Analyzing and Predicting Youth Substance Use Based on Family Engagement, Religious Background, Educational Environment, and Personal Experiences

Abstract:

This study leverages the National Survey on Drug Use and Health (NSDUH) to forecast youth substance consumption and the initial age of substance exposure using decision trees and ensemble models. Focusing on predictors like family engagement, educational influences, religious beliefs, and involvement in developmental group activities, this research underscores how early life experiences shape attitudes towards substance use. By integrating behavioral and environmental variables, our main aim through this study is to predict the type of substance that may be used or whether the substance would be used or not, and the age at which initial use occurs. The findings advocate for the critical role of familial and community support in mitigating risky behaviors, providing insights for targeted preventative strategies.

Introduction:

Alcohol consumption among youth is a significant public health issue that can lead to adverse effects on individual health and societal well-being. This study employs data from the National Survey on Drug Use and Health (NSDUH), an annual survey designed to gather information about substance use and related behaviors from U.S. residents aged 12 and older. Through this study, we aim to unearth potential risk and protective factors that are integral to understanding youth behavior towards alcohol. These insights are crucial for developing targeted prevention and intervention strategies that could significantly diminish alcohol consumption among young people.

Our predictive models not only seek to identify the likelihood of alcohol consumption but also aim to categorize potential substance preferences among youth—ranging from alcohol to marijuana and tobacco. Additionally, we utilize regression techniques to estimate the age at which youths might first engage with alcohol. The outcomes of this research will provide valuable guidance for public health initiatives aimed at curbing substance use among adolescents.

Theoretical Background:

Tree-based methods

Tree-based models are highly versatile tools for machine learning, capable of addressing both classification and regression challenges. These models work by partitioning the dataset into subsets based on specific conditions applied to the predictor variables. This process involves recursively dividing the dataset, placing the most significant variables at higher levels of the tree, and discarding less relevant ones.

A typical decision tree is composed of two main types of nodes:

- Decision Nodes: These nodes represent the points where the data is split according to a certain condition. Each decision node has multiple branches leading to other nodes based on the outcomes of the condition.

- Leaf Nodes: These nodes are the terminal points of the tree where no further splitting occurs. Leaf nodes represent the outcome or decision of the model.

The advantages of Tree-based models are that they're easy to interpret and visualize, making them accessible for explaining it to non-technical audiences. They provide insights into the importance of variables used in the prediction process. Decision trees can make predictions quickly, which is good for applications needing real-time decisions.

Overfitting: When decision trees are too deep, they tend to memorize the training data rather than learning to generalize from it, leading to poor predictive performance on new data. To address overfitting, trees can be pruned by cutting off branches that have little influence on the final output. This process helps in simplifying the model and enhancing its generalization abilities. Pruning methods include cost-complexity pruning, reduced error pruning, and minimum description length pruning.

Ensemble Methods

Ensemble methods are advanced techniques in machine learning that combine multiple models to achieve better accuracy and model performance than individual models alone. The two primary types of ensemble methods discussed in this context are Bagging and Boosting.

Bagging involves creating multiple models (usually decision trees), each trained on a different subset of the training dataset. After training, the predictions from all models are combined (typically by averaging) to form a final prediction. This technique reduces variance and helps avoid overfitting.

Random Forest: An extension of the bagging technique specifically for tree-based models. In a Random Forest, each tree in the ensemble is built from a random sample of the data, and only a subset of features is chosen at random at each split. This diversity among the trees leads to a more robust and accurate ensemble.

Boosting: Boosting builds an ensemble by sequentially training models, each compensating for the weaknesses of its predecessors. The final model, therefore, is a weighted average of all the models where weights may be assigned according to the individual model's accuracy. Boosting is particularly useful for improving the performance of models that are weak by themselves but can achieve better accuracy when combined.

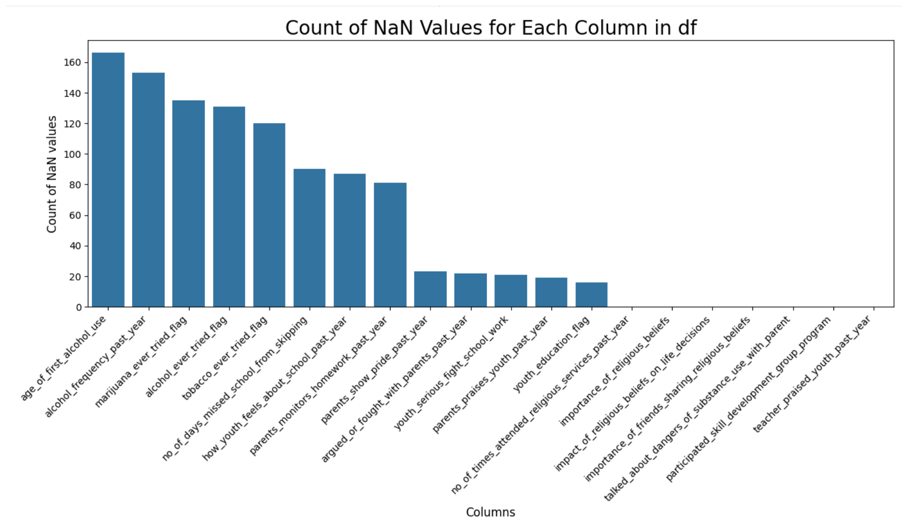
Ensemble methods like Bagging and Boosting are powerful strategies for enhancing the performance of predictive models in machine learning. The application of tree-based models and ensemble methods in predictive analytics represents a sophisticated approach to understanding complex datasets like those involved in studying youth drug use. Each method discussed brings unique advantages and potential drawbacks, making their thoughtful integration crucial to the success of a predictive model.

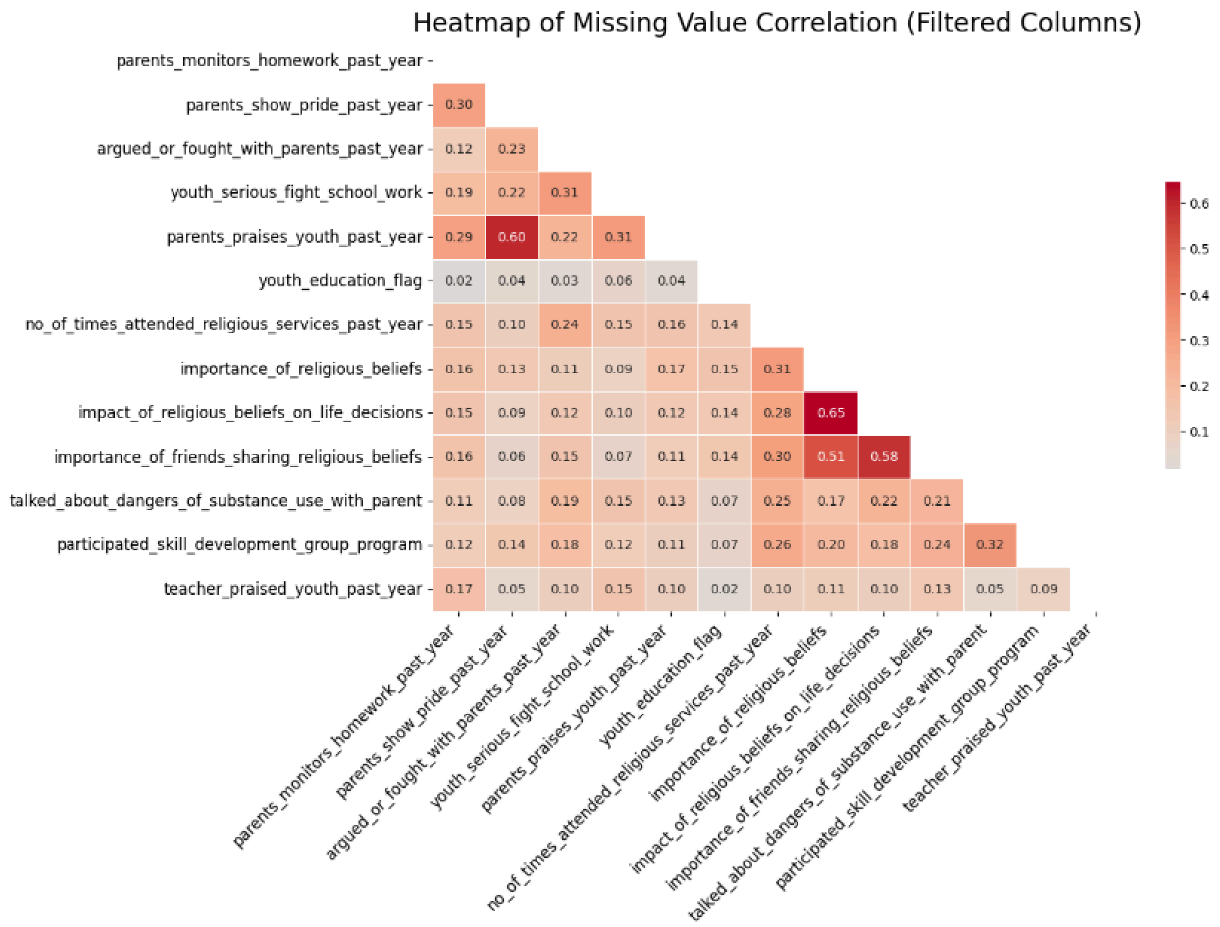
By carefully balancing the strengths and limitations of each method, this research not only advances our understanding of an important societal issue but also showcases the potential of machine learning in public health analytics.

Methodology:

The objective of this study was to predict both the likelihood of substance use among youth and the age at which they might first encounter substances such as alcohol, tobacco, and marijuana. The methodology was rigorously designed to analyze the National Survey on Drug Use and Health (NSDUH) dataset comprehensively, focusing particularly on cleaning the data and preparing it for accurate modeling.

Initial steps in the data preparation phase involved the removal of irrelevant variables that did not contribute to the prediction models. This was followed by a detailed examination of missing values within the dataset to ensure the integrity and completeness of the analysis. Variables that showed a strong correlation with alcohol consumption were specifically excluded to prevent multicollinearity, which could skew the predictive models.





Since removing these columns could introduce bias in the model, we imputed the missing values using the Simple Imputer method and replaced them with 0. The column EDUSCHGRD2 that describes the grade level of youth had many categories and for our analysis, knowing whether someone either went to school or not is enough. This variable was modified to a binary variable youth_education_flag with 98 denoting missing values or unknown response. We take care of this later in our analysis.

```
youth_education_flag
4      869
5      828
3      823
6      818
7      707
99     667
8      394
2      281
98      81
1       22
9        8
10       2
Name: count, dtype: int64
```

```
youth_education_flag
1     4752
0      667
98      81
Name: count, dtype: int64
```

The core of our methodology involved employing decision tree models to predict alcohol consumption based on a wide array of demographic and behavioral factors, including age, gender, race, educational attainment, and family income. These predictors were chosen due to their potential relevance to substance use patterns as indicated by prior research.

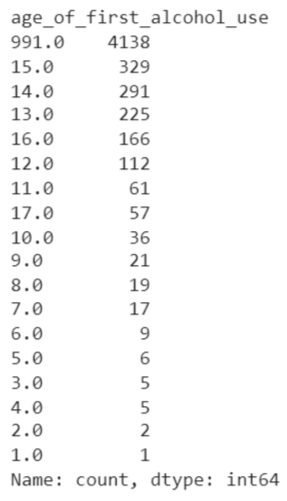
Classification Models:

Initially, we utilized decision trees for a simple binary classification task—to determine whether an individual has consumed alcohol. To enhance the robustness of our predictions, we applied cross-validation techniques, systematically partitioning the data into subsets, training the model on one subset, and validating it on another. This approach helps in minimizing bias and variance, ensuring that our model generalizes well to new data.

Subsequently, we expanded our analysis to a multi-class classification setting where the objective was to identify whether a respondent's substance use involved alcohol, marijuana, or tobacco. This was operationalized by integrating individual usage flags into a single 'substance use flag', thereby allowing for a comparative analysis across different types of substance use.

Regression Models for Age Prediction:

Parallel to our classification efforts, we developed a regression model aimed at predicting the age at which an individual first consumes alcohol. Recognizing outliers in the dataset—such as reports of alcohol consumption at improbably early ages—we corrected these by setting a reasonable minimum age threshold. This helped in maintaining the statistical integrity of our model.



```
age_of_first_alcohol_use
991.0    4138
15.0      329
14.0      291
13.0      225
16.0      166
12.0      112
11.0       61
17.0       57
10.0       36
9.0        21
8.0        19
7.0        17
6.0         9
5.0         6
3.0         5
4.0         5
2.0         2
1.0         1
Name: count, dtype: int64
```

Figure 1

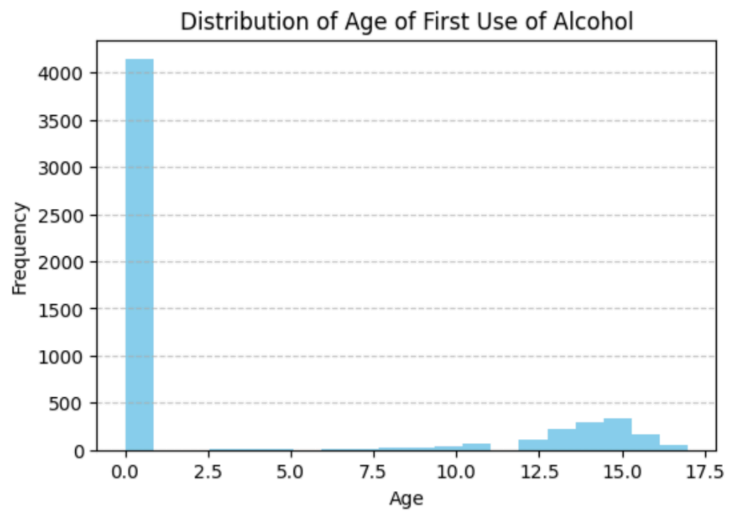
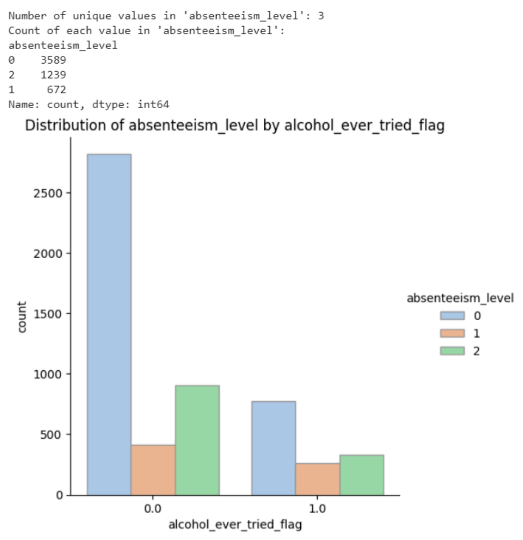


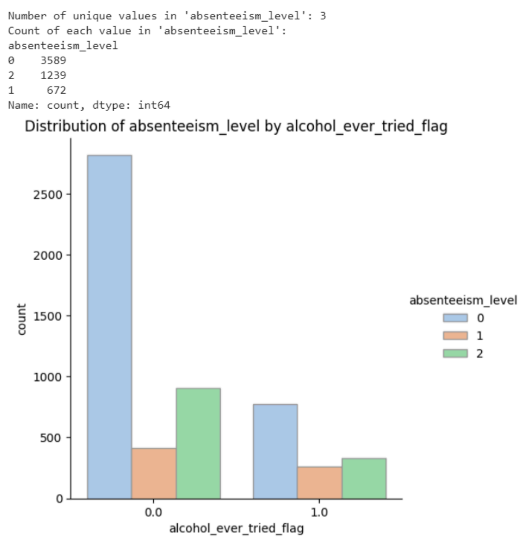
Figure 2

We thoroughly examined the distribution of ages at which alcohol consumption typically begins, as illustrated in Figure 2. Our observations indicated a common initiation age around 15 years, with a noteworthy proportion of youths yet to initiate alcohol use. These insights were crucial for setting up our regression model, which aims to predict the age of first alcohol use accurately.

The variable *eduskpcom* denotes the number of days youth missed school from skipping. This variable was also distributed over a range of numeric values. We modified this variable and split it into three categories and renamed it to absenteeism_level: low, high, medium as shown in the plots below.



```
Number of unique values in 'absenteeism_level': 3
Count of each value in 'absenteeism_level':
absenteeism_level
0    3589
2    1239
1     672
Name: count, dtype: int64
```

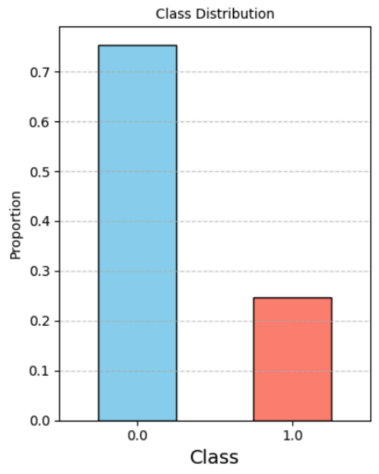


```
Number of unique values in 'absenteeism_level': 3
Count of each value in 'absenteeism_level':
absenteeism_level
0    3589
2    1239
1     672
Name: count, dtype: int64
```

Most of the variables had classes 1 or 2. To make the data consistent throughout the dataset, we modified these columns to have binary values 0(where it was 1) or 1(where it was 2).

Computational Results:

Binary Classification: In our binary classification task, the objective was to predict whether or not a youth would engage in alcohol consumption. We encountered a significant class imbalance in the dataset: out of 5,550 data points, 4,138 indicated no alcohol consumption among adults.



To tackle this imbalance and enhance the reliability of our predictive models, we implemented stratified sampling techniques. This approach ensured that each class was adequately represented during the training of our decision tree and ensemble models, thus mitigating potential biases due to uneven class distributions. The figure below shows the important variables of decision tree classifiers when fitting the model on training data. It is evident that relationship of youth with Parents and religious beliefs is a strong predictor.

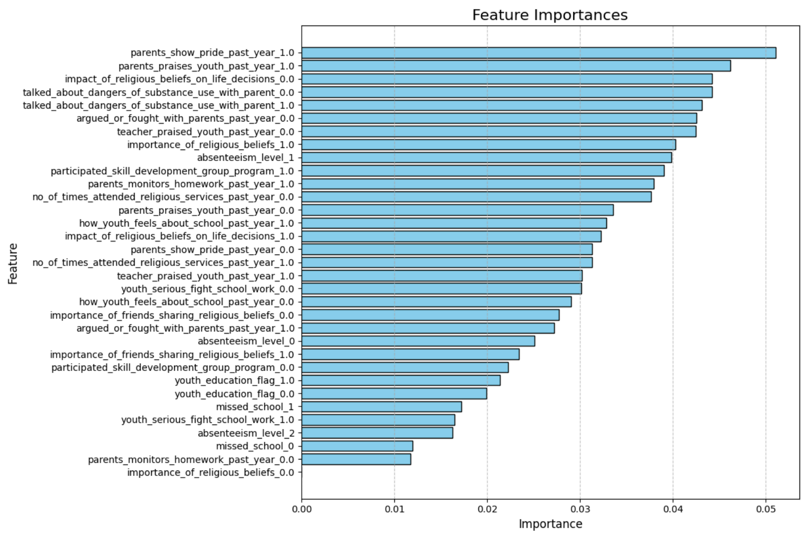
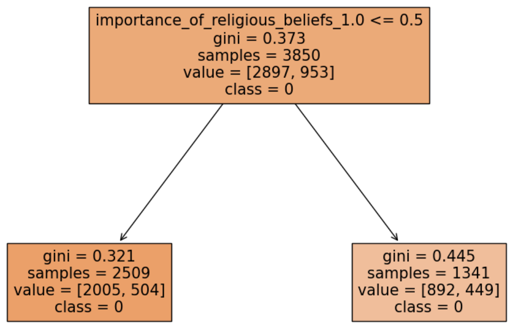
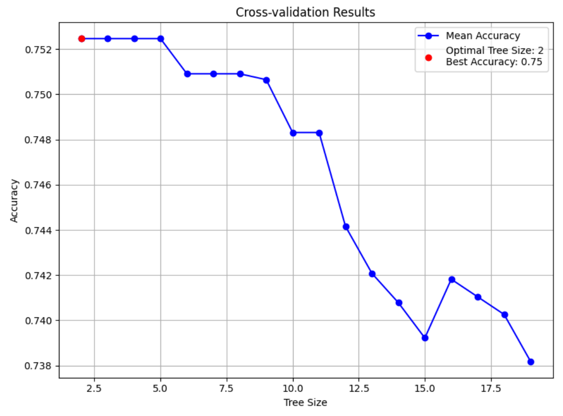


Figure 3: Importance of features in classifying whether a person will consume alcohol or not.





```
Optimal Tree Size: 2
Best Accuracy: 0.7524675324675325
```

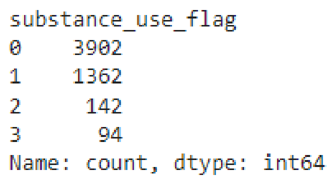
Pruned tree and cross validation results

It's particularly noteworthy that the pruned decision tree model achieved an accuracy of 75.2% on the test set, which is identical to that of a simpler, single-node unpruned decision tree with 71.82%. This observation is intriguing as it suggests that both models—despite their differences in complexity—encounter the same level of error when predicting alcohol consumption among youth.

It indicates that simplifying the decision tree, through the process of pruning, did not compromise its predictive power. In fact, the pruned tree maintained high accuracy while potentially offering benefits in terms of model interpretability and generalizability. This similarity in performance also raises questions about the underlying distribution of the data and the factors most predictive of substance use, suggesting that a minimal set of features may be adequate for achieving significant predictive accuracy.

These findings underscore the importance of feature selection and model complexity in the context of predictive modeling. They highlight that more complex models, such as an extensive decision tree, do not necessarily provide better accuracy than their simpler counterparts. This insight is crucial for optimizing model performance in practical applications, where simplicity and efficiency are often as valued as accuracy.

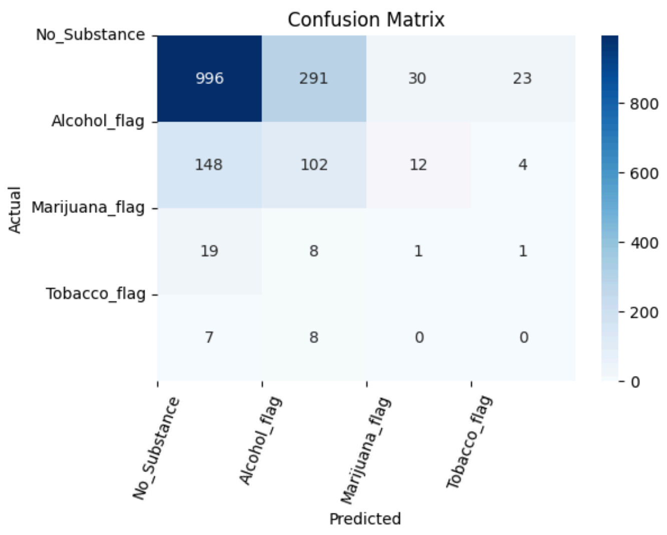
Multiclass classification: In our multiclass classification analysis, the aim is to predict whether an individual might use any type of substance. For this purpose, we have organized the possible outcomes into four distinct categories as previously described.



```
substance_use_flag
0    3902
1    1362
2     142
3      94
Name: count, dtype: int64
```

Figure 4: Multiclass variable

```
Accuracy: 0.666060606060606
```



Confusion matrix of multiclass classification

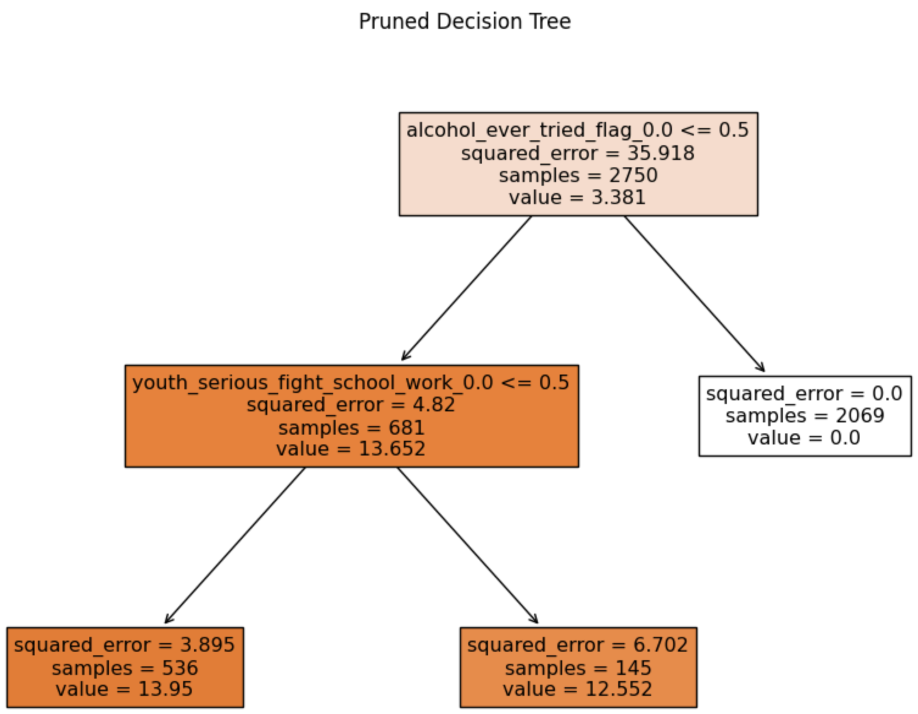
After exploring various modeling approaches, we found that the bagging model stood out as the most effective for analyzing our dataset. This ensemble method significantly improved prediction accuracy, particularly in scenarios involving non-use of substances, as evidenced by the detailed results shown in the confusion matrix.

The bagging model's strength lies in its robustness and ability to generalize across different cases, resulting in a commendable overall accuracy rate of 69.58%. It is important to highlight that while the model performs exceptionally well in predicting cases of non-substance use, there is still room for improvement in its predictive capabilities for other categories.

Further refinement of the model could potentially be achieved by incorporating more comprehensive data on youth behavior. By expanding our dataset to include additional behavioral indicators, we anticipate that the model's accuracy could be enhanced even further. This approach would allow for a more nuanced analysis of the factors influencing substance use among youth, thereby improving our ability to predict various outcomes more effectively.

Regression Model Analysis:

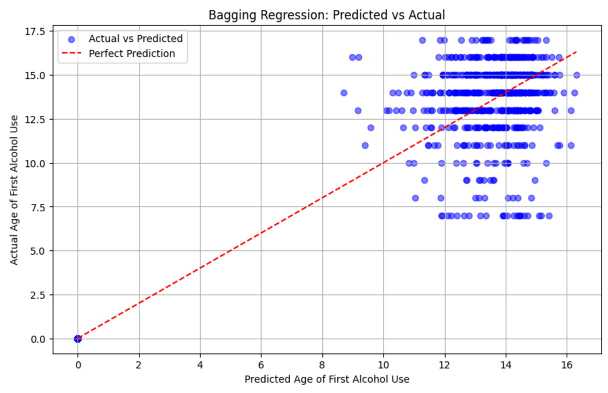
Our regression model aimed to predict the age at which young individuals are likely to have their first alcoholic drink. We employed a variety of modeling techniques to achieve this, with a focus on using a pruned decision tree for regression analysis. This method proved effective, allowing us to estimate the age of initial alcohol use with an impressive accuracy of approximately 96.84%, despite some inherent limitations in the data.

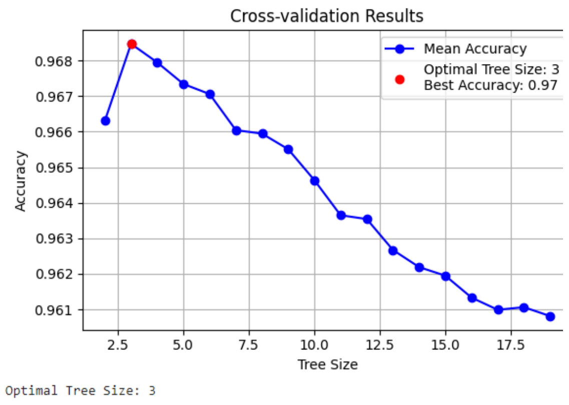


Throughout our analysis, it became evident that the presence of an "alcohol use" indicator within the pruned decision tree was a critical factor in predicting the onset age of drinking. Specifically, whether or not an individual had previously consumed alcohol was found to significantly influence the age at which they first experimented with alcohol.

Moreover, our study uncovered that experiences involving serious fights at school or work were the second most influential predictor in our model. This finding highlights the potential psychological impact of such violent incidents on young people's behavior, suggesting a possible link to earlier substance experimentation and subsequent long-term mental health issues, such as depression. This insight underscores the importance of considering social and behavioral contexts when assessing risk factors for early alcohol use.

Bagging proved to be highly accurate but the pruned decision tree has outperformed it with an accuracy of 96.84% as shown below.





Discussion:

The NSDUH dataset presents a comprehensive overview of behavioral and demographic variables that offer valuable insights into substance use patterns. While our study primarily focused on the impact of religious beliefs and educational experiences on youth alcohol consumption, these are just a fraction of the potential analyses that this rich dataset supports. Our findings highlight significant risk and protective factors that could be instrumental in shaping effective prevention and intervention strategies targeted at reducing substance use among youth.

Influential Factors in Substance Use: Our analysis revealed that parental communication about substance use, the child's academic achievements, and their attitudes towards school play pivotal roles in influencing their likelihood of consuming alcohol. Specifically, positive school experiences and open conversations about substance risks are linked to a lower likelihood of substance use, underscoring the importance of supportive educational and familial environments.

The decision tree model, which was pruned to optimize performance without compromising accuracy, demonstrated a commendable accuracy rate of 75.2% in predicting whether an individual would consume alcohol. This indicates the model's effectiveness in capturing the complex interplay of factors that contribute to alcohol consumption among youth.

Multiclass Classification Findings: In terms of multiclass classification, the goal was to ascertain broader substance use patterns, including alcohol, tobacco, and marijuana use. The results indicated that besides the factors mentioned above, participation in self-esteem and problem-solving groups also emerged as crucial predictors. This suggests that interventions aimed at building these skills could significantly mitigate substance use incidences.

However, it is important to acknowledge certain limitations in our study. The cross-sectional design of the NSDUH dataset restricts our ability to draw causal inferences about the relationships between behavioral factors and substance use. Additionally, the reliance on self-reported data might introduce biases, such as underreporting due to social desirability, which can affect the accuracy of our predictions. The relatively small sample size could also limit the generalizability of our findings to a broader population.

Despite these limitations, our study provides critical insights into the factors that can reduce the likelihood of alcohol and other substance use among youth. These insights reinforce the value of targeted educational and family-oriented interventions. Looking forward, extending this research to include longitudinal data could help clarify causal relationships and enhance the predictive power of our models. Furthermore, incorporating a larger and more diverse dataset could improve the generalizability of our results, providing a stronger basis for developing nationwide substance abuse prevention programs.

Conclusion:

Our research utilized decision tree models and tree-based ensemble methods, including Random Forest, Bagging, and Boosting, to predict alcohol consumption among youth. Employing data from the NSDUH survey, our models achieved commendable accuracies, highlighting their effectiveness in substance use prediction.

In binary classification tasks, the pruned decision tree model notably achieved an accuracy of 75.24%, while in a more focused analysis, it reached up to 96% accuracy using less than half of the initial predictors. These results not only demonstrate the efficiency of pruning in reducing model complexity but also underscore the potential of these models to identify early alcohol use accurately. Our findings are significant as they affirm that both decision tree models and ensemble techniques are robust tools for predicting whether an individual might engage in alcohol consumption. More importantly, these methods can accurately estimate the age of first alcohol use, providing crucial data for preventive health strategies.

Additional studies could expand on this work by incorporating more diverse datasets or by applying these techniques to predict other types of substance use and behavioral outcomes.

Overall, the effectiveness of our predictive models provides valuable insights for public health officials and educators seeking to implement evidence-based strategies to reduce alcohol misuse among youth. As we continue to refine these models and enhance their accuracy, they hold the potential to significantly impact public health initiatives by enabling more precise and timely interventions.

References:

Substance Abuse and Mental Health Services Administration. (2020). National Survey on Drug Use and Health (NSDUH) 2020. [Codebook]. Retrieved from https://www.datafiles.samhsa.gov/sites/default/files/field-uploads-protected/studies/NSDUH2020/NSDUH-2020-datasets/NSDUH-2020-DS0001/NSDUH-2020-DS0001-info/NSDUH2020-DS0001-info-codebook.pdf.

Appendix:

```
# @title Loading libraries
import pandas as pd
import numpy as np
import missingno as msno
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import preprocessing
from sklearn.impute import SimpleImputer
import matplotlib.pyplot as plt

from sklearn.tree import DecisionTreeClassifier, export_text, plot_tree, DecisionTreeRegressor
from sklearn.preprocessing import OneHotEncoder, LabelBinarizer
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestRegressor,
GradientBoostingRegressor,RandomForestClassifier, GradientBoostingClassifier
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.inspection import PartialDependenceDisplay
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
from sklearn.inspection import PartialDependenceDisplay
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OrdinalEncoder
from sklearn.metrics import classification_report, accuracy_score
from sklearn.metrics import confusion_matrix

# @title Loading Data
file_path_csv = 'youth_data.csv'
youth_data = pd.read_csv(file_path_csv)
youth_data.head()
# @title Verifying the shape of the dataframe
youth_data.shape

# @title There are 5500 records and 79 variables. Now, I want to map all variables to their
questions in the codebook, to understand the variables.

"""
'iralcfy': ALCOHOL FREQUENCY PAST YEAR
'irmjfy': MARIJUANA FREQUENCY PAST YEAR
'ircigfm': CIG FREQUENCY PAST MONTH
'IRSMKLSS30N': SMOKELESS TOBACCO FREQUENCY PAST MONTH
'iralcfm': ALCOHOL FREQUENCY PAST MONTH
```

'irmjfm': MARIJUANA FREQUENCY PAST MONTH
'ircigage': CIGARETTE AGE OF FIRST USE
'irsmklsstry': SMOKELESS TOBACCO AGE OF FIRST USE
'iralcage': ALCOHOL AGE OF FIRST USE -
'irmjage': MARIJUANA AGE OF FIRST USE
'mrjflag': MARIJUANA - EVER USED
'alcflag': ILLICIT DRUG, TOBACCO PRD, OR ALCOHOL - EVER USED
'tobflag': ANY TOBACCO - EVER USED
'alcydays': # OF DAYS USED ALCOHOL IN PAST YEAR
'mrjydays': # OF DAYS USED MARIJUANA IN PAST YEAR
'alcmdays': # OF DAYS USED ALCOHOL IN PAST MONTH
'mrjmdays': # OF DAYS USED MARIJUANA IN PAST MONTH
'cigmdays': # OF DAYS USED CIGARETTES IN PAST MONTH
'smklsmdays': # OF DAYS USED SMOKELESS TOBACCO IN PAST MONTH
'schfelt': HOW YTH FELT: ABOUT GOING TO SCHOOL IN PST YR
'tchgjob': TEACHER LET YTH KNOW DOING GOOD JOB IN PST YR
'avggrade': GRADE AVERAGE FOR LAST GRADING PERIOD COMPLETED
'stndscig': STUDENTS IN YTH GRADE SMOKE CIGARETTES
'stndsmj': STUDENTS IN YTH GRADE USE MARIJUANA
'stndalc': STUDENTS IN YTH GRADE DRINK ALCOHOLIC BEVERAGES
'stnddnk': PARENTS CHECK IF HOMEWORK DONE IN PST YR
'parchkhw': PARENTS CHECK IF HOMEWORK DONE IN PST YR
'parhlphw': PARENTS HELP WITH HOMEWORK IN PST YR
'PRCHORE2': PARENTS MAKE YTH DO CHORES AROUND HOUSE IN PST YR
'PRLMTTV2': PARENTS LIMIT AMOUNT OF TV IN PST YR
'parlmtsn': PARENTS LIMIT TIME OUT ON SCHOOL NIGHT IN PST YR
'PRGDJOB2': PARENTS TELL YTH HAD DONE GOOD JOB IN PST YR
'PRPROUD2': PARENTS TELL YTH PROUD OF THINGS DONE IN PST YR
'argupar': TIMES ARGUED/HAD A FIGHT WITH ONE PARENT IN PST YR
'YOFIGHT2': YOUTH HAD SERIOUS FIGHT AT SCHOOL/WORK
'YOGRPFT2': YOUTH FOUGHT WITH GROUP VS OTHER GROUP
'YOHGUN2': YOUTH CARRIED A HANDGUN
'YOSELL2': YOUTH SOLD ILLEGAL DRUGS
'YOSTOLE2': YOUTH STOLE/TRIED TO STEAL ITEM >$50
'YOATTAK2': YOUTH ATTACKED WITH INTENT TO SERIOUSLY HARM
'PRPKCIG2': YTH THINK: PARENTS FEEL ABT YTH SMOKE PACK CIG/DAY
'PRMJEVR2': YTH THINK: PARENTS FEEL ABT YTH TRY MARIJUANA
'prmjmo': YTH THINK: PARENTS FEEL ABT YTH USE MARIJUANA MNTHLY
'PRALDLY2': YTH THINK: PARNTS FEEL ABT YTH DRK 1-2 ALC BEV/DAY
'YFLPKCG2': HOW YTH FEELS: PEERS SMOKE PACK/DAY CIG
'YFLTMRJ2': HOW YTH FEELS: PEERS TRY MARIJUANA
'yflmjmo': HOW YTH FEELS: PEERS USING MARIJUANA MONTHLY
'YFLADLY2': HOW YTH FEELS: PEERS DRNK 1-2 ALC BEV/DAY
'FRDPCIG2': YTH THINK: CLSE FRND FEEL ABT YTH SMK 1+ PAC DAILY
'FRDMEVR2': YTH THINK: CLOSE FRNDS FEEL ABT YTH TRY MARIJUANA
'frdmjmon': YTH THINK: CLSE FRNDS FEEL ABT YTH USE MARIJUANA MON

```
'FRDADLY2': YTH THINK: CLSE FRND FEEL ABT YTH HAVE 1-2 ALC/DAY
'talkprob': WHO YTH TALKS WITH ABOUT SERIOUS PROBLEMS
'PRTALK3': TALKED WITH PARENT ABOUT DANGER TOB/ALC/DRG
'PRBSOLV2': PARTICIPATED IN PRBSLV/COMMSKILL/SELFESTEEM GROUP
'PREVIOL2': PARTICIPATED IN VIOLENCE PREVENTION PROGRAM
'PRVDRGO2': PARTICIPATED IN SUBSTANCE PREV PROGRAM OUTSIDE SCHOOL
'GRPCNSL2': PARTICIPATED IN PROGRAM TO HELP SUBSTANCE USE
'PREGPGM2': PARTICIPATED IN PREG/STD PREVENTION PROGRAM
'YTHACT2': YTH PARTICIPATED IN YOUTH ACTIVITIES
'DRPRVME3': YTH SEEN ALC OR DRUG PREVENTION MESSAGE OUTSIDE
SCHOOL
'ANYEDUC3': YTH HAD ANY DRUG OR ALCOHOL EDUCATION IN SCHOOL
'rlgattd': NUMBER TIMES ATTEND RELIGIOUS SERVICES IN PST YR
'rlgimpt': RELIGIOUS BELIEFS VERY IMPORTANT IN LIFE
'rlgdcsn': RELIGIOUS BELIEFS INFLUENCE LIFE DECISIONS
'rlgfrnd': IMPORTANT FOR FRIENDS TO SHARE RELIGIOUS BELIEFS
'irsex': IMPUTATION REVISED GENDER
'NEWRACE2': RACE/HISPANICITY RECODE (7 LEVELS)
'HEALTH2': OVERALL HEALTH RECODE
'eduschlgo': NOW GOING TO SCHOOL
'EDUSCHGRD2': WHAT GRADE IN NOW/WILL BE IN
'eduskpcom': HOW MANY DAYS MISSED SCHOOL FROM SKIPPING (COMBINED)
'imother': MOTHER IN HH
'ifather': FATHER IN HH
'income': TOTAL FAMILY INCOME RECODE
'govtprog': PARTICIPATED IN ONE OR MORE GOVT ASSIST PROGRAMS
'POVERTY3': POVERTY LEVEL (% OF US CENSUS POVERTY THRESHOLD)
'PDEN10': POPULATION DENSITY 2010 - THREE LEVELS
'COUTYP4': COUNTY METRO/NONMETRO STATUS (2013 3-LEVEL)
"""
print('')
# @title I'm going to seggregate the columns based on their categories like related to Alcohol,
Marijuana, etc.

## FAMILY 15 cols
'''
'imother': MOTHER IN HH
'ifather': FATHER IN HH
'income': TOTAL FAMILY INCOME RECODE
'govtprog': PARTICIPATED IN ONE OR MORE GOVT ASSIST PROGRAMS
'POVERTY3': POVERTY LEVEL (% OF US CENSUS POVERTY THRESHOLD)
'PDEN10': POPULATION DENSITY 2010 - THREE LEVELS
'COUTYP4': COUNTY METRO/NONMETRO STATUS (2013 3-LEVEL)
'NEWRACE2': RACE/HISPANICITY RECODE (7 LEVELS)
'talkprob': WHO YTH TALKS WITH ABOUT SERIOUS PROBLEMS
'rlgattd': NUMBER TIMES ATTEND RELIGIOUS SERVICES IN PST YR
```

```
'rlgimpt': RELIGIOUS BELIEFS VERY IMPORTANT IN LIFE
'rlgdcsn': RELIGIOUS BELIEFS INFLUENCE LIFE DECISIONS
'rlgfrnd': IMPORTANT FOR FRIENDS TO SHARE RELIGIOUS BELIEFS
'irsex': IMPUTATION REVISED GENDER
'HEALTH2': OVERALL HEALTH RECODE
'''

## SCHOOL 8 cols

'''
'schfelt': HOW YTH FELT: ABOUT GOING TO SCHOOL IN PST YR
'tchgjob': TEACHER LET YTH KNOW DOING GOOD JOB IN PST YR
'avggrade': GRADE AVERAGE FOR LAST GRADING PERIOD COMPLETED
'ANYEDUC3': YTH HAD ANY DRUG OR ALCOHOL EDUCATION IN SCHOOL
'DRPRVME3': YTH SEEN ALC OR DRUG PREVENTION MESSAGE OUTSIDE
SCHOOL
'eduschlgo': NOW GOING TO SCHOOL
'eduskpcom': HOW MANY DAYS MISSED SCHOOL FROM SKIPPING (COMBINED)
'EDUSCHGRD2': WHAT GRADE IN NOW/WILL BE IN
'''

## PARENTS 10 cols
'''
'stnddnk': PARENTS CHECK IF HOMEWORK DONE IN PST YR
'parchkhw': PARENTS CHECK IF HOMEWORK DONE IN PST YR
'parhlphw': PARENTS HELP WITH HOMEWORK IN PST YR
'PRCHORE2': PARENTS MAKE YTH DO CHORES AROUND HOUSE IN PST YR
'PRLMTTV2': PARENTS LIMIT AMOUNT OF TV IN PST YR
'parlmtsn': PARENTS LIMIT TIME OUT ON SCHOOL NIGHT IN PST YR
'PRGDJOB2': PARENTS TELL YTH HAD DONE GOOD JOB IN PST YR
'PRPROUD2': PARENTS TELL YTH PROUD OF THINGS DONE IN PST YR
'argupar': TIMES ARGUED/HAD A FIGHT WITH ONE PARENT IN PST YR
'PRTALK3': TALKED WITH PARENT ABOUT DANGER TOB/ALC/DRG
'''

## CIGARETTE 7 cols

'''
'ircigfm': CIG FREQUENCY PAST MONTH
'ircigage': CIGARETTE AGE OF FIRST USE
'cigmdays': # OF DAYS USED CIGARETTES IN PAST MONTH
'stndscig': STUDENTS IN YTH GRADE SMOKE CIGARETTES
'PRPKCIG2': YTH THINK: PARENTS FEEL ABT YTH SMOKE PACK CIG/DAY
'YFLPKCG2': HOW YTH FEELS: PEERS SMOKE PACK/DAY CIG
'FRDPCIG2': YTH THINK: CLSE FRND FEEL ABT YTH SMK 1+ PAC DAILY
'''
```

```
## Alcohol 10 cols

'''
'iralcfy': ALCOHOL FREQUENCY PAST YEAR
'iralcfm': ALCOHOL FREQUENCY PAST MONTH
'iralcage': ALCOHOL AGE OF FIRST USE
'alcflag': ILLICIT DRUG, TOBACCO PRD, OR ALCOHOL - EVER USED
'alcydays': # OF DAYS USED ALCOHOL IN PAST YEAR
'alcmdays': # OF DAYS USED ALCOHOL IN PAST MONTH
'stndalc': STUDENTS IN YTH GRADE DRINK ALCOHOLIC BEVERAGES
'PRALDLY2': YTH THINK: PARNTS FEEL ABT YTH DRK 1-2 ALC BEV/DAY
'YFLADLY2': HOW YTH FEELS: PEERS DRNK 1-2 ALC BEV/DAY
'FRDADLY2': YTH THINK: CLSE FRND FEEL ABT YTH HAVE 1-2 ALC/DAY
'''

## Marijuana 13 cols

'''
'irmjfy': MARIJUANA FREQUENCY PAST YEAR
'irmjfm': MARIJUANA FREQUENCY PAST MONTH
'irmjage': MARIJUANA AGE OF FIRST USE
'mrjflag': MARIJUANA - EVER USED
'mrjydays': # OF DAYS USED MARIJUANA IN PAST YEAR
'mrjmdays': # OF DAYS USED MARIJUANA IN PAST MONTH
'stndsmj':  STUDENTS IN YTH GRADE USE MARIJUANA
'PRMJEVR2': YTH THINK: PARENTS FEEL ABT YTH TRY MARIJUANA
'prmjmo': YTH THINK: PARENTS FEEL ABT YTH USE MARIJUANA MNTHLY
'YFLTMRJ2': HOW YTH FEELS: PEERS TRY MARIJUANA
'yflmjmo': HOW YTH FEELS: PEERS USING MARIJUANA MONTHLY
'FRDMEVR2': YTH THINK: CLOSE FRNDS FEEL ABT YTH TRY MARIJUANA
'frdmjmon': YTH THINK: CLSE FRNDS FEEL ABT YTH USE MARIJUANA MON
'''

## tobacco 4 cols

'''
'IRSMKLSS30N': SMOKELESS TOBACCO FREQUENCY PAST MONTH
'irsmklsstry': SMOKELESS TOBACCO AGE OF FIRST USE
'tobflag': ANY TOBACCO - EVER USED
'smklsmdays': # OF DAYS USED SMOKELESS TOBACCO IN PAST MONTH
'''

## youth fights 6 cols

'''
```

```
'YOFIGHT2': YOUTH HAD SERIOUS FIGHT AT SCHOOL/WORK
'YOGRPFT2': YOUTH FOUGHT WITH GROUP VS OTHER GROUP
'YOHGUN2': YOUTH CARRIED A HANDGUN
'YOSELL2': YOUTH SOLD ILLEGAL DRUGS
'YOSTOLE2': YOUTH STOLE/TRIED TO STEAL ITEM > $50
'YOATTAK2': YOUTH ATTACKED WITH INTENT TO SERIOUSLY HARM
'''

## participation in activities 6 cols

'''
'PRBSOLV2': PARTICIPATED IN PRBSLV/COMMSKILL/SELFESTEEM GROUP
'PREVIOL2': PARTICIPATED IN VIOLENCE PREVENTION PROGRAM
'PRVDRGO2': PARTICIPATED IN SUBSTANCE PREV PROGRAM OUTSIDE SCHOOL
'GRPCNSL2': PARTICIPATED IN PROGRAM TO HELP SUBSTANCE USE
'PREGPGM2': PARTICIPATED IN PREG/STD PREVENTION PROGRAM
'YTHACT2': YTH PARTICIPATED IN YOUTH ACTIVITIES
'''
```

It is obvious that several factors could lead to youth doing substance abuse. And through this Machine Learning Analysis we'll try to predict:

1. Whether youth would consume alcohol or not? (Binary Classfiication)
2. Would youth consume Tobacco, Alcohol, Marijuana or nothing? (Multi class classification)
3. At what age would youth try Marijuana? (Regression)

Before we proceed, I'll clean the data further. So that our analysis is backed by good data and we can trust the insights more. This also helps in building the models.
Firstly, I want to make a new dataframe to only have columns that seem important or relevant based on their descriptions from the codebook.
Creating a dataframe of relevant columns related to youth's education status, drug use, family, religion and teachers:

```
df = youth_data[['iralcage', 'iralcfy', 'mrjflag', 'alcflag', 'tobflag',
 'eduskpcom', 'schfelt', 'parchkhw', 'PRPROUD2', 'argupar',
 'YOFIGHT2', 'PRGDJOB2', 'EDUSCHGRD2', 'rlgattd', 'rlgimpt',
 'rlgdcsn', 'rlgfrnd', 'PRTALK3', 'PRBSOLV2', 'tchgjob']].copy()

df.head()
```

Let's rename the columns names for better understanding and to make readable plots.

```
col_names = {
    'iralcage': 'age_of_first_alcohol_use',
    'iralcfy': 'alcohol_frequency_past_year',
    'mrjflag': 'marijuana_ever_tried_flag',
    'alcflag': 'alcohol_ever_tried_flag',
    'tobflag': 'tobacco_ever_tried_flag',
    'eduskpcom': 'no_of_days_missed_school_from_skipping',
    'schfelt': 'how_youth_feels_about_school_past_year',
    'parchkhw': 'parents_monitors_homework_past_year',
    'PRPROUD2': 'parents_show_pride_past_year',
    'argupar': 'argued_or_fought_with_parents_past_year',
    'YOFIGHT2': 'youth_serious_fight_school_work',
    'PRGDJOB2': 'parents_praises_youth_past_year',
    'EDUSCHGRD2': 'youth_education_flag',
    'rlgattd': 'no_of_times_attended_religious_services_past_year',
    'rlgimpt': 'importance_of_religious_beliefs',
    'rlgdcsn': 'impact_of_religious_beliefs_on_life_decisions',
    'rlgfrnd': 'importance_of_friends_sharing_religious_beliefs',
    'PRTALK3': 'talked_about_dangers_of_substance_use_with_parent',
    'PRBSOLV2': 'participated_skill_development_group_program',
    'tchgjob': 'teacher_praised_youth_past_year',
}
df = df.rename(columns=col_names)
df.head()
```

youth_education_flag: simplifying the EDUSCHGRD2 column that has grade levels, to make it a flag of whether youth has been to school or not.

```
df['youth_education_flag'].value_counts()
# @title making the variable binary by combining all grades(went to school) into one value 1 and respone 99 as 0(didn't go to school)
df['youth_education_flag'] = df['youth_education_flag'].replace(to_replace=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11], value=1)
df['youth_education_flag'] = df['youth_education_flag'].replace(to_replace=[99], value=0)
df['youth_education_flag'].value_counts()
```

There are 81 people who didn't answer about their education. This might influence the model. So I want to replace these values with NaN. And later perform imputation on it.

```
# Recoding the '98' values in 'youth_education_flag' to NaN for imputation
df['youth_education_flag'] = df['youth_education_flag'].replace(98, np.nan)

# Verify the recoding
education_flag_counts = df['youth_education_flag'].value_counts(dropna=False)

education_flag_counts
# @title Checking for missing values

missing_values = df.isnull().sum()
missing_values
```

Let's vosualize this to get a better view of the missing values

```
plt.figure(figsize=(15, 5))
sns.barplot(x=df.columns, y=df.isnull().sum().sort_values(ascending = False))
plt.title('Count of NaN Values for Each Column in df', fontsize=20)
plt.xticks(rotation=45, ha='right')
plt.ylabel('Count of NaN values', fontsize=12)
plt.xlabel('Columns', fontsize=12)
plt.show()
```

Missing values doesn't seem to be a big problem in our dataset, let's look into it more closely.

```
# First we will filter out the columns that have missing values, we don't need to analyse all columns.
columns_with_missing_values = df.columns[df.isnull().any()].tolist()

plt.figure(figsize=(20, 8))
sns.heatmap(df[columns_with_missing_values].isnull(), cbar=False, cmap='viridis')
plt.title('Matrix of Missing Values in df', fontsize=20)
plt.xlabel('Columns', fontsize=15)
plt.ylabel('Data Index', fontsize=15)
plt.xticks(rotation=45, ha='right', fontsize=10)
plt.yticks(fontsize=10)
plt.show()
```

- The matrix plot shows a non-random pattern of missingness for certain records and this could be a sign that the data are missing not at random(MNAR) or at least missing at random(MAR), but with some structure. This means that the probability of missing data on certain variables is influenced by other variables in the data.
- Cases when data has structured missingness, simple imputation techniques can introduce bias. So I want to explore other imputation methods.

- But first, let's also verify this pattern using a heatmap. Because heatmaps are good at showing correlations.

```
# Now, calculate the correlation matrix only for columns with missing values
missing_corr_filtered = df[columns_with_missing_values].isnull().corr()

# Create the mask for the upper triangle
mask_filtered = np.triu(np.ones_like(missing_corr_filtered, dtype=bool))

# Plotting the heatmap for the filtered correlation matrix
plt.figure(figsize=(12, 8))
sns.heatmap(missing_corr_filtered, mask=mask_filtered, cmap='coolwarm', center=0,
            annot=True, fmt=".2f", linewidths=.5, cbar_kws={"shrink": .5})
plt.xticks(rotation=45, ha='right', fontsize=12)
plt.yticks(rotation=0, fontsize=12)
plt.title('Heatmap of Missing Value Correlation (Filtered Columns)', fontsize=20)
plt.show()
```

Strong correlation between:

- impact_of_religious_beliefs_on_life_decisions and importance_of_friends_sharing_religious_beliefs
- parents_show_pride_past_year and parents_praises_youth_past_year

```
# @title addressing the missing values using imputation
```

```
imputer = SimpleImputer(strategy="constant")

df_imputed = pd.DataFrame(imputer.fit_transform(df), columns=df.columns)

missing_values = df_imputed.isnull().sum()
missing_values
```

Since we need to convert multiple columns to categorical variables and do one-hot-encoding. I'll be using these basic functions to make it easier to convert multiple variables.
- **convert_to_category**
- **one_hot_encoding**

I'm using the **find_unique_count_and_plot** function to get a clear picture of each variable and compare it against the response variable.

```
# Function to find unique count and plot distribution of a categorical variable
def find_unique_count_and_plot(frame, column, response=None):
    print(f"Number of unique values in '{column}': {frame[column].nunique()}")
    print(f"Count of each value in '{column}':\n{frame[column].value_counts()}")

    # If response variable is provided, plotting the distribution of the column against the
response
    if response is not None:
        sns.catplot(data=frame, x=response, hue=column, kind='count', palette="pastel",
edgecolor=".6")
        plt.title(f"Distribution of {column} by {response}")
        plt.show()

# Function to convert specified columns to 'category' dtype
def convert_to_category(dataframe, col_names):
    # for col in cols:
      # print(dataframe[col])
    dataframe[col_names.drop('alcohol_ever_tried_flag')] =
dataframe[col_names.drop('alcohol_ever_tried_flag')].astype('category')

      # dataframe[col] = dataframe[col].astype('category')
    return dataframe

#function to convert categorical variable into one hot encoded values
def one_hot_encoding(dataframe, col_name):
    dummy = pd.get_dummies(dataframe[[col_name]])
    res = pd.concat([dataframe, dummy ], axis=1)
    res.drop(col_name,inplace = True,axis = 1)
    return(res)

# @title Creating separate df for binary classification

# df_bc - df for binary classification
```

```
df_bc = df_imputed[['how_youth_feels_about_school_past_year',
'teacher_praised_youth_past_year', 'youth_education_flag',
                'no_of_days_missed_school_from_skipping',
'talked_about_dangers_of_substance_use_with_parent',
                'participated_skill_development_group_program',
'no_of_times_attended_religious_services_past_year',
                'importance_of_religious_beliefs',
'impact_of_religious_beliefs_on_life_decisions',
                'importance_of_friends_sharing_religious_beliefs',
'parents_praises_youth_past_year',
                'youth_serious_fight_school_work', 'argued_or_fought_with_parents_past_year',

'parents_show_pride_past_year','parents_monitors_homework_past_year','alcohol_ever_tried_
flag'
                ]].copy()
```

I looked at the unique values of each column using **find_unique_count_and_plot** function, and noticed that 13 of them have two classes 1 or 2, so I want them to be 0 or 1 for better readability and consistency throughout the dataset.

```
# @title changing the values to 0 or 1 where it has two classes 1 or 2

# find_unique_count_and_plot(df_bc,
'how_youth_feels_about_school_past_year','alcohol_ever_tried_flag')
# find_unique_count_and_plot(df_bc,
'teacher_praised_youth_past_year','alcohol_ever_tried_flag')
# find_unique_count_and_plot(df_bc,
'talked_about_dangers_of_substance_use_with_parent','alcohol_ever_tried_flag')

col_list = ['how_youth_feels_about_school_past_year', 'teacher_praised_youth_past_year',
          'talked_about_dangers_of_substance_use_with_parent',
'participated_skill_development_group_program',
          'no_of_times_attended_religious_services_past_year',
'importance_of_religious_beliefs',
          'impact_of_religious_beliefs_on_life_decisions',
'importance_of_friends_sharing_religious_beliefs',
          'parents_praises_youth_past_year', 'youth_serious_fight_school_work',
          'argued_or_fought_with_parents_past_year', 'parents_show_pride_past_year',
          'parents_monitors_homework_past_year']

# making the values 0 or 1 instead of 1 or 2
for col in col_list:
  df_bc[col].replace({2: 1, 1: 0}, inplace=True)

# plots after updating the columns classes to 0 and 1
for col in col_list:
  find_unique_count_and_plot(df_bc, col,'alcohol_ever_tried_flag')
# @title this column no_of_days_missed_school_from_skipping has a lot of categories.
find_unique_count_and_plot(df_bc,
'no_of_days_missed_school_from_skipping','alcohol_ever_tried_flag')
'no_of_days_missed_school_from_skipping' has a lot of categories, so I've planned  to make
this column simpler by having:
- one column that's 0 if no response and 1 if responded
- one column that is categorical based on the number of days missed. (low, medium, high)
basically a variable indicating the level of absenteeism
# @title creating two columns for col: no_of_days_missed_school_from_skipping
df_bc['missed_school'] = np.where(df_bc['no_of_days_missed_school_from_skipping'] > 0, 1,
0)

# Creating categorical variable indicating level of absenteeism
def categorize_absenteeism(days_missed):
    if days_missed == 0:
        return 0
    elif days_missed <= 5:
        return 1
    else:
        return 2

df_bc['absenteeism_level'] =
df_bc['no_of_days_missed_school_from_skipping'].apply(categorize_absenteeism)

# Dropping the original column
df_bc.drop(columns=['no_of_days_missed_school_from_skipping'], inplace=True)
find_unique_count_and_plot(df_bc, 'missed_school','alcohol_ever_tried_flag')
find_unique_count_and_plot(df_bc, 'absenteeism_level','alcohol_ever_tried_flag')
# @title converting all variables to binary variables and performing one-hot-encoding
convert_to_category(df_bc,df_bc.columns[:-1])

df_bc['absenteeism_level'] = df_bc['absenteeism_level'].astype('category')

for col in df_bc.columns:
  if col != 'alcohol_ever_tried_flag':
      df_bc = one_hot_encoding(df_bc,col)
# df_bc.info()
df_bc.columns[:-1]

# @title Decision tree - using stratify sampling because we have imbalance in the dataset
between class 0 and class 1

X = df_bc.drop(columns=['alcohol_ever_tried_flag'])  # predictors
y = df_bc['alcohol_ever_tried_flag']  # Target variable
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify
= y)

# Initializing the Decision Tree classifier
dt_classifier = DecisionTreeClassifier(random_state=42)

class_distribution = pd.Series(y).value_counts(normalize=True)
print("Target variable class distribution:")
print(class_distribution)

plt.figure(figsize=(4, 5))
class_distribution.plot(kind='bar', color=['skyblue', 'salmon'], edgecolor='black')
plt.title('Class Distribution', fontsize=10)
plt.xlabel('Class', fontsize=14)
plt.ylabel('Proportion', fontsize=10)
plt.xticks(rotation=0)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
# @title Decision tree performance on training data

#Training the classifier
dt_classifier.fit(X_train, y_train)

# Predictions on the training set
y_pred_train = dt_classifier.predict(X_train)

# Accuracy on the training set
accuracy_train = accuracy_score(y_train, y_pred_train)
print("Accuracy on training set:",  round(accuracy_train,4)*100)

# Confusion matrix on the training set
print("Confusion Matrix on training set:")
# print(confusion_matrix(y_train, y_pred_train))
confusion_matrix = pd.crosstab(index=y_pred_train, columns=y_train, rownames=[''])
confusion_matrix
# @title visualizing the tree
plt.figure(figsize=(50,50))
plot_tree(dt_classifier
        , filled=True
        , feature_names=X.columns
        #, class_names=[0, 1]
        , label='all'
        , fontsize=12)
plt.show()
# @title let's se how it performs on the test data
```

```
#Predictions on the testing set
y_pred = dt_classifier.predict(X_test)

#evaluating the classifier
accuracy = accuracy_score(y_test, y_pred)

confusion_matrix = pd.crosstab(index=y_pred, columns=y_test, rownames=[''])

print("Accuracy:",  round(accuracy,4)*100)
print(confusion_matrix)
# print("Classification Report:")
# report = classification_report(y_test, y_pred)
# print(report)
```

The model is good at correctly predicting class 0 but not good at predicting class 1.

```
# @title Let's analyse important features

feature_importances = dt_classifier.feature_importances_
feature_importance_df = pd.DataFrame({'Feature': X.columns, 'Importance':
feature_importances})
feature_importance_df = feature_importance_df.sort_values(by='Importance',
ascending=False)

# top_10_features = feature_importance_df.head(10)

# Plotting the feature importances
plt.figure(figsize=(12, 8))
plt.barh(feature_importance_df['Feature'], feature_importance_df['Importance'],
color='skyblue', edgecolor='black')
plt.xlabel('Importance', fontsize=12)
plt.ylabel('Feature', fontsize=12)
plt.title('Feature Importances', fontsize=16)
 # Inverting y-axis to show most important features at the top
plt.gca().invert_yaxis()
plt.grid(axis='x', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```

The plot confirms our assumptions that predictors related to family, religious beliefs and school is essential in predicting substance use.

Let's try other models

```
# @title random forest
# Initialize the Random Forest classifier
rf_classifier = RandomForestClassifier()

# Train the classifier
rf_classifier.fit(X_train, y_train)
```

```
# Predictions on the testing set
y_pred_rf = rf_classifier.predict(X_test)

# Evaluate the classifier
accuracy_rf = accuracy_score(y_test, y_pred_rf)
print("Random Forest Classifier Accuracy:", round(accuracy_rf,4)*100)

# # Classification report
# print("Classification Report:")
# print(classification_report(y_test, y_pred_rf))

# Confusion matrix
print("Confusion Matrix:")
confusion_matrix = pd.crosstab(index=y_pred, columns=y_test, rownames=['Predicted'],
colnames=['Actual'])

print(confusion_matrix)
the Random Forest Classifier performed slightly better in terms of accuracy compared to the
unpruned decision tree classifier
# fit Random Forests model
bag = RandomForestClassifier(max_features=X_train.shape[1],random_state = 42)
bag.fit(X_train,y_train)

print("Number of trees:", bag.n_estimators)
print("Number of features tried at each split:",bag.max_features)
print("Training score: {:.2f}%".format(bag.score(X_train,y_train)*100))
# Predict values
y_pred = bag.predict(X_test)
print("Test score: {:.2f}%".format(bag.score(X_test,y_test)*100))
confusion_matrix = pd.crosstab(index=y_pred, columns=y_test, rownames=[''])
print(confusion_matrix)
Bagging is slightly better at predicting class 1 compared to unpruned decision tree and random
forest.
let's limit the depth of the tree and see how it looks
let's limit the max depth

from sklearn.tree import export_graphviz
import graphviz

# Limiting the depth of the tree
dt_classifier = DecisionTreeClassifier(max_depth=3)
dt_classifier.fit(X_train, y_train)

dot_data = export_graphviz(dt_classifier, out_file=None,
                    feature_names=X_train.columns, class_names=["0", "1"],
                        filled=True, rounded=True, special_characters=True)
graph = graphviz.Source(dot_data)
graph
```

Root Node:

The root of the tree is the question about the importance_of_religious_beliefs_0.
The gini value here is 0.373. There are 3,850 samples at this node, and it is predominantly of class 0 (2,897 samples of class 0 vs. 953 of class 1).

Left Subtree (True Branch):

If the importance of religious beliefs is low, the tree further asks if parents_monitors_homework_past_year_0 is 0 or below 0.5.

If yes, it then considers youth_serious_fight_school_work_0.

This branch seems to be quite significant in size, with many samples and further splits, indicating that monitoring homework and fights over school work are important factors for classification when religious beliefs are of lesser importance.

Right Subtree (False Branch):

If the importance of religious beliefs is high, the tree looks at how the youth feels about school and attendance at religious services in the past year.
This side of the tree suggests that when religious beliefs are considered more important, feelings about school and religious service attendance play a role in the classification.

```
#Predictions on the testing set
y_pred = dt_classifier.predict(X_test)

#evaluating the classifier
accuracy = accuracy_score(y_test, y_pred)

confusion_matrix = pd.crosstab(index=y_pred, columns=y_test, rownames=[''])

print("Accuracy:",  round(accuracy,4)*100)
print(confusion_matrix)
```

Limiting max depth to 3 had a significant improvement with 75% accuracy and this tree is actually better at classifying the class 1. I reduced the tree size manually by reducing the depth, but let's find this value using cross-validation.

```
# Define the decision tree classifier
tree_classifier = DecisionTreeClassifier(random_state=42)
```

```
# Define the parameters to search over
params = {'max_leaf_nodes': range(2, 20)}

# Perform grid search with 10-fold cross-validation
grid_search = GridSearchCV(estimator=tree_classifier, param_grid=params, cv=10)
grid_search.fit(X_train, y_train)

# Get the results of cross-validation
cv_results = grid_search.cv_results_

# Find the best tree size
best_size = grid_search.best_params_['max_leaf_nodes']
best_score = grid_search.best_score_

# Plot the cross-validation results with enhanced style
plt.figure(figsize=(8, 6))
plt.plot(cv_results["param_max_leaf_nodes"], cv_results["mean_test_score"], 'o-', color='b',
label='Mean Accuracy')
plt.plot(best_size, best_score, 'ro', label=f'Optimal Tree Size: {best_size}\nBest Accuracy:
{best_score:.2f}')
plt.xlabel('Tree Size')
plt.ylabel('Accuracy')
plt.title('Cross-validation Results')
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()

print("Optimal Tree Size:", best_size)
print("Best Accuracy:", best_score)
# @title Creating pruned tree using the optimal size
from sklearn.tree import DecisionTreeClassifier, plot_tree

# Create the pruned decision tree classifier
pruned_tree = DecisionTreeClassifier(max_leaf_nodes=best_size, random_state=42)
pruned_tree.fit(X_train, y_train)

# Plot the pruned decision tree
plt.figure(figsize=(12, 8))
plot_tree(pruned_tree, filled=True, feature_names=X_train.columns, class_names=["0", "1"])
plt.title('Pruned Decision Tree')
plt.show()

# Predictions on the test set
y_pred_pruned = pruned_tree.predict(X_test)
```

```
# Confusion matrix
conf_matrix_pruned = pd.crosstab(index=y_pred_pruned, columns=y_test,
rownames=['Predicted'], colnames=['Actual'])
print("Confusion Matrix for Pruned Decision Tree:")
print(conf_matrix_pruned)
#Calculate the accuracy of the decision tree on the test data
accuracy = pruned_tree.score(X_test, y_test)
print("Accuracy: {:.2f}%".format(accuracy*100))
# @title gradient boosting
gb_classifier = GradientBoostingClassifier(random_state=42,n_estimators=3500,
max_depth=4)

# Training the classifier
gb_classifier.fit(X_train, y_train)

# Predictions on the test set
y_pred_gb = gb_classifier.predict(X_test)

# Accuracy on the test set
accuracy_gb = accuracy_score(y_test, y_pred_gb)
print("Gradient Boosting Classifier Accuracy:", round(accuracy_gb * 100, 2))

# Confusion matrix
conf_matrix_gb = pd.crosstab(index=y_pred_gb, columns=y_test,  rownames=['Predicted'],
colnames=['Actual'])
print("Confusion Matrix:")
print(conf_matrix_gb)
feature_importances = gb_classifier.feature_importances_

# Create a DataFrame to display feature importances
feature_importance_df = pd.DataFrame({'Feature': X_train.columns, 'Importance':
feature_importances})

# Sort the DataFrame by importance in descending order
feature_importance_df = feature_importance_df.sort_values(by='Importance',
ascending=False)


# Plotting the feature importances
plt.figure(figsize=(12, 8))
plt.barh(feature_importance_df['Feature'], feature_importance_df['Importance'],
color='skyblue', edgecolor='black')
plt.xlabel('Importance', fontsize=12)
plt.ylabel('Feature', fontsize=12)
plt.title('Feature Importances', fontsize=16)
 # Inverting y-axis to show most important features at the top
```

```
plt.gca().invert_yaxis()
plt.grid(axis='x', linestyle='--', alpha=0.7)
plt.tight_layout()
PartialDependenceDisplay.from_estimator(gb_classifier, X_train, ['absenteeism_level_1',
'parents_monitors_homework_past_year_0.0']);
```

- The upward slope of the line indicates a positive relationship between absenteeism level and the response variable. As absenteeism_level_1 increases, the response variable also increases.
- The partial_dependence on the y-axis here shows a negative relationship. As the monitoring by parents increases, the expected value of the target variable decreases.

```
# # @title Multiclass classification - merging the columns: alcohol_ever_tried_flag,
tobacco_ever_tried_flag and marijuana_ever_tried_flag

# Create a dictionary to map the values
mapping = {1: 1, 2: 2, 3: 3}

# Creating a list of the columns to use
columns = ['alcohol_ever_tried_flag', 'marijuana_ever_tried_flag', 'tobacco_ever_tried_flag']

# Create a new column using numpy.select
df_imputed['substance_use_flag'] = np.select(
    condlist=[df_imputed[col] == 1 for col in columns],
    choicelist=[mapping[i] for i in range(1, len(columns) + 1)],
    default=0
)

df_imputed['substance_use_flag'] .value_counts()
```

I'll be using df_bc that was used for binary classification. Because I plan to use most of those variables for the multi_class_classification too.

```
df_mc = df_bc.drop(columns=['alcohol_ever_tried_flag']).copy()

df_mc['substance_use_flag'] = df_imputed['substance_use_flag']
df_mc.columns
df_mc.head()
output = ['No_Substance','Alcohol_flag','Marijuana_flag','Tobacco_flag']

X = df_mc.drop(columns=['substance_use_flag'])
y = df_mc['substance_use_flag']

# Splitting the dataset into the training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42,
stratify=y)

# Initialize the Decision Tree classifier
dt_mc_classifier = DecisionTreeClassifier(random_state=42)

# Training the classifier
```

```
dt_mc_classifier.fit(X_train, y_train)

# Predictions on the testing set
y_pred = dt_mc_classifier.predict(X_test)

# Evaluate the classifier
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Confusion matrix
conf_matrix = pd.crosstab(index=y_pred, columns=y_test, rownames=[''])
# print("Confusion Matrix:")
# print(conf_matrix)
# Plotting the heatmap
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, cmap="Blues", fmt="d",
        xticklabels=df_mc['substance_use_flag'].unique(),
        yticklabels=df_mc['substance_use_flag'].unique())
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.yticks(range(4),[out for out in output],rotation = 0)
plt.xticks(range(4),[out for out in output],rotation = 70)
plt.title('Confusion Matrix')
plt.show()
The model is better on class 0 but pretty bad at predicting the output for class 1.

feature_importances = dt_mc_classifier.feature_importances_
feature_importance_df = pd.DataFrame({'Feature': X.columns, 'Importance':
feature_importances})
feature_importance_df = feature_importance_df.sort_values(by='Importance',
ascending=False)

# top_10_features = feature_importance_df.head(10)
feature_importance_df
# @title Random Forest
# Split the Data between test and train
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state= 42,
stratify=y)

# create Random Forest model with class_weight parameter
rf_classifier = RandomForestClassifier()

rf_classifier.fit(X_train, y_train)

# predict on test data
y_pred = rf_classifier.predict(X_test)
```

```
# create confusion matrix
confusion_matrix = pd.crosstab(index=y_pred, columns=y_test, rownames=[''])
print(confusion_matrix)
accuracy = rf_classifier.score(X_test, y_test)
print("Accuracy: {:.2f}%".format(accuracy*100))
Slightly better
# @title Let's try bagging

bagging_mc = RandomForestClassifier(max_features=X_train.shape[1],random_state = 42)
bagging_mc.fit(X_train,y_train)

print("Number of trees:", bagging_mc.n_estimators)
print("Number of features tried at each split:",bagging_mc.max_features)
print("Training score: {:.2f}%".format(bagging_mc.score(X_train,y_train)*100))

y_pred = bagging_mc.predict(X_test)

confusion_matrix = pd.crosstab(index=y_pred, columns=y_test, rownames=[''])
print(confusion_matrix)
accuracy = bagging_mc.score(X_test, y_test)
print("Accuracy: {:.2f}%".format(accuracy*100))
# @title Find optimal tree size

tree_classifier = DecisionTreeClassifier(random_state=42)

# Define the parameters to search over
params = {'max_leaf_nodes': range(2, 20)}

# Perform grid search with 10-fold cross-validation
grid_search = GridSearchCV(estimator=tree_classifier, param_grid=params, cv=10)
grid_search.fit(X_train, y_train)

# Get the results of cross-validation
cv_results = grid_search.cv_results_

# Find the best tree size
best_size = grid_search.best_params_['max_leaf_nodes']
best_score = grid_search.best_score_

# Plot the cross-validation results with enhanced style
plt.figure(figsize=(6, 4))
plt.plot(cv_results["param_max_leaf_nodes"], cv_results["mean_test_score"], 'o-', color='b',
label='Mean Accuracy')
plt.plot(best_size, best_score, 'ro', label=f'Optimal Tree Size: {best_size}\nBest Accuracy:
{best_score:.2f}')
```

```
plt.xlabel('Tree Size')
plt.ylabel('Accuracy')
plt.title('Cross-validation Results')
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()

print("Optimal Tree Size:", best_size)
print("Best Accuracy:", best_score)
# @title Creating pruned tree using the optimal size
from sklearn.tree import DecisionTreeClassifier, plot_tree

# Create the pruned decision tree classifier
pruned_tree = DecisionTreeClassifier(max_leaf_nodes=best_size, random_state=42)
pruned_tree.fit(X_train, y_train)

# Plot the pruned decision tree
plt.figure(figsize=(12, 8))
plot_tree(pruned_tree, filled=True, feature_names=X_train.columns, class_names=["0", "1"])
plt.title('Pruned Decision Tree')
plt.show()
# Predictions on the test set
y_pred_pruned = pruned_tree.predict(X_test)

# Confusion matrix
conf_matrix_pruned = pd.crosstab(index=y_pred_pruned, columns=y_test,
rownames=['Predicted'], colnames=['Actual'])
print("Confusion Matrix for Pruned Decision Tree:")
print(conf_matrix_pruned)
#Calculate the accuracy of the decision tree on the test data
accuracy = pruned_tree.score(X_test, y_test)
print("Accuracy: {:.2f}%".format(accuracy*100))
# @title gradient boosting
gb_classifier = GradientBoostingClassifier(random_state=1,n_estimators=3500,
max_depth=4)

gb_classifier.fit(X_train, y_train)
y_pred = gb_classifier.predict(X_test)

# Accuracy on the test set
accuracy_gb = accuracy_score(y_test, y_pred)
print("Gradient Boosting Classifier Accuracy:", round(accuracy_gb * 100, 2))

# Confusion matrix
conf_matrix_gb = pd.crosstab(index=y_pred, columns=y_test,  rownames=['Predicted'],
colnames=['Actual'])
print("Confusion Matrix:")
print(conf_matrix_gb)
feature_importances = gb_classifier.feature_importances_

# Create a DataFrame to display feature importances
feature_importance_df = pd.DataFrame({'Feature': X_train.columns, 'Importance':
feature_importances})

# Sort the DataFrame by importance in descending order
feature_importance_df = feature_importance_df.sort_values(by='Importance',
ascending=False)

# Plotting the feature importances
plt.figure(figsize=(10, 6))
plt.barh(feature_importance_df['Feature'], feature_importance_df['Importance'],
color='skyblue', edgecolor='black')
plt.xlabel('Importance', fontsize=12)
plt.ylabel('Feature', fontsize=12)
plt.title('Feature Importances', fontsize=16)
 # Inverting y-axis to show most important features at the top
plt.gca().invert_yaxis()
plt.grid(axis='x', linestyle='--', alpha=0.7)
plt.tight_layout()
# Function to convert specified columns to 'category' dtype
def convert_to_category(dataframe, col_names):

   dataframe[col] = dataframe[col].astype('category')
   return dataframe

#function to convert categorical variable into one hot encoded values
def one_hot_encoding(dataframe, col_name):
   dummy = pd.get_dummies(dataframe[[col_name]])
   res = pd.concat([dataframe, dummy ], axis=1)
   res.drop(col_name,inplace = True,axis = 1)
   return(res)
# @title regression problem
df_reg = df_bc.copy()

df_reg[['marijuana_ever_tried_flag', 'tobacco_ever_tried_flag', 'substance_use_flag']] =
df_imputed[['marijuana_ever_tried_flag', 'tobacco_ever_tried_flag', 'substance_use_flag']]

# Replacing extreme value (e.g., 991) with 0 for clarity
df_reg['age_of_first_alcohol_use'] = df_imputed['age_of_first_alcohol_use'].replace([991], 0)

plt.figure(figsize=(6, 4))
```

```
df_reg['age_of_first_alcohol_use'].hist(grid=False, color='skyblue', bins=20)
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.title('Distribution of Age of First Use of Alcohol')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
df_imputed['age_of_first_alcohol_use'].value_counts()
```

The data shows that some responded age as 1-7 for alcohol use, that doesn't seem right and can affect the model. Let's modify that to 7

```
# replace values 1-6 with 7 in the 'age_first_alcohol_age' column
df_reg['age_of_first_alcohol_use'] = df_reg['age_of_first_alcohol_use'].replace([1, 2, 3, 4, 5,
6], 7)
df_reg['age_of_first_alcohol_use'].value_counts()

# df_reg[['marijuana_ever_tried_flag', 'tobacco_ever_tried_flag']] =
df_imputed[['marijuana_ever_tried_flag', 'tobacco_ever_tried_flag']]

for col in ['marijuana_ever_tried_flag', 'tobacco_ever_tried_flag', 'alcohol_ever_tried_flag']:
    convert_to_category(df_reg,col)
    df_reg= one_hot_encoding(df_reg,col)
# df_imputed[['marijuana_ever_tried_flag', 'tobacco_ever_tried_flag']]
df_reg.columns
X = df_reg[df_reg.columns.difference(['age_of_first_alcohol_use'])]
y = df_reg['age_of_first_alcohol_use']
dt_reg = DecisionTreeRegressor()
dt_reg.fit(X, y)

tree_summary = export_text(dt_reg, feature_names=X.columns.tolist())
importances = pd.DataFrame({'feature_name': X.columns, 'importance':
dt_reg.feature_importances_})
importances = importances.sort_values('importance', ascending=False).reset_index(drop=True)
print(importances)
# plt.figure(figsize=(50,50))
# plot_tree(dt_reg
#          , filled=True
#          , feature_names=X.columns
#          #, class_names=[0, 1]
#          , label='all'
#          , fontsize=12)
# plt.show()
```

Let's try different models

```
# @title Random Forest

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state= 1,
stratify=y)
```

```
rf_classifier = RandomForestRegressor(n_estimators=3500, max_depth=4, random_state=1)

# fit Random Forest model
rf_classifier.fit(X_train, y_train)

# predict on test data
y_pred = rf_classifier.predict(X_test)

# plot predicted vs actual values
plt.figure(figsize=(10, 6))
plt.scatter(y_pred, y_test, alpha=0.5, color='blue', label='Actual vs Predicted')
plt.plot([min(y_pred), max(y_pred)], [min(y_pred), max(y_pred)], 'r--', label='Perfect
Prediction')
plt.xlabel('Predicted Age of First Alcohol Use')
plt.ylabel('Actual Age of First Alcohol Use')
plt.title('Random Forest Regression: Predicted vs Actual')
plt.legend()
plt.grid(True)
plt.show()

# find the MSE
print("Mean Squared Error: {:.2f}".format(mean_squared_error(y_test, y_pred)))
bag_reg = RandomForestRegressor(max_features=X_train.shape[1],random_state = 1)
bag_reg.fit(X_train,y_train)

print("Number of trees:", bag_reg.n_estimators)
print("Number of features tried at each split:",bag_reg.max_features)
print("Training score: {:.2f}%".format(bag_reg.score(X_train,y_train)*100))

# Predict values
y_pred = bag_reg.predict(X_test)

accuracy = bag_reg.score(X_test, y_test)
print("Accuracy: {:.2f}%".format(accuracy*100))


# plot predicted vs actual values
plt.figure(figsize=(10, 6))
plt.scatter(y_pred, y_test, alpha=0.5, color='blue', label='Actual vs Predicted')
plt.plot([min(y_pred), max(y_pred)], [min(y_pred), max(y_pred)], 'r--', label='Perfect
Prediction')
plt.xlabel('Predicted Age of First Alcohol Use')
plt.ylabel('Actual Age of First Alcohol Use')
plt.title('Bagging Regression: Predicted vs Actual')
plt.legend()
plt.grid(True)
plt.show()

# find the MSE
print("Mean Squared Error: {:.2f}".format(mean_squared_error(y_test, y_pred)))
# @title Find optimal tree size

tree_classifier = DecisionTreeRegressor(random_state= 1)

# Define the parameters to search over
params = {'max_leaf_nodes': range(2, 20)}

# Perform grid search with 10-fold cross-validation
grid_search = GridSearchCV(estimator=tree_classifier, param_grid=params, cv=10)
grid_search.fit(X_train, y_train)

# Get the results of cross-validation
cv_results = grid_search.cv_results_

# Find the best tree size
best_size = grid_search.best_params_['max_leaf_nodes']
best_score = grid_search.best_score_

# Plot the cross-validation results with enhanced style
plt.figure(figsize=(6, 4))
plt.plot(cv_results["param_max_leaf_nodes"], cv_results["mean_test_score"], 'o-', color='b',
label='Mean Accuracy')
plt.plot(best_size, best_score, 'ro', label=f'Optimal Tree Size: {best_size}\nBest Accuracy:
{best_score:.2f}')
plt.xlabel('Tree Size')
plt.ylabel('Accuracy')
plt.title('Cross-validation Results')
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()

print("Optimal Tree Size:", best_size)
print("Best Accuracy:", best_score)
# @title Creating pruned tree using the optimal size
from sklearn.tree import plot_tree

# Create the pruned decision tree classifier
pruned_tree = DecisionTreeRegressor(max_leaf_nodes=best_size, random_state=42)
pruned_tree.fit(X_train, y_train)
y_pred = pruned_tree.predict(X_test)
# Plot the pruned decision tree/
```

```
plt.figure(figsize=(12, 8))
plot_tree(pruned_tree, filled=True, feature_names=X_train.columns, class_names=["0", "1"])
plt.title('Pruned Decision Tree')
plt.show()
accuracy = pruned_tree.score(X_test, y_test)
print("Accuracy: {:.2f}%".format(accuracy*100))

# @title gradient boosting

# fit Gradient Boosting model
boost_reg = GradientBoostingRegressor(n_estimators=5000, max_depth=4
                              , random_state=1)
boost_reg.fit(X_train, y_train)

# create a dataframe of feature importances and their corresponding column names
importances_boost_reg = pd.DataFrame({'Feature': X_train.columns, 'Importance':
boost_reg.feature_importances_})
importances_boost_reg = importances_boost_reg.sort_values('Importance',
ascending=False).reset_index(drop=True)

# display the table
print(importances_boost_reg)
# Predict values
y_pred = boost_reg.predict(X_test)
y_pred = np.clip(y_pred, 0, 365)
accuracy = boost_reg.score(X_test, y_test)
print("Accuracy: {:.2f}%".format(accuracy*100))


# plot predicted vs actual values
plt.figure(figsize=(10, 6))
plt.scatter(y_pred, y_test, alpha=0.5, color='blue', label='Actual vs Predicted')
plt.plot([min(y_pred), max(y_pred)], [min(y_pred), max(y_pred)], 'r--', label='Perfect
Prediction')
plt.xlabel('Predicted Age of First Alcohol Use')
plt.ylabel('Actual Age of First Alcohol Use')
plt.title('Bagging Regression: Predicted vs Actual')
plt.legend()
plt.grid(True)
plt.show()

# find the MSE
print("Mean Squared Error: {:.2f}".format(mean_squared_error(y_test, y_pred)))
```

the model seems to be a good fit for the data, as evidenced by the clustering of points near the perfect prediction line and a relatively low MSE.