Predicting Use of Harmful Substances Based on Parental Involvement, Religious Beliefs, and Education

Abstract:

It is fair to assume that the parental involvement, education, and religion beliefs affect the outcome on how child will perceive the use of harmful substance later in their life. This research, based on the National Survey on Drug Use and Health (NSDUH), aims to predict alcohol consumption among youth using behavioral factors and variables such as religious beliefs, parent-child relationship, academic factors, and participation in self-esteem and problem-solving groups were analyzed. The study highlights the importance of family and community support in reducing harmful behaviors among youth. Decision tree models were employed in this study to anticipate the behavior of young people in terms of consuming alcohol later in life. Additionally, the research expands to predicting the age at which young people will have their first encounter with alcohol.

Introduction:

Alcohol consumption among youth is a significant public health concern with potentially harmful consequences for both individuals and society. In this study, we aim to predict alcohol consumption among youth using data from the National Survey on Drug Use and Health (NSDUH). The NSDUH is an annual survey that collects data on substance use and related behaviors among individuals aged 12 years and older in the United States. Our attention is directed towards various factors such as religious convictions, the dynamic between parents and their children, academic performance, and engagement in groups aimed at improving self-esteem and problem-solving abilities. By analyzing these factors, we hope to identify potential risk and protective factors that could inform prevention and intervention efforts to reduce alcohol consumption among youth.

Theoretical Background:

Tree-based models use an algorithmic approach to split a data set based on certain conditions, making them suitable for regression and classification problems. The predictor variables are recursively divided into subsets to create the tree, with more significant variables at the top and insignificant ones discarded. A decision tree has two types of nodes: decision nodes, and leaf nodes. Decision nodes make choices and have multiple branches, while leaf nodes have no more branches and represent the output. The advantages of using tree-based models are easy interpretability, variable importance identification, and fast predictions. When decision trees are allowed to grow too large, they may become overfitted, resulting in less accurate predictions on new data. As a solution, pruned decision trees were developed to prevent overfitting.

A pruned decision tree is a decision tree that has undergone a process of reducing its size by removing branches that have little or no impact on the outcome variable, thereby enhancing its generalization and predictive performance. The basic idea behind pruning is to build a complete decision tree and then remove the branches that do not improve the model's performance on the test data. Pruning can be performed in various ways, including cost-complexity pruning, reduced error pruning, and minimum description length pruning. In summary, pruned decision trees are a powerful and widely used technique in machine learning that helps to improve the predictive performance of decision trees, thereby enhancing the model's accuracy and generalization.

While pruned decision trees are effective in improving the predictive performance of decision trees by reducing overfitting, ensemble methods are a popular machine learning technique that involve combining multiple models to produce a more accurate prediction than any individual model could achieve on its own. Two common types of ensemble methods are Bagging and Boosting.

Bagging, short for bootstrap aggregating, involves fitting multiple models on different subsets of a training dataset. The predictions from these models are then combined to produce a final prediction. The idea behind bagging is that by using different subsets of the training data to fit each model, the ensemble is able to capture different patterns in the data, leading to a more robust and accurate prediction.

Random Forest is an extension of bagging that is particularly effective for tree-based models. In a Random Forest, each individual tree randomly selects a subset of the predictors to split each node. By using different predictors for each tree, the ensemble is able to capture different aspects of the data, leading to a more accurate prediction.

Boosting is another ensemble method that involves fitting multiple models sequentially to correct errors made by previous models. The final prediction is a weighted sum of the individual model predictions, with more weight given to models that perform better on the training data. Boosting is particularly effective for improving the accuracy of weak models, such as decision trees.

In summary, ensemble methods like Bagging and Boosting are powerful techniques for improving the accuracy of machine learning models. Bagging improves the performance of models by fitting multiple models on different subsets of the training data and combining the predictions. Random Forest is a specific type of bagging that is particularly effective for tree-based models. Boosting improves the performance of weak models by fitting multiple models sequentially and weighting their predictions.

Methodology:

The aim of this research was to predict the use of alcohol and other harmful substances and to determine the age at which individuals are likely to start using them. The methodology involved a thorough analysis of the NSDUH dataset, with a focus on removing irrelevant variables and null values. To identify the correlation between alcohol consumption and other substances, variables with a strong correlation to alcohol consumption were removed.

To better understand the impact of missing data on the dataset, we analyzed the pattern of missing data, as shown in Figure 1. We then used MICE to replace the null values in the dataset, ensuring that our analysis was based on complete data. Our methodology aimed to provide an accurate prediction of alcohol consumption and other substance use, considering all relevant factors and minimizing the impact of missing data on our analysis.

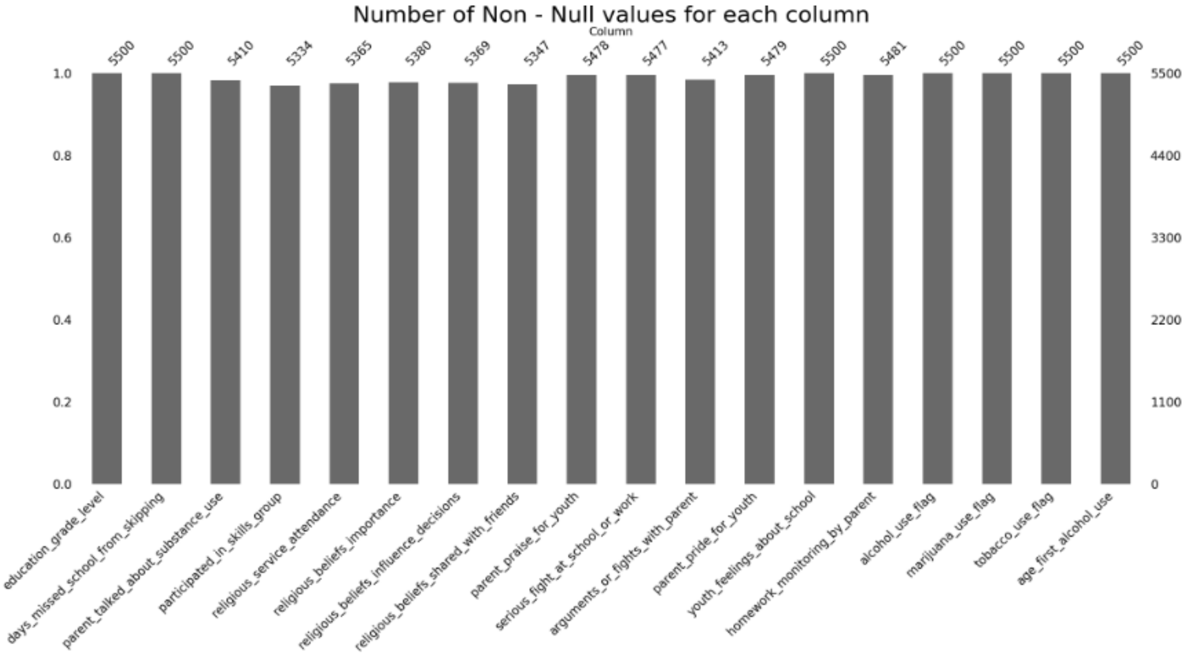

Figure 1: Number of non-null values for each column out of 5500 records.

To accurately predict alcohol consumption, we employed decision tree models for classification tasks using a range of demographic and behavioral factors such as age, gender, race, education level, and family income as predictors. To evaluate the performance, we utilized cross-validation techniques. Later, we extended our model to multiclass classification to identify the usage of alcohol, marijuana, or tobacco. The multiclass classification model combined the flags for alcohol, tobacco, and marijuana use to create a single variable called substance use flag, which allowed us to analyze their impact on substance use habits.

To broaden the scope of our research, we aimed to predict whether an individual will consume alcohol and the age at which they will begin doing so. To achieve this objective, we developed a regression model. Initially, we examined the data to identify any outliers, and we found that some individuals reported consuming alcohol before the age of 7, which is highly improbable. We resolved this issue by replacing such values with 7 as the age indicator when a person first tried drinking alcohol. Furthermore, we examined the distribution of the age at which youth reported their first alcohol consumption, which is presented in Figure 2. We observed that many teenagers tend to take their first sip at the age of 15, and there is a significant number of individuals who have not yet consumed alcohol. To test whether our regression model can accurately forecast this behavior, we will evaluate its performance. In summary, our methodology involves identifying outliers and addressing them, analyzing the distribution of age at which youth consume alcohol, testing our regression model's accuracy, and evaluating its performance.

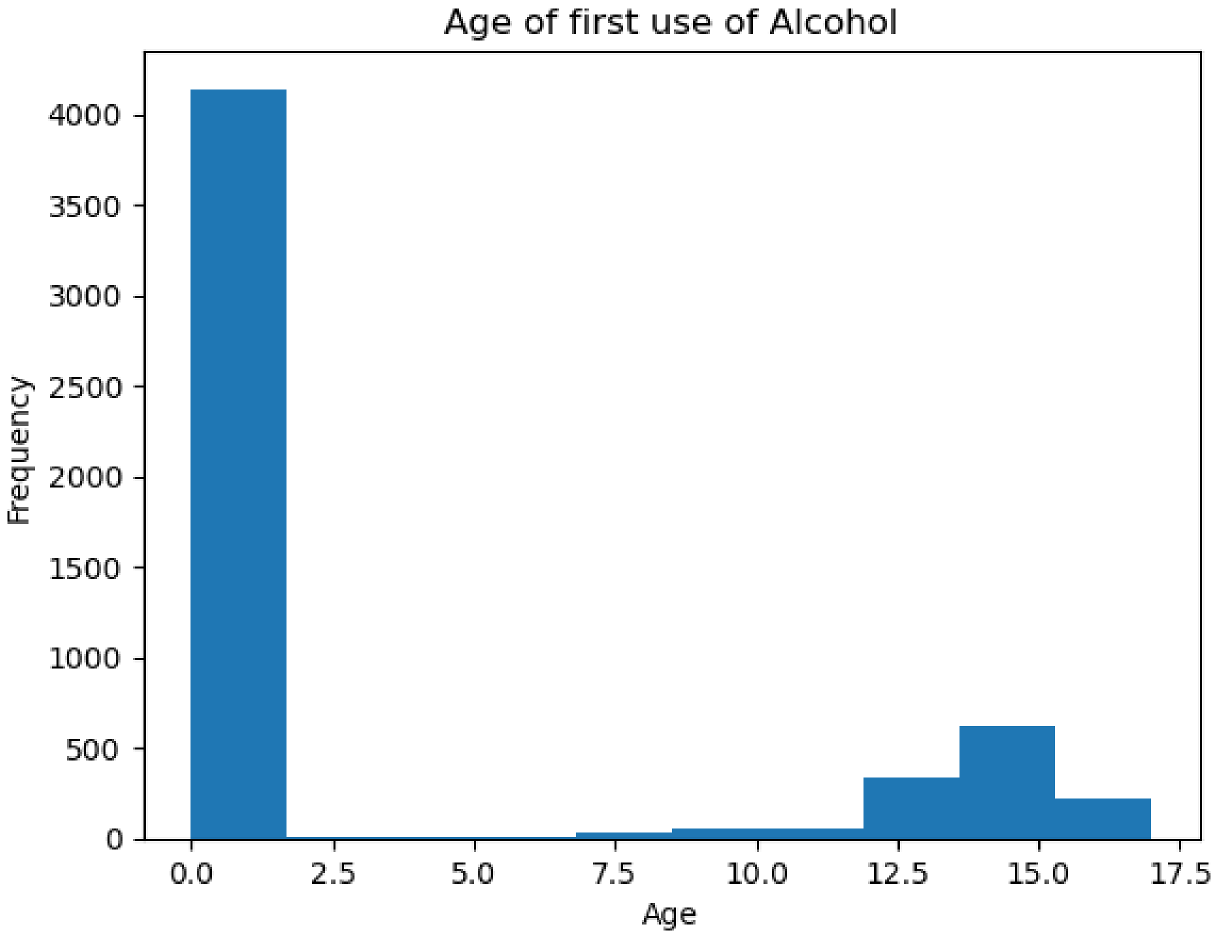

Figure 2: Frequency at what age people try their first drink

By using this methodology, we were able to accurately predict alcohol usage among youth and gain insights into the role of demographic and behavioral factors in predicting substance use. Our research findings have important implications for the development of interventions and policies aimed at preventing youth substance use.

Computational Results:

In a binary classification problem where the task was to predict whether youth would consume alcohol or not, the dataset was found to be imbalanced, with 4138 out of 5550 adult datapoints not consuming alcohol. To address this, we used stratified sampling and considered class imbalance while training various decision tree and ensemble models. Upon training the decision tree model, we discovered that the accuracy of the binary classification was highly dependent on the youth's feelings about attending school, emphasizing the importance of education in predicting alcohol consumption. Moreover, we observed that parental communication about substance use, and the child's accomplishments also had an impact on the decision. Interestingly, the number of school days missed had a relatively lesser impact on the prediction compared to other variables in the dataset. Figure 1 highlights the crucial features for predicting whether a person will consume alcohol or not.

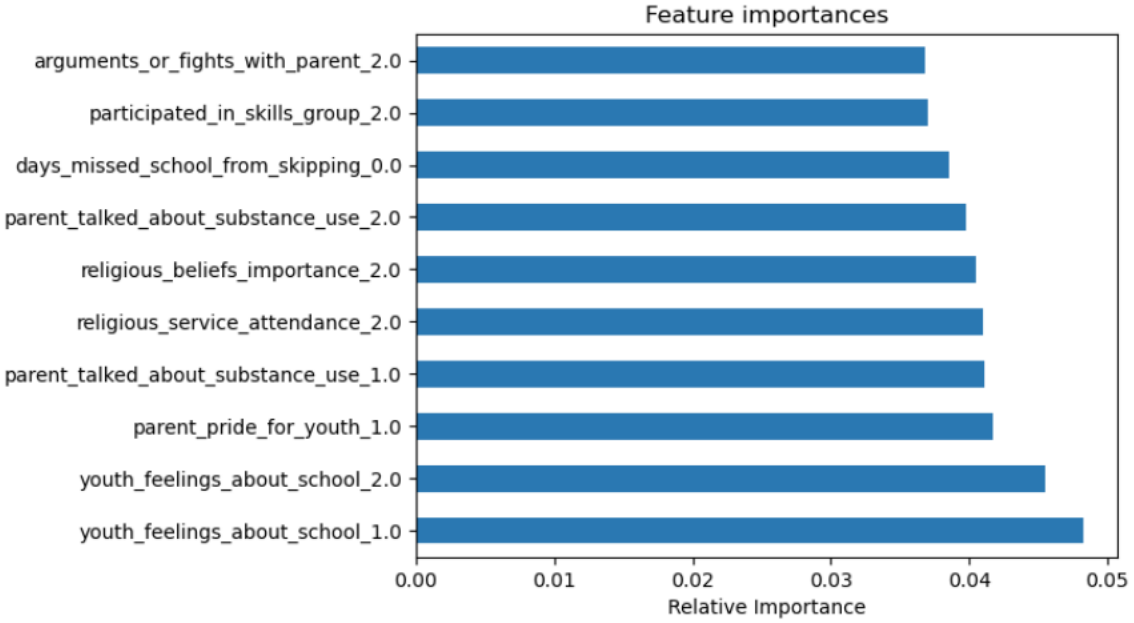

Figure 3: Importance of features in classifying whether a person will consume alcohol or not.

We got the same accuracy with pruned tree as a normal decision tree with just one single node. It's worth noting that we achieved a 75.2% accuracy on the test results when predicting whether a person will drink or not. Additionally, it's intriguing to observe that we're encountering the same level of error as the unpruned tree.

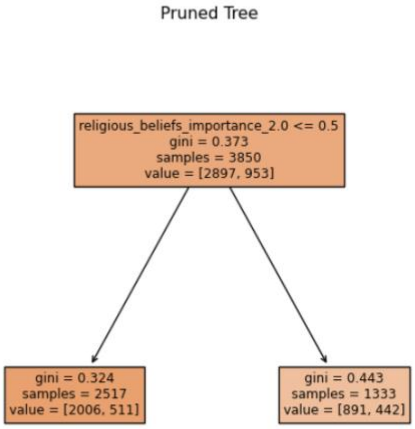

Figure 4.a: Pruned tree with accuracy 75.2

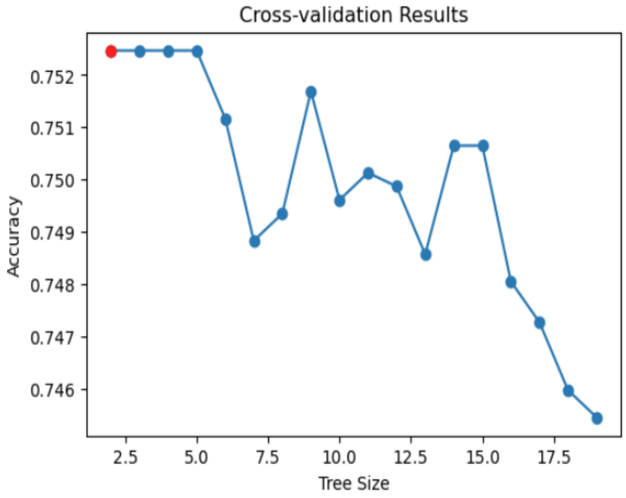

Figure 4.b: Cross-Validation accuracy VS tree size

Multiclass classification: In the context of multiclass classification, our goal is to determine whether a person will use any form of substance. To achieve this, we have categorized the output into four distinct classes, as previously mentioned.

Multi-class Classification

Substance Use	Output Value
Alcohol	1
Marijuana	2
Tobacco	3
No Substance Use	0

Table 1: Multiclass classification output

After attempting several methods, we discovered that the bagging model was the most effective for this dataset. However, if we had additional information about youth behavior, we could further enhance the accuracy of the model. This can be observed in the confusion matrix, which illustrates the model's performance. It is worth noting that the model performs well in predicting instances where there is no substance use. The model is currently achieving an accuracy rate of 65%.

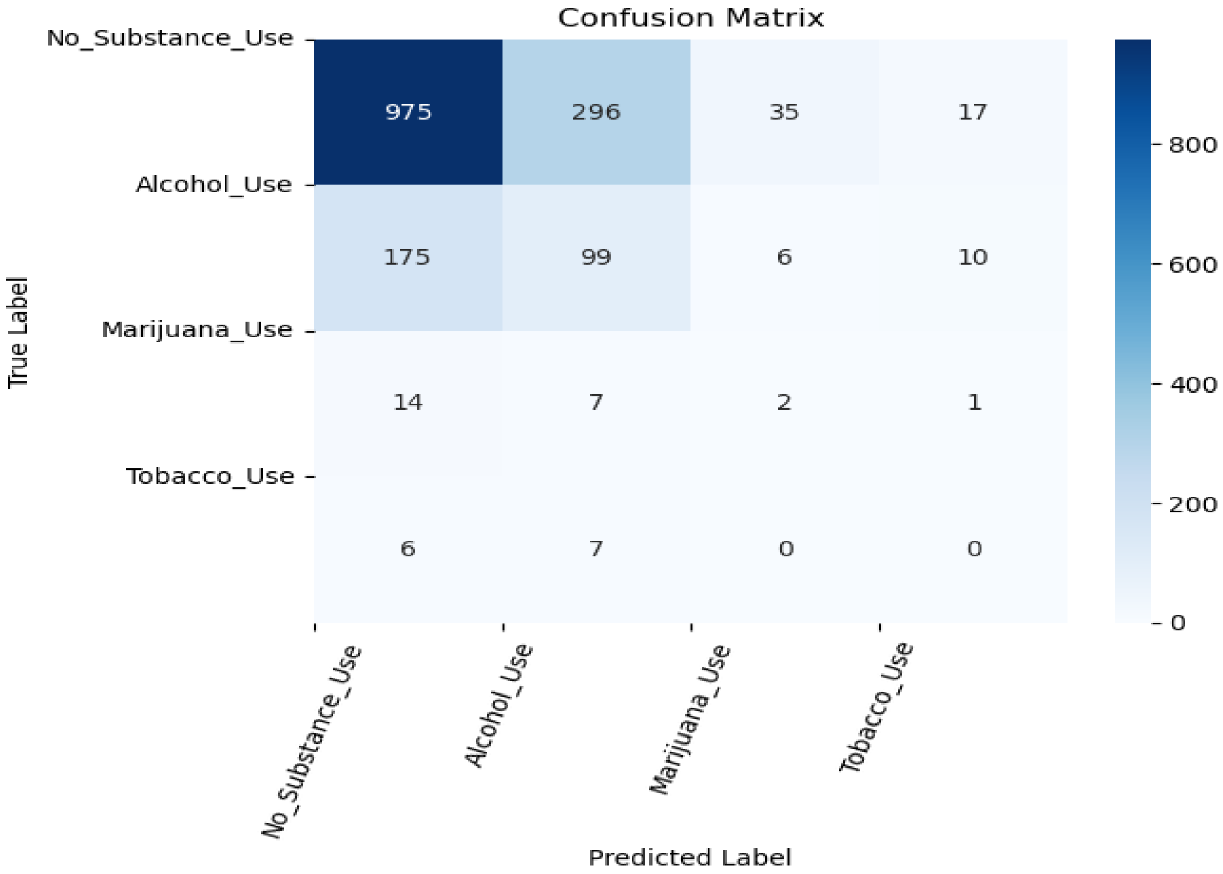

Figure 5: Confustion Matrix: Random Forest Classifier

Regression model: In the context of a regression problem, we were trying to predict the age at which young people will try alcohol for the first time. To accomplish this, we used various models to predict the output. One of the methods we used was pruned decision tree to do regression analysis, which allowed us to predict the age of the first drink with an accuracy of approximately 96.84%, considering the limitations of the data we had.

During our analysis, we found that the "alcohol use" flag in the pruned tree was a significant predictor of the age at which a person started drinking. This means that whether a person has used alcohol or not can have a significant impact on the age at which they try it for the first time. Additionally, we found that the occurrence of serious fights at school or work was the second most important predictor for determining the age of first drink. This suggests that such incidents can have a significant impact on a person's behavior and mental health, potentially leading to long-term health concerns like depression.

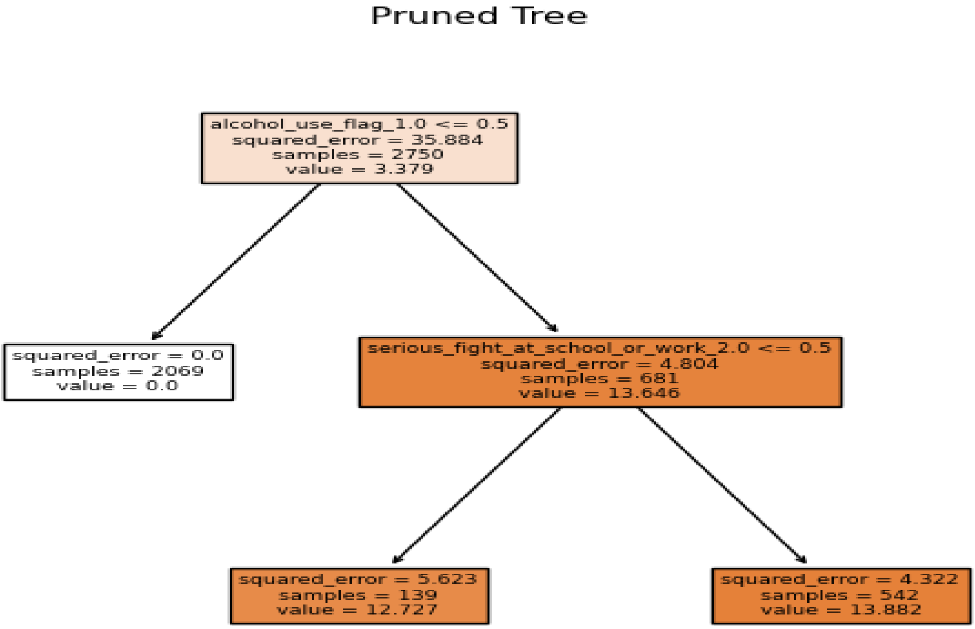

Figure 6: Regression Decision Tree

To visualize our findings, we created a decision tree, which is a tool used to predict outcomes based on different conditions. The tree starts with a condition about whether or not a person has used alcohol. If they have used it very little or not at all, the tree goes to the left and predicts a certain outcome. If they have used alcohol more than a little, the tree goes to the right and looks at whether or not they have been involved in a serious fight at school or work. Based on the different conditions, the tree predicts different outcomes, represented by numerical values.

Our findings were based on data from a sample of 2750 people, and the accuracy of the predictions varied depending on the conditions that applied to each group of people. Overall, our analysis sheds light on important factors that can impact the age at which young people try alcohol for the first time and highlights the importance of considering these factors in prevention and intervention strategies.

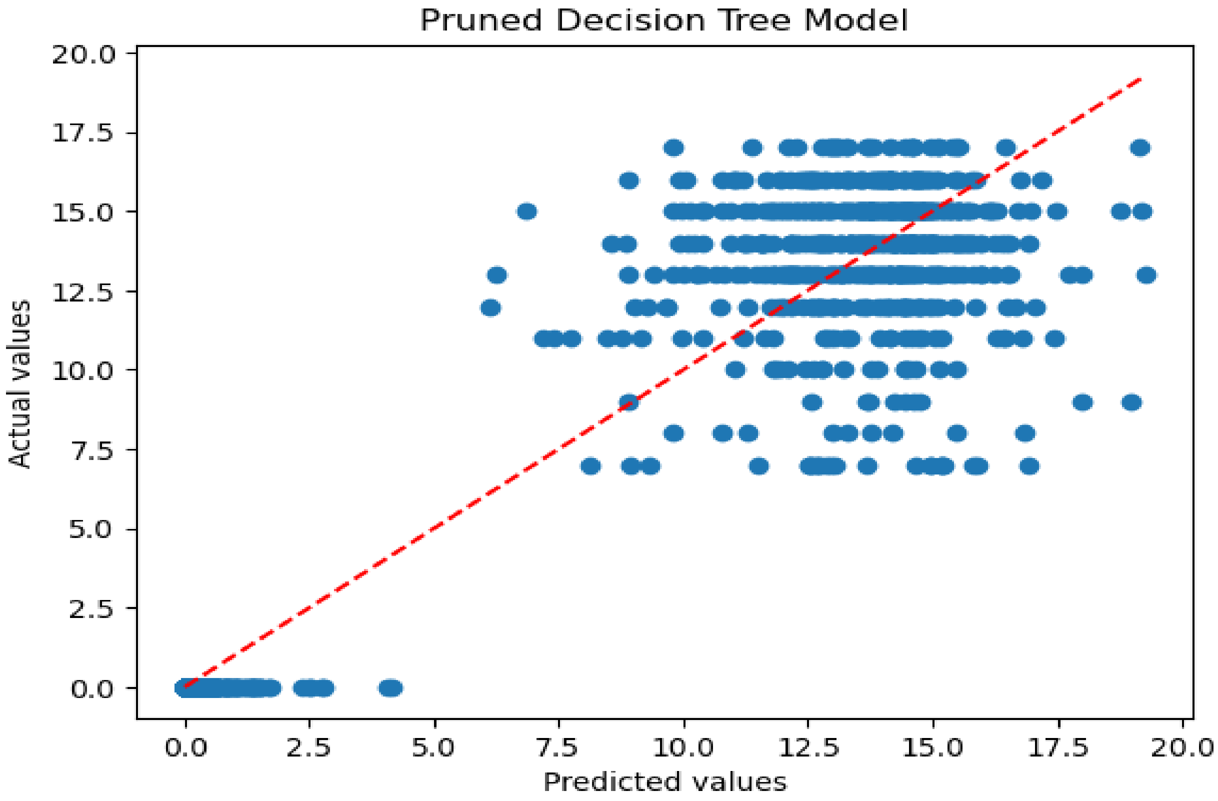

Figure 7: Predicted age value VS Actual Age value

Looking at the pruned decision tree output model graph, we can observe that the predicted values are generally close to the actual values, indicating that our regression model performed well in predicting the age at which a person will try alcohol for the first time, given the limitations of the data.

Discussion:

It is worth noting that the NSDUH dataset includes a vast array of data that can be used to answer many more questions related to substance use and other topics. Our study's specific focus on religious beliefs and school behavior was just one aspect of the available data. Nonetheless, our findings provide valuable insights into the potential risk and protective factors that can inform prevention and intervention efforts aimed at reducing substance use among youth. Our findings indicate that these factors significantly influence a person's likelihood of consuming substances later in life. Therefore, it is essential to engage students in fostering positive relationships with the people in their lives, including parents and teachers, to reduce the risk of harmful behaviors.

The study aimed to predict alcohol consumption among youth using various behavioral factors and to identify potential risk and protective factors for informing prevention and intervention efforts. The findings showed that parental communication about substance use, academic accomplishments, and the child's feelings about attending school were highly influential in predicting whether a person would consume alcohol or not. The decision tree model achieved an accuracy of 75.2% on test results, indicating its effectiveness in predicting alcohol consumption. Furthermore, the model's accuracy was not compromised by pruning.

Regarding multiclass classification, the study aimed to determine whether a person would use any form of substance, including alcohol, tobacco, and marijuana. The analysis revealed that parental communication, academic performance, and participation in self-esteem and problem-solving groups were significant predictors of substance use.

However, the study had limitations such as the cross-sectional nature of the data, which limits the ability to infer causality, and self-reported data on substance use subject to social desirability bias, which could impact the accuracy of predictions. Additionally, the dataset's sample size was relatively small, which may limit the generalizability of findings to the larger population.

In conclusion, the study provides valuable insights into potential risk and protective factors for reducing alcohol consumption among youth. The findings underscore the need for targeted interventions aimed at reducing substance use among youth and highlight the importance of education and family support in reducing harmful behaviors.

Conclusions:

Our study aimed to predict alcohol consumption using decision tree models and tree-based ensemble models such as Random Forest, Bagging, and Boosting. We utilized data from the NSDUH survey and found that the models were able to achieve impressive accuracies. For the binary classification problem, the decision tree after pruning yielded an accuracy of 75.09%. Furthermore, the Pruned Decision Tree model demonstrated an accuracy of 96%, using less than half the number of predictors.

These findings suggest that decision tree models and ensemble methods are effective tools for predicting if an individual will consume alcohol or not. They can also be utilized to determine the age at which a person is likely to try their first drink or whether a young person will try alcohol. The models' performance is encouraging and highlights their potential in aiding early intervention strategies to prevent alcohol misuse.

Overall, our study contributes to the existing body of literature on the use of decision tree models and ensemble methods in predicting alcohol consumption. These tools offer a promising approach for identifying individuals who are at risk of alcohol misuse and providing appropriate 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/NSDUH-2020/NSDUH-2020-datasets/NSDUH-2020-DS0001/NSDUH-2020-DS0001-info/NSDUH-2020-DS0001-info-codebook.pdf.

Appendix:

```
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, Deci
sionTreeRegressor
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.preprocessing import LabelEncoder
from sklearn.preprocessing import OrdinalEncoder
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
from sklearn.inspection import PartialDependenceDisplay
```

```
df = pd.read_csv('youth_data.csv')
```

```
# WE WANT TO KNOW IF THERE'S IMPACT OF PARENT ECONOMICAL THINGS ON IF YOUTH W
ILL TRY Alcohol OR NOT
```

Technical Requirements You must include one example of each of the following problem types:

- binary classification (e.g. has or has not used cigarettes)
- multi-class classification (e.g. differentiate between seldom, sometimes, and frequent marijuana use)
- regression (e.g. how many days per year a person has used alcohol)

Let’s try to answer the question if person has or has not tried alcohol based on various factors.

To answer this question we need to be certain about what kinds of predictors we want to use for this model

```
df.shape
```

```
(5500, 80)
```

There are 5500 records and 80 variables.

We will first try to do binary classification using this dataset

- Question we are trying to answer here is if person have tried alcohol or not

Let’s find missing values in the dateset

- PRTALK3 RC-TALKED WITH PARENT ABOUT DANGER TOB/ALC/DRG
- PRBSOLV2 RC-PARTICIPATED IN PRBSLV/COMMSKILL/SELFESTEEM GROUP
- rlgattd RC-NUMBER TIMES ATTEND RELIGIOUS SERVICES IN PST YR (1 – 25 or more , 2 – less than 25)
- rlgimpt - RC-RELIGIOUS BELIEFS VERY IMPORTANT IN LIFE
- rlgdcsn - RC-RELIGIOUS BELIEFS INFLUENCE LIFE DECISIONS
- rlgfrnd - RC-IMPORTANT FOR FRIENDS TO SHARE RELIGIOUS BELIEFS
- EDUSCHGRD2 - WHAT GRADE IN NOW/WILL BE IN
- PRGDJOB2 - PARENTS TELL YTH HAD DONE GOOD JOB IN PST YR
- YOFIGHT2 - RC-YOUTH HAD SERIOUS FIGHT AT SCHOOL/WORK
- argupar - RC-TIMES ARGUED/HAD A FIGHT WITH ONE PARENT IN PST YR
- PRPROUD2 - RC-PARENTS TELL YTH PROUD OF THINGS DONE IN PST YR
- schfelt - RC-HOW YTH FELT: ABOUT GOING TO SCHOOL IN PST YR
- parchkhw - RC-PARENTS CHECK IF HOMEWORK DONE IN PST YR
- eduskpcom - HOW MANY DAYS MISSED SCHOOL FROM SKIPPING (COMBINED)

```
df_final = df[['EDUSCHGRD2',  'eduskpcom','PRTALK3','PRBSOLV2',\
                       'rlgattd','rlgimpt','rlgdcsn','rlgfrnd','PRGDJOB2','YO
FIGHT2',\
                       'argupar', 'PRPROUD2','schfelt','parchkhw','alcflag','
mrjflag', 'tobflag','iralcage']].copy()

# create a dictionary to map old variable names to new variable names
new_names = {
    'PRTALK3': 'parent_talked_about_substance_use',
    'PRBSOLV2': 'participated_in_skills_group',
    'rlgattd': 'religious_service_attendance',
    'rlgimpt': 'religious_beliefs_importance',
    'rlgdcsn': 'religious_beliefs_influence_decisions',
    'rlgfrnd': 'religious_beliefs_shared_with_friends',
    'EDUSCHGRD2': 'education_grade_level',
    'PRGDJOB2': 'parent_praise_for_youth',
    'YOFIGHT2': 'serious_fight_at_school_or_work',
    'argupar': 'arguments_or_fights_with_parent',
    'PRPROUD2': 'parent_pride_for_youth',
    'schfelt': 'youth_feelings_about_school',
    'parchkhw': 'homework_monitoring_by_parent',
    'eduskpcom': 'days_missed_school_from_skipping',
    'alcflag': 'alcohol_use_flag',
    'mrjflag': 'marijuana_use_flag',
    'tobflag': 'tobacco_use_flag',
    'iralcfy': 'alcohol_frequency_past_year',
    'iralcage' : 'age_first_alcohol_use'
}

# rename the columns in the dataframe
df_final = df_final.rename(columns=new_names)
```

Education grade level was not categorical and there were values which were indicating if person have taken any education or not. We have replaced it in a way that it is binary indicator

If person have taken any education or not.

```
# use the replace function to create the new variable
df_final['education_grade_level'] = df_final['education_grade_level'].replace
(to_replace=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11], value=1)
df_final['education_grade_level'] = df_final['education_grade_level'].replace
(to_replace=[99], value=0)

# print the resulting dataframe
df_final['education_grade_level'].value_counts()
```

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

```
msno.bar(df_final)
plt.title('Number of Non - Null values for each column',fontsize = 30)
plt.ylabel('Count of non-NAN values', fontsize = 15)
plt.xlabel('Column', fontsize = 15);
```

png

```
msno.matrix(df_final, labels = True)
```

```
<AxesSubplot:>
```

png

We can see that there are missing values for certain columns and by seeing the graph it’s interesting to note that the data is not missing at random. There are certain records which have same field missing. We can see how we want to impute that data.

We can see that the following number of recrods are missing for each of the variable.

```
msno.heatmap(df_final, fontsize = 12);
```

png

We can also see that there’s correlation between the missing values using the missing value heat map. To replace the missing values in the dataset we are using MICE libary

```
df_final.isna().sum()[df_final.isna().sum()>0].sort_values(ascending = False)
.plot(kind = 'bar', figsize = (20,6), color = 'k', alpha = 0.6)
```

```
<AxesSubplot:>
```

png

Imputated Values

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

```
df_impute = df_final.copy()
```

```
df_impute = pd.DataFrame(imputer.fit_transform(df_impute), columns = df_input
e.columns)
```

```
df_impute.notna().sum().plot(kind = 'bar',figsize = (18,4),alpha = 0.6)
plt.xlabel('Column', fontsize = 15)
plt.ylabel('Count of non-NAN values', fontsize = 15)
plt.tick_params(labelsize = 13)
```

png

We can see that there are 0 null values in the dataset now

Does religion or parental behavior , religious belief , education and youth behavior in society impacts drinking alcohol?

Define functions

```
def find_unique_count(frame , column , response):
    print('Number of nan values in the dataset: ', frame[column].isna().sum()
)
    print('Number of unique values: ' , frame[column].nunique())
    print('Count of each value:\n' ,frame[column].value_counts())
    sns.catplot(data = frame , x = response , hue = column, kind = 'count',pa
lette="pastel", edgecolor=".6")
```

```
def convert_to_category(df,col):
    df[col] = df[col].astype('category')
```

Binary Classification

```
df_binary = df_impute[['education_grade_level', 'days_missed_school_from_skip
ping',
        'parent_talked_about_substance_use', 'participated_in_skills_group',
        'religious_service_attendance', 'religious_beliefs_importance',
        'religious_beliefs_influence_decisions',
        'religious_beliefs_shared_with_friends', 'parent_praise_for_youth',
        'serious_fight_at_school_or_work', 'arguments_or_fights_with_parent',
        'parent_pride_for_youth', 'youth_feelings_about_school',
        'homework_monitoring_by_parent','alcohol_use_flag']].copy()
```

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

```
for col in df_binary.columns[:-1]:
    convert_to_category(df_binary,col)
    df_binary = onehotencoding(df_binary,col)
```

```
# Split the dataset into X and y variable
X = df_binary[df_binary.columns.difference(['alcohol_use_flag'])]
y = df_binary['alcohol_use_flag']
```

```
# fit decision tree model
tree_mar = DecisionTreeClassifier()
tree_mar.fit(X, y)
```

```
DecisionTreeClassifier()
```

```
tree_summary = export_text(tree_mar, feature_names=X.columns.tolist())
```

```
importances = pd.DataFrame({'feature_name': X.columns, 'importance': tree_mar
.feature_importances_})
importances = importances.sort_values('importance', ascending=False).reset_in
dex(drop=True)
print(importances)
```

```
                             feature_name  importance
0   parent_talked_about_substance_use_1.0    0.053478
1         youth_feelings_about_school_2.0    0.048302
2    days_missed_school_from_skipping_0.0    0.042280
3        religious_beliefs_importance_2.0    0.040506
4         youth_feelings_about_school_1.0    0.040148
..                                    ...         ...
60  days_missed_school_from_skipping_22.0    0.000000
61             parent_praise_for_youth_0.0    0.000000
62  days_missed_school_from_skipping_25.0    0.000000
63  days_missed_school_from_skipping_16.0    0.000000
64  days_missed_school_from_skipping_13.0    0.000000

[65 rows x 2 columns]
```

```
importances[:10]['feature_name']
```

```
0    parent_talked_about_substance_use_1.0
1          youth_feelings_about_school_2.0
2     days_missed_school_from_skipping_0.0
3         religious_beliefs_importance_2.0
4          youth_feelings_about_school_1.0
5         participated_in_skills_group_2.0
```

```
6          religious_service_attendance_2.0
7       arguments_or_fights_with_parent_2.0
8                parent_pride_for_youth_1.0
9               parent_praise_for_youth_1.0
Name: feature_name, dtype: object
```

```
importances[:10].plot(kind='barh')
plt.yticks(range(10),[feature for feature in importances[:10]['feature_name']
])
plt.xlabel('Relative Importance')
plt.legend('',frameon=False)
plt.title("Feature importances")
plt.show()
```

png

```
plt.figure(figsize=(50,50))
plot_tree(tree_mar
          , filled=True
          , feature_names=X.columns
          #, class_names=[0, 1]
          , label='all'
          , fontsize=12)
plt.show()
```

png

The above output looks too messy to visualize. Let’s see what output we are getting after running the model for the training dataset

Let’s try different models to predict the alcohol usage based on various parameters

Simple Decision Classifier

since the data is imbalanced, we will do stratify sampling and see how it performs

```
# 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 = 2,strati
fy=y)
# fit decision tree model
tree_mar_flag = DecisionTreeClassifier()
tree_mar_flag.fit(X_train, y_train)

# predict on test data
tree_pred = tree_mar_flag.predict(X_test)

# create confusion matrix
```

```
confusion_matrix = pd.crosstab(index=tree_pred, columns=y_test, rownames=['']
)
print(confusion_matrix)
```

```
alcohol_use_flag   0.0  1.0

0.0               1098  317
1.0                143   92
```

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

```
Accuracy: 72.12%
```

We can see that the model is doing well on class 0 but it is unable to predict the output accurately for the class 1.

Let's try Random Forest Classifier and see how it works in the training and test dataset

```
# Split the Data between test and train
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, rand
om_state=2, stratify=y)


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

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

# predict on test data
rf_pred = rf_model.predict(X_test)

# create confusion matrix
confusion_matrix = pd.crosstab(index=rf_pred, columns=y_test, rownames=[''])
print(confusion_matrix)
accuracy = rf_model.score(X_test, y_test)
print("Accuracy: {:.2f}%".format(accuracy*100))
```

```
alcohol_use_flag   0.0  1.0

0.0               1124  321
1.0                117   88
Accuracy: 73.45%
```

There's atleast improvement in predicting the output for class 0 .

Let's try bagging approach

```python
# fit Random Forests model
alcohol_bagging = RandomForestClassifier(max_features=X_train.shape[1],random
_state = 2)
alcohol_bagging.fit(X_train,y_train)

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

```
Number of trees: 100
Number of features tried at each split: 65
Training score: 87.30%
```

```python
# Predict values
y_pred_bag = alcohol_bagging.predict(X_test)

confusion_matrix = pd.crosstab(index=y_pred_bag, columns=y_test, rownames=[''
])
print(confusion_matrix)
```

```
alcohol_use_flag   0.0  1.0

0.0               1099  308
1.0                142  101
```

```python
# Calculate the accuracy of the decision tree on the test data
accuracy = alcohol_bagging.score(X_test, y_test)
print("Accuracy: {:.2f}%".format(accuracy*100))
```

```
Accuracy: 72.73%
```

```python
# fit decision tree model
tree_mar = DecisionTreeClassifier(random_state=7)
tree_mar.fit(X_train, y_train)

# cross-validation to determine optimal tree size
params = {'max_leaf_nodes': range(2, 20)}
cv_mar = GridSearchCV(tree_mar, params, cv=10)
cv_mar.fit(X_train, y_train)
cv_results = cv_mar.cv_results_

# find the best score for max leaf nodes
best_size = cv_mar.best_params_['max_leaf_nodes']
best_score = cv_mar.best_score_

# plot results of cross-validation
plt.figure(figsize=(6, 4))
plt.plot(cv_results["param_max_leaf_nodes"], cv_results["mean_test_score"], '
o-')
plt.plot(best_size, best_score, 'ro-')
plt.xlabel('Tree Size')
plt.ylabel('Accuracy')
plt.title('Cross-validation Results');
```

png

```
# prune tree using optimal size
prune_mar = DecisionTreeClassifier(max_leaf_nodes=best_size, random_state=7)
prune_mar.fit(X_train, y_train)

# plot pruned tree
plt.figure(figsize=(7,6))
plt.title('Pruned Tree')
plot_tree(prune_mar, feature_names=X_train.columns, filled=True);
plt.title('Pruned Tree');
```

png

```
# obtain predicted labels for test set
y_pred = prune_mar.predict(X_test)

# create confusion matrix
confusion_matrix = pd.crosstab(index=y_pred, columns=y_test, rownames=[''])
print(confusion_matrix)
```

```
alcohol_use_flag   0.0  1.0

0.0               1241  409
```

```
# Calculate the accuracy of the decision tree on the test data
accuracy = prune_mar.score(X_test, y_test)
print("Accuracy: {:.2f}%".format(accuracy*100))
```

```
Accuracy: 75.21%
```

```
# fit Gradient Boosting model
boost_alcohol = GradientBoostingClassifier(n_estimators=5000, max_depth=4
                                          , random_state=1)
boost_alcohol.fit(X_train, y_train)
```

```
GradientBoostingClassifier(max_depth=4, n_estimators=5000, random_state=1)
```

```
# create a dataframe of feature importances and their corresponding column na
mes
importances_boost_alcohol = pd.DataFrame({'Feature': X_train.columns, 'Import
ance': boost_alcohol.feature_importances_})
importances_boost_alcohol = importances_boost_alcohol.sort_values('Importance
', ascending=False).reset_index(drop=True)

# display the table
print(importances_boost_alcohol)
```

```
                                         Feature  Importance
0          arguments_or_fights_with_parent_2.0    0.055848
1             religious_beliefs_importance_2.0    0.053520
2              homework_monitoring_by_parent_2.0    0.049237
3    religious_beliefs_influence_decisions_2.0    0.044102
4          serious_fight_at_school_or_work_2.0    0.039174
..                                         ...         ...
60       days_missed_school_from_skipping_11.0    0.000946
61       days_missed_school_from_skipping_22.0    0.000590
62       days_missed_school_from_skipping_13.0    0.000577
63       days_missed_school_from_skipping_14.0    0.000477
64       days_missed_school_from_skipping_16.0    0.000111

[65 rows x 2 columns]
```

```
PartialDependenceDisplay.from_estimator(boost_alcohol, X_train, ['arguments_o
r_fights_with_parent_2.0', 'religious_beliefs_importance_2.0']);
```

png

```
# obtain predicted labels for test set
y_pred = boost_alcohol.predict(X_test)

# create confusion matrix
confusion_matrix = pd.crosstab(index=y_pred, columns=y_test, rownames=[''])
print(confusion_matrix)
```

```
alcohol_use_flag   0.0  1.0

0.0               1073  295
1.0                168  114
```

Multiclass classification

For multiclass classification we will merge the colum output of alcohol flag,tobacco flag and marijuana flag

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

# create a list of the columns to use
columns = ['alcohol_use_flag', 'marijuana_use_flag', 'tobacco_use_flag']

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

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

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

```
df_multi = df_impute[['education_grade_level', 'days_missed_school_from_skipp
ing',
       'parent_talked_about_substance_use', 'participated_in_skills_group',
       'religious_service_attendance', 'religious_beliefs_importance',
       'religious_beliefs_influence_decisions',
       'religious_beliefs_shared_with_friends', 'parent_praise_for_youth',
       'serious_fight_at_school_or_work', 'arguments_or_fights_with_parent',
       'parent_pride_for_youth', 'youth_feelings_about_school',
       'homework_monitoring_by_parent','substance_use_flag']].copy()
```

```
def convert_to_category(df,col):
    df[col] = df[col].astype('category')
```

```
for col in df_multi.columns[:-1]:
    convert_to_category(df_multi,col)
    df_multi = onehotencoding(df_multi,col)
```

```
X = df_multi[df_multi.columns.difference(['substance_use_flag'])]
y = df_multi['substance_use_flag']
```

```
# fit decision tree model
tree_mar = DecisionTreeClassifier()
tree_mar.fit(X, y)
```

```
DecisionTreeClassifier()
```

```
tree_summary = export_text(tree_mar, feature_names=X.columns.tolist())
```

```
importances = pd.DataFrame({'feature_name': X.columns, 'importance': tree_mar
.feature_importances_})
importances = importances.sort_values('importance', ascending=False).reset_in
dex(drop=True)
print(importances)
```

```
                             feature_name  importance
0   parent_talked_about_substance_use_2.0    0.045986
1         youth_feelings_about_school_2.0    0.043591
2     arguments_or_fights_with_parent_2.0    0.042422
3    days_missed_school_from_skipping_0.0    0.041216
4        religious_beliefs_importance_2.0    0.039368
..                                    ...         ...
60  days_missed_school_from_skipping_22.0    0.000600
61  days_missed_school_from_skipping_20.0    0.000516
62  days_missed_school_from_skipping_12.0    0.000293
```

```
63  days_missed_school_from_skipping_16.0    0.000000
64  days_missed_school_from_skipping_13.0    0.000000

[65 rows x 2 columns]
```

```
plt.figure(figsize=(50,50))
plot_tree(tree_mar
          , filled=True
          , feature_names=X.columns
          #, class_names=[0, 1]
          , label='all'
          , fontsize=12)
plt.show()
```

png

The above output looks too messy to visualize. Let's see what output we are getting after running the model for the training dataset

Let's try different models to predict the alcohol usage based on various parameters

Simple Decision Classifier

since the data is imbalanced, we will do stratify sampling and see how it performs

```
output = ['No_Substance_Use','Alcohol_Use','Marijuana_Use','Tobacco_Use']

# 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 = 2,strati
fy=y)
# fit decision tree model
tree_mar_flag = DecisionTreeClassifier()
tree_mar_flag.fit(X_train, y_train)

# predict on test data
tree_pred = tree_mar_flag.predict(X_test)

# create confusion matrix
confusion_matrix = pd.crosstab(index=tree_pred, columns=y_test, rownames=['']
)
# create a heatmap of the confusion matrix
sns.heatmap(confusion_matrix, annot=True, cmap="Blues", fmt="d")

# add labels to the plot
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.yticks(range(4),[out for out in output],rotation = 0)
plt.xticks(range(4),[out for out in output],rotation = 70)
plt.show()
```

png

We can see that the model is doing well on class 0 but it is unable to predict the output accurately for the class 1.

Let’s try Random Forest Classifier and see how it works in the training and test dataset

```
# Split the Data between test and train
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, rand
om_state=2, stratify=y)


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

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

# predict on test data
rf_pred = rf_model.predict(X_test)

# create confusion matrix
confusion_matrix = pd.crosstab(index=rf_pred, columns=y_test, rownames=[''])
print(confusion_matrix)
```

```
substance_use_flag     0    1   2   3

0                   1028  309  38  20
1                    135   92   5   8
2                      5    5   0   0
3                      2    3   0   0
```

There’s atleast improvement in predicting the output for class 0 .

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

```
Accuracy: 67.88%
```

Let’s try bagging approach

```
# fit Random Forests model
multi_bagging = RandomForestClassifier(max_features=X_train.shape[1],random_s
tate = 2)
multi_bagging.fit(X_train,y_train)
```

```
print("Number of trees:", multi_bagging.n_estimators)
print("Number of features tried at each split:",multi_bagging.max_features)
print("Training score: {:.2f}%".format(multi_bagging.score(X_train,y_train)*1
00))
```

```
Number of trees: 100
Number of features tried at each split: 65
Training score: 86.21%
```

```
# Predict values
y_pred_bag = multi_bagging.predict(X_test)

confusion_matrix = pd.crosstab(index=y_pred_bag, columns=y_test, rownames=[''
])
print(confusion_matrix)
```

```
substance_use_flag    0    1   2   3

0                   995  309  36  19
1                   165   91   7   9
2                     4    5   0   0
3                     6    4   0   0
```

```
# Calculate the accuracy of the decision tree on the test data
accuracy = multi_bagging.score(X_test, y_test)
print("Accuracy: {:.2f}%".format(accuracy*100))
```

```
Accuracy: 65.82%
```

```
# fit decision tree model
tree_mar = DecisionTreeClassifier(random_state=7)
tree_mar.fit(X_train, y_train)

# cross-validation to determine optimal tree size
params = {'max_leaf_nodes': range(2, 20)}
cv_mar = GridSearchCV(tree_mar, params, cv=10)
cv_mar.fit(X_train, y_train)
cv_results = cv_mar.cv_results_

# find the best score for max leaf nodes
best_size = cv_mar.best_params_['max_leaf_nodes']
best_score = cv_mar.best_score_

# plot results of cross-validation
plt.figure(figsize=(6, 4))
plt.plot(cv_results["param_max_leaf_nodes"], cv_results["mean_test_score"], '
o-')
plt.plot(best_size, best_score, 'ro-')
plt.xlabel('Tree Size')
plt.ylabel('Accuracy')
plt.title('Cross-validation Results');
```

png

```
# prune tree using optimal size
prune_mar = DecisionTreeClassifier(max_leaf_nodes=best_size, random_state=7)
prune_mar.fit(X_train, y_train)

# plot pruned tree
plt.figure(figsize=(7,6))
plt.title('Pruned Tree')
plot_tree(prune_mar, feature_names=X_train.columns, filled=True);
```

png

```
# obtain predicted labels for test set
y_pred = prune_mar.predict(X_test)

# create confusion matrix
confusion_matrix = pd.crosstab(index=y_pred, columns=y_test, rownames=[''])
print(confusion_matrix)
```

```
substance_use_flag     0    1   2   3

0                   1170  409  43  28
```

```
# Calculate the accuracy of the decision tree on the test data
accuracy = prune_mar.score(X_test, y_test)
print("Accuracy: {:.2f}%".format(accuracy*100))
```

```
Accuracy: 70.91%
```

```
# fit Gradient Boosting model
boost = GradientBoostingClassifier(n_estimators=5000, max_depth=4
                                         , random_state=1)
boost.fit(X_train, y_train)
```

```
GradientBoostingClassifier(max_depth=4, n_estimators=5000, random_state=1)
```

```
# create a dataframe of feature importances and their corresponding column na
mes
importances_boost = pd.DataFrame({'Feature': X_train.columns, 'Importance': b
oost.feature_importances_})
importances_boost = importances_boost.sort_values('Importance', ascending=Fal
se).reset_index(drop=True)

# display the table
print(importances_boost)
```

```
                                  Feature  Importance
0       homework_monitoring_by_parent_2.0    0.062809
1        religious_beliefs_importance_2.0    0.061512
2     arguments_or_fights_with_parent_2.0    0.060894
3         participated_in_skills_group_2.0    0.038284
```

```
4     serious_fight_at_school_or_work_2.0    0.036727
..                                    ...         ...
60  days_missed_school_from_skipping_11.0    0.000849
61  days_missed_school_from_skipping_22.0    0.000768
62  days_missed_school_from_skipping_14.0    0.000647
63  days_missed_school_from_skipping_16.0    0.000149
64  days_missed_school_from_skipping_13.0    0.000097

[65 rows x 2 columns]
```

```
# obtain predicted labels for test set
y_pred = boost.predict(X_test)

# create confusion matrix
confusion_matrix = pd.crosstab(index=y_pred, columns=y_test, rownames=[''])
print(confusion_matrix)
```

```
substance_use_flag    0    1   2   3

0                   963  292  34  20
1                   187  102   9   8
2                    11   10   0   0
3                     9    5   0   0
```

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

```
Accuracy: 64.55%
```

Regression Problem

```
df_regressor = df_impute.copy()

df_regressor['age_first_alcohol_use'] = df_regressor['age_first_alcohol_use']
.replace([991], 0)

df_regressor['age_first_alcohol_use'].hist(grid =
                                          False)
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.title('Age of first use of Alcohol')
```

```
Text(0.5, 1.0, 'Age of first use of Alcohol')
```

png

```
# having the first alcohol age below age of 6 is unheard of . Let's make thes
e as 7.
df_impute['age_first_alcohol_use'].value_counts()
```

```
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: age_first_alcohol_use, dtype: int64
```

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

```
df_regressor['age_first_alcohol_use'].value_counts()
```

```
0.0     4138
15.0     329
14.0     291
13.0     225
16.0     166
12.0     112
11.0      61
17.0      57
7.0       45
10.0      36
9.0       21
8.0       19
Name: age_first_alcohol_use, dtype: int64
```

```
for col in df_regressor.columns[:-2]:
    convert_to_category(df_regressor,col)
    df_regressor= onehotencoding(df_regressor,col)
```

```
X = df_regressor[df_regressor.columns.difference(['alcohol_frequency_past_yea
r','age_first_alcohol_use'])]
y = df_regressor['age_first_alcohol_use']
```

```
# fit decision tree model
tree_mar = DecisionTreeRegressor()
tree_mar.fit(X, y)
```

```
DecisionTreeRegressor()
```

```
tree_summary = export_text(tree_mar, feature_names=X.columns.tolist())
#print(tree_summary)
```

```
importances = pd.DataFrame({'feature_name': X.columns, 'importance': tree_mar
.feature_importances_})
importances = importances.sort_values('importance', ascending=False).reset_in
dex(drop=True)
print(importances)
```

```
                                 feature_name  importance
0                        alcohol_use_flag_0.0    0.972885
1         serious_fight_at_school_or_work_2.0    0.001563
2         arguments_or_fights_with_parent_2.0    0.001226
3            participated_in_skills_group_1.0    0.001151
4            religious_service_attendance_1.0    0.001135
..                                        ...         ...
67                       alcohol_use_flag_1.0    0.000000
68  religious_beliefs_influence_decisions_0.0    0.000000
69          homework_monitoring_by_parent_0.0    0.000000
70      days_missed_school_from_skipping_11.0    0.000000
71                parent_praise_for_youth_0.0    0.000000

[72 rows x 2 columns]
```

```
plt.figure(figsize=(50,50))
plot_tree(tree_mar
          , filled=True
          , feature_names=X.columns
          #, class_names=[0, 1]
          , label='all'
          , fontsize=12)
plt.show()
```

png

The above output looks too messy to visualize. Let's see what output we are getting after running the model for the training dataset

Let's try different models to predict the alcohol usage based on various parameters

Simple Decision Classifier

since the data is imbalanced, we will do stratify sampling and see how it performs

Let's try Random Forest Classifier and see how it works in the training and test dataset

```
# Split the Data between test and train
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, rand
om_state=2, stratify=y)


# create Random Forest model with class_weight parameter
rf_model = RandomForestRegressor(n_estimators=5000, max_depth=4
                                          , random_state=1
                                          )

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

# predict on test data
rf_pred = rf_model.predict(X_test)

# plot predicted vs actual values
plt.scatter(rf_pred, y_test)
plt.plot([min(rf_pred), max(rf_pred)], [min(rf_pred), max(rf_pred)], 'r--')
plt.xlabel('Predicted values')
plt.ylabel('Actual values')
plt.title('Bagging model');
# find the MSE
print("Mean Squared Error: {:.2f}".format(mean_squared_error(y_test, rf_pred)
))
```

```
Mean Squared Error: 1.14
```

png

Let’s try bagging approach

```
# fit Random Forests model
alcohol_bagging = RandomForestRegressor(max_features=X_train.shape[1],random_
state = 2)
alcohol_bagging.fit(X_train,y_train)

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

```
Number of trees: 100
Number of features tried at each split: 72
Training score: 99.09%
```

```
# Predict values
y_pred_bag = alcohol_bagging.predict(X_test)

# plot predicted vs actual values
```

```
plt.scatter(y_pred_bag, y_test)
plt.plot([min(y_pred_bag), max(y_pred_bag)], [min(y_pred_bag), max(y_pred_bag
)], 'r--')
plt.xlabel('Predicted values')
plt.ylabel('Actual values')
plt.title('Bagging model');
# find the MSE
print("Mean Squared Error: {:.2f}".format(mean_squared_error(y_test, y_pred_b
ag)))
```

```
Mean Squared Error: 1.35
```

png

```
# Calculate the accuracy of the decision tree on the test data
accuracy = alcohol_bagging.score(X_test, y_test)
print("Accuracy: {:.2f}%".format(accuracy*100))
```

```
Accuracy: 96.25%
```

```
# fit decision tree model
tree_mar = DecisionTreeRegressor(random_state=7)
tree_mar.fit(X_train, y_train)

# cross-validation to determine optimal tree size
params = {'max_leaf_nodes': range(2, 20)}
cv_mar = GridSearchCV(tree_mar, params, cv=10)
cv_mar.fit(X_train, y_train)
cv_results = cv_mar.cv_results_

# find the best score for max leaf nodes
best_size = cv_mar.best_params_['max_leaf_nodes']
best_score = cv_mar.best_score_

# plot results of cross-validation
plt.figure(figsize=(6, 4))
plt.plot(cv_results["param_max_leaf_nodes"], cv_results["mean_test_score"], '
o-')
plt.plot(best_size, best_score, 'ro-')
plt.xlabel('Tree Size')
plt.ylabel('Accuracy')
plt.title('Cross-validation Results');
```

png

```
# prune tree using optimal size
prune_mar = DecisionTreeRegressor(max_leaf_nodes=best_size, random_state=7)
prune_mar.fit(X_train, y_train)

# plot pruned tree
plt.figure(figsize=(7,6))
```

```
plot_tree(prune_mar, feature_names=X_train.columns, filled=True)
plt.title('Pruned Tree');
```

png

```
# Calculate the accuracy of the decision tree on the test data
accuracy = prune_mar.score(X_test, y_test)
print("Accuracy: {:.2f}%".format(accuracy*100))
```

```
Accuracy: 96.84%
```

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

```
GradientBoostingRegressor(max_depth=4, n_estimators=5000, random_state=1)
```

```
# create a dataframe of feature importances and their corresponding column na
mes
importances_boost_alcohol = pd.DataFrame({'Feature': X_train.columns, 'Import
ance': boost_alcohol.feature_importances_})
importances_boost_alcohol = importances_boost_alcohol.sort_values('Importance
', ascending=False).reset_index(drop=True)

# display the table
print(importances_boost_alcohol)
```

```
                                      Feature    Importance
0                         alcohol_use_flag_1.0  5.548216e-01
1                         alcohol_use_flag_0.0  4.237421e-01
2          serious_fight_at_school_or_work_2.0  1.523897e-03
3    religious_beliefs_shared_with_friends_1.0  8.815725e-04
4                  parent_praise_for_youth_2.0  8.725426e-04
..                                         ...           ...
67       days_missed_school_from_skipping_16.0  5.800253e-08
68       days_missed_school_from_skipping_22.0  5.747186e-08
69       days_missed_school_from_skipping_97.0  4.897885e-09
70       days_missed_school_from_skipping_14.0  0.000000e+00
71       days_missed_school_from_skipping_11.0  0.000000e+00

[72 rows x 2 columns]
```

```
# obtain predicted labels for test set
y_pred = boost_alcohol.predict(X_test)
y_pred = np.clip(y_pred, 0, 365)
# plot predicted vs actual values
plt.scatter(y_pred, y_test)
plt.plot([min(y_pred), max(y_pred)], [min(y_pred), max(y_pred)], 'r--')
plt.xlabel('Predicted values')
plt.ylabel('Actual values')
plt.title('Pruned Decision Tree Model');
# find the MSE
print("Mean Squared Error: {:.2f}".format(mean_squared_error(y_test, y_pred))
)
```

```
Mean Squared Error: 1.85
```

png