Breaking the Cycle: Predicting Adolescent Substance Abuse through Socioeconomic and Criminal Factors

Group information

Family name	First name	Student ID	Email address	UTORID
Mehdiyar	Parmis	1005781307	parmis.mehdiyar@mail.utoronoto.ca	mehdiya2
Chim	Clifton	1005827857	clifton.chim@mail.utoronto.ca	chimchin

Introduction

This project aims to predict adolescent groups most vulnerable to substance abuse (alcohol, cannabis, narcotics) based on various socioeconomic characteristics such as income, poverty status, family structure, education, and criminal record data. Substance abuse among American adolescents is a significant public health concern, with nearly 14% of adolescents aged 12 to 17 reporting using illicit drugs in the past year. According to the National Institute on Drug Abuse, approximately 35% of highschool seniors reported using marijuana in the past year, and over 55% reported consuming alcohol.

Past researchers in public health economics have not reached consensus about the direction of the relationship between socioeconomic status and substance abuse using the traditional analytical methods, including linear regression models. However, we believe individuals from lower socioeconomic backgrounds may face a range of economic and social disadvantages, including limited access to education and employment opportunities, inadequate housing, and exposure to crime and violence. These factors can increase the likelihood of individuals engaging in criminal behavior, particularly when combined with other risk factors such as drug and alcohol abuse, mental health issues, and peer pressure. Therefore, in addition to examining socioeconomic factors, this project also incorporates criminal record data to provide a more comprehensive understanding of the predictors of substance abuse in adolescents.

Identifying the groups that are most at risk can inform the development of targeted prevention and intervention programs. For example, programs that provide support for low-income families or offer vocational training and employment opportunities for at-risk adolescents may be effective in reducing the likelihood of substance abuse. Understanding the relationship between socioeconomic factors and substance abuse can also inform policies aimed at reducing social inequalities and improving overall health and well-being. For example, policies that address

income inequality or increase access to healthcare services may help reduce the risk of substance abuse among vulnerable groups.

Moreover, identifying the most important features in predicting substance abuse in adolescents can provide valuable insights into the underlying factors contributing to the issue. By identifying the key predictors, we can prioritize intervention and prevention efforts more effectively. Therefore, understanding the most important predictors can guide the development of more effective prevention and intervention programs. For instance, if the most important predictor is poverty status, this finding may suggest that interventions targeting poverty reduction, such as increased access to job training or education programs, may be most effective in reducing substance abuse rates.

Background

Numerous studies have investigated the relationship between socioeconomic status (SES) and substance abuse, but the findings are not consistent. Some researchers suggest that individuals with higher SES are more likely to engage in substance abuse, while others suggest that lower SES is associated with substance abuse-related problems. This lack of consensus highlights the need for stronger models to predict substance abuse in adolescents to develop effective prevention and intervention strategies targeted towards vulnerable groups.

For instance, the work of Humensky (2010) aimed to investigate the relationship between SES and substance use among US adolescents. The author used data from the National Longitudinal Study of Adolescent Health, which is a nationally representative study of US adolescents, to examine the association between SES and substance use in early adulthood. The study employed regression frameworks and found that adolescents with higher SES were more likely to engage in binge drinking and marijuana use in early adulthood, even after controlling for other factors such as gender, race/ethnicity, and parental monitoring. The author asscociated this to greater disposable income, less parental supervision, and more exposure to substance use through peer networks. The study highlighted the need for future research to investigate the mechanisms underlying the association between SES and substance use to develop effective prevention and intervention strategies.

Differently, the study by Baptiste-Roberts and Hossain (2018) employed logistic regression analyses and found that individuals with lower income, lower education levels, and who were unemployed had higher odds of reporting substance abuse-related problems. This study explored the relationship between SES and self-reported substance abuse-related problems in the US using data from the National Survey on Drug Use and Health (NSDUH). The authors argued that the association between SES and substance abuse-related problems may be explained by a range of factors, including economic stress, social disadvantage, and poor mental health. This study also suggested that future research should investigate the specific factors through which SES affects substance abuse-related problems, in order to develop targeted interventions for disadvantaged populations.

Finally, Lemstra et al. (2008) add to the previous studies by conducting a meta-analysis of 10 studies to investigate the relationship between socioeconomic status and substance use among adolescents aged 10-15 years. They found a significant association between higher SES and increased alcohol use, which is consistent with Humensky's findings. However, unlike the other studies, they did not find a significant association between SES and marijuana use. These findings suggest that the relationship between SES and substance use may vary depending on the type of substance being used. The authors recommend future research to further explore the interaction between socioeconomic factors and specific substance uses to develop even more focused policies.

To the extent of our knowledge, machine learning tools have not been used extensively in this literature yet, and the lack of consensus on whether socioeconomic status impacts alcohol and substance use positively or negatively makes this a great case for non-linear analysis with specific focus on the interaction between different socioeconomic factors and specific substances Therefore, this project aims to improve the prediction tools for identifying adolescent groups most vulnerable to substance abuse (alcohol, cannabis, narcotics) based on various socioeconomic characteristics using machine learning. By advancing upon traditional analytical methods and accounting for more complex relationships, this study hopes to contribute to the development of more effective and specific prevention and intervention strategies to reduce substance use among vulnerable groups.

Data

Data Collection

Our data was extracted by picking 44 variables of interest relating to socioeconomic status, family structure, education and substance usage from the National Longitudinal Survery of Youth (https://www.nlsinfo.org/content/cohorts/nlsy97/get-data). Given the nature of the information and legality of access to substances and alcohol for adolescents, Self reports have been the most popular and accessible data source in this literature, as seen in both Baptiste-Roberts and Humensky's work. The NLSY particularly is a nationally representative survey conducted by the US Bureau of Labor Statistics, collecting data from over 12,000 individuals aged 14-21. The survey includes a wealth of demographic, socioeconomic, and health-related data, making it a valuable resource for the purpose of our analysis. Particularly, it contains data on alcohol and drug use among the cohort members, including the frequency and onset of use.

Data Construction

We selected 44 variables of interest relating to socioeconomic status, family structure, education, engagement in other illegal activities and substance usage. The variables were chosen based on common sense and the findings of previous studies. Next, we constructed a dataset (csv) using the selected variables of interest. However, the dataset required substantive

cleaning and wrangling. Many columns were categorical scales that needed to be converted into binary variables for our analysis. Furthermore, the dataset used negative numbers to indicate null. We fixed the null values by removing the negative numbers, and then renamed the remaining columns to more intuitive names. We also fixed any incorrect data types and turned categorical variables into dummies.

Data Cleaning

For missing observations, we first inspected the columns with the most missing information to consider dropping them if they were not of extreme importance. We wanted a good balance between features, outputs, and observations. Dropping all missing observations would have left us with a very small sample. After inspection, we dropped the 5 columns with the most missing values, as these columns included data on sentence to correctional institutions, probation, having run away, skipped school, and respondent's own income. These variables were not of extreme importance to our research and would have reduced our sample size significantly. This left us with 2090 observations and 38 features (before turning categorical variables, such as religion, race, and marital status into dummies).

For survey questions where responses were on a Cantril ladder scale, particularly frequencies of substance usage or involvement in illegal activities, we extracted two variables. The first variable was whether the participant had ever engaged in the behavior, and the second variable was whether the participant had passed the maximum threshold on the Cantril ladder scale of how frequently they engaged in that activity. We used the former for detecting early signs of vulnerability to substance usage and the latter for detecting substance abuse patterns. After all the cleaning, we had 43 features and 13 targets.

Model Selection

In order to predict adolescent substance use based on socioeconomic factors, we will employ three machine learning algorithms: decision tree, random forest, and gradient boosted trees. These tree-based methods can efficiently detect intricate relationships between socioeconomic factors and substance use, thus offering an enhancement to the predictive power of existing models.

The decision tree model is especially well-suited to our research question, as it is a supervised machine learning algorithm that is capable of handling both categorical and continuous data. We can use decision trees to identify the most significant SES predictors of substance use, amongst factors such as income and poverty status, family structure, education level, and more. The hierarchical structure of decision trees allows us to differentiate the most critical parameters and interactions between them based on purity indices. This is especially valuable for informing public health economics policy-making. Furthermore, the easy-to-interpret

decision tree visualization made it an ideal tool to convey information to policymakers and stakeholders.

In order to address the issue of overfitting and enhance the accuracy of our predictions, we also use ensemble extensions of the tree model. Random forest for example trains multiple decision trees and aggregates them to get more accurate predictions. Similarly, gradient boosted decision tree combine multiple weak learners to create a robust learner. Given the complex relationships between input features and the target variable, ensemble methods are particularly useful for our predicting problem. These methods are able to handle a large number of predictor variables and identify important predictors, which is important in the context of targeted and focused prevention or intervention policy making. Finally, ensemble tree methods have been shown to outperform single tree methods in predictive accuracy and out of sample generalizability.

Estimation

In this section, we tune parameters for the optimal performance of our machine learning models. This process allows us to fine-tune our models to achieve the best possible performance and accuracy in predicting substance usage in young adults out of the sample. We focused on optimizing the parameters for decision tree, random forest, and gradient boosted trees for each of the 13 targets.

To begin, we split the data into a training set, consisting of 75% of the data, and a testing set consisting of the remaining 25% of the data. We then used a grid search with a 5-fold cross-validation to identify the best parameters for each model, for each target.

For the decision tree, we tuned the maximum depth of the tree, the minimum number of samples required to split a node, the minimum number of samples required to be at a leaf node, and the criterion used to measure the quality of a split (either Gini impurity or information gain).

For the random forest, we tuned the number of trees in the forest and the maximum number of features considered for each split (either the square root or logarithm of the total number of features - two popular options).

For the gradient boosted trees, we tuned the number of trees, the learning rate, and the maximum number of features considered for each split (againn either the square root or logarithm of the total number of features).

After identifying the best parameters for each model, we used these parameters to train our models and make predictions on the test set. This process allowed us to identify the best-performing models for each of our 13 targets, which we used to generate our final predictions.

Results

After optimizing the parameters for decision tree, random forest, and gradient boosted trees, we computed accuracy and recall scores for each of the 13 targets. Since our main goal was to identify the most vulnerable adolescents, we gave more importance to recall than precision. To achieve this, we added recall rate into the score evaluation and determined where it was maximized. To gain further insight, we computed the Top 5 Features for each model. Additionally, for visualization purposes, we plotted a graph of the decision tree, which can be found in Appendix 2 - Analysis.

Cannabis:

In predicting whether a candidate had ever smoked cannabis (target 9), the decision tree had the highest accuracy and recall, and predicted that having ever shoplifted was the most important features. In predicting whether a candidate had smoked cannabis more than 10 times this year (target 1), the gradient-boosted tree, had the highest accuracy and recall, and predicted that having ever shoplifted was the most important feature. In predicting whether a candidate had ever sold cannabis (target 11), the gradient boosted decision tree had the highest accuracy and recall, and predicted that having ever threatened to hit someone was the most important feature.

Alcohol:

In predicting whether a candidate had ever had 6 drinks or more at once (target 13), random forest had the highest accuracy by very little margin, but gradient-boosted trees had the highest recall rate. Boosted trees marked having ever destructed someone's property as the most important feature. In predicting whether a candidate had had 6 drinks or more at once, more than 3 times over the last month (target 5), boosted trees had the highest accuracy and recall, and marked being male as the most important feature.

Hard drugs and narcotics:

In predicting whether a candidate had ever used cocaine (target 6), random forest had the highest accuracy and recall, and marked having frequent religious attendance as the most important feature. In predicting whether a candidate had ever used other narcotics (target 7), decision tree had the highest recall with marginally less accuracy that the other models, and marked having ever broke in as the most important feature. In predicting whether a candidate had ever sold hard drugs (target 12), decision tree had the highest recall with marginally less accuracy that the other models, and marked having ever shoplifted as the most important feature.

Chemicals and other drugs:

In predicting whether a candidate had ever gotten high from chemicals (target 10), random forest had the lowest accuracy but almost perfect recall rate, and marked estimated net family income as the most important feature.

Finally, in predicting whether a candidate had ever gotten high from chemicals more than 10 times this year (target 2), ever used any other drugs (target 8), sold hard drugs more than 10 times this year (target 4) or sold cannabis more than 10 times this year (target 3), our results were inconclusive with high accuracy rates and near-0 recall rates, due to very few observations who fit this criterion. The models marked completely different features as important in each case, and it is important to note that these results do not hold a strong predictive value and have likely not captured the true patterns of behaviours of interest.

Conclusion

In conclusion, our models were able to predict the likelihood of adolescents engaging in drug and alcohol use, abuse and sale with high accuracy, but generally low recall rates - i.e. identifying all the volnurable adolescents. We also aimed to identify the key risk factors associated with adolescent drug and alcohol use, and by analyzing data from a large survey, we trained decision tree, random forest, and gradient boosted trees models to predict 13 different drug and alcohol use outcomes. We found that no single model performed universally better than others, with the best performing model varying depending on the target outcome.

One of the main limitations of our study is the reliance on self-reported data, which can be subject to recall bias and social desirability bias. Additionally, we cannot establish causality between the important features and the predictions as some features may have been effects, rather than causes, of drug and alcohol usage. This means that our findings only provide insight into the associations between the identified features and drug and alcohol usage, rather than definitive proof of causality.Another limitation is that our machine learning models could not help make many inferences about how exactly features affect adolescents' drug usage due to the nature of the models. While the models can accurately predict drug usage based on the available data, they cannot explain the underlying mechanisms that drive these relationships.

To address these limitations, future research should consider using objective measures of drug and alcohol use, such as biological markers or medical records, to supplement self-reported data. Additionally, research should use longitudinal data to better understand the temporal relationships between the identified features and drug and alcohol usage. Furthermore, future studies could incorporate other methods, such as clustering or structural equation modeling, to better explore the complex relationships between the identified features and drug and alcohol use. Particularly, in terms of addressing the issue of low recall rates, clustering techniques could help identify subgroups of adolescents with higher risk profiles. Additionally, our findings

underscore the importance of larger and more representative sample sizes, particularly when it comes to modeling the complex behaviors of drug sales.

Overall, our study contributes to the existing literature by providing insights into the features that are most strongly associated with drug and alcohol use among adolescents. We also developed strong predictin algorithms that although have limited interpretability, could help public health economists idenntify the most volnurable adolescents in terms of alcohol and substabce uses. However, the limitations to our approach must be acknowledged, and future research should continue to build on our findings to better understand the underlying factors that contribute to adolescent substance use.

References

1- Humensky, J.L. Are adolescents with high socioeconomic status more likely to engage in alcohol and illicit drug use in early adulthood?. Subst Abuse Treat Prev Policy 5, 19 (2010). https://doi.org/10.1186/1747-597X-5-19

2- Baptiste-Roberts K, Hossain M. Socioeconomic Disparities and Self-reported Substance Abuse-related Problems. Addict Health. 2018 Apr;10(2):112-122. https://doi.org/10.22122/ahj.v10i2.561.

3- Lemstra, Mark, et al. "A Meta-Analysis of Marijuana and Alcohol Use by Socio-Economic Status in Adolescents Aged 10-15 Years." Canadian Journal of Public Health / Revue Canadienne de Sante'e Publique, vol. 99, no. 3, 2008, pp. 172–77. JSTOR, http://www.jstor.org/stable/41995073. Accessed 1 Mar. 2023.

Appendix

Appendix 1 - Cleaning

The cleaning process includes removing null values (marked by negative values in the original dataset), renaming the remaining columns to more intuitive names, fixing any incorrect data types and turning categorical variables into dummies.

For missig observations, we first inspected the columnns with most missing information to consider dropping them if they were not of extreme importance, as we wanted a good balance between features, outputs and observations. Dropping all missing observations would have left us with a very small sample.

In [ ]:
```
# Importing libraries and setting up access to Drive

import pandas as pd
import numpy as np
```

```
pd.set_option('display.max_columns', None)


from google.colab import drive
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

In [ ]:
```
df = pd.read_csv('/content/gdrive/MyDrive/UofT/ECO480/480_project/default.csv')
```

In [ ]:
```
# Getting rid of columns with most missing variables:

# Replace all values below 0 with NaN
df = df.mask(df < 0, np.nan)

# Inspecting the columns with most missing observations
missing_counts = df.isnull().sum()
missing_counts_sorted = missing_counts.sort_values(ascending=False)
missing_counts_sorted.head(44)
```

Out[ ]:
```
POLICE-7       9937
POLICE-6       9937
DELIN-3        8800
DELIN-1        8788
Q13-5          5787
Q12-4          4801
H40-BPAR-6     4317
MIL-6          4102
TNFI_TRUNC     2491
POVSTATUS      2491
HGC-FATHER     1806
URBAN-RURAL    1220
DELIN-14        993
DELIN-13        988
DELIN-15        969
DELIN-12        964
DELIN-4         952
DELIN-18        938
DELIN-8         910
DELIN-10        901
DELIN-7         898
DELIN-6         898
DELIN-11        894
DELIN-9         892
DELIN-5         886
HGC-MOTHER      808
DRUG-32         718
DRUG-26         685
DRUG-22         675
POLICE-1        557
Q3-3            160
Q3-4            152
R_REL-1_COL      56
Q9-72            20
FAM-28A          18
R_REL-3          17
MARSTAT-KEY       2
FAM-2A            1
SAMPLE_ID         0
CASEID            0
SAMPLE_RACE       0
SAMPLE_SEX        0
FAM-1B            0
FAMSIZE           0
dtype: int64
```

After inspection, we drop the 5 columns with most missing values, as these columns included data on sentence to correctional institutions, probation, having ran away, skipped school, and respondant's own income. These variables were not of extreme importance to our research and would have reduced our sample size significantly.

In [ ]:
```python
# Get the column names of the 5 columns with the most missing values

cols_to_drop = list(missing_counts_sorted.head(5).index)

# Drop the columns from the DataFrame
df = df.drop(columns=cols_to_drop)
```

```
# Drop the rest of missing observations manually
df = df.dropna()

# Drop the ID column and reset index, as about 2000 observations are left
df = df.drop(columns=['CASEID'])
df = df.reset_index(drop=True)

df
```

Out[ ]:

	H40-BPAR-6	FAM-1B	FAM-2A	HGC-MOTHER	HGC-FATHER	FAM-28A	R_REL-1_COL	R_REL-3	Q9-72	Q3-3	Q3-4	MIL-6	SAMI
0	1.0	18	1.0	12.0	12.0	1.0	7.0	1.0	0.0	13.0	12.0	0.0	
1	0.0	20	1.0	9.0	6.0	7.0	9.0	2.0	1.0	12.0	12.0	0.0	
2	1.0	20	1.0	12.0	12.0	3.0	7.0	2.0	0.0	12.0	12.0	0.0	
3	1.0	21	1.0	12.0	16.0	2.0	9.0	2.0	0.0	13.0	12.0	0.0	
4	1.0	21	1.0	10.0	12.0	3.0	7.0	1.0	0.0	12.0	12.0	0.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
2085	1.0	19	1.0	12.0	4.0	4.0	2.0	4.0	0.0	13.0	12.0	0.0	
2086	0.0	21	1.0	12.0	12.0	4.0	2.0	4.0	1.0	11.0	10.0	1.0	
2087	1.0	21	1.0	14.0	16.0	3.0	5.0	2.0	0.0	16.0	16.0	0.0	
2088	1.0	18	1.0	14.0	12.0	3.0	2.0	1.0	0.0	12.0	12.0	0.0	
2089	1.0	19	1.0	12.0	12.0	2.0	7.0	3.0	0.0	14.0	13.0	0.0	

2090 rows × 38 columns

In [ ]:
```
# Cleaning: Parmis

# ================================================================================
# adding dummies for categorical colunmns

# ======== Religion ========
# create dummy variables
dummy_df = pd.get_dummies(df['R_REL-1_COL'], prefix='rel')
# rename columns and drop old column
religion_map = {'rel_0.0': 'NO RELIGION', 'rel_1.0': 'PROTESTANT', 'rel_2.0':
                'BAPTIST', 'rel_3.0': 'EPISCOPALIAN', 'rel_4.0': 'LUTHERAN',
                'rel_5.0': 'METHODIST', 'rel_6.0': 'PRESBYTERIAN', 'rel_7.0':
                'ROMAN CATHOLIC', 'rel_8.0': 'JEWISH', 'rel_9.0': 'OTHER'}
religion_map = {k: v.lower() for k, v in religion_map.items()}
dummy_df = dummy_df.rename(columns=religion_map)
dummy_df = dummy_df.add_prefix('rel_')
df = df.drop('R_REL-1_COL', axis=1)
# concatenate new columns to original dataframe
df = pd.concat([df, dummy_df], axis=1)

# ======== Race ========
# create dummy variables
dummy_df = pd.get_dummies(df['SAMPLE_RACE'], prefix='race')
```

```
# rename columns and drop old column
race_map = {'race_1': 'hispanic', 'race_2': 'black', 'race_3':
                'non_black_or_hispanic'}
dummy_df = dummy_df.rename(columns=race_map)
dummy_df = dummy_df.add_prefix('race_')
df = df.drop('SAMPLE_RACE', axis=1)
# concatenate new columns to original dataframe
df = pd.concat([df, dummy_df], axis=1)

# ================================================================================
# Renaming columns
df = df.rename(columns={'H40-BPAR-6': 'mother_alive', 'FAM-1B': 'age',
                        'FAM-2A': 'born_in_US', 'HGC-MOTHER': 'education_mother'
                        , 'HGC-FATHER': 'education_father', 'FAM-28A':
                        'num_siblings', 'R_REL-3': 'rel_attndnce_freq', 'Q9-72'
                        : 'has_children', 'Q3-3': 'highest_grade_attended',
                        'Q3-4': 'highest_grade_completed', 'MIL-6':
                        'been_in_armed_forces', 'SAMPLE_SEX': 'male'})

# ================================================================================
# Fixing notations based on NLSY guidelines
# ======== Born in US ========
df['born_in_US'] = df['born_in_US'].replace(2, 0)

# ======== Religion frequency ========
df['rel_attndnce_freq'] = df['rel_attndnce_freq'].apply(
    lambda x: 0 if x <= 2 else 1)

# ======== male ========
df['male'] = df['male'].replace(2, 0)

# Sample ID is irrelevant
df = df.drop('SAMPLE_ID', axis=1)
```

In [ ]:

```
# Cleaning: Clifton
# ================================================================================

# ======== Marital Status ========
# create dummy variables
dummy_df = pd.get_dummies(df['MARSTAT-KEY'], prefix='maritalstatus')
# rename columns and drop old column
maritalstatus_map = {'maritalstatus_0.0': 'NEVER MARRIED', 'maritalstatus_1.0':
                     'MARRIED', 'maritalstatus_2.0': 'SEPARATED',
                     'maritalstatus_3.0': 'DIVORCED', 'maritalstatus_5.0':
                     'REMARRIED', 'maritalstatus_6.0': 'WIDOWED'}
maritalstatus_map = {k: v.lower() for k, v in maritalstatus_map.items()}
dummy_df = dummy_df.rename(columns=maritalstatus_map)
dummy_df = dummy_df.add_prefix('mar_stat_')
df = df.drop('MARSTAT-KEY', axis=1)
# concatenate new columns to original dataframe
df = pd.concat([df, dummy_df], axis=1)

# ======== Ever intentionally damaged property? ========
df['DELIN-4'] = df['DELIN-4'].apply(lambda x: 1 if x != 0 else 0)

# ======== Ever had physical fight at school or work? ========
df['DELIN-5'] = df['DELIN-5'].apply(lambda x: 1 if x != 0 else 0)
```

```
# ======== Ever shoplifted ========
df['DELIN-6'] = df['DELIN-6'].apply(lambda x: 1 if x != 0 else 0)

# ======== Ever stolen other's belongings less than 50 ========
df['DELIN-7'] = df['DELIN-7'].apply(lambda x: 1 if x != 0 else 0)

# ======== Frequency of stolen other's belongings more than 50 ========
df['DELIN-8'] = df['DELIN-8'].apply(lambda x: 1 if x != 0 else 0)

# ======== Ever used force to obtain things ========
# create dummy variables
df['DELIN-9'] = df['DELIN-9'].apply(lambda x: 1 if x != 0 else 0)

# ======== Ever seriously threatened to hit/ hit someone ========
df['DELIN-10'] = df['DELIN-10'].apply(lambda x: 1 if x != 0 else 0)

# ======== Ever attacked or intended to injure or kill someone ========
df['DELIN-11'] = df['DELIN-11'].apply(lambda x: 1 if x != 0 else 0)

# ======== Ever smoked cannabis this year? ========
df['ever_cannabis'] = df['DELIN-12'].copy() # using DELIN-12 again for freq
df['ever_cannabis'] = df['ever_cannabis'].apply(lambda x: 1 if x != 0 else 0)

# ======== Smoked cannabis more than 10 times this year? ========
df['DELIN-12'] = df['DELIN-12'].apply(lambda x: 1 if x > 4 else 0)

# ======== Ever used other drugs or chemical to get high this year? ========
df['ever_drugs_chemicals_high'] = df['DELIN-13'].copy()
# using DELIN-13 again for freq
df['ever_drugs_chemicals_high'] = df['ever_drugs_chemicals_high'].apply(
    lambda x: 1 if x != 0 else 0)

# ======== Used drugs or chemicals to get high more than 10 times? ========
df['DELIN-13'] = df['DELIN-13'].apply(lambda x: 1 if x > 4 else 0)

# ======== Ever sold cannabis this year? ========
df['ever_sold_cannabis'] = df['DELIN-14'].copy() # using DELIN-14 again for freq
df['ever_sold_cannabis'] = df['ever_sold_cannabis'].apply(
    lambda x: 1 if x != 0 else 0)

# ======== Sold cannabis more than 10 times this year? ========
df['DELIN-14'] = df['DELIN-14'].apply(lambda x: 1 if x > 4 else 0)

# ======== Ever sold hard drugs this year? ========
df['ever_sold_hard_drugs'] = df['DELIN-15'].copy()
# using DELIN-15 again for freq
df['ever_sold_hard_drugs'] = df['ever_sold_hard_drugs'].apply(
    lambda x: 1 if x != 0 else 0)

# ======== Sold hard drugs more than 10 times this year? ========
df['DELIN-15'] = df['DELIN-15'].apply(lambda x: 1 if x > 4 else 0)

# ======== Ever broke into a building ========
df['DELIN-18'] = df['DELIN-18'].apply(lambda x: 1 if x != 0 else 0)

# ======== Ever had 6 or more drinks at once in last month ========
df['ever_<6_drinks_at_once'] = df['Q12-4'].copy()
# using Q12-4 again for freq
df['ever_<6_drinks_at_once'] = df['ever_<6_drinks_at_once'].apply(
```

```
    lambda x: 1 if x != 0 else 0)

# ======== Has had 6 or more drinks more than 3 times in last month ========
df['Q12-4'] = df['Q12-4'].apply(lambda x: 1 if x > 2 else 0)


# ================================================================================
# Renaming columns
df = df.rename(columns={'POVSTATUS': 'in_poverty', 'POLICE-1':
                        'ever_stopped_by_police', 'DRUG-22': 'ever_cocaine',
                        'DRUG-26': 'ever_other_narcotics', 'DRUG-32':
                        'ever_other_drugs', 'URBAN-RURAL': 'urban','FAMSIZE':
                        'fam_size', 'DELIN-4': 'ever_destructed_property',
                        'DELIN-5': 'ever_physical_fight', 'DELIN-6':
                        'ever_shoplifted','DELIN-7': 'ever_stolen_<50',
                        'DELIN-8': 'ever_stolen_>50', 'DELIN-9':
                        'ever_used_force', 'DELIN-10': 'ever_threatened_hit',
                        'DELIN-11': 'ever_attacked_injured_killed', 'DELIN-12':
                        'cannabis_this_year_>10', 'DELIN-13':
                        'drugs_chemical_high_this_year_>10',
                        'DELIN-14': 'sold_can_this_year_>10', 'DELIN-15':
                        'sold_hard_drugs_this_year_>10', 'DELIN-18':
                        'ever_broke_in', 'TNFI_TRUNC':
                        'estimated_fam_income_this_year', 'Q12-4':
                        'had_+6_drinks_>3'})
```

Appendix 2 - Analysis

Decision tree, random forest, gradient boosted trees. If we want to over-identify (for identifying the most volnureable adolescents possible), we would care more about recall than precision, hence why I have added recall rate into score evaluation to see where it is maximized.

In [ ]:
```
# Modules
import inspect
import numpy as np
import pandas as pd

from sklearn import ensemble, metrics, model_selection, preprocessing, tree
from matplotlib import pyplot
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score
from sklearn.metrics import accuracy_score, recall_score
```

In [ ]:
```
# separating targets

targets = {}

for col in ['cannabis_this_year_>10', 'drugs_chemical_high_this_year_>10',
            'sold_can_this_year_>10', 'sold_hard_drugs_this_year_>10',
            'had_+6_drinks_>3', 'ever_cocaine', 'ever_other_narcotics',
            'ever_other_drugs', 'ever_cannabis', 'ever_drugs_chemicals_high',
            'ever_sold_cannabis', 'ever_sold_hard_drugs',
```

```
            'ever_<6_drinks_at_once']:
            new_df = pd.DataFrame(df[col])
            new_df.columns = [col]
            targets[col] = new_df
            df = df.drop(col, axis=1)

taregt_cannabis_this_year_10 = targets['cannabis_this_year_>10']
target_drugs_chemical_high_this_year_10 = targets[
    'drugs_chemical_high_this_year_>10']
target_sold_can_this_year_10 = targets['sold_can_this_year_>10']
target_sold_hard_drugs_this_year_10 = targets['sold_hard_drugs_this_year_>10']
target_had_6_drinks_3 = targets['had_+6_drinks_>3']
target_ever_cocaine = targets['ever_cocaine']
target_ever_other_narcotics = targets['ever_other_narcotics']
target_ever_other_drugs = targets['ever_other_drugs']
target_ever_cannabis = targets['ever_cannabis']
target_ever_drugs_chemicals_high = targets['ever_drugs_chemicals_high']
target_ever_sold_cannabis = targets['ever_sold_cannabis']
target_ever_sold_hard_drugs = targets['ever_sold_hard_drugs']
target_ever_6_drinks_at_once = targets['ever_<6_drinks_at_once']
```

Target 1: Smoked cannabis more than 10 times this year

In [ ]:
```
# ================================ Target 1 ================================:
target = taregt_cannabis_this_year_10.values

# sample split:
X_train, X_test, y_train, y_test = model_selection.train_test_split(
    df, target, train_size=.75, test_size=.25,
    shuffle=True, random_state=1)

np.random.seed(0)

# ============================== tree tuning ==============================:
# Define the parameter grid to search
param_grid = {'max_depth': [2, 4, 6, 8, 10],
              'min_samples_split': [2, 4, 6, 8, 10],
              'min_samples_leaf': [1, 2, 3, 4, 5],
              'criterion': ['gini', 'entropy']}

# Create a decision tree classifier object
dtc = DecisionTreeClassifier()

# Create a GridSearchCV object
grid_search = GridSearchCV(dtc, param_grid, cv=5)

# Fit the GridSearchCV object to the data
grid_search.fit(X_train, y_train)

# Print the best parameters and best score
print("Best parameters: ", grid_search.best_params_)
print("Best score: ", grid_search.best_score_)
```

```
Best parameters:  {'criterion': 'gini', 'max_depth': 2, 'min_samples_leaf': 1, 'mi
n_samples_split': 2}
Best score:  0.7268411306241224
```

In [ ]:
```
# ================================ Target 1 ================================:
# ========================== random forest tuning ==========================:
np.random.seed(0)

# Define the parameter grid to search
param_grid = {'n_estimators': [50, 100, 200],
              'max_features': ['sqrt', 'log2']}

# Create a random forest classifier object
rfc = RandomForestClassifier()

# Create a GridSearchCV object
grid_search = GridSearchCV(rfc, param_grid, cv=5)

# Fit the GridSearchCV object to the data
grid_search.fit(X_train, y_train.ravel())

# Print the best parameters and best score
print("Best parameters: ", grid_search.best_params_)
print("Best score: ", grid_search.best_score_)
```

```
Best parameters:  {'max_features': 'log2', 'n_estimators': 200}
Best score:  0.7192018884434586
```

In [ ]:
```
# ================================ Target 1 ================================:
# ========================== gradient-boost tuning ==========================:
np.random.seed(0)

# Define the parameter grid to search
param_grid = {'n_estimators': [50, 100, 200],
              'learning_rate': [0.05, 0.1, 0.2],
              'max_features': ['sqrt', 'log2']}

# Create a gradient boosted classifier object
gbc = GradientBoostingClassifier()

# Create a GridSearchCV object
grid_search = GridSearchCV(gbc, param_grid, cv=5)

# Fit the GridSearchCV object to the data
grid_search.fit(X_train, y_train.ravel())

# Print the best parameters and best score
print("Best parameters: ", grid_search.best_params_)
print("Best score: ", grid_search.best_score_)
```

```
Best parameters:  {'learning_rate': 0.05, 'max_features': 'sqrt', 'n_estimators':
200}
Best score:  0.7281170509350643
```

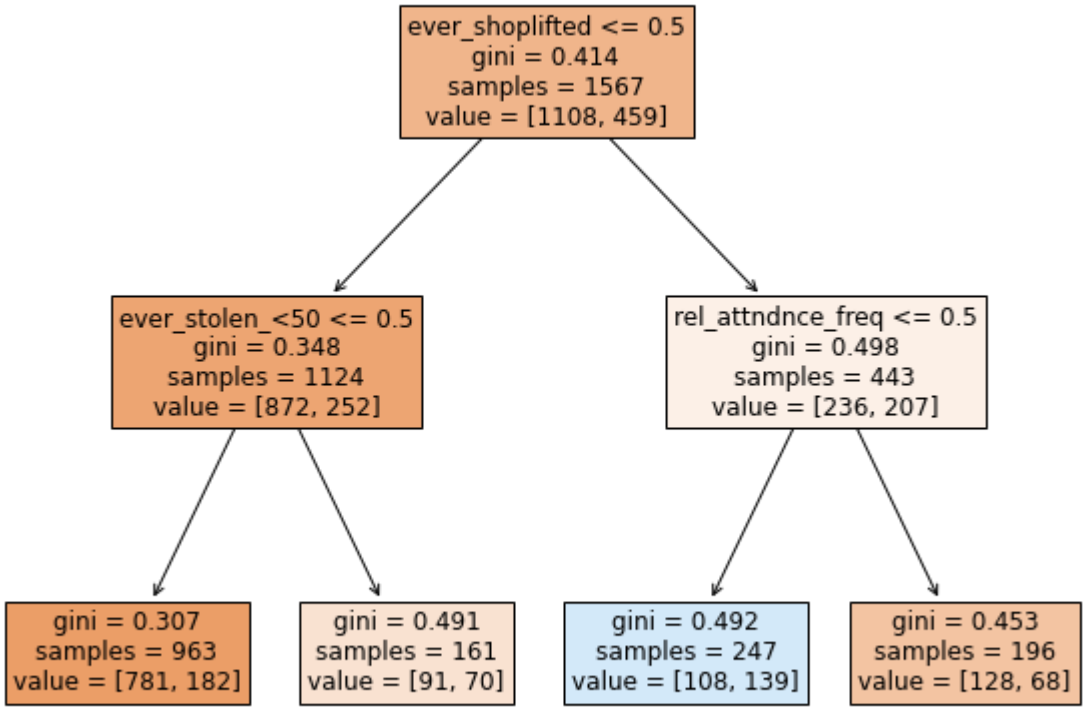
In [ ]:
```
# ================================ Target 1 ================================:

# Fit decision tree
dt_model = DecisionTreeClassifier(criterion='gini', max_depth=2, random_state=0,
                                  min_samples_leaf=1, min_samples_split=2)
dt_model.fit(X_train, y_train.ravel())
dt_pred = dt_model.predict(X_test)
dt_acc = accuracy_score(y_test.ravel(), dt_pred)
```

```
# Fit random forest
rf_model = RandomForestClassifier(max_features='log2', n_estimators=200,
                                  random_state=0)
rf_model.fit(X_train, y_train.ravel())
rf_pred = rf_model.predict(X_test)
rf_acc = accuracy_score(y_test.ravel(), rf_pred)

# Fit gradient boosted trees
gbt_model = GradientBoostingClassifier(learning_rate=0.05, max_features='sqrt',
                                       n_estimators=200, random_state=0)
gbt_model.fit(X_train, y_train.ravel())
gbt_pred = gbt_model.predict(X_test)
gbt_acc = accuracy_score(y_test.ravel(), gbt_pred)

# Calculate accuracy and recall scores for each model
dt_acc = accuracy_score(y_test, dt_pred)
rf_acc = accuracy_score(y_test, rf_pred)
gbt_acc = accuracy_score(y_test, gbt_pred)

dt_recall = recall_score(y_test, dt_pred)
rf_recall = recall_score(y_test, rf_pred)
gbt_recall = recall_score(y_test, gbt_pred)

# Compute top 5 most important features for each model
dt_feat_importances = pd.Series(dt_model.feature_importances_,
                                index=X_train.columns
                                ).nlargest(5).index.tolist()

rf_feat_importances = pd.Series(rf_model.feature_importances_,
                                index=X_train.columns
                                ).nlargest(5).index.tolist()

gbt_feat_importances = pd.Series(gbt_model.feature_importances_,
                                 index=X_train.columns
                                 ).nlargest(5).index.tolist()


# Create table of important features, accuracy and recall scores
scores = {'Model': ['Decision Tree', 'Random Forest', 'Gradient Boosted Trees'],
          'Accuracy': [dt_acc, rf_acc, gbt_acc],
          'Recall': [dt_recall, rf_recall, gbt_recall],
          'Top 5 Features': [dt_feat_importances, rf_feat_importances,
                             gbt_feat_importances]}
scores = pd.DataFrame(scores)
pd.set_option('display.max_colwidth', None)
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
scores
```

Out[ ]:

	Model	Accuracy	Recall	Top 5 Features
0	Decision Tree	0.705545	0.279221	[ever_shoplifted, ever_stolen_<50, rel_attndnce_freq, mother_alive, age]
1	Random Forest	0.734226	0.246753	[estimated_fam_income_this_year, education_father, num_siblings, fam_size, education_mother]
2	Gradient Boosted Trees	0.736138	0.305195	[ever_shoplifted, estimated_fam_income_this_year, rel_attndnce_freq, ever_destructed_property, education_father]

In [ ]:
```
# ================================= Target 1 ================================:
# plot tree:
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

plt.figure(figsize=(10,8))
plot_tree(dt_model, filled=True, fontsize=12, feature_names=df.columns)
plt.show()
```



Target 2: Has gotten high from other drugs or chemicals more than 10 times this year

In [ ]:
```
# ================================= Target 2 ================================:
target2 = target_drugs_chemical_high_this_year_10.values

# sample split:
```

```
X_train, X_test, y_train, y_test = model_selection.train_test_split(
    df, target2, train_size=.75, test_size=.25,
    shuffle=True, random_state=2)

np.random.seed(2)

# ============================== tree tuning ==============================:
# Define the parameter grid to search
param_grid = {'max_depth': [2, 4, 6, 8, 10],
              'min_samples_split': [2, 4, 6, 8, 10],
              'min_samples_leaf': [1, 2, 3, 4, 5],
              'criterion': ['gini', 'entropy']}

# Create a decision tree classifier object
dtc2 = DecisionTreeClassifier()

# Create a GridSearchCV object
grid_search = GridSearchCV(dtc2, param_grid, cv=5)

# Fit the GridSearchCV object to the data
grid_search.fit(X_train, y_train)

# Print the best parameters and best score
print("Best parameters: ", grid_search.best_params_)
print("Best score: ", grid_search.best_score_)
```

```
Best parameters:  {'criterion': 'entropy', 'max_depth': 2, 'min_samples_leaf': 1,
'min_samples_split': 2}
Best score:  0.9355466921715065
```

In [ ]:
```
# ================================ Target 2 ================================:
# ========================== random forest tuning ==========================:
np.random.seed(2)

# Define the parameter grid to search
param_grid = {'n_estimators': [50, 100, 150, 200],
              'max_features': ['sqrt', 'log2']}

# Create a random forest classifier object
rfc2 = RandomForestClassifier()

# Create a GridSearchCV object
grid_search2 = GridSearchCV(rfc2, param_grid, cv=5)

# Fit the GridSearchCV object to the data
grid_search2.fit(X_train, y_train.ravel())

# Print the best parameters and best score
print("Best parameters: ", grid_search2.best_params_)
print("Best score: ", grid_search2.best_score_)
```

```
Best parameters:  {'max_features': 'sqrt', 'n_estimators': 150}
Best score:  0.9355466921715065
```

In [ ]:
```
# ================================ Target 2 ================================:
# ========================== gradient-boost tuning ==========================:
np.random.seed(2)

# Define the parameter grid to search
```

```
param_grid = {'n_estimators': [50, 100, 200],
              'learning_rate': [0.001, 0.01, 0.05, 0.1, 0.15, 0.2, 0.3, 0.5],
              'max_features': ['sqrt', 'log2']}

# Create a gradient boosted classifier object
gbc2 = GradientBoostingClassifier()

# Create a GridSearchCV object
grid_search = GridSearchCV(gbc2, param_grid, cv=5)

# Fit the GridSearchCV object to the data
grid_search.fit(X_train, y_train.ravel())

# Print the best parameters and best score
print("Best parameters: ", grid_search.best_params_)
print("Best score: ", grid_search.best_score_)
```

```
Best parameters:  {'learning_rate': 0.001, 'max_features': 'sqrt', 'n_estimators':
50}
Best score:  0.9355466921715065
```

In [ ]:

```
# ================================ Target 2 ================================:

# Fit decision tree
dt_model_2 = DecisionTreeClassifier(criterion='entropy', max_depth=2, random_state
                                    min_samples_leaf=1, min_samples_split=2)
dt_model_2.fit(X_train, y_train.ravel())
dt_pred_2 = dt_model_2.predict(X_test)
dt_acc_2 = accuracy_score(y_test.ravel(), dt_pred_2)

# Fit random forest
rf_model_2 = RandomForestClassifier(max_features='sqrt', n_estimators=150,
                                    random_state=2)
rf_model_2.fit(X_train, y_train.ravel())
rf_pred_2 = rf_model_2.predict(X_test)
rf_acc_2 = accuracy_score(y_test.ravel(), rf_pred_2)

# Fit gradient boosted trees
gbt_model_2 = GradientBoostingClassifier(learning_rate=0.001, max_features='sqrt',
                                         n_estimators=50, random_state=2)
gbt_model_2.fit(X_train, y_train.ravel())
gbt_pred_2 = gbt_model_2.predict(X_test)
gbt_acc_2 = accuracy_score(y_test.ravel(), gbt_pred_2)

# Calculate accuracy and recall scores for each model
dt_acc_2 = accuracy_score(y_test, dt_pred_2)
rf_acc_2 = accuracy_score(y_test, rf_pred_2)
gbt_acc_2 = accuracy_score(y_test, gbt_pred_2)

dt_recall_2 = recall_score(y_test, dt_pred_2)
rf_recall_2 = recall_score(y_test, rf_pred_2)
gbt_recall_2 = recall_score(y_test, gbt_pred_2)

# Compute top 5 most important features for each model
dt2_feat_importances = pd.Series(dt_model_2.feature_importances_,
                                 index=X_train.columns
                                 ).nlargest(5).index.tolist()

rf2_feat_importances = pd.Series(rf_model_2.feature_importances_,
```

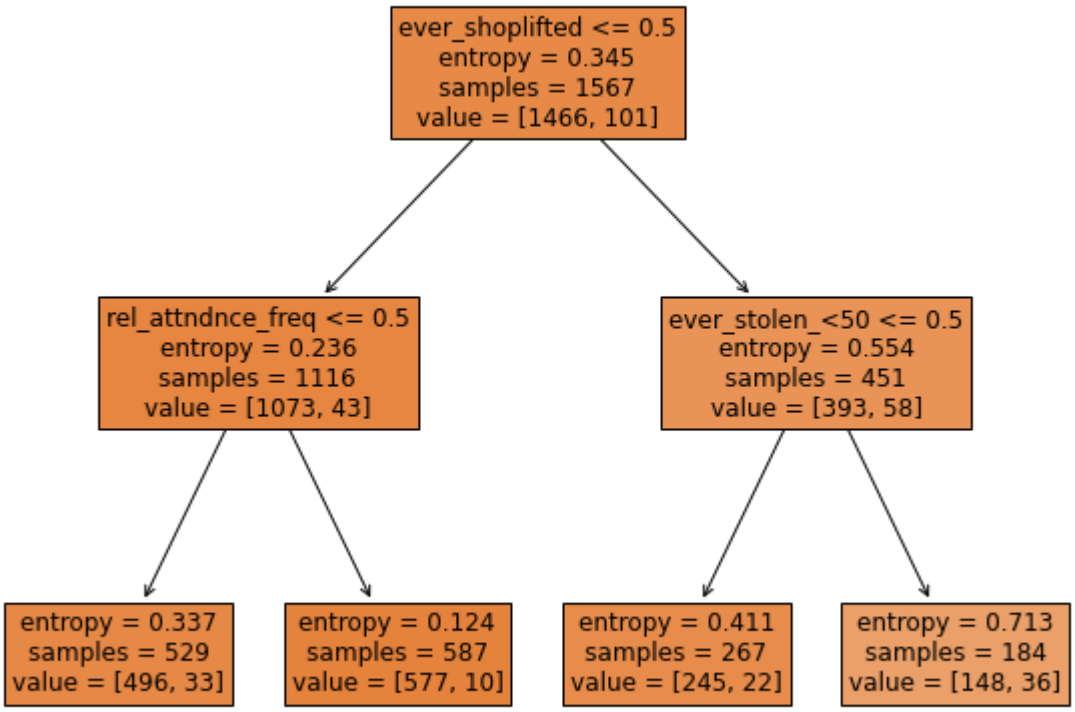
```
                                    index=X_train.columns
                                    ).nlargest(5).index.tolist()

gbt2_feat_importances = pd.Series(gbt_model_2.feature_importances_,
                                    index=X_train.columns
                                    ).nlargest(5).index.tolist()


# Create table of important features, accuracy and recall scores
scores = {'Model': ['Decision Tree', 'Random Forest', 'Gradient Boosted Trees'],
          'Accuracy': [dt_acc_2, rf_acc_2, gbt_acc_2],
          'Recall': [dt_recall_2, rf_recall_2, gbt_recall_2],
          'Top 5 Features': [dt2_feat_importances, rf2_feat_importances,
                             gbt2_feat_importances]}
scores = pd.DataFrame(scores)
pd.set_option('display.max_colwidth', None)
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
scores
```

Out[ ]:

	Model	Accuracy	Recall	Top 5 Features
0	Decision Tree	0.938815	0.0	[ever_shoplifted, rel_attndnce_freq, ever_stolen_<50, mother_alive, age]
1	Random Forest	0.938815	0.0	[estimated_fam_income_this_year, education_father, education_mother, num_siblings, fam_size]
2	Gradient Boosted Trees	0.938815	0.0	[ever_shoplifted, ever_destructed_property, ever_stopped_by_police, ever_broke_in, ever_stolen_<50]

In [ ]:

```
# ================================ Target 2 ================================:
# plot tree:
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

plt.figure(figsize=(10,8))
plot_tree(dt_model_2, filled=True, fontsize=12, feature_names=df.columns)
plt.show()
```



Target 3: Has sold cannabis more than 10 times this year

In [ ]:

```
# ================================ Target 3 ================================:
target3 = target_sold_can_this_year_10.values

# sample split:
X_train, X_test, y_train, y_test = model_selection.train_test_split(
    df, target3, train_size=.75, test_size=.25,
    shuffle=True, random_state=3)

np.random.seed(3)

# ============================== tree tuning ==============================:
# Define the parameter grid to search
param_grid = {'max_depth': [2, 4, 6, 8, 10],
              'min_samples_split': [2, 4, 6, 8, 10, 12, 14],
              'min_samples_leaf': [1, 2, 3, 4, 5, 6],
              'criterion': ['gini', 'entropy']}

# Create a decision tree classifier object
dtc3 = DecisionTreeClassifier()

# Create a GridSearchCV object
grid_search = GridSearchCV(dtc3, param_grid, cv=5)

# Fit the GridSearchCV object to the data
grid_search.fit(X_train, y_train)

# Print the best parameters and best score
```

```
print("Best parameters: ", grid_search.best_params_)
print("Best score: ", grid_search.best_score_)
```

```
Best parameters:  {'criterion': 'gini', 'max_depth': 6, 'min_samples_leaf': 6, 'mi
n_samples_split': 14}
Best score:  0.9795791701430577
```

In [ ]:
```
# ================================ Target 3 ================================:
# ========================== random forest tuning ==========================:
np.random.seed(3)

# Define the parameter grid to search
param_grid = {'n_estimators': [100, 150, 200],
              'max_features': ['sqrt', 'log2']}

# Create a random forest classifier object
rfc3 = RandomForestClassifier()

# Create a GridSearchCV object
grid_search = GridSearchCV(rfc3, param_grid, cv=5)

# Fit the GridSearchCV object to the data
grid_search.fit(X_train, y_train.ravel())

# Print the best parameters and best score
print("Best parameters: ", grid_search.best_params_)
print("Best score: ", grid_search.best_score_)
```

```
Best parameters:  {'max_features': 'sqrt', 'n_estimators': 100}
Best score:  0.9789422274678984
```

In [ ]:
```
# ================================ Target 3 ================================:
# ========================== gradient-boost tuning ==========================:
np.random.seed(3)

# Define the parameter grid to search
param_grid = {'n_estimators': [100, 150, 200],
              'learning_rate': [0.001, 0.005, 0.01, 0.02],
              'max_features': ['sqrt', 'log2']}

# Create a gradient boosted classifier object
gbc3 = GradientBoostingClassifier()

# Create a GridSearchCV object
grid_search = GridSearchCV(gbc3, param_grid, cv=5)

# Fit the GridSearchCV object to the data
grid_search.fit(X_train, y_train.ravel())

# Print the best parameters and best score
print("Best parameters: ", grid_search.best_params_)
print("Best score: ", grid_search.best_score_)
```

```
Best parameters:  {'learning_rate': 0.001, 'max_features': 'sqrt', 'n_estimators':
100}
Best score:  0.9789422274678984
```

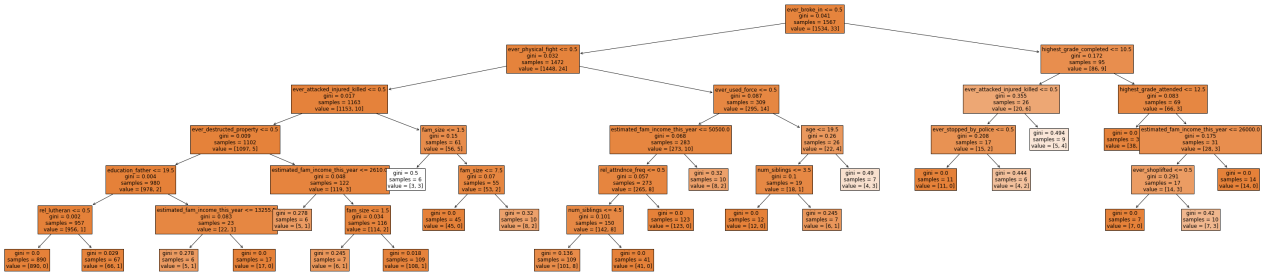
In [ ]:
```
# ================================ Target 3 ================================:

# Fit decision tree
```

```
dt_model_3 = DecisionTreeClassifier(criterion='gini', max_depth=6, random_state=3,
                                    min_samples_leaf=6, min_samples_split=14)
dt_model_3.fit(X_train, y_train.ravel())
dt_pred_3 = dt_model_3.predict(X_test)
dt_acc_3 = accuracy_score(y_test.ravel(), dt_pred_3)

# Fit random forest
rf_model_3 = RandomForestClassifier(max_features='sqrt', n_estimators=100,
                                    random_state=3)
rf_model_3.fit(X_train, y_train.ravel())
rf_pred_3 = rf_model_3.predict(X_test)
rf_acc_3 = accuracy_score(y_test.ravel(), rf_pred_3)

# Fit gradient boosted trees
gbt_model_3 = GradientBoostingClassifier(learning_rate=0.001, max_features='sqrt',
                                         n_estimators=100, random_state=3)
gbt_model_3.fit(X_train, y_train.ravel())
gbt_pred_3 = gbt_model_3.predict(X_test)
gbt_acc_3 = accuracy_score(y_test.ravel(), gbt_pred_3)

# Calculate accuracy and recall scores for each model
dt_acc_3 = accuracy_score(y_test, dt_pred_3)
rf_acc_3 = accuracy_score(y_test, rf_pred_3)
gbt_acc_3 = accuracy_score(y_test, gbt_pred_3)

dt_recall_3 = recall_score(y_test, dt_pred_3)
rf_recall_3 = recall_score(y_test, rf_pred_3)
gbt_recall_3 = recall_score(y_test, gbt_pred_3)

# Compute top 5 most important features for each model
dt3_feat_importances = pd.Series(dt_model_3.feature_importances_,
                                 index=X_train.columns
                                 ).nlargest(5).index.tolist()

rf3_feat_importances = pd.Series(rf_model_3.feature_importances_,
                                 index=X_train.columns
                                 ).nlargest(5).index.tolist()

gbt3_feat_importances = pd.Series(gbt_model_3.feature_importances_,
                                  index=X_train.columns
                                  ).nlargest(5).index.tolist()


# Create table of important features, accuracy and recall scores
scores = {'Model': ['Decision Tree', 'Random Forest', 'Gradient Boosted Trees'],
          'Accuracy': [dt_acc_3, rf_acc_3, gbt_acc_3],
          'Recall': [dt_recall_3, rf_recall_3, gbt_recall_3],
          'Top 5 Features': [dt2_feat_importances, rf3_feat_importances,
                             gbt3_feat_importances]}
scores = pd.DataFrame(scores)
pd.set_option('display.max_colwidth', None)
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
scores
```

Out[ ]:

	Model	Accuracy	Recall	Top 5 Features
0	Decision Tree	0.959847	0.0	[ever_shoplifted, rel_attndnce_freq, ever_stolen_<50, mother_alive, age]
1	Random Forest	0.959847	0.0	[estimated_fam_income_this_year, fam_size, education_father, education_mother, num_siblings]
2	Gradient Boosted Trees	0.959847	0.0	[ever_broke_in, ever_destructed_property, ever_physical_fight, estimated_fam_income_this_year, ever_attacked_injured_killed]

In [ ]:
```
# ================================== Target 3 ==================================:
# plot tree:
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

plt.figure(figsize=(54,12))
plot_tree(dt_model_3, filled=True, fontsize=12, feature_names=df.columns)
plt.show()
```



Target 4: Has sold hard drugs more than 10 times this year

In [ ]:
```
# ================================== Target 4 ==================================:
target4 = target_sold_hard_drugs_this_year_10.values

# sample split:
X_train, X_test, y_train, y_test = model_selection.train_test_split(
    df, target4, train_size=.75, test_size=.25,
    shuffle=True, random_state=4)

np.random.seed(4)

# ============================== tree tuning ==============================:
# Define the parameter grid to search
param_grid = {'max_depth': [2, 4, 6, 8, 10, 12, 14],
              'min_samples_split': [2, 4, 6, 8, 10],
              'min_samples_leaf': [1, 2, 3, 4, 5],
              'criterion': ['gini', 'entropy']}

# Create a decision tree classifier object
dtc4 = DecisionTreeClassifier()

# Create a GridSearchCV object
grid_search = GridSearchCV(dtc4, param_grid, cv=5)

# Fit the GridSearchCV object to the data
```

```
grid_search.fit(X_train, y_train)

# Print the best parameters and best score
print("Best parameters: ", grid_search.best_params_)
print("Best score: ", grid_search.best_score_)
```

```
Best parameters:  {'criterion': 'gini', 'max_depth': 2, 'min_samples_leaf': 1, 'mi
n_samples_split': 2}
Best score:  0.9948963187562321
```

In [ ]:
```
# ================================ Target 4 ================================:
# ========================== random forest tuning ==========================:
np.random.seed(4)

# Define the parameter grid to search
param_grid = {'n_estimators': [100, 150, 200, 250, 300],
              'max_features': ['sqrt', 'log2']}

# Create a random forest classifier object
rfc4 = RandomForestClassifier()

# Create a GridSearchCV object
grid_search = GridSearchCV(rfc4, param_grid, cv=5)

# Fit the GridSearchCV object to the data
grid_search.fit(X_train, y_train.ravel())

# Print the best parameters and best score
print("Best parameters: ", grid_search.best_params_)
print("Best score: ", grid_search.best_score_)
```

```
Best parameters:  {'max_features': 'sqrt', 'n_estimators': 100}
Best score:  0.9948963187562321
```

In [ ]:
```
# ================================ Target 4 ================================:
# ========================== gradient-boost tuning ==========================:
np.random.seed(4)

# Define the parameter grid to search
param_grid = {'n_estimators': [50, 100, 150, 200],
              'learning_rate': [0.01, 0.02, 0.03, 0.04, 0.05, 0.1, 0.2, 0.5],
              'max_features': ['sqrt', 'log2']}

# Create a gradient boosted classifier object
gbc4 = GradientBoostingClassifier()

# Create a GridSearchCV object
grid_search = GridSearchCV(gbc4, param_grid, cv=5)

# Fit the GridSearchCV object to the data
grid_search.fit(X_train, y_train.ravel())

# Print the best parameters and best score
print("Best parameters: ", grid_search.best_params_)
print("Best score: ", grid_search.best_score_)
```

```
Best parameters:  {'learning_rate': 0.01, 'max_features': 'sqrt', 'n_estimators':
50}
Best score:  0.9948963187562321
```

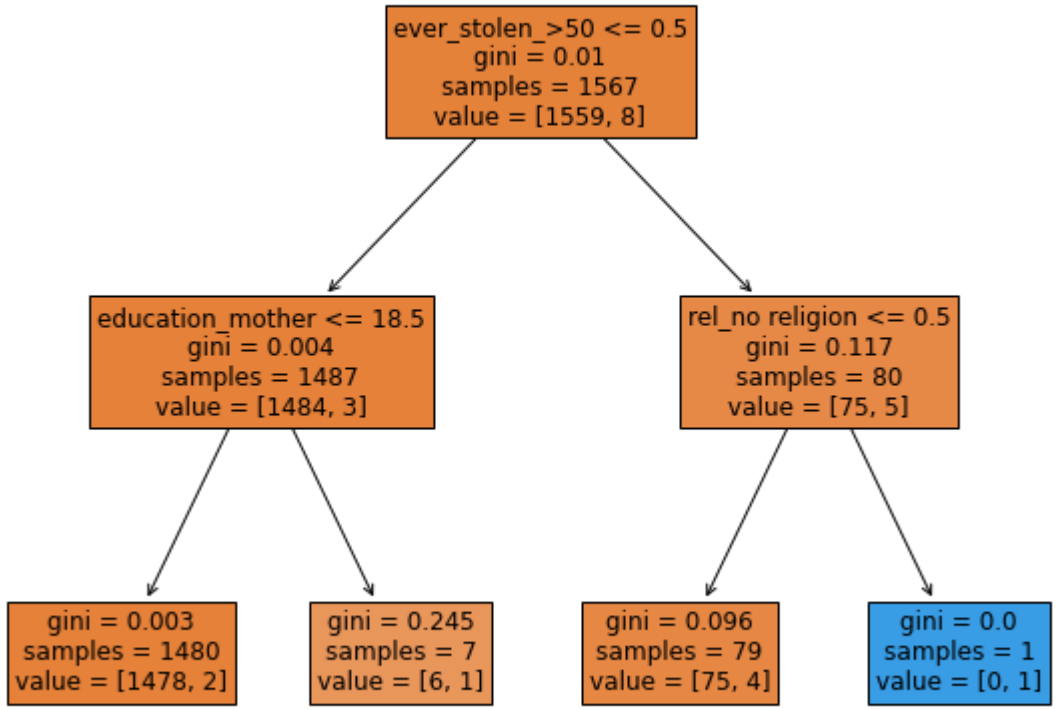
In [ ]:

```
# ================================= Target 4 ================================:

# Fit decision tree
dt_model_4 = DecisionTreeClassifier(criterion='gini', max_depth=2, random_state=4,
                                    min_samples_leaf=1, min_samples_split=2)
dt_model_4.fit(X_train, y_train.ravel())
dt_pred_4 = dt_model_4.predict(X_test)
dt_acc_4 = accuracy_score(y_test.ravel(), dt_pred_4)

# Fit random forest
rf_model_4 = RandomForestClassifier(max_features='sqrt', n_estimators=100,
                                    random_state=4)
rf_model_4.fit(X_train, y_train.ravel())
rf_pred_4 = rf_model_4.predict(X_test)
rf_acc_4 = accuracy_score(y_test.ravel(), rf_pred_4)

# Fit gradient boosted trees
gbt_model_4 = GradientBoostingClassifier(learning_rate=0.01, max_features='sqrt',
                                         n_estimators= 50, random_state=4)
gbt_model_4.fit(X_train, y_train.ravel())
gbt_pred_4 = gbt_model_4.predict(X_test)
gbt_acc_4 = accuracy_score(y_test.ravel(), gbt_pred_4)

# Calculate accuracy and recall scores for each model
dt_acc_4 = accuracy_score(y_test, dt_pred_4)
rf_acc_4 = accuracy_score(y_test, rf_pred_4)
gbt_acc_4 = accuracy_score(y_test, gbt_pred_4)

dt_recall_4 = recall_score(y_test, dt_pred_4)
rf_recall_4 = recall_score(y_test, rf_pred_4)
gbt_recall_4 = recall_score(y_test, gbt_pred_4)

# Compute top 5 most important features for each model
dt4_feat_importances = pd.Series(dt_model_4.feature_importances_,
                                 index=X_train.columns
                                 ).nlargest(5).index.tolist()

rf4_feat_importances = pd.Series(rf_model_4.feature_importances_,
                                 index=X_train.columns
                                 ).nlargest(5).index.tolist()

gbt4_feat_importances = pd.Series(gbt_model_4.feature_importances_,
                                  index=X_train.columns
                                  ).nlargest(5).index.tolist()


# Create table of important features, accuracy and recall scores
scores = {'Model': ['Decision Tree', 'Random Forest', 'Gradient Boosted Trees'],
          'Accuracy': [dt_acc_4, rf_acc_4, gbt_acc_4],
          'Recall': [dt_recall_4, rf_recall_4, gbt_recall_4],
          'Top 5 Features': [dt4_feat_importances, rf4_feat_importances,
                             gbt4_feat_importances]}
scores = pd.DataFrame(scores)
pd.set_option('display.max_colwidth', None)
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
scores
```

Out[ ]:

	Model	Accuracy	Recall	Top 5 Features
0	Decision Tree	0.988528	0.0	[rel_no religion, ever_stolen_>50, education_mother, mother_alive, age]
1	Random Forest	0.992352	0.0	[estimated_fam_income_this_year, education_father, education_mother, num_siblings, fam_size]
2	Gradient Boosted Trees	0.992352	0.0	[ever_stolen_>50, education_mother, fam_size, estimated_fam_income_this_year, ever_used_force]

In [ ]:
```
# ================================ Target 4 ================================:
# plot tree:
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

plt.figure(figsize=(10,8))
plot_tree(dt_model_4, filled=True, fontsize=12, feature_names=df.columns)
plt.show()
```



Target 5: Has had 6 drinks or more at once, more than 3 times over the last month

In [ ]:
```
# ================================ Target 5 ================================:
target5 = target_had_6_drinks_3.values

# sample split:
X_train, X_test, y_train, y_test = model_selection.train_test_split(
```

```
    df, target5, train_size=.75, test_size=.25,
    shuffle=True, random_state=5)

np.random.seed(4)

# ============================== tree tuning ==============================:
# Define the parameter grid to search
param_grid = {'max_depth': [2, 4, 6, 8, 10, 12, 14, ],
              'min_samples_split': [2, 4, 6, 8, 10, 12, 14, 16, 18, 20],
              'min_samples_leaf': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
              'criterion': ['gini', 'entropy']}

# Create a decision tree classifier object
dtc5 = DecisionTreeClassifier()

# Create a GridSearchCV object
grid_search = GridSearchCV(dtc5, param_grid, cv=5)

# Fit the GridSearchCV object to the data
grid_search.fit(X_train, y_train)

# Print the best parameters and best score
print("Best parameters: ", grid_search.best_params_)
print("Best score: ", grid_search.best_score_)
```

```
Best parameters:  {'criterion': 'gini', 'max_depth': 2, 'min_samples_leaf': 1, 'mi
n_samples_split': 2}
Best score:  0.7785596548706782
```

In [ ]:
```
# ================================ Target 5 ================================:
# ========================== random forest tuning ==========================:
np.random.seed(5)

# Define the parameter grid to search
param_grid = {'n_estimators': [50, 100, 150, 200, 250, 300],
              'max_features': ['sqrt', 'log2']}

# Create a random forest classifier object
rfc5 = RandomForestClassifier()

# Create a GridSearchCV object
grid_search = GridSearchCV(rfc5, param_grid, cv=5)

# Fit the GridSearchCV object to the data
grid_search.fit(X_train, y_train.ravel())

# Print the best parameters and best score
print("Best parameters: ", grid_search.best_params_)
print("Best score: ", grid_search.best_score_)
```

```
Best parameters:  {'max_features': 'sqrt', 'n_estimators': 150}
Best score:  0.7728271707942451
```

In [ ]:
```
# ================================ Target 5 ================================:
# ========================== gradient-boost tuning ==========================:
np.random.seed(5)

# Define the parameter grid to search
param_grid = {'n_estimators': [200, 300, 400, 500],
```

```
              'learning_rate': [0.01, 0.02, 0.03, 0.04, 0.05],
              'max_features': ['sqrt', 'log2']}

# Create a gradient boosted classifier object
gbc5 = GradientBoostingClassifier()

# Create a GridSearchCV object
grid_search = GridSearchCV(gbc5, param_grid, cv=5)

# Fit the GridSearchCV object to the data
grid_search.fit(X_train, y_train.ravel())

# Print the best parameters and best score
print("Best parameters: ", grid_search.best_params_)
print("Best score: ", grid_search.best_score_)
```

```
Best parameters:  {'learning_rate': 0.03, 'max_features': 'log2', 'n_estimators':
400}
Best score:  0.7798457499847378
```

In [ ]:
```
# ================================ Target 5 ================================:

# Fit decision tree
dt_model_5 = DecisionTreeClassifier(criterion='gini', max_depth=2, random_state=5,
                                    min_samples_leaf=1, min_samples_split=2)
dt_model_5.fit(X_train, y_train.ravel())
dt_pred_5 = dt_model_5.predict(X_test)
dt_acc_5 = accuracy_score(y_test.ravel(), dt_pred_5)

# Fit random forest
rf_model_5 = RandomForestClassifier(max_features='sqrt', n_estimators=150,
                                    random_state=5)
rf_model_5.fit(X_train, y_train.ravel())
rf_pred_5 = rf_model_5.predict(X_test)
rf_acc_5 = accuracy_score(y_test.ravel(), rf_pred_5)

# Fit gradient boosted trees
gbt_model_5 = GradientBoostingClassifier(learning_rate=0.03, max_features='log2',
                                         n_estimators=400, random_state=5)
gbt_model_5.fit(X_train, y_train.ravel())
gbt_pred_5 = gbt_model_5.predict(X_test)
gbt_acc_5 = accuracy_score(y_test.ravel(), gbt_pred_5)

# Calculate accuracy and recall scores for each model
dt_acc_5 = accuracy_score(y_test, dt_pred_5)
rf_acc_5 = accuracy_score(y_test, rf_pred_5)
gbt_acc_5 = accuracy_score(y_test, gbt_pred_5)

dt_recall_5 = recall_score(y_test, dt_pred_5)
rf_recall_5 = recall_score(y_test, rf_pred_5)
gbt_recall_5 = recall_score(y_test, gbt_pred_5)

# Compute top 5 most important features for each model
dt5_feat_importances = pd.Series(dt_model_5.feature_importances_,
                                 index=X_train.columns
                                 ).nlargest(5).index.tolist()

rf5_feat_importances = pd.Series(rf_model_5.feature_importances_,
                                 index=X_train.columns
```

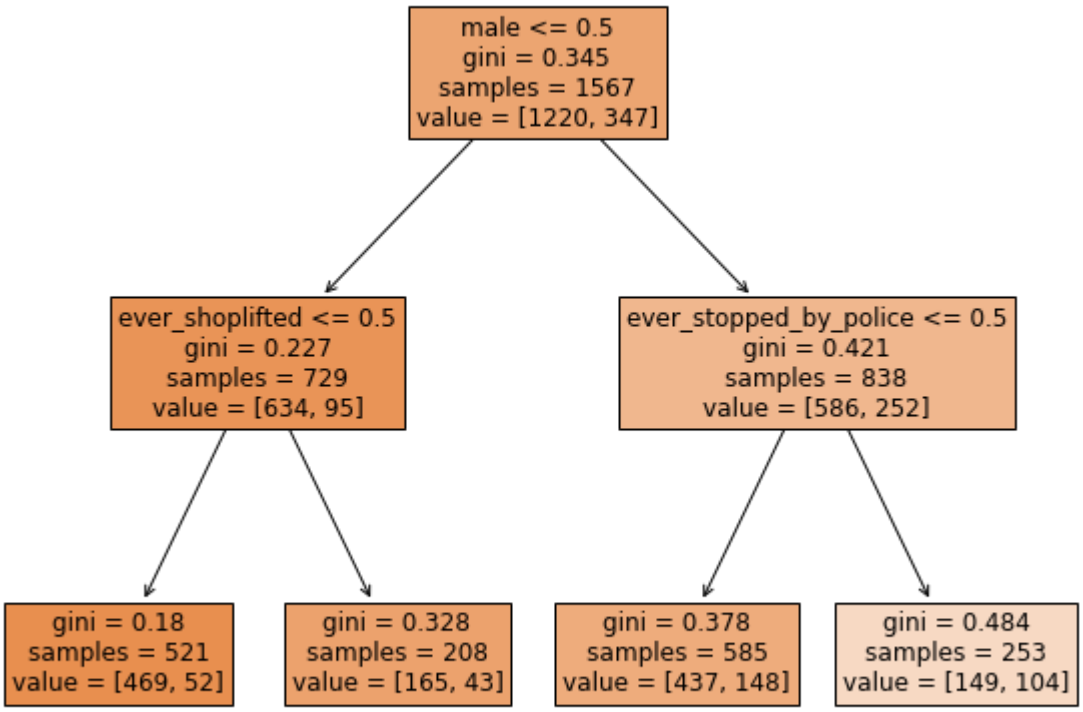
```
                                ).nlargest(5).index.tolist()

gbt5_feat_importances = pd.Series(gbt_model_5.feature_importances_,
                                  index=X_train.columns
                                  ).nlargest(5).index.tolist()


# Create table of important features, accuracy and recall scores
scores = {'Model': ['Decision Tree', 'Random Forest', 'Gradient Boosted Trees'],
          'Accuracy': [dt_acc_5, rf_acc_5, gbt_acc_5],
          'Recall': [dt_recall_5, rf_recall_5, gbt_recall_5],
          'Top 5 Features': [dt5_feat_importances, rf5_feat_importances,
                             gbt5_feat_importances]}
scores = pd.DataFrame(scores)
pd.set_option('display.max_colwidth', None)
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
scores
```

Out[ ]:

	Model	Accuracy	Recall	Top 5 Features
0	Decision Tree	0.762906	0.000000	[male, ever_stopped_by_police, ever_shoplifted, mother_alive, age]
1	Random Forest	0.739962	0.032258	[estimated_fam_income_this_year, education_father, fam_size, num_siblings, education_mother]
2	Gradient Boosted Trees	0.768642	0.096774	[male, estimated_fam_income_this_year, ever_stopped_by_police, education_father, ever_destructed_property]

In [ ]:

```
# ================================= Target 5 ================================:
# plot tree:
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

plt.figure(figsize=(10,8))
plot_tree(dt_model_5, filled=True, fontsize=12, feature_names=df.columns)
plt.show()
```



Target 6: Has ever done cocaine

In [ ]:

```
# ================================ Target 6 ================================:
target6 = target_ever_cocaine.values

# sample split:
X_train, X_test, y_train, y_test = model_selection.train_test_split(
    df, target6, train_size=.75, test_size=.25,
    shuffle=True, random_state=6)

np.random.seed(4)

# ============================== tree tuning ==============================:
# Define the parameter grid to search
param_grid = {'max_depth': [2, 4, 6, 8, 10, 12, 14, 16, 18, 20],
              'min_samples_split': [2, 4, 6, 8, 10, 12, 14],
              'min_samples_leaf': [1, 2, 3, 4, 5, 6],
              'criterion': ['gini', 'entropy']}

# Create a decision tree classifier object
dtc6 = DecisionTreeClassifier()

# Create a GridSearchCV object
grid_search = GridSearchCV(dtc6, param_grid, cv=5)

# Fit the GridSearchCV object to the data
grid_search.fit(X_train, y_train)

# Print the best parameters and best score
```

```
print("Best parameters: ", grid_search.best_params_)
print("Best score: ", grid_search.best_score_)
```

```
Best parameters:  {'criterion': 'entropy', 'max_depth': 2, 'min_samples_leaf': 1,
'min_samples_split': 2}
Best score:  0.7906839502655625
```

In [ ]:
```
# ================================ Target 6 ================================:
# ========================== random forest tuning ==========================:
np.random.seed(6)

# Define the parameter grid to search
param_grid = {'n_estimators': [50, 100, 150, 200, 250, 300],
              'max_features': ['sqrt', 'log2']}

# Create a random forest classifier object
rfc6 = RandomForestClassifier()

# Create a GridSearchCV object
grid_search = GridSearchCV(rfc6, param_grid, cv=5)

# Fit the GridSearchCV object to the data
grid_search.fit(X_train, y_train.ravel())

# Print the best parameters and best score
print("Best parameters: ", grid_search.best_params_)
print("Best score: ", grid_search.best_score_)
```

```
Best parameters:  {'max_features': 'sqrt', 'n_estimators': 250}
Best score:  0.7983455770130847
```

In [ ]:
```
# ================================ Target 6 ================================:
# ========================== gradient-boost tuning ==========================:
np.random.seed(6)

# Define the parameter grid to search
param_grid = {'n_estimators': [10, 20, 30, 40, 50, 100, 200, 300, 400],
              'learning_rate': [0.01, 0.02, 0.03, 0.04, 0.05],
              'max_features': ['sqrt', 'log2']}

# Create a gradient boosted classifier object
gbc6 = GradientBoostingClassifier()

# Create a GridSearchCV object
grid_search = GridSearchCV(gbc6, param_grid, cv=5)

# Fit the GridSearchCV object to the data
grid_search.fit(X_train, y_train.ravel())

# Print the best parameters and best score
print("Best parameters: ", grid_search.best_params_)
print("Best score: ", grid_search.best_score_)
```

```
Best parameters:  {'learning_rate': 0.05, 'max_features': 'log2', 'n_estimators':
100}
Best score:  0.8028163855029404
```

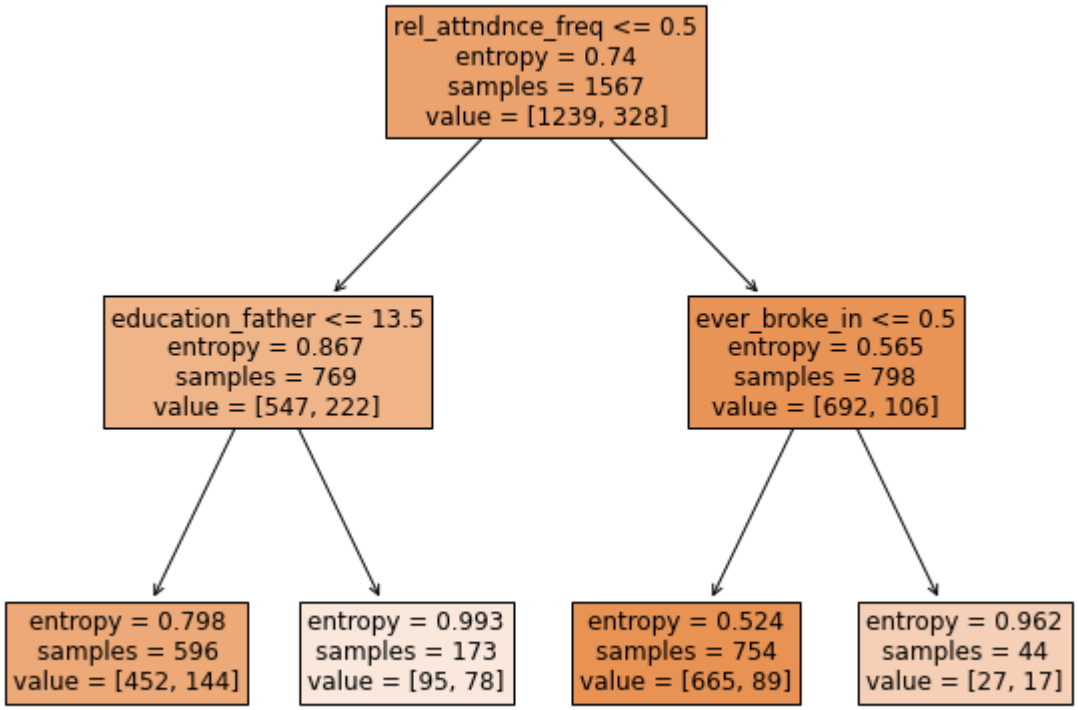
In [ ]:
```
# ================================ Target 6 ================================:

# Fit decision tree
```

```
dt_model_6 = DecisionTreeClassifier(criterion='entropy', max_depth=2, random_state
                                    min_samples_leaf=1, min_samples_split=2)
dt_model_6.fit(X_train, y_train.ravel())
dt_pred_6 = dt_model_6.predict(X_test)
dt_acc_6 = accuracy_score(y_test.ravel(), dt_pred_6)

# Fit random forest
rf_model_6 = RandomForestClassifier(max_features='sqrt', n_estimators=250,
                                    random_state=6)
rf_model_6.fit(X_train, y_train.ravel())
rf_pred_6 = rf_model_6.predict(X_test)
rf_acc_6 = accuracy_score(y_test.ravel(), rf_pred_6)

# Fit gradient boosted trees
gbt_model_6 = GradientBoostingClassifier(learning_rate=0.05, max_features='log2',
                                         n_estimators=100, random_state=6)
gbt_model_6.fit(X_train, y_train.ravel())
gbt_pred_6 = gbt_model_6.predict(X_test)
gbt_acc_6 = accuracy_score(y_test.ravel(), gbt_pred_6)

# Calculate accuracy and recall scores for each model
dt_acc_6 = accuracy_score(y_test, dt_pred_6)
rf_acc_6 = accuracy_score(y_test, rf_pred_6)
gbt_acc_6 = accuracy_score(y_test, gbt_pred_6)

dt_recall_6 = recall_score(y_test, dt_pred_6)
rf_recall_6 = recall_score(y_test, rf_pred_6)
gbt_recall_6 = recall_score(y_test, gbt_pred_6)

# Compute top 5 most important features for each model
dt6_feat_importances = pd.Series(dt_model_6.feature_importances_,
                                 index=X_train.columns
                                 ).nlargest(5).index.tolist()

rf6_feat_importances = pd.Series(rf_model_6.feature_importances_,
                                 index=X_train.columns
                                 ).nlargest(5).index.tolist()

gbt6_feat_importances = pd.Series(gbt_model_6.feature_importances_,
                                  index=X_train.columns
                                  ).nlargest(5).index.tolist()


# Create table of important features, accuracy and recall scores
scores = {'Model': ['Decision Tree', 'Random Forest', 'Gradient Boosted Trees'],
          'Accuracy': [dt_acc_6, rf_acc_6, gbt_acc_6],
          'Recall': [dt_recall_6, rf_recall_6, gbt_recall_6],
          'Top 5 Features': [dt6_feat_importances, rf6_feat_importances,
                             gbt6_feat_importances]}
scores = pd.DataFrame(scores)
pd.set_option('display.max_colwidth', None)
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
scores
```

Out[ ]:

	Model	Accuracy	Recall	Top 5 Features
0	Decision Tree	0.774379	0.000000	[rel_attndnce_freq, education_father, ever_broke_in, mother_alive, age]
1	Random Forest	0.787763	0.084746	[estimated_fam_income_this_year, education_father, num_siblings, education_mother, fam_size]
2	Gradient Boosted Trees	0.778203	0.050847	[rel_attndnce_freq, education_father, estimated_fam_income_this_year, ever_stolen_<50, ever_broke_in]

In [ ]:
```
# ================================= Target 6 ================================:
# plot tree:
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

plt.figure(figsize=(10,8))
plot_tree(dt_model_6, filled=True, fontsize=12, feature_names=df.columns)
plt.show()
```



Target 7: Has ever done other narcotics

In [ ]:
```
# ================================= Target 7 ================================:
target7 = target_ever_other_narcotics.values

# sample split:
X_train, X_test, y_train, y_test = model_selection.train_test_split(
```

```
    df, target7, train_size=.75, test_size=.25,
    shuffle=True, random_state=7)

np.random.seed(7)

# ============================== tree tuning ==============================:
# Define the parameter grid to search
param_grid = {'max_depth': [2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30
              'min_samples_split': [2, 4, 6, 8, 10, 12, 14],
              'min_samples_leaf': [1, 2, 3, 4, 5, 6],
              'criterion': ['gini', 'entropy']}

# Create a decision tree classifier object
dtc7 = DecisionTreeClassifier()

# Create a GridSearchCV object
grid_search = GridSearchCV(dtc7, param_grid, cv=5)

# Fit the GridSearchCV object to the data
grid_search.fit(X_train, y_train)

# Print the best parameters and best score
print("Best parameters: ", grid_search.best_params_)
print("Best score: ", grid_search.best_score_)
```

```
Best parameters:  {'criterion': 'gini', 'max_depth': 2, 'min_samples_leaf': 3, 'mi
n_samples_split': 2}
Best score:  0.94192832868684
```

In [ ]:
```
# ================================ Target 7 ================================:
# ========================== random forest tuning ==========================:
np.random.seed(7)

# Define the parameter grid to search
param_grid = {'n_estimators': [50, 100, 150, 200, 250, 300],
              'max_features': ['sqrt', 'log2']}

# Create a random forest classifier object
rfc7 = RandomForestClassifier()

# Create a GridSearchCV object
grid_search = GridSearchCV(rfc7, param_grid, cv=5)

# Fit the GridSearchCV object to the data
grid_search.fit(X_train, y_train.ravel())

# Print the best parameters and best score
print("Best parameters: ", grid_search.best_params_)
print("Best score: ", grid_search.best_score_)
```

```
Best parameters:  {'max_features': 'sqrt', 'n_estimators': 50}
Best score:  0.94192832868684
```

In [ ]:
```
# ================================ Target 7 ================================:
# ========================== gradient-boost tuning ==========================:
np.random.seed(7)

# Define the parameter grid to search
param_grid = {'n_estimators': [10, 20, 30, 40, 50, 100, 200, 300, 400],
```

```
              'learning_rate': [0.01, 0.02, 0.03, 0.04, 0.05, 0.1, 0.2],
              'max_features': ['sqrt', 'log2']}

# Create a gradient boosted classifier object
gbc7 = GradientBoostingClassifier()

# Create a GridSearchCV object
grid_search = GridSearchCV(gbc7, param_grid, cv=5)

# Fit the GridSearchCV object to the data
grid_search.fit(X_train, y_train.ravel())

# Print the best parameters and best score
print("Best parameters: ", grid_search.best_params_)
print("Best score: ", grid_search.best_score_)
```

```
Best parameters:  {'learning_rate': 0.1, 'max_features': 'sqrt', 'n_estimators': 3
0}
Best score:  0.9425673063226225
```

In [ ]:
```
# ================================ Target 7 ================================:

# Fit decision tree
dt_model_7 = DecisionTreeClassifier(criterion='gini', max_depth=2, random_state=7,
                                    min_samples_leaf=3, min_samples_split=2)
dt_model_7.fit(X_train, y_train.ravel())
dt_pred_7 = dt_model_7.predict(X_test)
dt_acc_7 = accuracy_score(y_test.ravel(), dt_pred_7)

# Fit random forest
rf_model_7 = RandomForestClassifier(max_features='sqrt', n_estimators=50,
                                    random_state=7)
rf_model_7.fit(X_train, y_train.ravel())
rf_pred_7 = rf_model_7.predict(X_test)
rf_acc_7 = accuracy_score(y_test.ravel(), rf_pred_7)

# Fit gradient boosted trees
gbt_model_7 = GradientBoostingClassifier(learning_rate=0.1, max_features='sqrt',
                                         n_estimators=30, random_state=7)
gbt_model_7.fit(X_train, y_train.ravel())
gbt_pred_7 = gbt_model_7.predict(X_test)
gbt_acc_7 = accuracy_score(y_test.ravel(), gbt_pred_7)

# Calculate accuracy and recall scores for each model
dt_acc_7 = accuracy_score(y_test, dt_pred_7)
rf_acc_7 = accuracy_score(y_test, rf_pred_7)
gbt_acc_7 = accuracy_score(y_test, gbt_pred_7)

dt_recall_7 = recall_score(y_test, dt_pred_7)
rf_recall_7 = recall_score(y_test, rf_pred_7)
gbt_recall_7 = recall_score(y_test, gbt_pred_7)

# Compute top 5 most important features for each model
dt7_feat_importances = pd.Series(dt_model_7.feature_importances_,
                                 index=X_train.columns
                                 ).nlargest(5).index.tolist()

rf7_feat_importances = pd.Series(rf_model_7.feature_importances_,
                                 index=X_train.columns
```

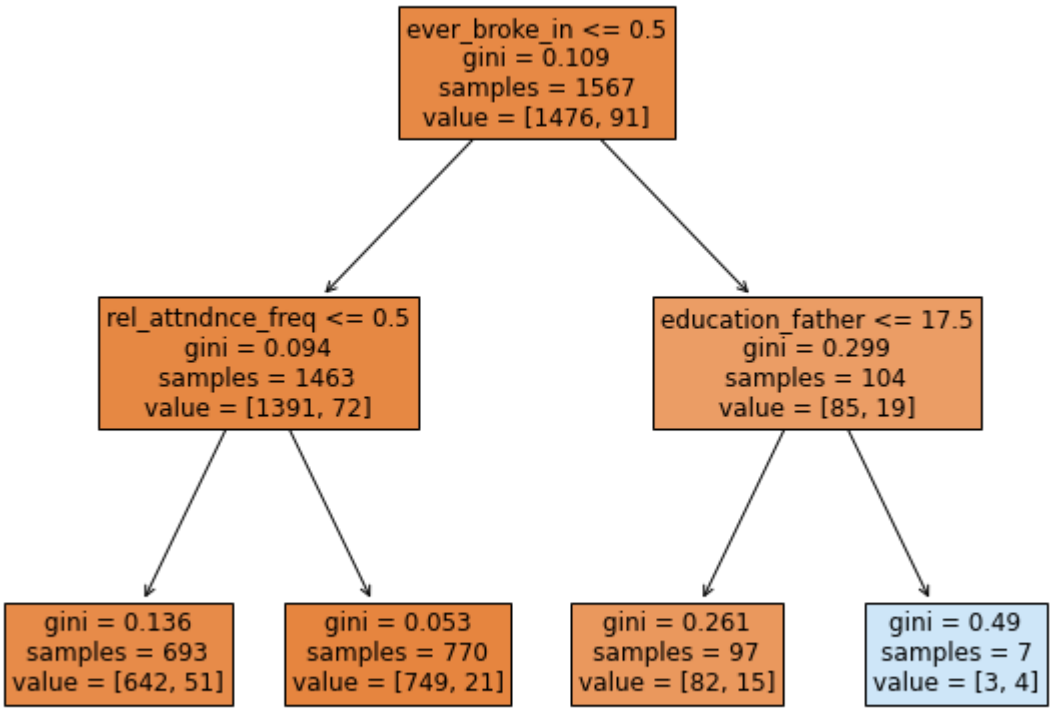
```
                                  ).nlargest(5).index.tolist()

gbt7_feat_importances = pd.Series(gbt_model_7.feature_importances_,
                                  index=X_train.columns
                                  ).nlargest(5).index.tolist()


# Create table of important features, accuracy and recall scores
scores = {'Model': ['Decision Tree', 'Random Forest', 'Gradient Boosted Trees'],
          'Accuracy': [dt_acc_7, rf_acc_7, gbt_acc_7],
          'Recall': [dt_recall_7, rf_recall_7, gbt_recall_7],
          'Top 5 Features': [dt7_feat_importances, rf7_feat_importances,
                             gbt7_feat_importances]}
scores = pd.DataFrame(scores)
pd.set_option('display.max_colwidth', None)
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
scores
```

Out[ ]:

	Model	Accuracy	Recall	Top 5 Features
0	Decision Tree	0.931166	0.029412	[ever_broke_in, education_father, rel_attndnce_freq, mother_alive, age]
1	Random Forest	0.934990	0.000000	[estimated_fam_income_this_year, education_father, education_mother, fam_size, num_siblings]
2	Gradient Boosted Trees	0.934990	0.000000	[ever_broke_in, rel_attndnce_freq, education_mother, highest_grade_attended, education_father]

In [ ]:

```
# ================================ Target 7 ================================:
# plot tree:
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

plt.figure(figsize=(10,8))
plot_tree(dt_model_7, filled=True, fontsize=12, feature_names=df.columns)
plt.show()
```



Target 8: Has ever done other drugs

In [ ]:

```
# ================================ Target 8 ================================:
target8 = target_ever_other_drugs.values

# sample split:
X_train, X_test, y_train, y_test = model_selection.train_test_split(
    df, target8, train_size=.75, test_size=.25,
    shuffle=True, random_state=8)

np.random.seed(4)

# ============================== tree tuning ==============================:
# Define the parameter grid to search
param_grid = {'max_depth': [2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30
              'min_samples_split': [2, 4, 6, 8, 10, 12, 14],
              'min_samples_leaf': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
              'criterion': ['gini', 'entropy']}

# Create a decision tree classifier object
dtc8 = DecisionTreeClassifier()

# Create a GridSearchCV object
grid_search = GridSearchCV(dtc8, param_grid, cv=5)

# Fit the GridSearchCV object to the data
grid_search.fit(X_train, y_train)

# Print the best parameters and best score
```

```
print("Best parameters: ", grid_search.best_params_)
print("Best score: ", grid_search.best_score_)
```

```
Best parameters:  {'criterion': 'gini', 'max_depth': 2, 'min_samples_leaf': 1, 'mi
n_samples_split': 2}
Best score:  0.9910665228627826
```

In [ ]:
```
# ================================ Target 8 ================================:
# ========================== random forest tuning ==========================:
np.random.seed(8)

# Define the parameter grid to search
param_grid = {'n_estimators': [50, 100, 150, 200, 250, 300],
              'max_features': ['sqrt', 'log2']}

# Create a random forest classifier object
rfc8 = RandomForestClassifier()

# Create a GridSearchCV object
grid_search = GridSearchCV(rfc8, param_grid, cv=5)

# Fit the GridSearchCV object to the data
grid_search.fit(X_train, y_train.ravel())

# Print the best parameters and best score
print("Best parameters: ", grid_search.best_params_)
print("Best score: ", grid_search.best_score_)
```

```
Best parameters:  {'max_features': 'sqrt', 'n_estimators': 50}
Best score:  0.9904275452269997
```

In [ ]:
```
# ================================ Target 8 ================================:
# ========================== gradient-boost tuning ==========================:
np.random.seed(8)

# Define the parameter grid to search
param_grid = {'n_estimators': [10, 20, 30, 40, 50, 100, 200, 300, 400],
              'learning_rate': [0.01, 0.02, 0.03, 0.04, 0.05, 0.1, 0.2],
              'max_features': ['sqrt', 'log2']}

# Create a gradient boosted classifier object
gbc8 = GradientBoostingClassifier()

# Create a GridSearchCV object
grid_search = GridSearchCV(gbc8, param_grid, cv=5)

# Fit the GridSearchCV object to the data
grid_search.fit(X_train, y_train.ravel())

# Print the best parameters and best score
print("Best parameters: ", grid_search.best_params_)
print("Best score: ", grid_search.best_score_)
```

```
Best parameters:  {'learning_rate': 0.03, 'max_features': 'sqrt', 'n_estimators':
400}
Best score:  0.9910665228627826
```

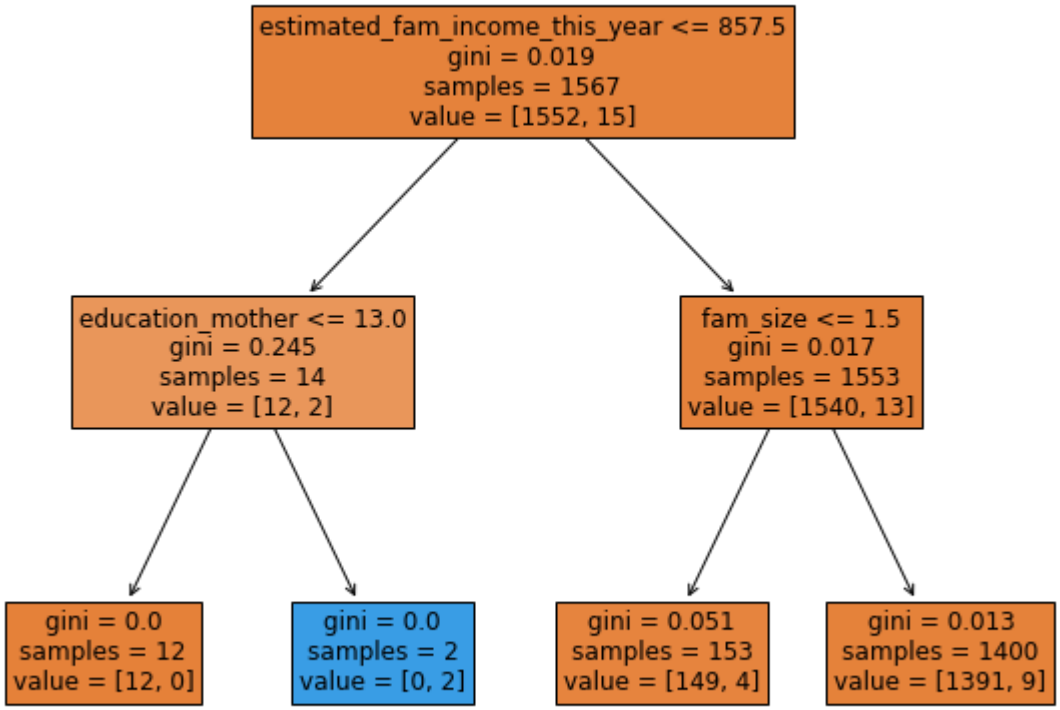
In [ ]:
```
# ================================ Target 8 ================================:

# Fit decision tree
```

```
dt_model_8 = DecisionTreeClassifier(criterion='gini', max_depth=2, random_state=8,
                                    min_samples_leaf=1, min_samples_split=2)
dt_model_8.fit(X_train, y_train.ravel())
dt_pred_8 = dt_model_8.predict(X_test)
dt_acc_8 = accuracy_score(y_test.ravel(), dt_pred_8)

# Fit random forest
rf_model_8 = RandomForestClassifier(max_features='sqrt', n_estimators=50,
                                    random_state=8)
rf_model_8.fit(X_train, y_train.ravel())
rf_pred_8 = rf_model_8.predict(X_test)
rf_acc_8 = accuracy_score(y_test.ravel(), rf_pred_8)

# Fit gradient boosted trees
gbt_model_8 = GradientBoostingClassifier(learning_rate=0.03, max_features='sqrt',
                                         n_estimators=400, random_state=8)
gbt_model_8.fit(X_train, y_train.ravel())
gbt_pred_8 = gbt_model_8.predict(X_test)
gbt_acc_8 = accuracy_score(y_test.ravel(), gbt_pred_8)

# Calculate accuracy and recall scores for each model
dt_acc_8 = accuracy_score(y_test, dt_pred_8)
rf_acc_8 = accuracy_score(y_test, rf_pred_8)
gbt_acc_8 = accuracy_score(y_test, gbt_pred_8)

dt_recall_8 = recall_score(y_test, dt_pred_8)
rf_recall_8 = recall_score(y_test, rf_pred_8)
gbt_recall_8 = recall_score(y_test, gbt_pred_8)

# Compute top 5 most important features for each model
dt8_feat_importances = pd.Series(dt_model_8.feature_importances_,
                                 index=X_train.columns
                                 ).nlargest(5).index.tolist()

rf8_feat_importances = pd.Series(rf_model_8.feature_importances_,
                                 index=X_train.columns
                                 ).nlargest(5).index.tolist()

gbt8_feat_importances = pd.Series(gbt_model_8.feature_importances_,
                                  index=X_train.columns
                                  ).nlargest(5).index.tolist()


# Create table of important features, accuracy and recall scores
scores = {'Model': ['Decision Tree', 'Random Forest', 'Gradient Boosted Trees'],
          'Accuracy': [dt_acc_8, rf_acc_8, gbt_acc_8],
          'Recall': [dt_recall_8, rf_recall_8, gbt_recall_8],
          'Top 5 Features': [dt8_feat_importances, rf8_feat_importances,
                             gbt8_feat_importances]}
scores = pd.DataFrame(scores)
pd.set_option('display.max_colwidth', None)
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
scores
```

Out[ ]:

	Model	Accuracy	Recall	Top 5 Features
0	Decision Tree	0.994264	0.0	[education_mother, estimated_fam_income_this_year, fam_size, mother_alive, age]
1	Random Forest	0.996176	0.0	[estimated_fam_income_this_year, education_mother, fam_size, education_father, num_siblings]
2	Gradient Boosted Trees	0.996176	0.0	[estimated_fam_income_this_year, education_mother, education_father, fam_size, ever_used_force]

In [ ]:
```
# ================================ Target 8 ================================:
# plot tree:
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

plt.figure(figsize=(10,8))
plot_tree(dt_model_8, filled=True, fontsize=12, feature_names=df.columns)
plt.show()
```



Target 9: Has ever smoked cannabis

In [ ]:
```
# ================================ Target 9 ================================:
target9 = target_ever_cannabis.values

# sample split:
X_train, X_test, y_train, y_test = model_selection.train_test_split(
    df, target9, train_size=.75, test_size=.25,
```

```
    shuffle=True, random_state=9)

np.random.seed(9)

# ============================== tree tuning ==============================:
# Define the parameter grid to search
param_grid = {'max_depth': [2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30
              'min_samples_split': [2, 4, 6, 8, 10, 12, 14],
              'min_samples_leaf': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
              'criterion': ['gini', 'entropy']}

# Create a decision tree classifier object
dtc9 = DecisionTreeClassifier()

# Create a GridSearchCV object
grid_search = GridSearchCV(dtc9, param_grid, cv=5)

# Fit the GridSearchCV object to the data
grid_search.fit(X_train, y_train)

# Print the best parameters and best score
print("Best parameters: ", grid_search.best_params_)
print("Best score: ", grid_search.best_score_)
```

```
Best parameters:  {'criterion': 'gini', 'max_depth': 4, 'min_samples_leaf': 1, 'mi
n_samples_split': 10}
Best score:  0.6630552898801408
```

In [ ]:
```
# ================================ Target 9 ================================:
# ========================== random forest tuning ==========================:
np.random.seed(9)

# Define the parameter grid to search
param_grid = {'n_estimators': [50, 100, 150, 200, 250, 300],
              'max_features': ['sqrt', 'log2']}

# Create a random forest classifier object
rfc9 = RandomForestClassifier()

# Create a GridSearchCV object
grid_search = GridSearchCV(rfc9, param_grid, cv=5)

# Fit the GridSearchCV object to the data
grid_search.fit(X_train, y_train.ravel())

# Print the best parameters and best score
print("Best parameters: ", grid_search.best_params_)
print("Best score: ", grid_search.best_score_)
```

```
Best parameters:  {'max_features': 'sqrt', 'n_estimators': 250}
Best score:  0.6592173541441972
```

In [ ]:
```
# ================================ Target 9 ================================:
# ========================== gradient-boost tuning ==========================:
np.random.seed(9)

# Define the parameter grid to search
param_grid = {'n_estimators': [10, 20, 30, 40, 50, 100, 200, 300, 400],
              'learning_rate': [0.01, 0.02, 0.03, 0.04, 0.05, 0.1, 0.2],
```

```
              'max_features': ['sqrt', 'log2']}

# Create a gradient boosted classifier object
gbc9 = GradientBoostingClassifier()

# Create a GridSearchCV object
grid_search = GridSearchCV(gbc9, param_grid, cv=5)

# Fit the GridSearchCV object to the data
grid_search.fit(X_train, y_train.ravel())

# Print the best parameters and best score
print("Best parameters: ", grid_search.best_params_)
print("Best score: ", grid_search.best_score_)
```

```
Best parameters:  {'learning_rate': 0.05, 'max_features': 'log2', 'n_estimators':
100}
Best score:  0.6770883783398791
```

In [ ]:

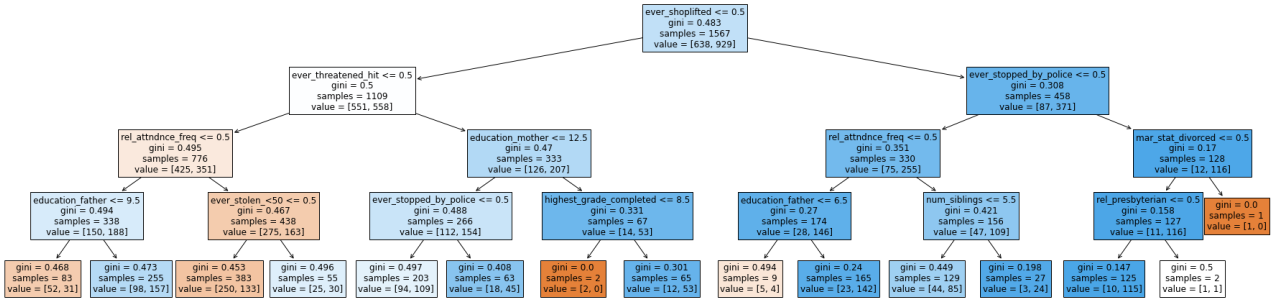
```
# ================================ Target 9 ================================:

# Fit decision tree
dt_model_9 = DecisionTreeClassifier(criterion='gini', max_depth=4, random_state=9,
                                    min_samples_leaf=1, min_samples_split=10)
dt_model_9.fit(X_train, y_train.ravel())
dt_pred_9 = dt_model_9.predict(X_test)
dt_acc_9 = accuracy_score(y_test.ravel(), dt_pred_9)

# Fit random forest
rf_model_9 = RandomForestClassifier(max_features='sqrt', n_estimators=250,
                                    random_state=9)
rf_model_9.fit(X_train, y_train.ravel())
rf_pred_9 = rf_model_9.predict(X_test)
rf_acc_9 = accuracy_score(y_test.ravel(), rf_pred_9)

# Fit gradient boosted trees
gbt_model_9 = GradientBoostingClassifier(learning_rate=0.05, max_features='log2',
                                         n_estimators=100, random_state=9)
gbt_model_9.fit(X_train, y_train.ravel())
gbt_pred_9 = gbt_model_9.predict(X_test)
gbt_acc_9 = accuracy_score(y_test.ravel(), gbt_pred_9)

# Calculate accuracy and recall scores for each model
dt_acc_9 = accuracy_score(y_test, dt_pred_9)
rf_acc_9 = accuracy_score(y_test, rf_pred_9)
gbt_acc_9 = accuracy_score(y_test, gbt_pred_9)

dt_recall_9 = recall_score(y_test, dt_pred_9)
rf_recall_9 = recall_score(y_test, rf_pred_9)
gbt_recall_9 = recall_score(y_test, gbt_pred_9)

# Compute top 5 most important features for each model
dt9_feat_importances = pd.Series(dt_model_9.feature_importances_,
                                 index=X_train.columns
                                 ).nlargest(5).index.tolist()

rf9_feat_importances = pd.Series(rf_model_9.feature_importances_,
                                 index=X_train.columns
                                 ).nlargest(5).index.tolist()
```

```
gbt9_feat_importances = pd.Series(gbt_model_9.feature_importances_,
                                  index=X_train.columns
                                  ).nlargest(5).index.tolist()


# Create table of important features, accuracy and recall scores
scores = {'Model': ['Decision Tree', 'Random Forest', 'Gradient Boosted Trees'],
          'Accuracy': [dt_acc_9, rf_acc_9, gbt_acc_9],
          'Recall': [dt_recall_9, rf_recall_9, gbt_recall_9],
          'Top 5 Features': [dt9_feat_importances, rf9_feat_importances,
                             gbt9_feat_importances]}
scores = pd.DataFrame(scores)
pd.set_option('display.max_colwidth', None)
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
scores
```

Out[ ]:

	Model	Accuracy	Recall	Top 5 Features
0	Decision Tree	0.650096	0.794304	[ever_shoplifted, rel_attndnce_freq, ever_threatened_hit, education_father, ever_stopped_by_police]
1	Random Forest	0.644359	0.781646	[estimated_fam_income_this_year, education_father, fam_size, num_siblings, education_mother]
2	Gradient Boosted Trees	0.646272	0.768987	[ever_shoplifted, rel_attndnce_freq, ever_threatened_hit, ever_stolen_<50, ever_stopped_by_police]

In [ ]:
```
# ================================== Target 9 ==================================:
# plot tree:
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

plt.figure(figsize=(32,8))
plot_tree(dt_model_9, filled=True, fontsize=12, feature_names=df.columns)
plt.show()
```



Target 10: Has ever gotten high from chemicals or other drugs

In [ ]:
```
# ================================== Target 10 ==================================:
target10 = target_ever_drugs_chemicals_high.values

# sample split:
X_train, X_test, y_train, y_test = model_selection.train_test_split(
    df, target10, train_size=.75, test_size=.25,
```

```
    shuffle=True, random_state=10)

np.random.seed(10)

# ============================== tree tuning ==============================:
# Define the parameter grid to search
param_grid = {'max_depth': [2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30
              'min_samples_split': [2, 4, 6, 8, 10, 12, 14],
              'min_samples_leaf': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
              'criterion': ['gini', 'entropy']}

# Create a decision tree classifier object
dtc10 = DecisionTreeClassifier()

# Create a GridSearchCV object
grid_search = GridSearchCV(dtc10, param_grid, cv=5)

# Fit the GridSearchCV object to the data
grid_search.fit(X_train, y_train)

# Print the best parameters and best score
print("Best parameters: ", grid_search.best_params_)
print("Best score: ", grid_search.best_score_)
```

```
Best parameters:  {'criterion': 'gini', 'max_depth': 2, 'min_samples_leaf': 1, 'mi
n_samples_split': 2}
Best score:  0.7581408599743595
```

In [ ]:
```
# ================================ Target 10 ================================:
# ========================== random forest tuning ==========================:
np.random.seed(10)

# Define the parameter grid to search
param_grid = {'n_estimators': [50, 100, 150, 200, 250, 300],
              'max_features': ['sqrt', 'log2']}

# Create a random forest classifier object
rfc10 = RandomForestClassifier()

# Create a GridSearchCV object
grid_search = GridSearchCV(rfc10, param_grid, cv=5)

# Fit the GridSearchCV object to the data
grid_search.fit(X_train, y_train.ravel())

# Print the best parameters and best score
print("Best parameters: ", grid_search.best_params_)
print("Best score: ", grid_search.best_score_)
```

```
Best parameters:  {'max_features': 'sqrt', 'n_estimators': 100}
Best score:  0.7677234895504773
```

In [ ]:
```
# ================================ Target 10 ================================:
# ========================== gradient-boost tuning ==========================:
np.random.seed(10)

# Define the parameter grid to search
param_grid = {'n_estimators': [10, 20, 30, 40, 50, 100, 200, 300, 400],
              'learning_rate': [0.01, 0.02, 0.03, 0.04, 0.05, 0.1, 0.2],
```

```
              'max_features': ['sqrt', 'log2']}

# Create a gradient boosted classifier object
gbc10 = GradientBoostingClassifier()

# Create a GridSearchCV object
grid_search = GridSearchCV(gbc10, param_grid, cv=5)

# Fit the GridSearchCV object to the data
grid_search.fit(X_train, y_train.ravel())

# Print the best parameters and best score
print("Best parameters: ", grid_search.best_params_)
print("Best score: ", grid_search.best_score_)
```

```
Best parameters:  {'learning_rate': 0.2, 'max_features': 'sqrt', 'n_estimators': 2
0}
Best score:  0.774090881341446
```

In [ ]:

```
# ================================ Target 10 ================================:

# Fit decision tree
dt_model_10 = DecisionTreeClassifier(criterion='gini', max_depth=2, random_state=1
                                     min_samples_leaf=1, min_samples_split=2)
dt_model_10.fit(X_train, y_train.ravel())
dt_pred_10 = dt_model_8.predict(X_test)
dt_acc_10 = accuracy_score(y_test.ravel(), dt_pred_10)

# Fit random forest
rf_model_10 = RandomForestClassifier(max_features='sqrt', n_estimators=100,
                                     random_state=10)
rf_model_10.fit(X_train, y_train.ravel())
rf_pred_10 = rf_model_9.predict(X_test)
rf_acc_10 = accuracy_score(y_test.ravel(), rf_pred_10)

# Fit gradient boosted trees
gbt_model_10 = GradientBoostingClassifier(learning_rate=0.2, max_features='sqrt',
                                          n_estimators=20, random_state=10)
gbt_model_10.fit(X_train, y_train.ravel())
gbt_pred_10 = gbt_model_10.predict(X_test)
gbt_acc_10 = accuracy_score(y_test.ravel(), gbt_pred_10)

# Calculate accuracy and recall scores for each model
dt_acc_10 = accuracy_score(y_test, dt_pred_10)
rf_acc_10 = accuracy_score(y_test, rf_pred_10)
gbt_acc_10 = accuracy_score(y_test, gbt_pred_10)

dt_recall_10 = recall_score(y_test, dt_pred_10)
rf_recall_10 = recall_score(y_test, rf_pred_10)
gbt_recall_10 = recall_score(y_test, gbt_pred_10)

# Compute top 5 most important features for each model
dt10_feat_importances = pd.Series(dt_model_10.feature_importances_,
                                  index=X_train.columns
                                  ).nlargest(5).index.tolist()

rf10_feat_importances = pd.Series(rf_model_10.feature_importances_,
                                  index=X_train.columns
                                  ).nlargest(5).index.tolist()
```

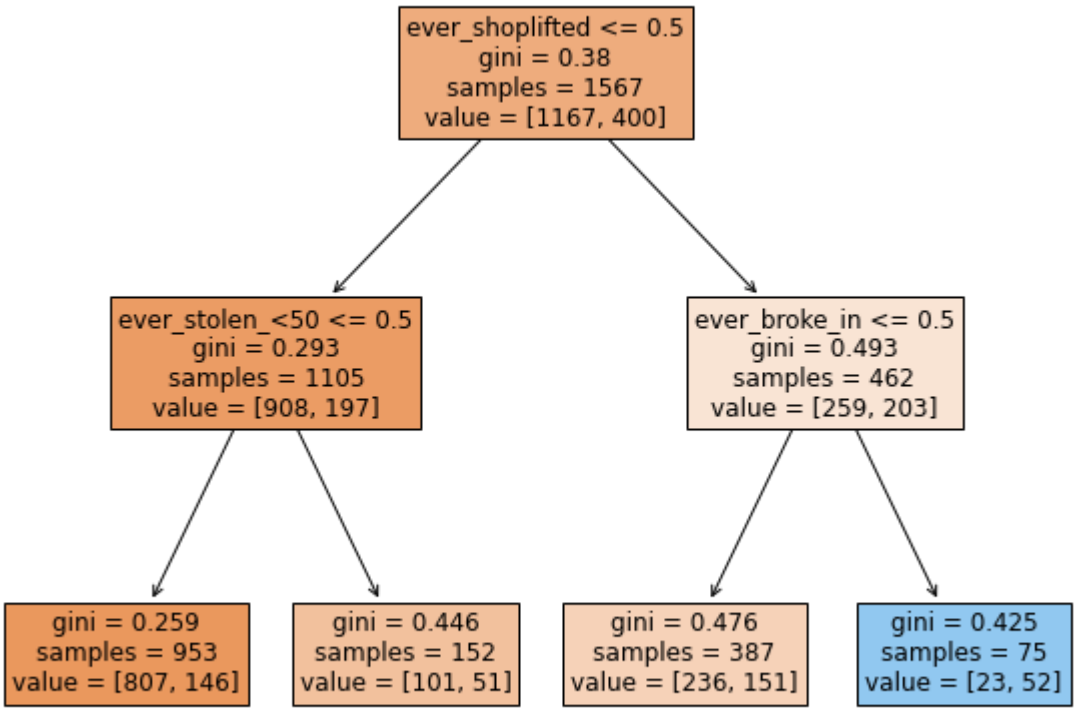
```
gbt10_feat_importances = pd.Series(gbt_model_10.feature_importances_,
                                   index=X_train.columns
                                   ).nlargest(5).index.tolist()


# Create table of important features, accuracy and recall scores
scores = {'Model': ['Decision Tree', 'Random Forest', 'Gradient Boosted Trees'],
          'Accuracy': [dt_acc_10, rf_acc_10, gbt_acc_10],
          'Recall': [dt_recall_10, rf_recall_10, gbt_recall_10],
          'Top 5 Features': [dt10_feat_importances, rf10_feat_importances,
                             gbt10_feat_importances]}
scores = pd.DataFrame(scores)
pd.set_option('display.max_colwidth', None)
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
scores
```

Out[ ]:

	Model	Accuracy	Recall	Top 5 Features
0	Decision Tree	0.797323	0.009346	[ever_shoplifted, ever_broke_in, ever_stolen_<50, mother_alive, age]
1	Random Forest	0.609943	0.971963	[estimated_fam_income_this_year, education_father, num_siblings, fam_size, education_mother]
2	Gradient Boosted Trees	0.793499	0.224299	[ever_shoplifted, ever_stolen_<50, ever_broke_in, ever_destructed_property, rel_attndnce_freq]

In [ ]:

```
# ================================= Target 10 ================================:
# plot tree:
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

plt.figure(figsize=(10,8))
plot_tree(dt_model_10, filled=True, fontsize=12, feature_names=df.columns)
plt.show()
```



Target 11: Has ever sold cannabis

In [ ]:

```
# ================================ Target 11 ================================:
target11 = target_ever_sold_cannabis.values

# sample split:
X_train, X_test, y_train, y_test = model_selection.train_test_split(
    df, target11, train_size=.75, test_size=.25,
    shuffle=True, random_state=11)

np.random.seed(11)

# ============================== tree tuning ==============================:
# Define the parameter grid to search
param_grid = {'max_depth': [2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30
              'min_samples_split': [2, 4, 6, 8, 10, 12, 14],
              'min_samples_leaf': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
              'criterion': ['gini', 'entropy']}

# Create a decision tree classifier object
dtc11 = DecisionTreeClassifier()

# Create a GridSearchCV object
grid_search = GridSearchCV(dtc11, param_grid, cv=5)

# Fit the GridSearchCV object to the data
grid_search.fit(X_train, y_train)

# Print the best parameters and best score
```

```
print("Best parameters: ", grid_search.best_params_)
print("Best score: ", grid_search.best_score_)
```

```
Best parameters:  {'criterion': 'entropy', 'max_depth': 2, 'min_samples_leaf': 1,
'min_samples_split': 2}
Best score:  0.87364115504365
```

In [ ]:
```
# ================================ Target 11 ================================:
# ========================== random forest tuning ==========================:
np.random.seed(11)

# Define the parameter grid to search
param_grid = {'n_estimators': [50, 100, 150, 200, 250, 300],
              'max_features': ['sqrt', 'log2']}

# Create a random forest classifier object
rfc11 = RandomForestClassifier()

# Create a GridSearchCV object
grid_search = GridSearchCV(rfc11, param_grid, cv=5)

# Fit the GridSearchCV object to the data
grid_search.fit(X_train, y_train.ravel())

# Print the best parameters and best score
print("Best parameters: ", grid_search.best_params_)
print("Best score: ", grid_search.best_score_)
```

```
Best parameters:  {'max_features': 'sqrt', 'n_estimators': 200}
Best score:  0.8774729858977228
```

In [ ]:
```
# ================================ Target 11 ================================:
# ========================== gradient-boost tuning ==========================:
np.random.seed(11)

# Define the parameter grid to search
param_grid = {'n_estimators': [50, 100, 200, 300, 400],
              'learning_rate': [0.01, 0.02, 0.03, 0.04, 0.05, 0.1, 0.2],
              'max_features': ['sqrt', 'log2']}

# Create a gradient boosted classifier object
gbc11 = GradientBoostingClassifier()

# Create a GridSearchCV object
grid_search = GridSearchCV(gbc11, param_grid, cv=5)

# Fit the GridSearchCV object to the data
grid_search.fit(X_train, y_train.ravel())

# Print the best parameters and best score
print("Best parameters: ", grid_search.best_params_)
print("Best score: ", grid_search.best_score_)
```

```
Best parameters:  {'learning_rate': 0.02, 'max_features': 'log2', 'n_estimators':
200}
Best score:  0.8806597342341427
```

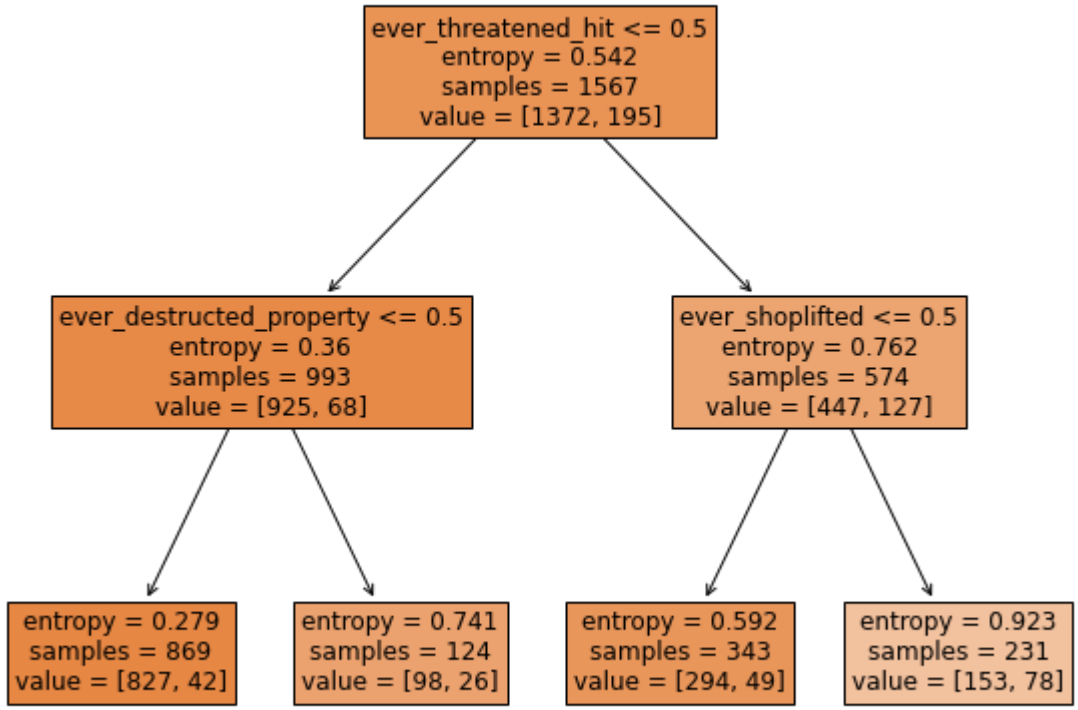
In [ ]:
```
# ================================ Target 11 ================================:

# Fit decision tree
```

```
dt_model_11 = DecisionTreeClassifier(criterion='entropy', max_depth=2, random_stat
                                     min_samples_leaf=1, min_samples_split=2)
dt_model_11.fit(X_train, y_train.ravel())
dt_pred_11 = dt_model_11.predict(X_test)
dt_acc_11 = accuracy_score(y_test.ravel(), dt_pred_11)

# Fit random forest
rf_model_11 = RandomForestClassifier(max_features='sqrt', n_estimators=200,
                                     random_state=11)
rf_model_11.fit(X_train, y_train.ravel())
rf_pred_11 = rf_model_11.predict(X_test)
rf_acc_11 = accuracy_score(y_test.ravel(), rf_pred_11)

# Fit gradient boosted trees
gbt_model_11 = GradientBoostingClassifier(learning_rate=0.02, max_features='log2',
                                          n_estimators= 200, random_state=11)
gbt_model_11.fit(X_train, y_train.ravel())
gbt_pred_11 = gbt_model_11.predict(X_test)
gbt_acc_11 = accuracy_score(y_test.ravel(), gbt_pred_11)

# Calculate accuracy and recall scores for each model
dt_acc_11 = accuracy_score(y_test, dt_pred_11)
rf_acc_11 = accuracy_score(y_test, rf_pred_11)
gbt_acc_11 = accuracy_score(y_test, gbt_pred_11)

dt_recall_11 = recall_score(y_test, dt_pred_11)
rf_recall_11 = recall_score(y_test, rf_pred_11)
gbt_recall_11 = recall_score(y_test, gbt_pred_11)

# Compute top 5 most important features for each model
dt11_feat_importances = pd.Series(dt_model_11.feature_importances_,
                                  index=X_train.columns
                                  ).nlargest(5).index.tolist()

rf11_feat_importances = pd.Series(rf_model_11.feature_importances_,
                                  index=X_train.columns
                                  ).nlargest(5).index.tolist()

gbt11_feat_importances = pd.Series(gbt_model_11.feature_importances_,
                                   index=X_train.columns
                                   ).nlargest(5).index.tolist()


# Create table of important features, accuracy and recall scores
scores = {'Model': ['Decision Tree', 'Random Forest', 'Gradient Boosted Trees'],
          'Accuracy': [dt_acc_11, rf_acc_11, gbt_acc_11],
          'Recall': [dt_recall_11, rf_recall_11, gbt_recall_11],
          'Top 5 Features': [dt11_feat_importances, rf11_feat_importances,
                             gbt11_feat_importances]}
scores = pd.DataFrame(scores)
pd.set_option('display.max_colwidth', None)
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
scores
```

Out[ ]:

	Model	Accuracy	Recall	Top 5 Features
0	Decision Tree	0.879541	0.000000	[ever_threatened_hit, ever_destructed_property, ever_shoplifted, mother_alive, age]
1	Random Forest	0.889101	0.111111	[estimated_fam_income_this_year, education_father, num_siblings, education_mother, fam_size]
2	Gradient Boosted Trees	0.891013	0.158730	[ever_threatened_hit, ever_stopped_by_police, ever_shoplifted, ever_destructed_property, ever_broke_in]

In [ ]:
```
# ================================ Target 11 ================================:
# plot tree:
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

plt.figure(figsize=(10,8))
plot_tree(dt_model_11, filled=True, fontsize=12, feature_names=df.columns)
plt.show()
```



Target 12: Has ever sold hard drugs

In [ ]:
```
# ================================ Target 12 ================================:
target12 = target_ever_sold_hard_drugs.values

# sample split:
X_train, X_test, y_train, y_test = model_selection.train_test_split(
    df, target10, train_size=.75, test_size=.25,
```

```
    shuffle=True, random_state=12)

np.random.seed(12)

# ============================== tree tuning ==============================:
# Define the parameter grid to search
param_grid = {'max_depth': [2, 4, 6, 8],
              'min_samples_split': [2, 4, 6, 8, 10, 12, 14],
              'min_samples_leaf': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
              'criterion': ['gini', 'entropy']}

# Create a decision tree classifier object
dtc12 = DecisionTreeClassifier()

# Create a GridSearchCV object
grid_search = GridSearchCV(dtc12, param_grid, cv=5)

# Fit the GridSearchCV object to the data
grid_search.fit(X_train, y_train)

# Print the best parameters and best score
print("Best parameters: ", grid_search.best_params_)
print("Best score: ", grid_search.best_score_)
```

```
Best parameters:  {'criterion': 'gini', 'max_depth': 4, 'min_samples_leaf': 9, 'mi
n_samples_split': 2}
Best score:  0.7708959931625323
```

In [ ]:
```
# ================================ Target 12 ================================:
# ============================ random forest tuning ============================:
np.random.seed(12)

# Define the parameter grid to search
param_grid = {'n_estimators': [50, 100, 150, 200, 250, 300],
              'max_features': ['sqrt', 'log2']}

# Create a random forest classifier object
rfc12 = RandomForestClassifier()

# Create a GridSearchCV object
grid_search = GridSearchCV(rfc12, param_grid, cv=5)

# Fit the GridSearchCV object to the data
grid_search.fit(X_train, y_train.ravel())

# Print the best parameters and best score
print("Best parameters: ", grid_search.best_params_)
print("Best score: ", grid_search.best_score_)
```

```
Best parameters:  {'max_features': 'log2', 'n_estimators': 100}
Best score:  0.775370871573635
```

In [ ]:
```
# ================================ Target 12 ================================:
# ============================ gradient-boost tuning ============================:
np.random.seed(12)

# Define the parameter grid to search
param_grid = {'n_estimators': [50, 100, 200, 300, 400],
              'learning_rate': [0.01, 0.02, 0.03, 0.04, 0.05, 0.1, 0.2],
```

```
              'max_features': ['sqrt', 'log2']}

# Create a gradient boosted classifier object
gbc12 = GradientBoostingClassifier()

# Create a GridSearchCV object
grid_search = GridSearchCV(gbc12, param_grid, cv=5)

# Fit the GridSearchCV object to the data
grid_search.fit(X_train, y_train.ravel())

# Print the best parameters and best score
print("Best parameters: ", grid_search.best_params_)
print("Best score: ", grid_search.best_score_)
```

```
Best parameters:  {'learning_rate': 0.04, 'max_features': 'log2', 'n_estimators':
100}
Best score:  0.7785576199100548
```

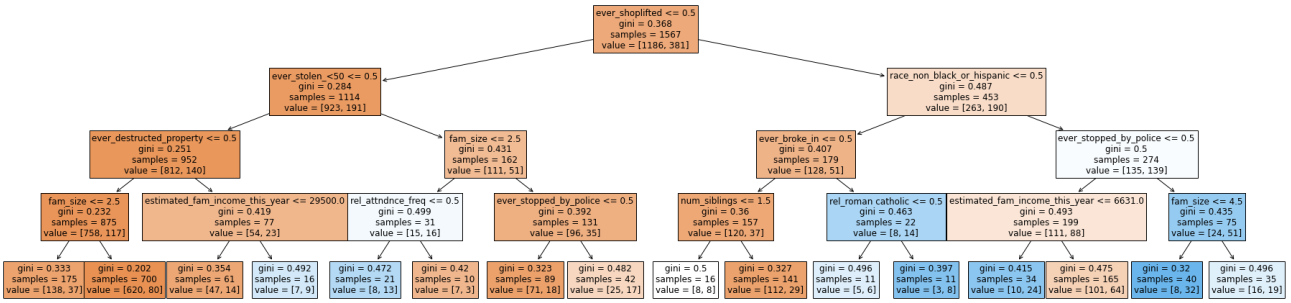
In [ ]:
```
# ================================ Target 12 ================================:

# Fit decision tree
dt_model_12 = DecisionTreeClassifier(criterion='gini', max_depth=4, random_state=1
                                     min_samples_leaf=9, min_samples_split=2)
dt_model_12.fit(X_train, y_train.ravel())
dt_pred_12 = dt_model_12.predict(X_test)
dt_acc_12 = accuracy_score(y_test.ravel(), dt_pred_12)

# Fit random forest
rf_model_12 = RandomForestClassifier(max_features='log2', n_estimators=100,
                                     random_state=12)
rf_model_12.fit(X_train, y_train.ravel())
rf_pred_12 = rf_model_12.predict(X_test)
rf_acc_12 = accuracy_score(y_test.ravel(), rf_pred_12)

# Fit gradient boosted trees
gbt_model_12 = GradientBoostingClassifier(learning_rate=0.04, max_features='log2',
                                          n_estimators=100, random_state=12)
gbt_model_12.fit(X_train, y_train.ravel())
gbt_pred_12 = gbt_model_12.predict(X_test)
gbt_acc_12 = accuracy_score(y_test.ravel(), gbt_pred_12)

# Calculate accuracy and recall scores for each model
dt_acc_12 = accuracy_score(y_test, dt_pred_12)
rf_acc_12 = accuracy_score(y_test, rf_pred_12)
gbt_acc_12 = accuracy_score(y_test, gbt_pred_12)

dt_recall_12 = recall_score(y_test, dt_pred_12)
rf_recall_12 = recall_score(y_test, rf_pred_12)
gbt_recall_12 = recall_score(y_test, gbt_pred_12)

# Compute top 5 most important features for each model
dt12_feat_importances = pd.Series(dt_model_12.feature_importances_,
                                  index=X_train.columns
                                  ).nlargest(5).index.tolist()

rf12_feat_importances = pd.Series(rf_model_12.feature_importances_,
                                  index=X_train.columns
                                  ).nlargest(5).index.tolist()
```

```
gbt12_feat_importances = pd.Series(gbt_model_12.feature_importances_,
                                   index=X_train.columns
                                   ).nlargest(5).index.tolist()


# Create table of important features, accuracy and recall scores
scores = {'Model': ['Decision Tree', 'Random Forest', 'Gradient Boosted Trees'],
          'Accuracy': [dt_acc_12, rf_acc_12, gbt_acc_12],
          'Recall': [dt_recall_12, rf_recall_12, gbt_recall_12],
          'Top 5 Features': [dt12_feat_importances, rf12_feat_importances,
                             gbt12_feat_importances]}
scores = pd.DataFrame(scores)
pd.set_option('display.max_colwidth', None)
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
scores
```

Out[ ]:

	Model	Accuracy	Recall	Top 5 Features
0	Decision Tree	0.774379	0.253968	[ever_shoplifted, race_non_black_or_hispanic, estimated_fam_income_this_year, ever_stopped_by_police, fam_size]
1	Random Forest	0.776291	0.174603	[estimated_fam_income_this_year, education_father, fam_size, num_siblings, education_mother]
2	Gradient Boosted Trees	0.768642	0.142857	[ever_shoplifted, ever_stolen_<50, ever_destructed_property, fam_size, ever_broke_in]

In [ ]:
```
# ================================== Target 12 ==================================:
# plot tree:
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

plt.figure(figsize=(32,8))
plot_tree(dt_model_12, filled=True, fontsize=12, feature_names=df.columns)
plt.show()
```



Target 13: Has ever had more than 6 drinks at once

In [ ]:
```
# ================================== Target 13 ==================================:
target13 = target_ever_6_drinks_at_once.values

# sample split:
```

```
X_train, X_test, y_train, y_test = model_selection.train_test_split(
    df, target11, train_size=.75, test_size=.25,
    shuffle=True, random_state=13)

np.random.seed(4)

# ============================== tree tuning ==============================:
# Define the parameter grid to search
param_grid = {'max_depth': [2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30
              'min_samples_split': [2, 4, 6, 8, 10, 12, 14],
              'min_samples_leaf': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
              'criterion': ['gini', 'entropy']}

# Create a decision tree classifier object
dtc13 = DecisionTreeClassifier()

# Create a GridSearchCV object
grid_search = GridSearchCV(dtc13, param_grid, cv=5)

# Fit the GridSearchCV object to the data
grid_search.fit(X_train, y_train)

# Print the best parameters and best score
print("Best parameters: ", grid_search.best_params_)
print("Best score: ", grid_search.best_score_)
```

```
Best parameters:  {'criterion': 'gini', 'max_depth': 2, 'min_samples_leaf': 1, 'mi
n_samples_split': 2}
Best score:  0.8768360432225638
```

In [ ]:
```
# ================================ Target 13 ================================:
# ========================== random forest tuning ==========================:
np.random.seed(13)

# Define the parameter grid to search
param_grid = {'n_estimators': [50, 100, 150, 200, 250, 300],
              'max_features': ['sqrt', 'log2']}

# Create a random forest classifier object
rfc13 = RandomForestClassifier()

# Create a GridSearchCV object
grid_search = GridSearchCV(rfc13, param_grid, cv=5)

# Fit the GridSearchCV object to the data
grid_search.fit(X_train, y_train.ravel())

# Print the best parameters and best score
print("Best parameters: ", grid_search.best_params_)
print("Best score: ", grid_search.best_score_)
```

```
Best parameters:  {'max_features': 'log2', 'n_estimators': 200}
Best score:  0.8800207565983598
```

In [ ]:
```
# ================================ Target 13 ================================:
# ========================== gradient-boost tuning ==========================:
np.random.seed(13)

# Define the parameter grid to search
```

```
param_grid = {'n_estimators': [10, 20, 30, 40, 50, 100, 200, 300, 400],
              'learning_rate': [0.01, 0.02, 0.03, 0.04, 0.05, 0.1, 0.2],
              'max_features': ['sqrt', 'log2']}

# Create a gradient boosted classifier object
gbc13 = GradientBoostingClassifier()

# Create a GridSearchCV object
grid_search = GridSearchCV(gbc13, param_grid, cv=5)

# Fit the GridSearchCV object to the data
grid_search.fit(X_train, y_train.ravel())

# Print the best parameters and best score
print("Best parameters: ", grid_search.best_params_)
print("Best score: ", grid_search.best_score_)
```

```
Best parameters:  {'learning_rate': 0.01, 'max_features': 'sqrt', 'n_estimators':
400}
Best score:  0.8819417594269552
```

In [ ]:

```
# ================================ Target 13 ================================:

# Fit decision tree
dt_model_13 = DecisionTreeClassifier(criterion='gini', max_depth=2, random_state=1
                                     min_samples_leaf=1, min_samples_split=2)
dt_model_13.fit(X_train, y_train.ravel())
dt_pred_13 = dt_model_13.predict(X_test)
dt_acc_13 = accuracy_score(y_test.ravel(), dt_pred_13)

# Fit random forest
rf_model_13 = RandomForestClassifier(max_features='log2', n_estimators=200,
                                     random_state=13)
rf_model_13.fit(X_train, y_train.ravel())
rf_pred_13 = rf_model_13.predict(X_test)
rf_acc_13 = accuracy_score(y_test.ravel(), rf_pred_13)

# Fit gradient boosted trees
gbt_model_13 = GradientBoostingClassifier(learning_rate=0.01, max_features='sqrt',
                                          n_estimators=400, random_state=13)
gbt_model_13.fit(X_train, y_train.ravel())
gbt_pred_13 = gbt_model_13.predict(X_test)
gbt_acc_13 = accuracy_score(y_test.ravel(), gbt_pred_13)

# Calculate accuracy and recall scores for each model
dt_acc_13 = accuracy_score(y_test, dt_pred_13)
rf_acc_13 = accuracy_score(y_test, rf_pred_13)
gbt_acc_13 = accuracy_score(y_test, gbt_pred_13)

dt_recall_13 = recall_score(y_test, dt_pred_13)
rf_recall_13 = recall_score(y_test, rf_pred_13)
gbt_recall_13 = recall_score(y_test, gbt_pred_13)

# Compute top 5 most important features for each model
dt13_feat_importances = pd.Series(dt_model_13.feature_importances_,
                                  index=X_train.columns
                                  ).nlargest(5).index.tolist()

rf13_feat_importances = pd.Series(rf_model_13.feature_importances_,
```

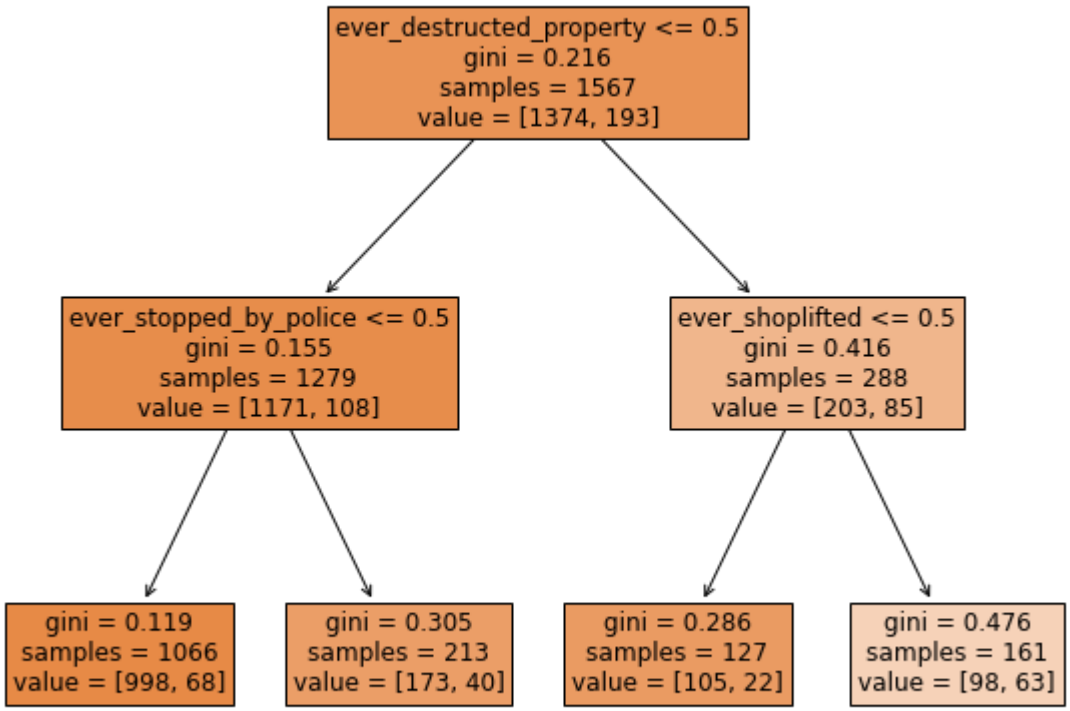
```
                                    index=X_train.columns
                                    ).nlargest(5).index.tolist()

gbt13_feat_importances = pd.Series(gbt_model_13.feature_importances_,
                                    index=X_train.columns
                                    ).nlargest(5).index.tolist()


# Create table of important features, accuracy and recall scores
scores = {'Model': ['Decision Tree', 'Random Forest', 'Gradient Boosted Trees'],
          'Accuracy': [dt_acc_13, rf_acc_13, gbt_acc_13],
          'Recall': [dt_recall_13, rf_recall_13, gbt_recall_13],
          'Top 5 Features': [dt13_feat_importances, rf13_feat_importances,
                             gbt13_feat_importances]}
scores = pd.DataFrame(scores)
pd.set_option('display.max_colwidth', None)
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
scores
```

Out[ ]:

	Model	Accuracy	Recall	Top 5 Features
0	Decision Tree	0.875717	0.000000	[ever_destructed_property, ever_shoplifted, ever_stopped_by_police, mother_alive, age]
1	Random Forest	0.881453	0.061538	[estimated_fam_income_this_year, num_siblings, education_father, fam_size, education_mother]
2	Gradient Boosted Trees	0.879541	0.076923	[ever_destructed_property, ever_shoplifted, ever_stopped_by_police, ever_broke_in, estimated_fam_income_this_year]

In [ ]:

```
# ================================= Target 13 ================================:
# plot tree:
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

plt.figure(figsize=(10,8))
plot_tree(dt_model_13, filled=True, fontsize=12, feature_names=df.columns)
plt.show()
```


ever_destructed_property <= 0.5
gini = 0.216
samples = 1567
value = [1374, 193]
ever_stopped_by_police <= 0.5
gini = 0.155
samples = 1279
value = [1171, 108]
ever_shoplifted <= 0.5
gini = 0.416
samples = 288
value = [203, 85]
gini = 0.119
samples = 1066
value = [998, 68]
gini = 0.305
samples = 213
value = [173, 40]
gini = 0.286
samples = 127
value = [105, 22]
gini = 0.476
samples = 161
value = [98, 63]