns
```

[13]:
```
Index(['Alc_Frq_Year', 'Alc_Frq_Month', 'Alc_use_Age', 'Alc_Use',
       'Alc_Last_Year', 'Alc_Last_Month', 'Exp_of_School', 'Teacher_Feedback',
       'Last_Avg_Grade', 'Used_Alc', 'Used_Alc_Week',
       'Parents_CheckHW_LastYear', 'Parents_HelpHW_LastYear',
       'Youth_doing_HChores', 'Parents_Limit_TV', 'Parents_Limit_Snacks',
       'Parents_Appreciation', 'Proud_Parents', 'Argument_Parents',
       'Youth_Fight', 'Youth_Group_Fight', 'Youth_have_Gun',
       'Youth_Sell_Drugs', 'Youth_Steals', 'Youth_Attacked',
       'Parent Alcohol Daily', 'Peer_Drinking', 'Friend_Feedback',
       'Share_Problems', 'Talked_with_Parents', 'Part_Extracurricular',
       'Part_Violence_Prevention', 'Part_Substance_Prevention',
       'Part_Help_Substance_Use', 'Part_Preg/STD_Prevention', 'Part_Youth_Act',
       'Yth_seen_alc+drug_prevention_ad', 'Education_On_alc+drug',
       'Number_Religion_Attend', 'Yth_Believe_Religion Imp',
       'Religion_Influence', 'Religius_Friend', 'Gender', 'Race',
       'Health_Condition', 'Attending_School', 'Educlation_Level',
       'School_Skipped', 'Individual_Mother', 'Individual_Father', 'Income',
       'Part_Gov_Program', 'Poverty_Level', 'Population Density',
       'Metro_Size'],
      dtype='object')
```

[14]:
```
# Now lets check the different values in the variables
for col in df.columns:
    print(f"Column: {col}")
    print(df[col].value_counts())
    print("\n")
```

```
Column: Alc_Frq_Year
991    3171
993     217
1       144
2       112
3        92
       …
61        1
45        1
37        1
47        1
82        1
Name: Alc_Frq_Year, Length: 80, dtype: int64


Column: Alc_Frq_Month
91.0    3171
93.0     756
```

```
1.0      160
2.0      105
3.0       52
4.0       28
5.0       16
6.0       13
10.0       9
7.0        9
13.0       5
9.0        5
1.5        3
15.0       3
14.0       3
8.0        3
23.0       2
18.0       1
12.0       1
20.0       1
11.0       1
Name: Alc_Frq_Month, dtype: int64


Column: Alc_use_Age
991    3171
15      297
14      258
13      192
16      152
12       88
17       49
11       47
10       31
9        16
8        15
7        12
6         8
5         4
4         3
3         2
2         1
1         1
Name: Alc_use_Age, dtype: int64


Column: Alc_Use
0    3171
1    1176
Name: Alc_Use, dtype: int64
```

```
Column: Alc_Last_Year
6    3388
1     579
2     239
3      84
4      56
5       1
Name: Alc_Last_Year, dtype: int64


Column: Alc_Last_Month
5    3927
1     268
2      96
3      53
4       3
Name: Alc_Last_Month, dtype: int64


Column: Exp_of_School
1    3153
2    1194
Name: Exp_of_School, dtype: int64


Column: Teacher_Feedback
1.0    3198
2.0    1149
Name: Teacher_Feedback, dtype: int64


Column: Last_Avg_Grade
2.0    4171
1.0     176
Name: Last_Avg_Grade, dtype: int64


Column: Used_Alc
2.0    3145
1.0    1202
Name: Used_Alc, dtype: int64


Column: Used_Alc_Week
2.0    3981
1.0     366
```

```
Name: Used_Alc_Week, dtype: int64


Column: Parents_CheckHW_LastYear
1.0    3569
2.0     778
Name: Parents_CheckHW_LastYear, dtype: int64


Column: Parents_HelpHW_LastYear
1.0    3467
2.0     880
Name: Parents_HelpHW_LastYear, dtype: int64


Column: Youth_doing_HChores
1.0    3898
2.0     449
Name: Youth_doing_HChores, dtype: int64


Column: Parents_Limit_TV
2.0    2488
1.0    1859
Name: Parents_Limit_TV, dtype: int64


Column: Parents_Limit_Snacks
1.0    2800
2.0    1547
Name: Parents_Limit_Snacks, dtype: int64


Column: Parents_Appreciation
1.0    3728
2.0     619
Name: Parents_Appreciation, dtype: int64


Column: Proud_Parents
1.0    3655
2.0     692
Name: Proud_Parents, dtype: int64


Column: Argument_Parents
1.0    3448
2.0     899
```

```
Name: Argument_Parents, dtype: int64


Column: Youth_Fight
2.0    3739
1.0     608
Name: Youth_Fight, dtype: int64


Column: Youth_Group_Fight
2.0    3929
1.0     418
Name: Youth_Group_Fight, dtype: int64


Column: Youth_have_Gun
2.0    4148
1.0     199
Name: Youth_have_Gun, dtype: int64


Column: Youth_Sell_Drugs
2.0    4289
1.0      58
Name: Youth_Sell_Drugs, dtype: int64


Column: Youth_Steals
2.0    4214
1.0     133
Name: Youth_Steals, dtype: int64


Column: Youth_Attacked
2.0    4160
1.0     187
Name: Youth_Attacked, dtype: int64


Column: Parent Alcohol Daily
1.0    3996
2.0     351
Name: Parent Alcohol Daily, dtype: int64


Column: Peer_Drinking
1.0    3976
2.0     371
```

```
Name: Peer_Drinking, dtype: int64


Column: Friend_Feedback
1.0    3953
2.0     394
Name: Friend_Feedback, dtype: int64


Column: Share_Problems
2.0    4122
1.0     225
Name: Share_Problems, dtype: int64


Column: Talked_with_Parents
1.0    2693
2.0    1654
Name: Talked_with_Parents, dtype: int64


Column: Part_Extracurricular
2.0    3162
1.0    1185
Name: Part_Extracurricular, dtype: int64


Column: Part_Violence_Prevention
2.0    3930
1.0     417
Name: Part_Violence_Prevention, dtype: int64


Column: Part_Substance_Prevention
2.0    3903
1.0     444
Name: Part_Substance_Prevention, dtype: int64


Column: Part_Help_Substance_Use
2.0    4189
1.0     158
Name: Part_Help_Substance_Use, dtype: int64


Column: Part_Preg/STD_Prevention
2.0    4116
1.0     231
```

```
Name: Part_Preg/STD_Prevention, dtype: int64


Column: Part_Youth_Act
2.0    3732
1.0     615
Name: Part_Youth_Act, dtype: int64


Column: Yth_seen_alc+drug_prevention_ad
1.0    3229
2.0    1118
Name: Yth_seen_alc+drug_prevention_ad, dtype: int64


Column: Education_On_alc+drug
1.0    3128
2.0    1219
Name: Education_On_alc+drug, dtype: int64


Column: Number_Religion_Attend
2.0    3150
1.0    1197
Name: Number_Religion_Attend, dtype: int64


Column: Yth_Believe_Religion Imp
1.0    2779
2.0    1568
Name: Yth_Believe_Religion Imp, dtype: int64


Column: Religion_Influence
1.0    2444
2.0    1903
Name: Religion_Influence, dtype: int64


Column: Religius_Friend
2.0    3281
1.0    1066
Name: Religius_Friend, dtype: int64


Column: Gender
1    2193
2    2154
```

```
Name: Gender, dtype: int64


Column: Race
1    2502
7     848
2     431
6     263
5     237
3      51
4      15
Name: Race, dtype: int64


Column: Health_Condition
2.0    1789
1.0    1514
3.0     863
4.0     181
Name: Health_Condition, dtype: int64


Column: Attending_School
1     3770
2      543
94      20
98       5
97       5
11       2
85       2
Name: Attending_School, dtype: int64


Column: Educlation_Level
6     687
5     676
4     633
7     618
3     588
99    543
8     355
2     183
98     40
1      16
9       7
10      1
Name: Educlation_Level, dtype: int64
```

```
Column: School_Skipped
0     2870
99     778
1      267
2      140
98      68
3       67
4       42
5       27
10      14
7       13
15       9
9        8
6        7
94       6
8        5
18       4
12       4
20       3
14       2
97       2
13       2
30       2
22       2
25       2
11       1
16       1
23       1
Name: School_Skipped, dtype: int64


Column: Individual_Mother
1    4029
2     304
3      14
Name: Individual_Mother, dtype: int64


Column: Individual_Father
1    3342
2     986
3      19
Name: Individual_Father, dtype: int64


Column: Income
4    2368
```

```
2     918
3     588
1     473
Name: Income, dtype: int64


Column: Part_Gov_Program
2    3554
1     793
Name: Part_Gov_Program, dtype: int64


Column: Poverty_Level
3    2927
2     771
1     649
Name: Poverty_Level, dtype: int64


Column: Population Density
2    2255
1    1790
3     302
Name: Population Density, dtype: int64


Column: Metro_Size
1    1928
2    1596
3     823
Name: Metro_Size, dtype: int64
```

Now we will redo the allignments in some of the columns where we are getting some values as 991,993,85,94,97,98,99 and other odd values.

First Columns: Alc__Frq__Month

In this column the data provided is of alcohol frequncy last month. In we have 91 and 93 as never used and did not use last month which i am going assign as 0 as its have not been used for last month or ever. Lastly I will divide the days into as per quarter like 1 to 90 is 1, 91 to 180 as 2, 181 to 270 as 3 and 271 and above as 4.

[15]:
```
def reassign_1(value):
    if value in [991, 993]:
        return 0
    elif 1 <= value <= 90:
```

```
        return 1
    elif 91 <= value <= 180:
        return 2
    elif 181 <= value <= 270:
        return 3
    else:
        return 4

df['Alc_Frq_Year'] = df['Alc_Frq_Year'].apply(reassign_1)
print(df['Alc_Frq_Year'].value_counts())
```

```
0    3388
1     894
2      51
3      12
4       2
Name: Alc_Frq_Year, dtype: int64
```

Second Columns: Alc_Frq_Month

In this column the data provided is of alcohol frequncy last month. In we have 91 and 93 as never used and did not use last month which i am going assign as 0 as its have not been used for last month or ever. This varaiable is our Regression model's target variable.

[16]:
```
def reassign_2(value):
    if value in [91, 93]:
        return 0
    return value

df['Alc_Frq_Month'] = df['Alc_Frq_Month'].apply(reassign_2)

print(df['Alc_Frq_Month'].value_counts())
```

```
0.0     3927
1.0      160
2.0      105
3.0       52
4.0       28
5.0       16
6.0       13
10.0       9
7.0        9
13.0       5
9.0        5
1.5        3
15.0       3
14.0       3
```

```
8.0        3
23.0       2
18.0       1
12.0       1
20.0       1
11.0       1
Name: Alc_Frq_Month, dtype: int64
```

Third Columns: Alc_use_Age

In this column the data provided is of alcohol used at which age. Here we have some intersting data, we have seen some values also assigned where the age is even 1. We have never used as 991 in our data. Since the numbers on age between 1 to 5 is less and also it does not make much impact or even sence to have these numbers, so we will convert the age less than 6 and 991 as 0, 6 to 12 as 1 and 12 + as 2.

[17]:
```python
def reassign_3(value):
    if value <= 6 or value == 991:
        return 0
    elif value <= 12:
        return 1
    else:
        return 2

df['Alc_use_Age'] = df['Alc_use_Age'].apply(reassign_3)
print(df['Alc_use_Age'].value_counts())
```

```
0    3190
2     948
1     209
Name: Alc_use_Age, dtype: int64
```

Fourth Columns: Attending_School

In this column the data provided is of the youth is attending the school or not. The data has some values like 85, 97,98 and 99 which tells that either its bad data or refued or blank or left blank and its numbers are much less so we will assign these under No as 2. 11 provides details where the youth is going to school but is not regular and missing the school. So we can assign that 11 to 1.

[18]:
```python
def reassign_4(value):
    if value in [85, 94, 97, 98]:
        return 2
    elif value == 11:
        return 1
    else:
        return value

df['Attending_School'] = df['Attending_School'].apply(reassign_4)
```

```python
print(df['Attending_School'].value_counts())
```

```
1    3772
2     575
Name: Attending_School, dtype: int64
```

Fifth Columns: Educlation_Level

In this column the data provided is of the level of Education a youth is studying currently. We have differnt valaues ranging from less than 5, 6th, 7th, 8th and till college. We also have 98 and 99 which tells either its blank or left on purpose. So in this part we will reassign values where 98 and 99 will be counted as 0, less 5th Grade will assigned as 1 as its already have been. We will assign value 2 for classes 6th to 12th as 2 and for College as 3.

[19]:
```python
def reassign_5(value):
    if value in [99, 98]:
        return 0
    elif value < 5:
        return 1
    elif value < 9:
        return 2
    else:
        return 3

df['Educlation_Level'] = df['Educlation_Level'].apply(reassign_5)
print(df['Educlation_Level'].value_counts())
```

```
2    2336
1    1420
0     583
3       8
Name: Educlation_Level, dtype: int64
```

Sixth Columns: School_Skipped

In this column the data provided speaks about the number of days youth has skipped school and there are some values like 94, 97,98,99 which is alternative of either blank, skip, dont't know or refusd to say anything. These will be assignment a value of 5. While we will divide this data into 1 to 7 as 1, 8 to 14 as 2, 15 to 21 as 3 and 22 to 30 as 4.

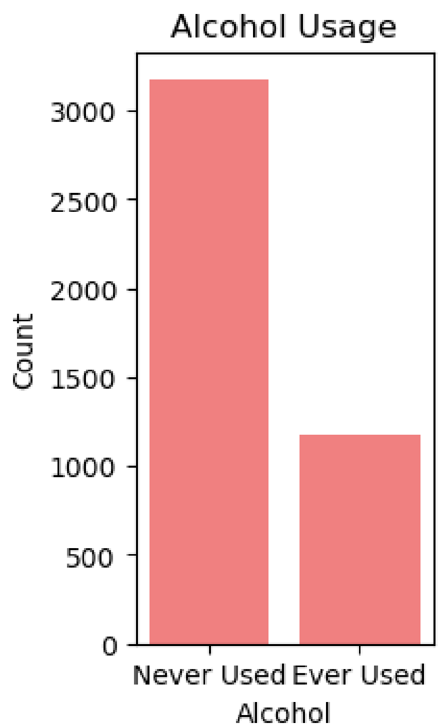
[20]:
```python
def reassign_6(value):
    if value in [94, 97, 98, 99]:
        return 5
    elif value == 0:
        return 0
    elif value <= 7:
```

```
        return 1
    elif value <= 14:
        return 2
    elif value <= 21:
        return 3
    elif value <= 30:
        return 4
    else:
        return None

df['School_Skipped'] = df['School_Skipped'].apply(reassign_6)
print(df['School_Skipped'].value_counts())
```

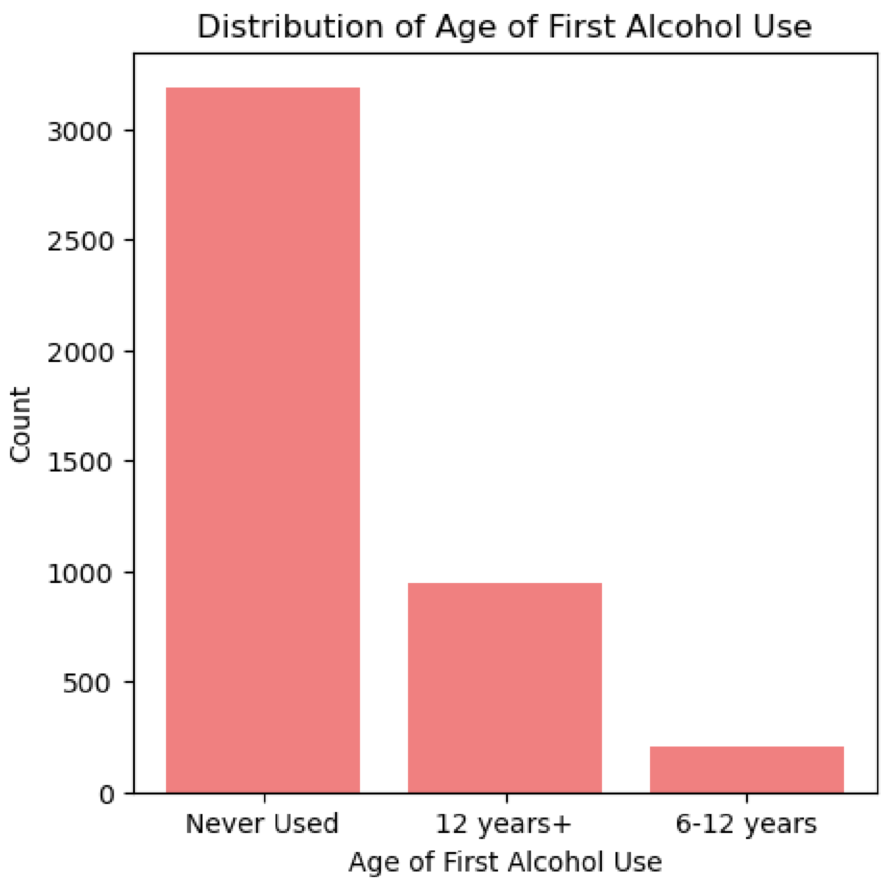
```
0    2870
5     854
1     563
2      36
3      17
4       7
Name: School_Skipped, dtype: int64
```

[21]:

```
# Lets do some feature checks with Graphs
plt.figure(figsize=(2, 4))
plt.bar(['Never Used', 'Ever Used'], df['Alc_Use'].map({0: 'Never Used', 1:␣
 ↪'Ever Used'}).value_counts(), color='lightcoral')
plt.title('Alcohol Usage')
plt.xlabel('Alcohol')
plt.ylabel('Count')
plt.show()
```



[22]:
```
age_labels = {0: 'Never Used', 1: '6-12 years', 2: '12 years+'}
plt.figure(figsize=(5, 5))
plt.bar(df['Alc_use_Age'].map(age_labels).value_counts().index,␣
↪df['Alc_use_Age'].value_counts(), color='lightcoral')
plt.title('Distribution of Age of First Alcohol Use')
plt.xlabel('Age of First Alcohol Use')
plt.ylabel('Count')
plt.show()
```



0.0.2 Model Building

Now its all the columns are perfectly alligned and we can now proceed to do Binary Classification. For this we are considering the target varaiable as Alc_Use. Some of the substance columns are not required for our Binary classification we will create a new dataset for this porpose.

[23]:
```
col_not_required = ['Alc_Frq_Year', 'Alc_Frq_Month', 'Alc_Last_Year',␣
↪'Alc_use_Age', 'Alc_Last_Month', 'Used_Alc']
binary_df = df.drop(columns=col_not_required)
print(binary_df.head())
```

```
   Alc_Use  Exp_of_School  Teacher_Feedback  Last_Avg_Grade  Used_Alc_Week  \
0        1              1               1.0             2.0            2.0
1        0              1               1.0             2.0            2.0
2        1              1               1.0             2.0            2.0
3        0              1               1.0             2.0            2.0
5        0              1               1.0             2.0            2.0

   Parents_CheckHW_LastYear  Parents_HelpHW_LastYear  Youth_doing_HChores  \
```

```
0                       1.0                      1.0                  2.0
1                       1.0                      2.0                  1.0
2                       1.0                      1.0                  1.0
3                       1.0                      1.0                  1.0
5                       1.0                      2.0                  1.0

   Parents_Limit_TV  Parents_Limit_Snacks  …  Attending_School  \
0               2.0                   2.0  …                 1
1               2.0                   2.0  …                 1
2               2.0                   1.0  …                 1
3               1.0                   2.0  …                 1
5               1.0                   2.0  …                 1

   Educlation_Level  School_Skipped  Individual_Mother  Individual_Father  \
0                 2               0                  1                  1
1                 2               0                  1                  1
2                 1               0                  1                  1
3                 2               0                  1                  1
5                 2               0                  1                  2

   Income  Part_Gov_Program  Poverty_Level  Population Density  Metro_Size
0       4                 2              3                   2           2
1       4                 2              3                   1           1
2       4                 1              3                   1           1
3       2                 2              1                   2           2
5       4                 2              3                   1           1

[5 rows x 49 columns]
```

Let's start our first model Decision Tree Classifier.

[24]:
```python
X = binary_df.drop('Alc_Use', axis=1)
y = binary_df['Alc_Use']
```

[25]:
```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30,␣
↪random_state=1)
```

[26]:
```python
param_grid = {
    'max_depth': [3, 5, 7, 10],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
}
```

[27]:
```python
clf = DecisionTreeClassifier()
```

[28]:
```python
grid_search = GridSearchCV(estimator=clf,
                           param_grid=param_grid,
                           scoring='accuracy',
```

```
                              cv=5,
                              verbose=1,
                              n_jobs=-1)
```

[29]:
```
grid_search.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 36 candidates, totalling 180 fits
```

[29]:
```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(), n_jobs=-1,
             param_grid={'max_depth': [3, 5, 7, 10],
                         'min_samples_leaf': [1, 2, 4],
                         'min_samples_split': [2, 5, 10]},
             scoring='accuracy', verbose=1)
```

[30]:
```
best_params_dt = grid_search.best_params_
best_score_dt = grid_search.best_score_
print("Best Parameters:\n", best_params_dt)
print("Best Accuracy:\n", best_score_dt)
```

```
Best Parameters:
 {'max_depth': 5, 'min_samples_leaf': 4, 'min_samples_split': 5}
Best Accuracy:
 0.745560560884971
```

[31]:
```
best_clf_dt = grid_search.best_estimator_
best_clf_dt.fit(X_train, y_train)
```

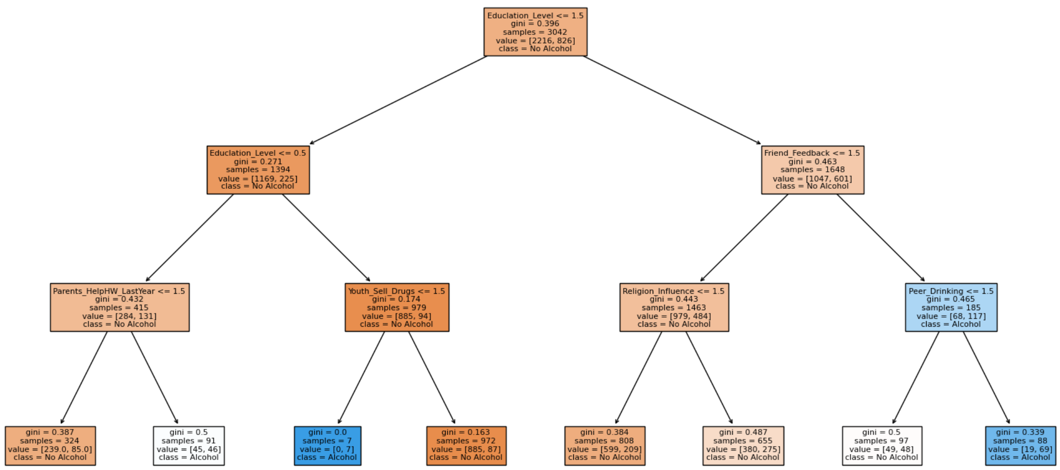
[31]:
```
DecisionTreeClassifier(max_depth=5, min_samples_leaf=4, min_samples_split=5)
```

[32]:
```
y_pred = best_clf_dt.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)
print('Accuracy:', accuracy)
print('Confusion Matrix:\n', conf_matrix)
print('Classification Report:\n', classification_rep)
```

```
Accuracy: 0.7448275862068966
Confusion Matrix:
 [[854 101]
 [232 118]]
Classification Report:
               precision    recall  f1-score   support

           0       0.79      0.89      0.84       955
           1       0.54      0.34      0.41       350

    accuracy                           0.74      1305
```

```
   macro avg       0.66      0.62      0.63      1305
weighted avg       0.72      0.74      0.72      1305
```

[33]:
```
clf_prun_DT = DecisionTreeClassifier(max_depth=3)
clf_prun_DT.fit(X_train, y_train)
plt.figure(figsize=(20, 10))
plot_tree(clf_prun_DT, filled=True, feature_names=X.columns, class_names=['No␣
↪Alcohol', 'Alcohol'])
plt.show()
```



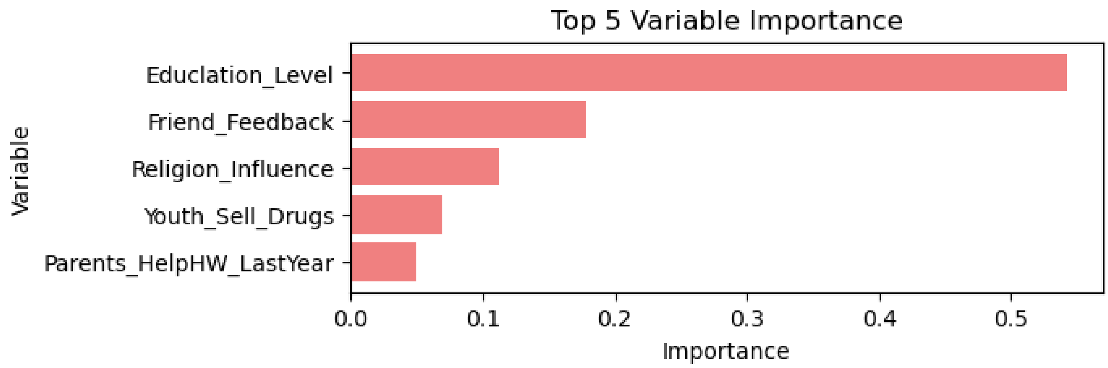
[34]:
```
var_importance = clf_prun_DT.feature_importances_
var_importance_df = pd.DataFrame({'Variable': X.columns, 'Importance':␣
↪var_importance})
var_importance_df = var_importance_df.sort_values(by='Importance',␣
↪ascending=False)
print(var_importance_df)
```

```
                         Variable  Importance
39                Educlation_Level    0.542596
20                 Friend_Feedback    0.179072
33              Religion_Influence    0.112653
15                Youth_Sell_Drugs    0.069060
5          Parents_HelpHW_LastYear    0.050347
19                   Peer_Drinking    0.046271
0                    Exp_of_School    0.000000
35                          Gender    0.000000
28                   Part_Youth_Act    0.000000
29  Yth_seen_alc+drug_prevention_ad    0.000000
```

```
30              Education_On_alc+drug     0.000000
31             Number_Religion_Attend     0.000000
32           Yth_Believe_Religion Imp     0.000000
34                    Religius_Friend     0.000000
36                               Race     0.000000
26             Part_Help_Substance_Use    0.000000
37                   Health_Condition     0.000000
38                   Attending_School     0.000000
40                     School_Skipped     0.000000
41                  Individual_Mother     0.000000
42                  Individual_Father     0.000000
43                             Income     0.000000
44                   Part_Gov_Program     0.000000
45                      Poverty_Level     0.000000
46                 Population Density     0.000000
27           Part_Preg/STD_Prevention     0.000000
24           Part_Violence_Prevention     0.000000
25         Part_Substance_Prevention      0.000000
11                   Argument_Parents     0.000000
2                      Last_Avg_Grade     0.000000
3                       Used_Alc_Week     0.000000
4            Parents_CheckHW_LastYear     0.000000
6                 Youth_doing_HChores     0.000000
7                    Parents_Limit_TV     0.000000
8                Parents_Limit_Snacks     0.000000
9               Parents_Appreciation      0.000000
10                     Proud_Parents      0.000000
12                       Youth_Fight      0.000000
1                   Teacher_Feedback      0.000000
13                 Youth_Group_Fight      0.000000
14                    Youth_have_Gun      0.000000
16                      Youth_Steals      0.000000
17                    Youth_Attacked      0.000000
18              Parent Alcohol Daily      0.000000
21                    Share_Problems      0.000000
22               Talked_with_Parents      0.000000
23               Part_Extracurricular     0.000000
47                        Metro_Size      0.000000
```

[35]:
```python
var_importance_5 = var_importance_df.head(5)
plt.figure(figsize=(6, 2))
plt.barh(var_importance_5['Variable'], var_importance_5['Importance'],
 ↪color='lightcoral')
plt.xlabel('Importance')
plt.ylabel('Variable')
plt.title('Top 5 Variable Importance')
plt.gca().invert_yaxis()
```

```
plt.show()
```



Now let's move to Bagging method

[36]:
```
base_classifier = RandomForestClassifier()
```

[37]:
```
param_grid = {
    'n_estimators': [30, 40, 50],
    'max_samples': [0.5, 0.7, 1.0]
}
```

[38]:
```
bagging_classifier = BaggingClassifier(base_classifier, random_state=42)
```

[39]:
```
grid_search = GridSearchCV(estimator=bagging_classifier,
                           param_grid=param_grid,
                           scoring='accuracy',
                           cv=5,
                           verbose=1,
                           n_jobs=-1)

grid_search.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 9 candidates, totalling 45 fits
```

[39]:
```
GridSearchCV(cv=5,
             estimator=BaggingClassifier(estimator=RandomForestClassifier(),
                                         random_state=42),
             n_jobs=-1,
             param_grid={'max_samples': [0.5, 0.7, 1.0],
                         'n_estimators': [30, 40, 50]},
             scoring='accuracy', verbose=1)
```

[40]:
```
best_params_bg = grid_search.best_params_
best_score_bg = grid_search.best_score_
print("Best Parameters:\n", best_params_bg)
print("Best Accuracy:\n", best_score_bg)
```

```
Best Parameters:
 {'max_samples': 0.5, 'n_estimators': 30}
Best Accuracy:
 0.76233309567021
```

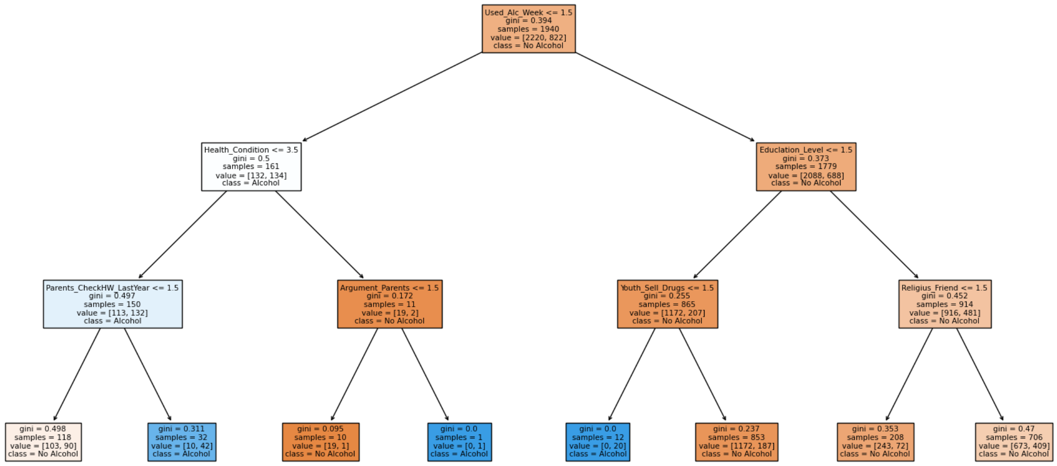
[41]:
```
best_bagging_classifier = grid_search.best_estimator_
best_bagging_classifier.fit(X_train, y_train)
```

[41]:
```
BaggingClassifier(estimator=RandomForestClassifier(), max_samples=0.5,
                  n_estimators=30, random_state=42)
```

[42]:
```
y_pred_bag = best_bagging_classifier.predict(X_test)
Accu_bag = accuracy_score(y_test, y_pred_bag)
conf_matrix_bag = confusion_matrix(y_test, y_pred_bag)
class_rep_bag = classification_report(y_test, y_pred_bag)
print('Bagging Accuracy:', Accu_bag)
print('Confusion Matrix with Bagging:\n', conf_matrix_bag)
print('Classification Report with Bagging:\n', class_rep_bag)
```

```
Bagging Accuracy: 0.7639846743295019
Confusion Matrix with Bagging:
 [[916  39]
 [269  81]]
Classification Report with Bagging:
              precision    recall  f1-score   support

           0       0.77      0.96      0.86       955
           1       0.68      0.23      0.34       350

    accuracy                           0.76      1305
   macro avg       0.72      0.60      0.60      1305
weighted avg       0.75      0.76      0.72      1305
```

[43]:
```
bag_tree = RandomForestClassifier(max_depth=3, n_estimators=1, random_state=42)
bag_tree.fit(X_train, y_train)
plt.figure(figsize=(20, 10))
plot_tree(bag_tree.estimators_[0], filled=True, feature_names=X.columns,␣
 ↪class_names=['No Alcohol', 'Alcohol'])
plt.show()
```



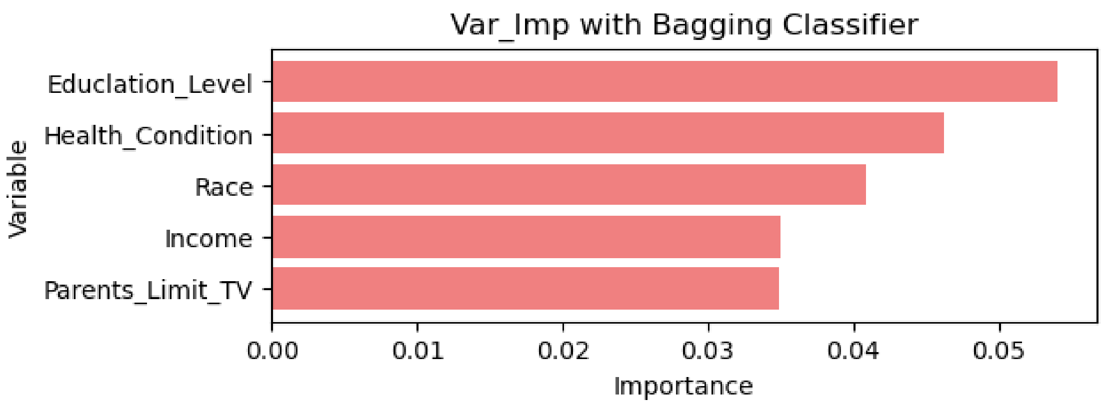
[44]:
```python
var_imp_bag = np.mean([tree.feature_importances_ for tree in␣
↪best_bagging_classifier.estimators_], axis=0)
var_imp_df_bag = pd.DataFrame({'Variable': X.columns, 'Importance':␣
↪var_imp_bag})
var_imp_df_bag = var_imp_df_bag.sort_values(by='Importance', ascending=False)
print('Variable Importance with Bagging Classifier:\n', var_imp_df_bag)
```

```
Variable Importance with Bagging Classifier:
                            Variable  Importance
39                  Educlation_Level    0.054045
37                  Health_Condition    0.046170
36                              Race    0.040823
43                            Income    0.034958
7                   Parents_Limit_TV    0.034871
47                        Metro_Size    0.034359
40                    School_Skipped    0.033060
46                Population Density    0.029208
35                            Gender    0.028149
8               Parents_Limit_Snacks    0.025627
22               Talked_with_Parents    0.024898
33                Religion_Influence    0.024843
0                     Exp_of_School    0.024802
32           Yth_Believe_Religion Imp    0.023882
20                   Friend_Feedback    0.023836
11                  Argument_Parents    0.023740
4           Parents_CheckHW_LastYear    0.023610
5            Parents_HelpHW_LastYear    0.023394
45                     Poverty_Level    0.023377
1                   Teacher_Feedback    0.023006
```

```
19                   Peer_Drinking    0.022524
30             Education_On_alc+drug    0.022439
23              Part_Extracurricular    0.021615
31             Number_Religion_Attend    0.021512
3                     Used_Alc_Week    0.021423
29  Yth_seen_alc+drug_prevention_ad    0.020292
34                  Religius_Friend    0.018098
42                Individual_Father    0.017961
10                   Proud_Parents    0.017552
9             Parents_Appreciation    0.017400
18              Parent Alcohol Daily    0.017370
12                     Youth_Fight    0.016853
13               Youth_Group_Fight    0.015907
28                  Part_Youth_Act    0.014449
6              Youth_doing_HChores    0.013716
44                 Part_Gov_Program    0.013626
25        Part_Substance_Prevention    0.011780
41                Individual_Mother    0.011073
16                     Youth_Steals    0.010852
15                 Youth_Sell_Drugs    0.010108
21                   Share_Problems    0.009584
38                 Attending_School    0.009491
14                   Youth_have_Gun    0.008998
24         Part_Violence_Prevention    0.008599
2                    Last_Avg_Grade    0.007317
17                   Youth_Attacked    0.007090
27         Part_Preg/STD_Prevention    0.006434
26          Part_Help_Substance_Use    0.005276
```

[45]:

```python
var_bag_imp = var_imp_df_bag.head(5)
plt.figure(figsize=(6, 2))
plt.barh(var_bag_imp['Variable'], var_bag_imp['Importance'], color='lightcoral')
plt.xlabel('Importance')
plt.ylabel('Variable')
plt.title('Var_Imp with Bagging Classifier')
plt.gca().invert_yaxis()
plt.show()
```



Now Let's check the accuracy and other parameters as per Random Forest Classifier

[46]:
```
param_grid_rf = {
    'n_estimators': [30, 40, 50],
    'max_depth': [3, 5, 7, 10],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
```

[47]:
```
rf_classifier = RandomForestClassifier()
```

[48]:
```
grid_search_rf = GridSearchCV(estimator=rf_classifier,
                              param_grid=param_grid_rf,
                              scoring='accuracy',
                              cv=5,
                              verbose=1,
                              n_jobs=-1)
```

[49]:
```
grid_search_rf.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 108 candidates, totalling 540 fits
```

[49]:
```
GridSearchCV(cv=5, estimator=RandomForestClassifier(), n_jobs=-1,
             param_grid={'max_depth': [3, 5, 7, 10],
                         'min_samples_leaf': [1, 2, 4],
                         'min_samples_split': [2, 5, 10],
                         'n_estimators': [30, 40, 50]},
             scoring='accuracy', verbose=1)
```

[50]:
```
best_params_rf = grid_search_rf.best_params_
best_score_rf = grid_search_rf.best_score_
print("Best Parameters:\n", best_params_rf)
print("Best Accuracy:\n", best_score_rf)
```

```
Best Parameters:
 {'max_depth': 10, 'min_samples_leaf': 2, 'min_samples_split': 2,
'n_estimators': 40}
Best Accuracy:
 0.7682455060063953
```

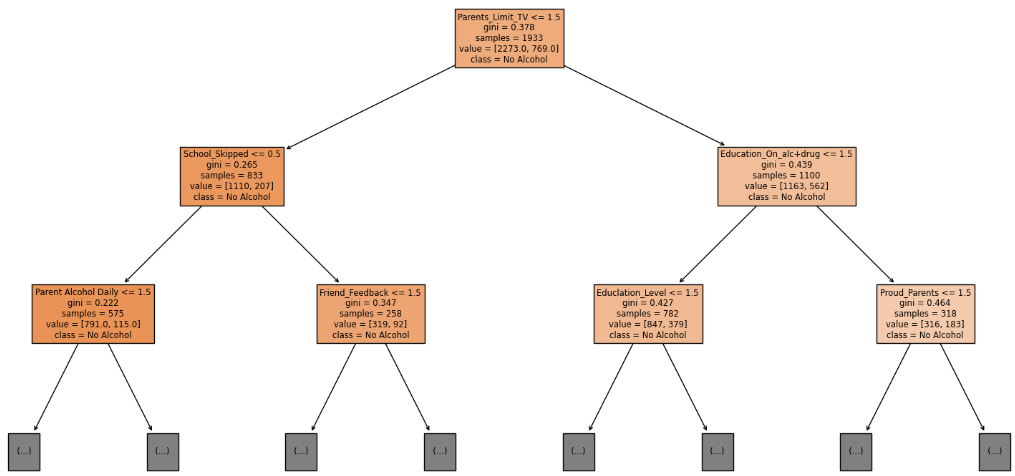
[51]:
```
best_rf_classifier = grid_search_rf.best_estimator_
best_rf_classifier.fit(X_train, y_train)
```

[51]:
```
RandomForestClassifier(max_depth=10, min_samples_leaf=2, n_estimators=40)
```

[52]:
```
y_pred_rf = best_rf_classifier.predict(X_test)
rf_accuracy = accuracy_score(y_test, y_pred_rf)
rf_conf_matrix = confusion_matrix(y_test, y_pred_rf)
rf_class_rep = classification_report(y_test, y_pred_rf)
print('RF Accuracy:', rf_accuracy)
print('RF Confusion Matrix:\n', rf_conf_matrix)
print('RF Classification Report:\n', rf_class_rep)
```

```
RF Accuracy: 0.7547892720306514
RF Confusion Matrix:
 [[913  42]
 [278  72]]
RF Classification Report:
               precision    recall  f1-score   support

           0       0.77      0.96      0.85       955
           1       0.63      0.21      0.31       350

    accuracy                           0.75      1305
   macro avg       0.70      0.58      0.58      1305
weighted avg       0.73      0.75      0.71      1305
```

[53]:
```
plt.figure(figsize=(20, 10))
plot_tree(best_rf_classifier.estimators_[0], filled=True, feature_names=X.
 ↪columns, class_names=['No Alcohol', 'Alcohol'], max_depth=2)
plt.show()
```



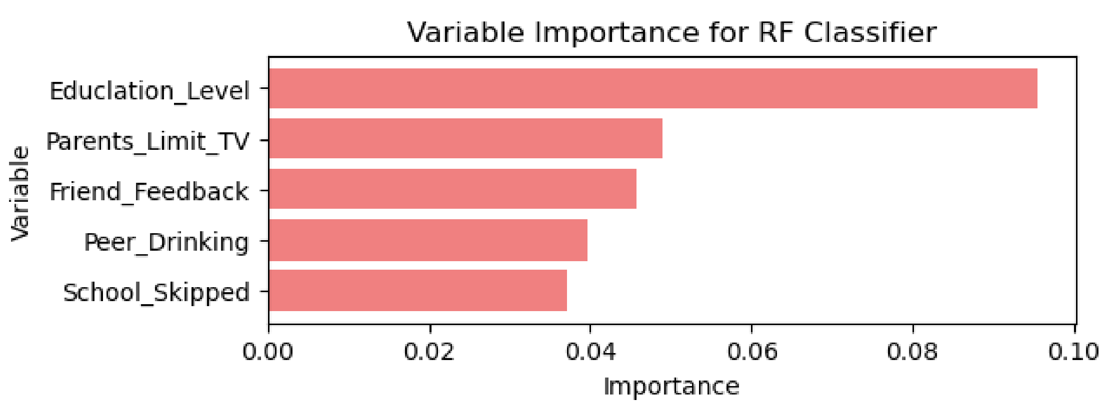
[54]:
```python
rf_var_imp = best_rf_classifier.feature_importances_
rf_var_imp_df = pd.DataFrame({'Variable': X.columns, 'Importance': rf_var_imp})
rf_var_imp_df = rf_var_imp_df.sort_values(by='Importance', ascending=False)
rf_var_imp_df
```

[54]:

	Variable	Importance
39	Educlation_Level	0.095450
7	Parents_Limit_TV	0.048970
20	Friend_Feedback	0.045746
19	Peer_Drinking	0.039774
40	School_Skipped	0.037226
36	Race	0.033842
4	Parents_CheckHW_LastYear	0.030724
3	Used_Alc_Week	0.029285
43	Income	0.029081
37	Health_Condition	0.028832
15	Youth_Sell_Drugs	0.027455
33	Religion_Influence	0.027097
11	Argument_Parents	0.026226
32	Yth_Believe_Religion Imp	0.025501
5	Parents_HelpHW_LastYear	0.024181
46	Population Density	0.023890
47	Metro_Size	0.023723
18	Parent Alcohol Daily	0.023566
45	Poverty_Level	0.022381
23	Part_Extracurricular	0.019883
0	Exp_of_School	0.018909
8	Parents_Limit_Snacks	0.017977

```
9             Parents_Appreciation    0.016816
1                 Teacher_Feedback    0.016810
35                          Gender    0.016783
31          Number_Religion_Attend    0.016420
13               Youth_Group_Fight    0.016331
12                     Youth_Fight    0.015226
29  Yth_seen_alc+drug_prevention_ad    0.015179
34                 Religius_Friend    0.014789
16                    Youth_Steals    0.014709
10                  Proud_Parents    0.014171
30            Education_On_alc+drug    0.013925
22               Talked_with_Parents    0.013115
42               Individual_Father    0.012859
38                Attending_School    0.011958
44                 Part_Gov_Program    0.011750
28                  Part_Youth_Act    0.010135
6               Youth_doing_HChores    0.009900
25        Part_Substance_Prevention    0.008700
2                   Last_Avg_Grade    0.008043
21                  Share_Problems    0.007416
41               Individual_Mother    0.007414
14                  Youth_have_Gun    0.007075
24        Part_Violence_Prevention    0.006249
17                  Youth_Attacked    0.005640
27        Part_Preg/STD_Prevention    0.005390
26         Part_Help_Substance_Use    0.003478
```

[55]:
```
var_rf_imp = rf_var_imp_df.head(5)
plt.figure(figsize=(6, 2))
plt.barh(var_rf_imp['Variable'], var_rf_imp['Importance'], color='lightcoral')
plt.xlabel('Importance')
plt.ylabel('Variable')
plt.title('Variable Importance for RF Classifier')
plt.gca().invert_yaxis()
plt.show()
```



Now we will use the Boosting Technique for check the values and importance

[56]:
```
param_grid_gbm = {
    'n_estimators': [30, 40,50],
    'max_depth': [3, 5, 7],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
```

[57]:
```
boost_classifier = GradientBoostingClassifier()
```

[58]:
```
grid_search_gbm = GridSearchCV(estimator=boost_classifier,
                               param_grid=param_grid_gbm,
                               scoring='accuracy',
                               cv=5,
                               verbose=1,
                               n_jobs=-1)
```

[59]:
```
grid_search_gbm.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 81 candidates, totalling 405 fits
```

[59]:
```
GridSearchCV(cv=5, estimator=GradientBoostingClassifier(), n_jobs=-1,
             param_grid={'max_depth': [3, 5, 7], 'min_samples_leaf': [1, 2, 4],
                         'min_samples_split': [2, 5, 10],
                         'n_estimators': [30, 40, 50]},
             scoring='accuracy', verbose=1)
```

[60]:
```
best_params_gbm = grid_search_gbm.best_params_
best_score_gbm = grid_search_gbm.best_score_
print("Best Parameters:\n", best_params_gbm)
print("Best Accuracy:\n", best_score_gbm)
```

```
Best Parameters:
 {'max_depth': 3, 'min_samples_leaf': 1, 'min_samples_split': 5, 'n_estimators':
50}
Best Accuracy:
 0.7659515167228416
```

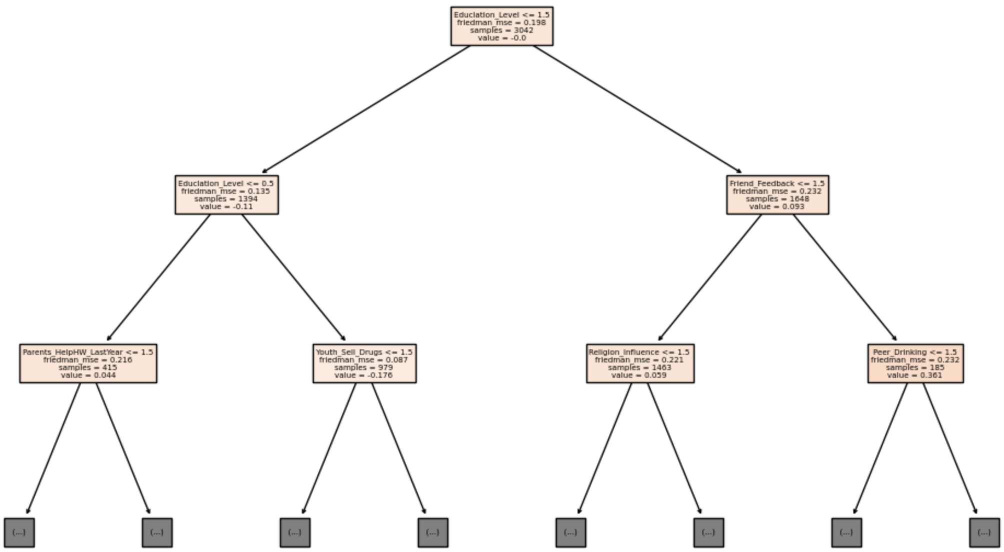
[61]:
```
best_boost_classifier = grid_search_gbm.best_estimator_
best_boost_classifier.fit(X_train, y_train)
```

[61]:
```
GradientBoostingClassifier(min_samples_split=5, n_estimators=50)
```

[62]:
```
y_pred_gbm = best_boost_classifier.predict(X_test)
accuracy_gbm = accuracy_score(y_test, y_pred_gbm)
conf_matrix_gbm = confusion_matrix(y_test, y_pred_gbm)
class_report_gbm = classification_report(y_test, y_pred_gbm)

print('Gradient Boosting Accuracy:', accuracy_gbm)
print('Gradient Boosting Confusion Matrix:\n', conf_matrix_gbm)
print('Gradient Boosting Classification Report:\n', class_report_gbm)
```

```
Gradient Boosting Accuracy: 0.7693486590038314
Gradient Boosting Confusion Matrix:
 [[899  56]
 [245 105]]
Gradient Boosting Classification Report:
               precision    recall  f1-score   support

           0       0.79      0.94      0.86       955
           1       0.65      0.30      0.41       350

    accuracy                           0.77      1305
   macro avg       0.72      0.62      0.63      1305
weighted avg       0.75      0.77      0.74      1305
```

[63]:
```
gbm_tree = best_boost_classifier.estimators_[0, 0]
plt.figure(figsize=(13, 8))
plot_tree(gbm_tree, filled=True, feature_names=X.columns, class_names=['No␣
↪Alcohol', 'Alcohol'], max_depth=2)
plt.show()
```



[64]:
```
var_imp_boost = best_boost_classifier.feature_importances_
var_imp_boost_df = pd.DataFrame({'Variable': X.columns, 'Importance':␣
↪var_imp_boost})
var_imp_boost_df = var_imp_boost_df.sort_values(by='Importance',␣
↪ascending=False)
var_imp_boost_df
```

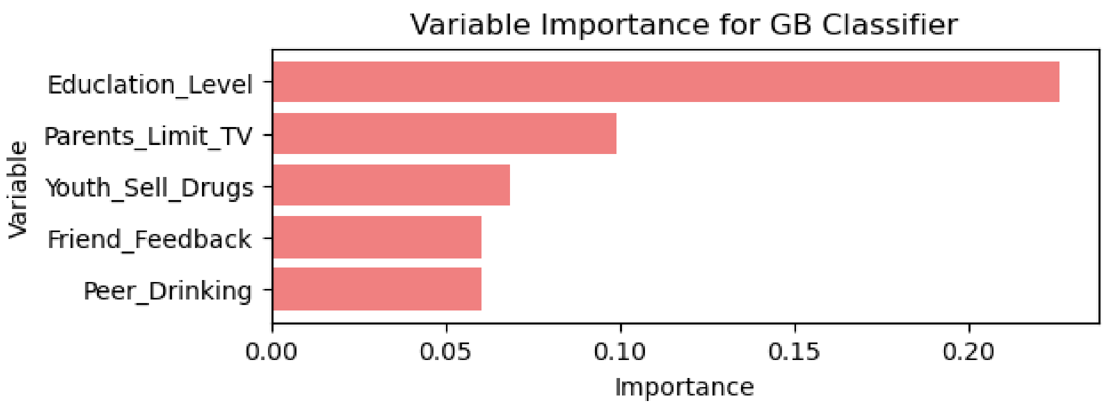
[64]:

	Variable	Importance
39	Educlation_Level	0.225858
7	Parents_Limit_TV	0.099254
15	Youth_Sell_Drugs	0.068251
20	Friend_Feedback	0.060200
19	Peer_Drinking	0.060174
3	Used_Alc_Week	0.053144
36	Race	0.039845
11	Argument_Parents	0.031895
4	Parents_CheckHW_LastYear	0.030808
33	Religion_Influence	0.029665
18	Parent Alcohol Daily	0.029556
40	School_Skipped	0.028119
23	Part_Extracurricular	0.027525
5	Parents_HelpHW_LastYear	0.023394
32	Yth_Believe_Religion Imp	0.021613
16	Youth_Steals	0.020489
31	Number_Religion_Attend	0.018662

```
13                  Youth_Group_Fight    0.017407
35                             Gender    0.012947
37                   Health_Condition    0.009460
34                    Religius_Friend    0.009353
45                      Poverty_Level    0.007970
9              Parents_Appreciation    0.006953
28                     Part_Youth_Act    0.006517
21                     Share_Problems    0.005786
25         Part_Substance_Prevention    0.005287
6                 Youth_doing_HChores    0.005156
43                             Income    0.005046
0                       Exp_of_School    0.004924
46                 Population Density    0.003881
8                Parents_Limit_Snacks    0.003811
17                     Youth_Attacked    0.003757
29   Yth_seen_alc+drug_prevention_ad    0.003388
41                  Individual_Mother    0.003149
47                         Metro_Size    0.003134
2                      Last_Avg_Grade    0.003115
42                  Individual_Father    0.002987
12                        Youth_Fight    0.002832
1                    Teacher_Feedback    0.002118
22                Talked_with_Parents    0.001237
10                      Proud_Parents    0.000713
27          Part_Preg/STD_Prevention    0.000622
30              Education_On_alc+drug    0.000000
26           Part_Help_Substance_Use    0.000000
38                   Attending_School    0.000000
14                     Youth_have_Gun    0.000000
44                    Part_Gov_Program    0.000000
24          Part_Violence_Prevention    0.000000
```

[65]:

```python
var_boost_imp = var_imp_boost_df.head(5)
plt.figure(figsize=(6, 2))
plt.barh(var_boost_imp['Variable'], var_boost_imp['Importance'],
  color='lightcoral')
plt.xlabel('Importance')
plt.ylabel('Variable')
plt.title('Variable Importance for GB Classifier')
plt.gca().invert_yaxis()
plt.show()
```



Now since we have 4 different models for our Binary Classification i.e., Decision Tree Classifier with an accuracy score of 74.48%, Bagging Classifier (RF with Bagging) with an accuracy score of 76.39 %, Random Forest Classifier with an accuracy of 76.09 % and Gradient boosting with an accuracy score of 76.93 %. Out of these 4 RF with Gradient Boosting Classifier has performed the best with the highest accuracy.

0.0.3 Multi-Class classification

We will now move forward for multi-class classification and we will consider the target varaiable as "Alc_Last_Month" and will create a new dataset for this purpose.

[66]:
```
col_not_required = ['Alc_Frq_Month', 'Alc_Last_Year', 'Alc_use_Age', 'Alc_Use',␣
 ↪'Used_Alc', 'Alc_Last_Month']
multi_df = df.drop(columns=col_not_required)
print(multi_df.head())
```

```
   Alc_Frq_Year  Exp_of_School  Teacher_Feedback  Last_Avg_Grade  \
0             0              1               1.0             2.0
1             0              1               1.0             2.0
2             0              1               1.0             2.0
3             0              1               1.0             2.0
5             0              1               1.0             2.0

   Used_Alc_Week  Parents_CheckHW_LastYear  Parents_HelpHW_LastYear  \
0            2.0                       1.0                      1.0
1            2.0                       1.0                      2.0
2            2.0                       1.0                      1.0
3            2.0                       1.0                      1.0
5            2.0                       1.0                      2.0

   Youth_doing_HChores  Parents_Limit_TV  Parents_Limit_Snacks  …  \
0                  2.0               2.0                   2.0  …
1                  1.0               2.0                   2.0  …
```

```
2                   1.0                 2.0                    1.0  …
3                   1.0                 1.0                    2.0  …
5                   1.0                 1.0                    2.0  …

   Attending_School  Educlation_Level  School_Skipped  Individual_Mother  \
0                 1                 2               0                  1
1                 1                 2               0                  1
2                 1                 1               0                  1
3                 1                 2               0                  1
5                 1                 2               0                  1

   Individual_Father  Income  Part_Gov_Program  Poverty_Level  \
0                  1       4                 2              3
1                  1       4                 2              3
2                  1       4                 1              3
3                  1       2                 2              1
5                  2       4                 2              3

   Population Density  Metro_Size
0                   2           2
1                   1           1
2                   1           1
3                   2           2
5                   1           1

[5 rows x 49 columns]
```

[67]:
```
X = multi_df.drop('Alc_Frq_Year', axis=1)
y = multi_df['Alc_Frq_Year']
X_train_m, X_test_m, y_train_m, y_test_m = train_test_split(X, y, test_size=0.
 ↪3, random_state=42)
```

[68]:
```
param_grid_dt = {
    'max_depth': [3, 5, 7, 10],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
```

[69]:
```
dt_classifier = DecisionTreeClassifier(random_state=42)
```

[70]:
```
grid_search_dt = GridSearchCV(estimator=dt_classifier,
                              param_grid=param_grid_dt,
                              scoring='accuracy',
                              cv=5,
                              verbose=1,
                              n_jobs=-1)
```

[71]:
```python
grid_search_dt.fit(X_train_m, y_train_m)
```

```
Fitting 5 folds for each of 36 candidates, totalling 180 fits
```

[71]:
```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(random_state=42), n_jobs=-1,
             param_grid={'max_depth': [3, 5, 7, 10],
                         'min_samples_leaf': [1, 2, 4],
                         'min_samples_split': [2, 5, 10]},
             scoring='accuracy', verbose=1)
```

[72]:
```python
best_params_dt = grid_search_dt.best_params_
best_score_dt = grid_search_dt.best_score_
print("Best Parameters:\n", best_params_dt)
print("Best Accuracy:\n", best_score_dt)
```

```
Best Parameters:
 {'max_depth': 3, 'min_samples_leaf': 1, 'min_samples_split': 2}
Best Accuracy:
 0.7935577089274912
```

[73]:
```python
best_dt_classifier = grid_search_dt.best_estimator_
best_dt_classifier.fit(X_train_m, y_train_m)
```

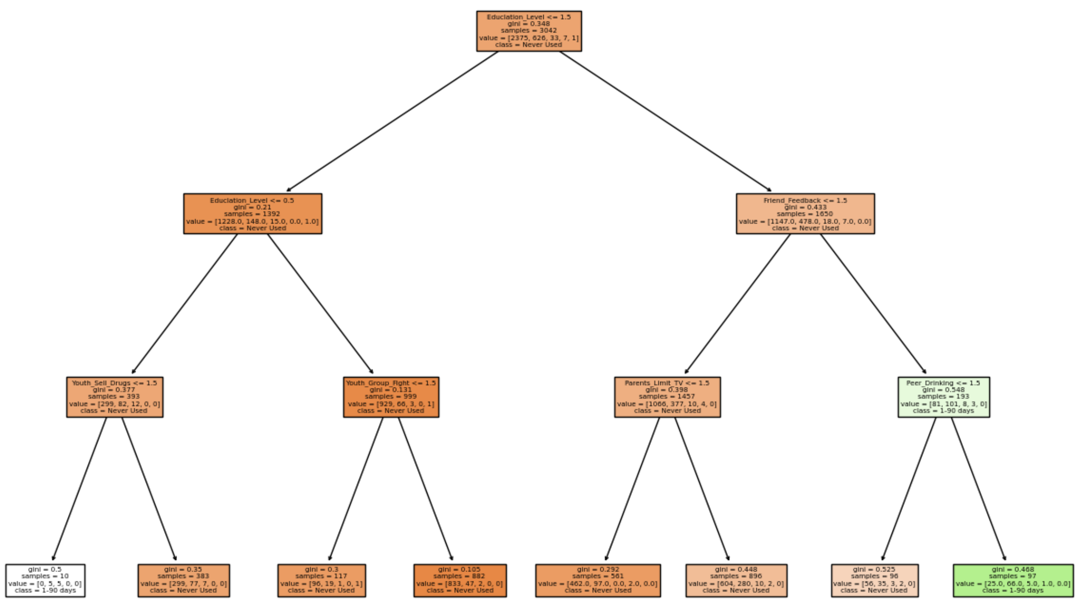
[73]:
```
DecisionTreeClassifier(max_depth=3, random_state=42)
```

[74]:
```python
y_pred_dt = best_dt_classifier.predict(X_test_m)
accuracy_dt = accuracy_score(y_test_m, y_pred_dt)
conf_matrix_dt = confusion_matrix(y_test_m, y_pred_dt)
class_report_dt = classification_report(y_test_m, y_pred_dt)

print('DT Classifier Accuracy:', accuracy_dt)
print('DT Classifier Confusion Matrix:\n', conf_matrix_dt)
print('DT Classifier Classification Report:\n', class_report_dt)
```

```
DT Classifier Accuracy: 0.7770114942528735
DT Classifier Confusion Matrix:
 [[996  17   0   0   0]
 [250  18   0   0   0]
 [ 17   1   0   0   0]
 [  4   1   0   0   0]
 [  1   0   0   0   0]]
DT Classifier Classification Report:
               precision    recall  f1-score   support

           0       0.79      0.98      0.87      1013
           1       0.49      0.07      0.12       268
           2       0.00      0.00      0.00        18
           3       0.00      0.00      0.00         5
```

```
              4       0.00      0.00      0.00         1

       accuracy                           0.78      1305
      macro avg       0.25      0.21      0.20      1305
   weighted avg       0.71      0.78      0.70      1305
```

[75]:
```python
plt.figure(figsize=(15, 10))
plot_tree(best_dt_classifier, filled=True, feature_names=X.columns,␣
 ↪class_names=['Never Used', '1-90 days', '91-180 days', '181-271 days', '271␣
 ↪days'], max_depth=3)
plt.show()
```



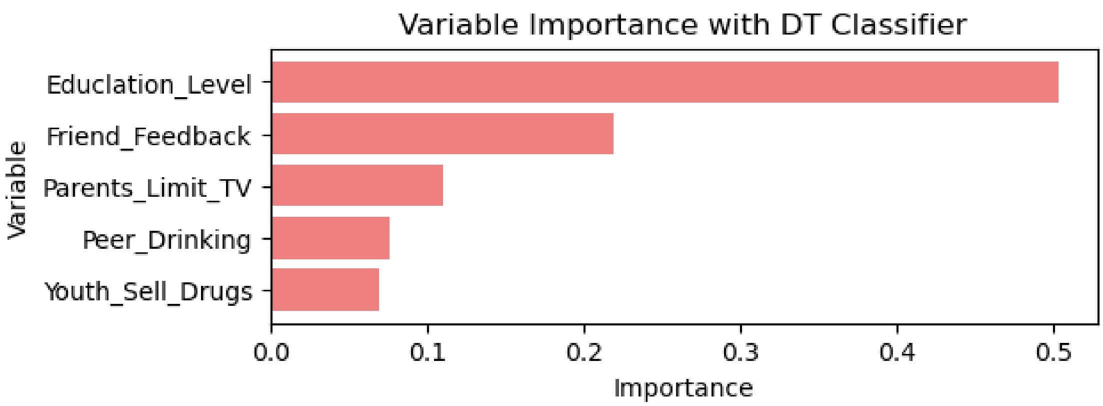
[76]:
```python
var_importance_dt = best_dt_classifier.feature_importances_
var_importance_dt_df = pd.DataFrame({'Variable': X.columns, 'Importance':␣
 ↪var_importance_dt})
var_importance_dt_df = var_importance_dt_df.sort_values(by='Importance',␣
 ↪ascending=False)
var_importance_dt_df
```

[76]:

	Variable	Importance
39	Educlation_Level	0.502911
20	Friend_Feedback	0.219603

7	Parents_Limit_TV	0.110538
19	Peer_Drinking	0.076038
15	Youth_Sell_Drugs	0.069307
13	Youth_Group_Fight	0.021603
36	Race	0.000000
29	Yth_seen_alc+drug_prevention_ad	0.000000
30	Education_On_alc+drug	0.000000
31	Number_Religion_Attend	0.000000
32	Yth_Believe_Religion Imp	0.000000
33	Religion_Influence	0.000000
34	Religius_Friend	0.000000
35	Gender	0.000000
0	Exp_of_School	0.000000
27	Part_Preg/STD_Prevention	0.000000
37	Health_Condition	0.000000
38	Attending_School	0.000000
40	School_Skipped	0.000000
41	Individual_Mother	0.000000
42	Individual_Father	0.000000
43	Income	0.000000
44	Part_Gov_Program	0.000000
45	Poverty_Level	0.000000
46	Population Density	0.000000
28	Part_Youth_Act	0.000000
24	Part_Violence_Prevention	0.000000
26	Part_Help_Substance_Use	0.000000
11	Argument_Parents	0.000000
2	Last_Avg_Grade	0.000000
3	Used_Alc_Week	0.000000
4	Parents_CheckHW_LastYear	0.000000
5	Parents_HelpHW_LastYear	0.000000
6	Youth_doing_HChores	0.000000
8	Parents_Limit_Snacks	0.000000
9	Parents_Appreciation	0.000000
10	Proud_Parents	0.000000
12	Youth_Fight	0.000000
25	Part_Substance_Prevention	0.000000
14	Youth_have_Gun	0.000000
16	Youth_Steals	0.000000
17	Youth_Attacked	0.000000
18	Parent Alcohol Daily	0.000000
21	Share_Problems	0.000000
22	Talked_with_Parents	0.000000
23	Part_Extracurricular	0.000000
1	Teacher_Feedback	0.000000
47	Metro_Size	0.000000

[77]:
```
var_dt = var_importance_dt_df.head(5)
plt.figure(figsize=(6, 2))
plt.barh(var_dt['Variable'], var_dt['Importance'], color='lightcoral')
plt.xlabel('Importance')
plt.ylabel('Variable')
plt.title('Variable Importance with DT Classifier')
plt.gca().invert_yaxis()
plt.show()
```



Now we will move towards the RF with Bagging Classifier

[78]:
```
param_grid_bag_rf = {
    'n_estimators': [30, 40 , 50],
    'max_samples': [0.5, 0.7, 1.0]
}
```

[79]:
```
base_rf_class = RandomForestClassifier()
```

[80]:
```
bag_rf_class = BaggingClassifier(base_rf_class, random_state=42)
```

[81]:
```
grid_search_bag_rf = GridSearchCV(estimator=bag_rf_class,
                                  param_grid=param_grid_bag_rf,
                                  scoring='accuracy',
                                  cv=5,
                                  verbose=1,
                                  n_jobs=-1)
```

[82]:
```
grid_search_bag_rf.fit(X_train_m, y_train_m)
```

```
Fitting 5 folds for each of 9 candidates, totalling 45 fits
```

[82]:
```
GridSearchCV(cv=5,
             estimator=BaggingClassifier(estimator=RandomForestClassifier(),
                                         random_state=42),
             n_jobs=-1,
             param_grid={'max_samples': [0.5, 0.7, 1.0],
                         'n_estimators': [30, 40, 50]},
             scoring='accuracy', verbose=1)
```

[83]:
```
best_params_bag_rf = grid_search_bag_rf.best_params_
best_score_bag_rf = grid_search_bag_rf.best_score_
print("Best Parameters:\n", best_params_bag_rf)
print("Best Accuracy:\n", best_score_bag_rf)
```

```
Best Parameters:
 {'max_samples': 0.5, 'n_estimators': 50}
Best Accuracy:
 0.7955308529945554
```

[84]:
```
best_bag_rf_class = grid_search_bag_rf.best_estimator_
best_bag_rf_class.fit(X_train_m, y_train_m)
```

[84]:
```
BaggingClassifier(estimator=RandomForestClassifier(), max_samples=0.5,
                  n_estimators=50, random_state=42)
```

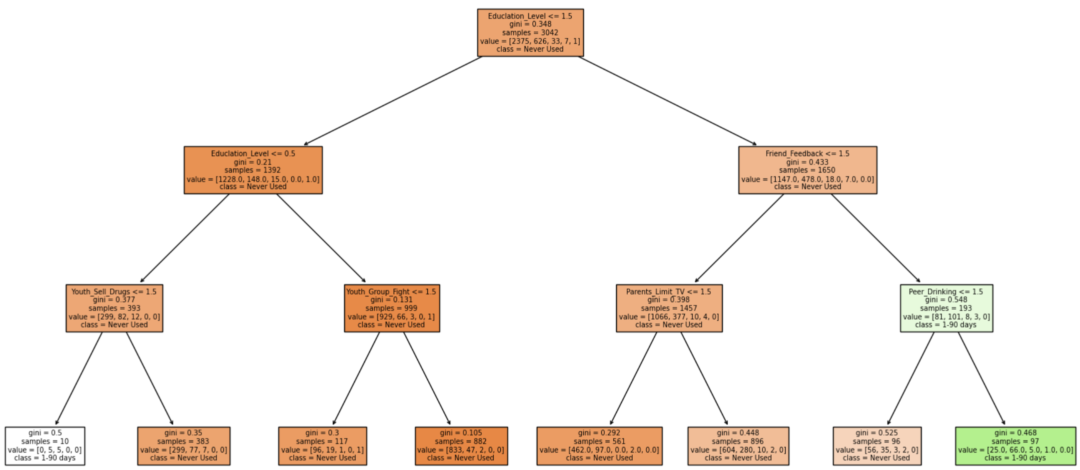
[85]:
```
y_pred_rfbag = best_bag_rf_class.predict(X_test_m)
acc_bag_rf = accuracy_score(y_test_m, y_pred_rfbag)
conf_matrix_rfbag = confusion_matrix(y_test_m, y_pred_rfbag)
class_rep_rfbag = classification_report(y_test_m, y_pred_rfbag)

print('RF Bagging Accuracy:', acc_bag_rf)
print('RF Bagging Confusion Matrix:\n', conf_matrix_rfbag)
print('RF Bagging Classification Report:\n', class_rep_rfbag)
```

```
RF Bagging Accuracy: 0.7862068965517242
RF Bagging Confusion Matrix:
 [[1000   13    0    0    0]
 [ 242   26    0    0    0]
 [  17    1    0    0    0]
 [   2    3    0    0    0]
 [   1    0    0    0    0]]
RF Bagging Classification Report:
               precision    recall  f1-score   support

           0       0.79      0.99      0.88      1013
           1       0.60      0.10      0.17       268
           2       0.00      0.00      0.00        18
           3       0.00      0.00      0.00         5
           4       0.00      0.00      0.00         1
```

```
     accuracy                           0.79      1305
    macro avg       0.28      0.22      0.21      1305
 weighted avg       0.74      0.79      0.72      1305
```

[86]:

```
base_rf_tree = DecisionTreeClassifier(max_depth=3)
base_rf_tree.fit(X_train_m, y_train_m)
plt.figure(figsize=(20, 10))
plot_tree(base_rf_tree, filled=True, feature_names=X.columns,␣
 ↪class_names=['Never Used', '1-90 days', '91-180 days', '181-271 days', '271␣
 ↪days'])
plt.show()
```



[87]:

```
var_imp_rfbag = best_bag_rf_class.estimators_[0].feature_importances_
varimp_rfbag_df = pd.DataFrame({'Variable': X.columns, 'Importance':␣
 ↪var_imp_rfbag})
varimp_rfbag_df = varimp_rfbag_df.sort_values(by='Importance', ascending=False)
varimp_rfbag_df
```

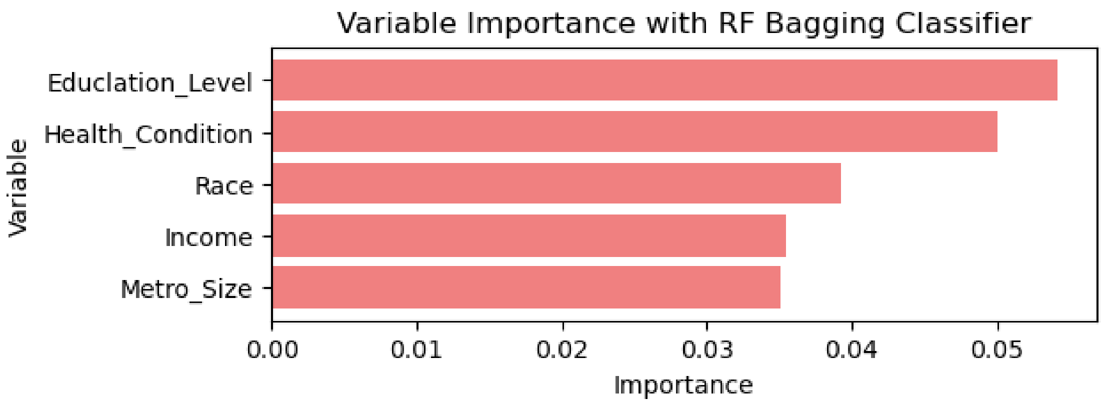
[87]:

	Variable	Importance
39	Educlation_Level	0.054208
37	Health_Condition	0.050008
36	Race	0.039307
43	Income	0.035414
47	Metro_Size	0.035122
40	School_Skipped	0.031985
7	Parents_Limit_TV	0.031628

```
46                  Population Density    0.030489
35                              Gender    0.029098
0                        Exp_of_School    0.025337
1                     Teacher_Feedback    0.024891
20                     Friend_Feedback    0.024852
33                  Religion_Influence    0.024646
22                   Talked_with_Parents  0.024502
30               Education_On_alc+drug    0.023876
8                  Parents_Limit_Snacks   0.023628
11                    Argument_Parents    0.023495
29   Yth_seen_alc+drug_prevention_ad      0.023347
32            Yth_Believe_Religion Imp    0.023271
19                       Peer_Drinking    0.022647
23                 Part_Extracurricular   0.022352
3                        Used_Alc_Week    0.022166
45                       Poverty_Level    0.020877
31              Number_Religion_Attend    0.020110
15                    Youth_Sell_Drugs    0.018661
5              Parents_HelpHW_LastYear    0.018616
34                     Religius_Friend    0.018217
18                 Parent Alcohol Daily   0.017846
9                 Parents_Appreciation    0.017821
4             Parents_CheckHW_LastYear    0.017144
42                   Individual_Father    0.016673
12                         Youth_Fight    0.016580
16                        Youth_Steals    0.015816
10                       Proud_Parents    0.015514
13                   Youth_Group_Fight    0.015111
6                   Youth_doing_HChores   0.014644
28                      Part_Youth_Act    0.014191
44                    Part_Gov_Program    0.010958
25          Part_Substance_Prevention     0.010855
41                   Individual_Mother    0.010459
14                      Youth_have_Gun    0.010381
17                      Youth_Attacked    0.009516
21                      Share_Problems    0.008285
38                    Attending_School    0.008191
24            Part_Violence_Prevention    0.008115
2                       Last_Avg_Grade    0.006771
27            Part_Preg/STD_Prevention    0.006568
26             Part_Help_Substance_Use    0.005812
```

[88]:
```
var_rfbag = varimp_rfbag_df.head(5)
plt.figure(figsize=(6, 2))
plt.barh(var_rfbag['Variable'], var_rfbag['Importance'], color='lightcoral')
plt.xlabel('Importance')
plt.ylabel('Variable')
```

```
plt.title('Variable Importance with RF Bagging Classifier')
plt.gca().invert_yaxis()
plt.show()
```



Now we will use Gradiant Boosting Classifier

[89]:
```
param_grid_boost = {
    'max_depth': [3, 5, 7],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
```

[90]:
```
multi_boost_class = GradientBoostingClassifier(random_state=42)
```

[91]:
```
grid_search_boost = GridSearchCV(estimator=multi_boost_class,
                                 param_grid=param_grid_boost,
                                 scoring='accuracy',
                                 cv=5,
                                 verbose=1,
                                 n_jobs=-1)
```

[92]:
```
grid_search_boost.fit(X_train_m, y_train_m)
```

```
Fitting 5 folds for each of 27 candidates, totalling 135 fits
```

[92]:
```
GridSearchCV(cv=5, estimator=GradientBoostingClassifier(random_state=42),
             n_jobs=-1,
             param_grid={'max_depth': [3, 5, 7], 'min_samples_leaf': [1, 2, 4],
                         'min_samples_split': [2, 5, 10]},
             scoring='accuracy', verbose=1)
```

```
[93]: best_params_boost = grid_search_boost.best_params_
best_score_boost = grid_search_boost.best_score_
print("Best Parameters:\n", best_params_boost)
print("Best Accuracy:\n", best_score_boost)
```

```
Best Parameters:
 {'max_depth': 3, 'min_samples_leaf': 4, 'min_samples_split': 2}
Best Accuracy:
 0.796848803042088
```

```
[94]: best_boost_class = grid_search_boost.best_estimator_
best_boost_class.fit(X_train_m, y_train_m)
```

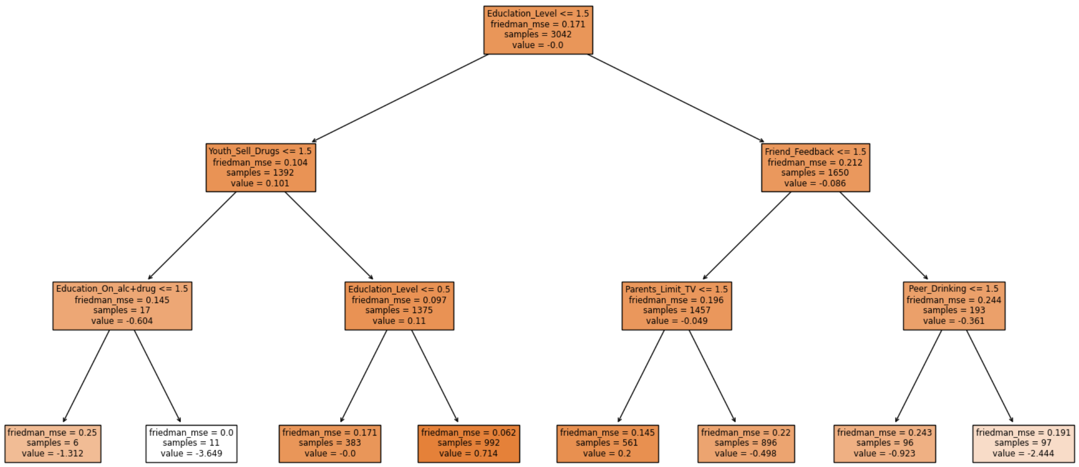
```
[94]: GradientBoostingClassifier(min_samples_leaf=4, random_state=42)
```

```
[95]: y_pred_boost = best_boost_class.predict(X_test_m)
multi_accu_boost = accuracy_score(y_test_m, y_pred_boost)
multi_confmatrix_boost = confusion_matrix(y_test_m, y_pred_boost)
multi_class_report_boost = classification_report(y_test_m, y_pred_boost)

print('GB Accuracy:', multi_accu_boost)
print('GB Confusion Matrix:\n', multi_confmatrix_boost)
print('GB Classification Report:\n', multi_class_report_boost)
```

```
GB Accuracy: 0.7915708812260537
GB Confusion Matrix:
 [[969  43   1   0   0]
 [203  64   1   0   0]
 [ 11   7   0   0   0]
 [  1   4   0   0   0]
 [  1   0   0   0   0]]
GB Classification Report:
               precision    recall  f1-score   support

           0       0.82      0.96      0.88      1013
           1       0.54      0.24      0.33       268
           2       0.00      0.00      0.00        18
           3       0.00      0.00      0.00         5
           4       0.00      0.00      0.00         1

    accuracy                           0.79      1305
   macro avg       0.27      0.24      0.24      1305
weighted avg       0.75      0.79      0.75      1305
```

```
[96]: best_tree = best_boost_class.estimators_[0, 0]
plt.figure(figsize=(20, 10))
```

```
plot_tree(best_tree, filled=True, feature_names=X.columns, class_names=['Never␣
↪Used', '1-90 days', '91-180 days', '181-271 days', '271 days'], max_depth=3)
plt.show()
```



[97]:
```
var_imp_boost = best_boost_class.feature_importances_
var_imp_boost_df = pd.DataFrame({'Variable': X.columns, 'Importance':␣
↪var_imp_boost})
var_imp_boost_df = var_imp_boost_df.sort_values(by='Importance',␣
↪ascending=False)
var_imp_boost_df
```

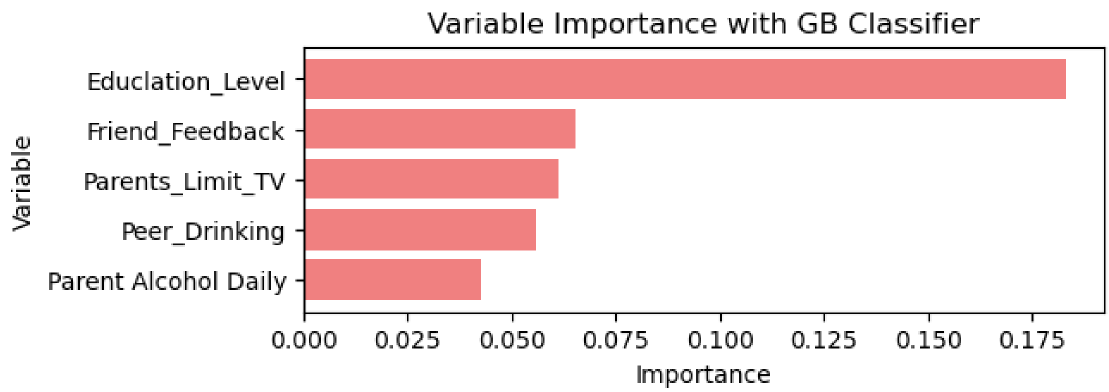
[97]:

	Variable	Importance
39	Educlation_Level	0.183103
20	Friend_Feedback	0.065534
7	Parents_Limit_TV	0.061447
19	Peer_Drinking	0.055803
18	Parent Alcohol Daily	0.042597
15	Youth_Sell_Drugs	0.038361
36	Race	0.035247
40	School_Skipped	0.032281
23	Part_Extracurricular	0.031506
11	Argument_Parents	0.031320
3	Used_Alc_Week	0.030961
16	Youth_Steals	0.028298
37	Health_Condition	0.027817
24	Part_Violence_Prevention	0.022751
4	Parents_CheckHW_LastYear	0.021889
43	Income	0.021314

```
25         Part_Substance_Prevention    0.019583
5           Parents_HelpHW_LastYear    0.019477
34                  Religius_Friend    0.018741
35                           Gender    0.016833
0                     Exp_of_School    0.016604
47                       Metro_Size    0.015142
32         Yth_Believe_Religion Imp    0.013966
28                    Part_Youth_Act   0.012663
12                      Youth_Fight    0.010260
45                    Poverty_Level    0.010099
22               Talked_with_Parents   0.009596
14                    Youth_have_Gun   0.009431
41                Individual_Mother    0.008647
29  Yth_seen_alc+drug_prevention_ad    0.007122
13                Youth_Group_Fight    0.006834
1                  Teacher_Feedback    0.006810
30             Education_On_alc+drug   0.006696
8               Parents_Limit_Snacks   0.006527
2                    Last_Avg_Grade    0.006096
33               Religion_Influence    0.006017
46               Population Density    0.005446
6                Youth_doing_HChores   0.005338
42                Individual_Father    0.004625
21                   Share_Problems    0.004585
31           Number_Religion_Attend    0.004535
10                   Proud_Parents     0.004202
27         Part_Preg/STD_Prevention    0.003610
17                   Youth_Attacked    0.003587
9              Parents_Appreciation    0.002560
38                 Attending_School    0.002379
44                  Part_Gov_Program   0.000984
26            Part_Help_Substance_Use  0.000779
```

[98]:

```
var_boost_m = var_imp_boost_df.head(5)
plt.figure(figsize=(6, 2))
plt.barh(var_boost_m['Variable'], var_boost_m['Importance'], color='lightcoral')
plt.xlabel('Importance')
plt.ylabel('Variable')
plt.title('Variable Importance with GB Classifier')
plt.gca().invert_yaxis()
plt.show()
```



Now we have 3 multi-class model and in comparision we have best model of the 3 is Gradient Boosting with a accuracy of 79.23% followed by Random Forest with Bagging model with a accuracy of 78.62% and the last in line is DT Model which has a accuracy of 77.70%.

0.0.4 Regression Model

We move to our final part using Regression models for prediction.

[99]:
```
col_not_required = ['Alc_Last_Year', 'Alc_use_Age', 'Alc_Use',␣
 ↪'Alc_Last_Month', 'Used_Alc', 'Alc_Frq_Year']
reg_df = df.drop(columns=col_not_required)
print(reg_df.head())
```

```
   Alc_Frq_Month  Exp_of_School  Teacher_Feedback  Last_Avg_Grade  \
0            0.0              1               1.0             2.0
1            0.0              1               1.0             2.0
2            0.0              1               1.0             2.0
3            0.0              1               1.0             2.0
5            0.0              1               1.0             2.0

   Used_Alc_Week  Parents_CheckHW_LastYear  Parents_HelpHW_LastYear  \
0            2.0                       1.0                      1.0
1            2.0                       1.0                      2.0
2            2.0                       1.0                      1.0
3            2.0                       1.0                      1.0
5            2.0                       1.0                      2.0

   Youth_doing_HChores  Parents_Limit_TV  Parents_Limit_Snacks  …  \
0                  2.0               2.0                   2.0  …
1                  1.0               2.0                   2.0  …
2                  1.0               2.0                   1.0  …
3                  1.0               1.0                   2.0  …
5                  1.0               1.0                   2.0  …
```

```
   Attending_School  Educlation_Level  School_Skipped  Individual_Mother  \
0                 1                 2               0                  1
1                 1                 2               0                  1
2                 1                 1               0                  1
3                 1                 2               0                  1
5                 1                 2               0                  1

   Individual_Father  Income  Part_Gov_Program  Poverty_Level  \
0                  1       4                 2              3
1                  1       4                 2              3
2                  1       4                 1              3
3                  1       2                 2              1
5                  2       4                 2              3

   Population Density  Metro_Size
0                   2           2
1                   1           1
2                   1           1
3                   2           2
5                   1           1

[5 rows x 49 columns]
```

[100]:
```
X = reg_df.drop('Alc_Frq_Month', axis=1)
y = reg_df['Alc_Frq_Month']
X_train_r, X_test_r, y_train_r, y_test_r = train_test_split(X, y, test_size=0.
↪3, random_state=42)
```

Decision Tree Regressor

[101]:
```
param_grid_dt = {
    'max_depth': [3, 5, 7],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
```

[102]:
```
dt_reg = DecisionTreeRegressor(random_state=42)
```

[103]:
```
grid_search_dt = GridSearchCV(estimator=dt_reg,
                              param_grid=param_grid_dt,
                              scoring='r2',
                              cv=5,
                              verbose=1,
                              n_jobs=-1)
```

[104]:
```
grid_search_dt.fit(X_train_r, y_train_r)
```

```
Fitting 5 folds for each of 27 candidates, totalling 135 fits
```

[104]:
```
GridSearchCV(cv=5, estimator=DecisionTreeRegressor(random_state=42), n_jobs=-1,
             param_grid={'max_depth': [3, 5, 7], 'min_samples_leaf': [1, 2, 4],
                         'min_samples_split': [2, 5, 10]},
             scoring='r2', verbose=1)
```

[105]:
```python
best_params_dt = grid_search_dt.best_params_
print("Best Parameters:\n", best_params_dt)
```

```
Best Parameters:
 {'max_depth': 3, 'min_samples_leaf': 4, 'min_samples_split': 10}
```

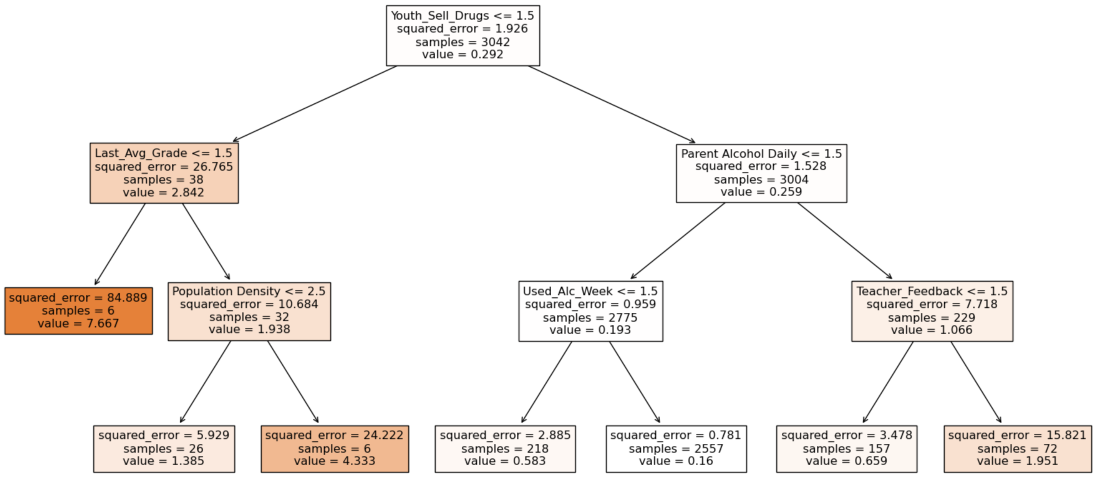
[106]:
```python
best_dt_reg = grid_search_dt.best_estimator_
best_dt_reg.fit(X_train_r, y_train_r)
```

[106]:
```
DecisionTreeRegressor(max_depth=3, min_samples_leaf=4, min_samples_split=10,
                      random_state=42)
```

[107]:
```python
y_pred_dt = best_dt_reg.predict(X_test_r)
mse_dt = mean_squared_error(y_test_r, y_pred_dt)
print('Decision Tree Regressor MSE:', mse_dt)
```

```
Decision Tree Regressor MSE: 1.9294968496848945
```

[108]:
```python
plt.figure(figsize=(20, 10))
plot_tree(best_dt_reg, filled=True, feature_names=X.columns)
plt.show()
```



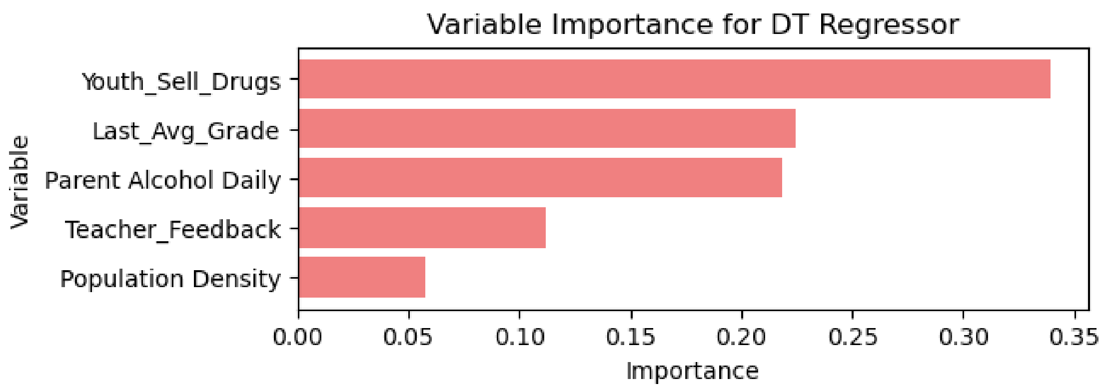
[109]:
```python
feature_importances = best_dt_reg.feature_importances_
var_imp_df = pd.DataFrame({'Variable': X.columns, 'Importance':␣
 ↪feature_importances})
var_imp_df = var_imp_df.sort_values(by='Importance', ascending=False)
var_imp_df
```

[109]:

	Variable	Importance
15	Youth_Sell_Drugs	0.339176
2	Last_Avg_Grade	0.224711
18	Parent Alcohol Daily	0.218304
1	Teacher_Feedback	0.111673
46	Population Density	0.057434
3	Used_Alc_Week	0.048702
0	Exp_of_School	0.000000
29	Yth_seen_alc+drug_prevention_ad	0.000000
30	Education_On_alc+drug	0.000000
31	Number_Religion_Attend	0.000000
32	Yth_Believe_Religion Imp	0.000000
33	Religion_Influence	0.000000
34	Religius_Friend	0.000000
35	Gender	0.000000
37	Health_Condition	0.000000
36	Race	0.000000
27	Part_Preg/STD_Prevention	0.000000
38	Attending_School	0.000000
39	Educlation_Level	0.000000
40	School_Skipped	0.000000
41	Individual_Mother	0.000000
42	Individual_Father	0.000000
43	Income	0.000000
44	Part_Gov_Program	0.000000
45	Poverty_Level	0.000000
28	Part_Youth_Act	0.000000
24	Part_Violence_Prevention	0.000000
26	Part_Help_Substance_Use	0.000000
12	Youth_Fight	0.000000
4	Parents_CheckHW_LastYear	0.000000
5	Parents_HelpHW_LastYear	0.000000
6	Youth_doing_HChores	0.000000
7	Parents_Limit_TV	0.000000
8	Parents_Limit_Snacks	0.000000
9	Parents_Appreciation	0.000000
10	Proud_Parents	0.000000
11	Argument_Parents	0.000000
13	Youth_Group_Fight	0.000000
25	Part_Substance_Prevention	0.000000
14	Youth_have_Gun	0.000000

```
16                      Youth_Steals     0.000000
17                    Youth_Attacked     0.000000
19                     Peer_Drinking     0.000000
20                   Friend_Feedback     0.000000
21                    Share_Problems     0.000000
22                Talked_with_Parents    0.000000
23              Part_Extracurricular     0.000000
47                        Metro_Size     0.000000
```

[110]:
```
varimp_reg_dt = var_imp_df.head(5)
plt.figure(figsize=(6, 2))
plt.barh(varimp_reg_dt['Variable'], varimp_reg_dt['Importance'],␣
 ↪color='lightcoral')
plt.xlabel('Importance')
plt.ylabel('Variable')
plt.title('Variable Importance for DT Regressor')
plt.gca().invert_yaxis()
plt.show()
```



RF Regressor with Boosting

[111]:
```
param_grid_rf = {
    'n_estimators': [30, 40, 50],
    'max_samples': [0.5, 0.7, 1.0]
}
```

[112]:
```
base_rf_regressor = RandomForestRegressor()
```

[113]:
```
bag_rf_regressor = BaggingRegressor(base_rf_regressor, random_state=42)
```

[114]:
```
grid_search_rf = GridSearchCV(estimator=bag_rf_regressor,
                              param_grid=param_grid_rf,
                              scoring='r2',
```

```
                              cv=5,
                              verbose=1,
                              n_jobs=-1)
```

[115]:
```
grid_search_rf.fit(X_train_r, y_train_r)
```

```
Fitting 5 folds for each of 9 candidates, totalling 45 fits
```

[115]:
```
GridSearchCV(cv=5,
             estimator=BaggingRegressor(estimator=RandomForestRegressor(),
                                        random_state=42),
             n_jobs=-1,
             param_grid={'max_samples': [0.5, 0.7, 1.0],
                         'n_estimators': [30, 40, 50]},
             scoring='r2', verbose=1)
```

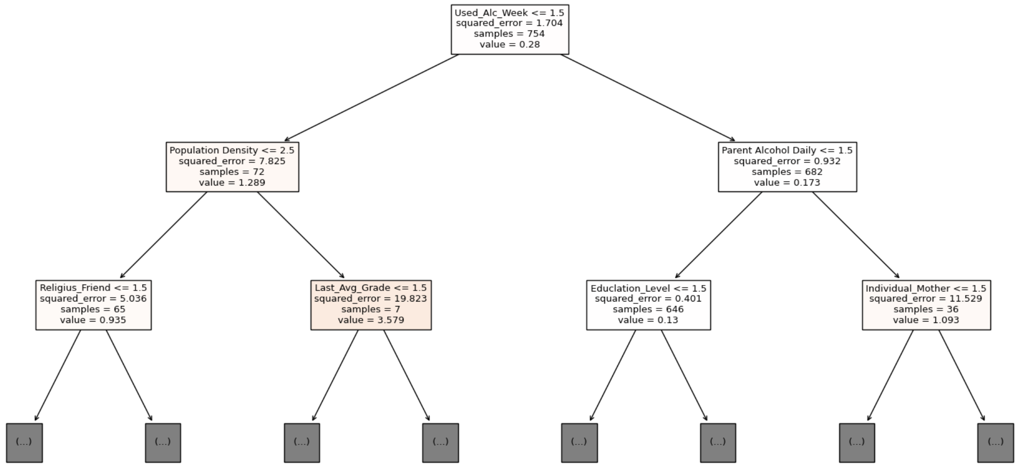
[116]:
```
best_estimator_rf = grid_search_rf.best_estimator_
best_params_rf = grid_search_rf.best_params_
print("Best Parameters:\n", best_estimator_rf)
print("Best Accuracy:\n", best_params_rf)
```

```
Best Parameters:
 BaggingRegressor(estimator=RandomForestRegressor(), max_samples=0.5,
                 n_estimators=30, random_state=42)
Best Accuracy:
 {'max_samples': 0.5, 'n_estimators': 30}
```

[117]:
```
y_pred_rf_bag = best_estimator_rf.predict(X_test_r)
mse_rf_bag = mean_squared_error(y_test_r, y_pred_rf_bag)
print('RF Regressor Bagging MSE:', mse_rf_bag)
```

```
RF Regressor Bagging MSE: 1.7647723669859512
```

[118]:
```
base_rf_tree = best_estimator_rf.estimators_[0].estimators_[0]
plt.figure(figsize=(20, 10))
plot_tree(base_rf_tree, filled=True, feature_names=X.columns, max_depth=2)
plt.show()
```



[119]:
```
var_imp_rfr_bag = best_estimator_rf.estimators_[0].feature_importances_
var_imp_rfrbag_df = pd.DataFrame({'Variable': X.columns, 'Importance':␣
↪var_imp_rfr_bag})
var_imp_rfrbag_df = var_imp_rfrbag_df.sort_values(by='Importance',␣
↪ascending=False)
var_imp_rfrbag_df
```

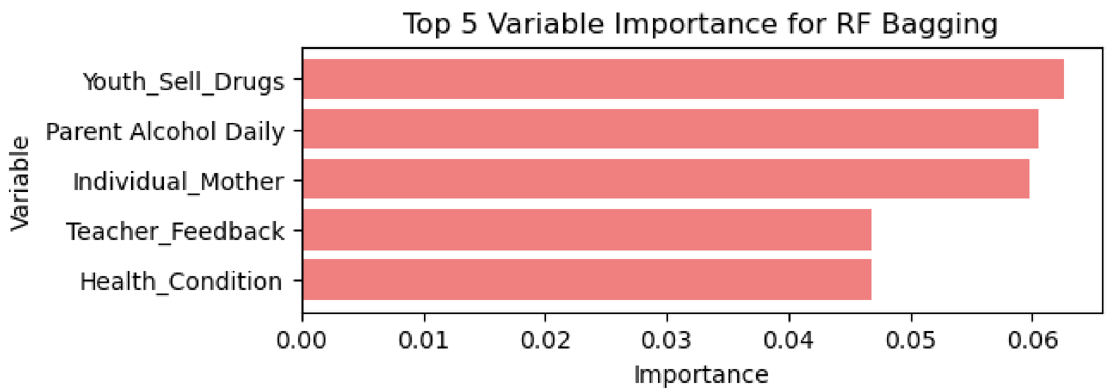
[119]:

	Variable	Importance
15	Youth_Sell_Drugs	0.062591
18	Parent Alcohol Daily	0.060457
41	Individual_Mother	0.059773
1	Teacher_Feedback	0.046773
37	Health_Condition	0.046771
3	Used_Alc_Week	0.040023
2	Last_Avg_Grade	0.039873
46	Population Density	0.038712
43	Income	0.035738
19	Peer_Drinking	0.031933
34	Religius_Friend	0.031608
47	Metro_Size	0.029170
36	Race	0.029164
0	Exp_of_School	0.025521
8	Parents_Limit_Snacks	0.023952
35	Gender	0.023614
22	Talked_with_Parents	0.023196
40	School_Skipped	0.022777
32	Yth_Believe_Religion Imp	0.022166
16	Youth_Steals	0.018430

```
29  Yth_seen_alc+drug_prevention_ad    0.018169
39                 Educlation_Level    0.017428
5                Parents_HelpHW_LastYear    0.016967
4               Parents_CheckHW_LastYear    0.016704
33               Religion_Influence    0.015859
42                Individual_Father    0.013646
24          Part_Violence_Prevention    0.013524
23              Part_Extracurricular    0.013090
10                    Proud_Parents    0.012950
13                Youth_Group_Fight    0.011804
7                  Parents_Limit_TV    0.011642
9              Parents_Appreciation    0.011495
14                   Youth_have_Gun    0.011442
20                  Friend_Feedback    0.011151
6                Youth_doing_HChores    0.010920
30            Education_On_alc+drug    0.010319
17                   Youth_Attacked    0.009898
38                 Attending_School    0.009848
11                 Argument_Parents    0.008892
31          Number_Religion_Attend    0.008457
45                    Poverty_Level    0.008391
12                      Youth_Fight    0.006726
25        Part_Substance_Prevention    0.004504
28                   Part_Youth_Act    0.004163
27        Part_Preg/STD_Prevention    0.003270
44                  Part_Gov_Program    0.003132
21                   Share_Problems    0.002157
26          Part_Help_Substance_Use    0.001210
```

[120]:

```
top5_var_imp_rfrbag_df = var_imp_rfrbag_df.head(5)
plt.figure(figsize=(6, 2))
plt.barh(top5_var_imp_rfrbag_df['Variable'],
 ↪top5_var_imp_rfrbag_df['Importance'], color='lightcoral')
plt.xlabel('Importance')
plt.ylabel('Variable')
plt.title('Top 5 Variable Importance for RF Bagging')
plt.gca().invert_yaxis()
plt.show()
```



Gradiant Boosting Regressor

[121]:
```
boosting_regressor = GradientBoostingRegressor(random_state=1)
```

[122]:
```
param_grid_boost = {
    'n_estimators': [30, 40, 50],
    'max_depth': [3, 5, 7],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
```

[123]:
```
grid_search_boost = GridSearchCV(estimator=boosting_regressor,
                                 param_grid=param_grid_boost,
                                 scoring='r2',
                                 cv=5,
                                 verbose=1,
                                 n_jobs=-1)
```

[124]:
```
grid_search_boost.fit(X_train_r, y_train_r)
```

```
Fitting 5 folds for each of 81 candidates, totalling 405 fits
```

[124]:
```
GridSearchCV(cv=5, estimator=GradientBoostingRegressor(random_state=1),
             n_jobs=-1,
             param_grid={'max_depth': [3, 5, 7], 'min_samples_leaf': [1, 2, 4],
                         'min_samples_split': [2, 5, 10],
                         'n_estimators': [30, 40, 50]},
             scoring='r2', verbose=1)
```

[125]:
```
best_estimator_boost = grid_search_boost.best_estimator_
best_params_boost = grid_search_boost.best_params_
print("Best Parameters:\n", best_estimator_boost)
print("Best Accuracy:\n", best_params_boost)
```

```
Best Parameters:
 GradientBoostingRegressor(min_samples_split=10, n_estimators=40,
random_state=1)
Best Accuracy:
 {'max_depth': 3, 'min_samples_leaf': 1, 'min_samples_split': 10,
'n_estimators': 40}
```

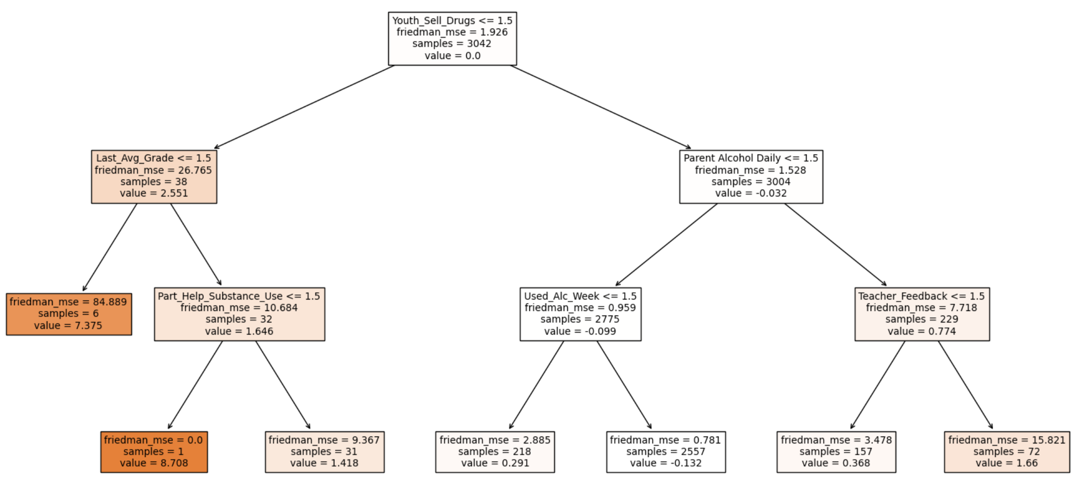
[126]:

```python
y_pred_boost = best_estimator_boost.predict(X_test_r)
mse_boost = mean_squared_error(y_test_r, y_pred_boost)
print('Gradient Boosting Regressor MSE:', mse_boost)
```

```
Gradient Boosting Regressor MSE: 2.082048518298126
```

[127]:

```python
base_tree = best_estimator_boost.estimators_[0][0]
plt.figure(figsize=(20, 10))
plot_tree(base_tree, filled=True, feature_names=X.columns, max_depth=3)
plt.show()
```



[128]:

```python
var_imp_boost_gbr = best_estimator_boost.feature_importances_
var_imp_df_gbr = pd.DataFrame({'Variable': X.columns, 'Importance': var_imp_boost_gbr})
var_imp_df_gbr = var_imp_df_gbr.sort_values(by='Importance', ascending=False)
var_imp_df_gbr
```

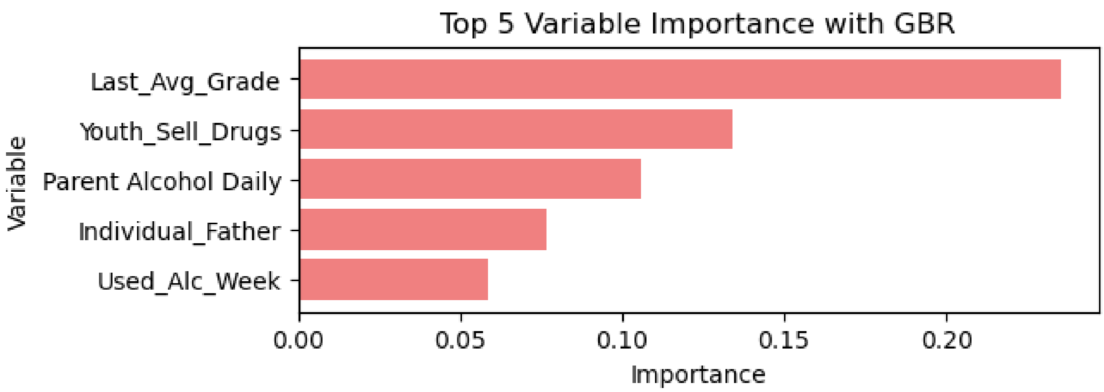
[128]:

	Variable	Importance
2	Last_Avg_Grade	0.235284
15	Youth_Sell_Drugs	0.134170
18	Parent Alcohol Daily	0.105800
42	Individual_Father	0.076716

```
3                        Used_Alc_Week    0.058764
26              Part_Help_Substance_Use    0.041415
43                               Income    0.038911
1                      Teacher_Feedback    0.034979
41                    Individual_Mother    0.034310
35                               Gender    0.022712
39                     Educlation_Level    0.022260
7                      Parents_Limit_TV    0.022189
20                      Friend_Feedback    0.019576
36                                 Race    0.019175
19                        Peer_Drinking    0.018398
5               Parents_HelpHW_LastYear    0.016415
17                       Youth_Attacked    0.014426
14                       Youth_have_Gun    0.014121
16                         Youth_Steals    0.009065
46                   Population Density    0.006937
37                     Health_Condition    0.006483
21                       Share_Problems    0.006132
34                      Religius_Friend    0.006051
11                     Argument_Parents    0.005578
6                    Youth_doing_HChores    0.004212
12                          Youth_Fight    0.004128
27             Part_Preg/STD_Prevention    0.003304
23                 Part_Extracurricular    0.003221
4              Parents_CheckHW_LastYear    0.003139
33                   Religion_Influence    0.003074
13                    Youth_Group_Fight    0.002389
40                       School_Skipped    0.002067
44                     Part_Gov_Program    0.001328
47                           Metro_Size    0.001234
31               Number_Religion_Attend    0.000917
28                       Part_Youth_Act    0.000712
9                 Parents_Appreciation    0.000408
45                        Poverty_Level    0.000000
0                        Exp_of_School    0.000000
38                     Attending_School    0.000000
32             Yth_Believe_Religion Imp    0.000000
30                Education_On_alc+drug    0.000000
29   Yth_seen_alc+drug_prevention_ad    0.000000
25             Part_Substance_Prevention    0.000000
22                   Talked_with_Parents    0.000000
10                        Proud_Parents    0.000000
8                  Parents_Limit_Snacks    0.000000
24              Part_Violence_Prevention    0.000000
```

[129]:
```
top5_var_imp_df_gbr = var_imp_df_gbr.head(5)
plt.figure(figsize=(6, 2))
```

```
plt.barh(top5_var_imp_df_gbr['Variable'], top5_var_imp_df_gbr['Importance'],␣
    ↪color='lightcoral')
plt.xlabel('Importance')
plt.ylabel('Variable')
plt.title('Top 5 Variable Importance with GBR')
plt.gca().invert_yaxis()
plt.show()
```



In the Regressor models we have the best model as Random Forest with Bagging Regressor came up with the lowest MSE of 1.764 in comparision with Decision Tree Regressor with MSE of 1.929 and Gradient Boosting Regressor which has a MSE of 2.082

[130]:
```
print("Binary Classification Results:")
print("Decision Tree Accuracy:", accuracy)
print("Bagging with RF Accuracy:", Accu_bag)
print("Random Forest Accuracy:", rf_accuracy)
print("Gradient Boosting Accuracy:", accuracy_gbm)
print("")

print("Multi-Class Classification Results:")
print("Decision Tree Accuracy:", accuracy_dt)
print("Bagging with Random Forest Accuracy:", acc_bag_rf)
print("Gradient Boosting Accuracy:", multi_accu_boost)
print("")

print("Regression Results:")
print("Decision Tree MSE:", mse_dt)
print("Bagging with Random Forest MSE:", mse_rf_bag)
print("Gradient Boosting MSE:", mse_boost)
```

```
Binary Classification Results:
Decision Tree Accuracy: 0.7448275862068966
```

```
Bagging with RF Accuracy: 0.7639846743295019
Random Forest Accuracy: 0.7547892720306514
Gradient Boosting Accuracy: 0.7693486590038314

Multi-Class Classification Results:
Decision Tree Accuracy: 0.7770114942528735
Bagging with Random Forest Accuracy: 0.7862068965517242
Gradient Boosting Accuracy: 0.7915708812260537

Regression Results:
Decision Tree MSE: 1.9294968496848945
Bagging with Random Forest MSE: 1.7647723669859512
Gradient Boosting MSE: 2.082048518298126
```

The overall results states that in Binary Classification we have the Gradiant Boosting as our best model with 77.77% accuracy. For the Multi-class Classification we have the again Gradiant Boosting Classification as our best best with an Accuracy of 79.23%. In the Regression analysis we have Bagging with Random Forest model which has the lowest MSE as 1.7647.

[ ]: