Analyzing and Predicting Substance Abuse in Youth using Decision Trees and Ensemble Methods

Abstract:

This study leverages the National Survey on Drug Use and Health (NSDUH) to explore different aspects of youth drug use using decision trees and ensemble methods. This project addresses three key predictions: identifying whether an individual uses alcohol with the help of binary classification, then estimating the frequency of alcohol use over the past year using multi-class classification, and finally predicting the age at which an individual first consumed alcohol using regression techniques. From our analysis, several variables proved to be significant predictors across different models. Factors related to educational achievements (EDUSCHGRD2) and demographic factors such as race (NEWRACE2), income levels (INCOME), and lifetime marijuana use (YFLMJMO) and alcohol consumption standards (STNDALC) were influential in predicting youth drug use behaviors. The binary classification gradient boosting model achieved a good accuracy of 83.12%, however the multi-class classification and regression tasks showed relatively poor performance but nonetheless gave us some insights about the factors related to youth substance abuse. These findings highlight the crucial role of both socio-demographic variables and substance use history in predicting drug use. The findings advocate for the critical role of family background and education in mitigating risky behaviors and provide insights for improving preventive strategies.

Introduction:

This study employs data from the National Survey on Drug Use and Health (NSDUH), an annual survey conducted to gather information about substance use and related behaviors from U.S. residents aged 12 and older. The dataset includes responses related to the use and frequency of the use of alcohol, marijuana, and tobacco. There is a group of variables that are responses to questions about youth's attitude regarding substance use, what they feel about substance use, as well as how they feel about their peers involved in substance use, and how their parents feel about substance use, whether they have been praised by their teachers or parents. Along with this, the dataset also has responses to whether youth was involved in a fight, or theft as well as their involvement in religious events and participation in team building or leadership activities.

Through this study, we aim to find important factors that can help us understand youth's behavior toward alcohol. The outcomes of this research will provide valuable guidance for public health initiatives that are aimed at reducing substance use among youth.

We have used decision trees and ensemble models to make three key predictions:

1. Identifying Alcohol Use - Binary Classification: The first task is to predict whether an individual uses alcohol. This is a binary classification where the model will predict a 'yes' or 'no' based on various predictors related to demographic details, youth experiences etc.
2. Estimating Frequency of Alcohol Use - Multi-Class Classification: The second task is where we are estimating the frequency of alcohol use over the past year. The model puts individuals into multiple groups based on how often they consumed alcohol in the last year.
3. Predicting Age of First Alcohol Use - Regression Analysis: The third task is to predict the age at which an individual consumed alcohol for the first time. This is done using regression techniques.

Theoretical background:

Decision tree is a supervised learning approach used for classification or regression tasks. Decision trees are built using a top-down greedy or recursive binary splitting. Initially all the observations belong to a single node at the top of the tree or the ‘root’ if we look at it upside down. Then it successively splits the predictor space, each split results in two new branches further down on the tree. It is called ‘greedy’ because at each step of the tree-building process, the best split is made at that particular step, rather than looking further ahead and selectively picking a split that will result in a better tree. The splits are made based on some splitting criterion such as Gini index– a measure of purity of the resulting split, For regression trees the splitting criterion is essentially a measure of residual squared sum, or RSS. The process of splitting goes on until a stopping criteria is reached, for instance, we may continue until no region contains more than five observations[2] or when no more improvement is expected in the performance. The stopping criterion can be specified in terms of tree depth and the optimal tree size is usually found using cross-validation with various tree sizes.

Tuning for decision trees is essentially performing a cross validation and pruning it with the optimal tree size found via cross validation. Pruning a tree mainly reduces the complexity of the model which means there may not be any significant improvement in the accuracy of the model but the overall tree would be easier to comprehend. Decision tree models tend to perform poorly when the feature space is large but due their ease of interpretability, these models are quite robust. Datasets with many categorical variables are also suitable for decision tree models as we don’t necessarily need to encode them. To improve the predictive power, ensemble methods like bagging, random forest, gradient boosting come into the picture.

Bootstrap aggregation or bagging is an ensemble technique that is used to reduce the variance of a statistical learning method. It is used in situations where it is challenging to directly compute the standard deviation of a quantity. The decision trees suffer from high variance. This means that if we randomly split the training data into two different sets, and build a decision tree to both sets, the results obtained could be quite different. If we have a set of n independent observations

Z1, . . . , Zn with variance σ^2 , the variance of the mean Z of the observations is given by $\frac{\sigma^2}{n}$. This simply means averaging a set of observations reduces variance. Hence, to reduce the variance and increase the test accuracy of a statistical learning method is to take many training sets from the dataset and build a separate model using each set and average the resulting predictions. Bagging is applied to regression trees by simply constructing B regression trees using B bootstrapped training sets, and averaging the resulting predictions. These trees are grown deep, and are not pruned. Hence each individual tree has high variance, but low bias. Averaging these B trees reduces the variance. Bagging has been demonstrated to give impressive improvements in accuracy by combining together hundreds or even thousands of trees into a single procedure[2]. In the context of classification problems, a simple bagging approach is to go by majority. For instance, for a given test observation, the class predicted by each of the B trees is recorded and a majority vote is taken, the final prediction is the most commonly occurring majority class among the B predictions. Although bagging is an improvement to decision trees but not when it comes to interpretability, hence we have something to the rescue called the 'feature importance'. We can display a summary of the importance of each predictor variable using the RSS (for bagging regression trees) or the Gini index (for bagging classification trees). In the case of bagging regression trees, we can record the total amount that the RSS is decreased due to splits over a given predictor, averaged over all B trees. A large value indicates an important predictor. Similarly, in the context of bagging classification trees, we can add up the total amount that the Gini index is decreased by splits over a given predictor, averaged over all B trees.

Random forests provide a further improvement over bagged trees with a small tweak that *decorrelates* the trees. Similar to bagging, we build a number of decision trees on bootstrapped training sets. But when building these decision trees, at each split, a random sample of m predictors is chosen from the full set of p predictors. The split is allowed to use only one of those m predictors. A fresh sample of m predictors is taken at each split, and typically we choose $m \approx \sqrt{p}$ meaning the number of predictors that are considered at each split are approximately equal to the square root of the total number of predictors. There's an interesting rationale behind this, if there is a strong predictor, that predictor will most likely be in all the bagged trees, meaning all trees end up becoming similar to each other. By limiting the number of predictors, other less strong predictors will have a higher chance of being able to determine an outcome. The resulting average trees are more reliable because of this phenomenon.

The ensemble models like bagging and random forest will produce an out-of-bag, OOB, error estimation, a measure of prediction error. When training these models, bootstrap samples of the original data are used, so the samples that are not used are called the "out-of-bag" samples. They are used to calculate the OOB error. Tuning this model will involve cross validating a range of m predictors to find the maximum number of features at each split.

In gradient boosting technique, the trees are grown successively, using a "slow" learning approach where information from the previous tree is used to improve the predictability of the decision tree. The current tree is trained in such a way that it corrects errors of the previous trees using residuals. There are three tuning parameters in the boosting method such as the learning rate, the number of trees and the number of splits in each tree that can be used.

Methodology:

Data Preparation:

This study focuses on analyzing youth's behavior based on factors related to their family, demographic details such as where they reside–urban or rural areas– and their experiences at school and religious beliefs. We removed data where the information about parents was either not known, deliberately not answered or the youth was 18+ years. We have also filtered out the data where the education background of an individual was unknown. Then data was further filtered to include individuals that belonged to densely populated or urban neighborhoods to address a particular group of youth that reside in urban areas. Features related to youth experiences' such as involvement in religious events, praise from teachers had a few missing values across the dataset, hence rows where response was either unknown or not answered were excluded from the dataset as it might contribute to the noise in the dataset. The dataset was split using 80% of the original data for the training set and 20% for the testing set for all the classification and regression tasks.

Models:

We merged demographic indicators and features related to youth experiences and alcohol flag for the binary classification task. A Decision tree was used to predict whether the individual had ever used alcohol. To improve the model, pruning was implemented by finding the optimal tree size using cross-validation technique.

Ensemble methods like random forest classifier, bagging and gradient booting were also used for the binary classification task. For all three methods, best parameters were found using grid search cross validation technique. For instance, an optimal number of maximum features was found for random forest, best n_estimators parameter and number of trees at each split for bagging was calculated and lastly, the optimal learning rate was found for the boosting method. Feature importances were analyzed for all the models.

For multi-class classification tasks, we are estimating the frequency of alcohol use over the past year using the 'ALCDAYS' as the target variable which has 5 categories based on the number of days. Similar to the binary classification task, we used a decision tree classifier, pruned decision tree, random forest and boosting. All models were cross-validated to find the optimal parameters.

For the regression task, the goal was to predict the age of initial alcohol use. The data was filtered to only include individuals over the age of 7 years and exclude where the response for the target feature IRALCAGE was unknown. Decision tree regressor, and ensemble methods like random forest and boosting regressor were employed to predict the age of initial alcohol consumption. Mean squared error was calculated for all the models since it is a regression task. Feature importances were analyzed to understand which predictors has the most influence.

Computational Results:

Binary classification task:

Table1 shows the results for the binary classification task which predicted whether an individual will use alcohol or not. The gradient boosting technique proved to be the best method for this task with an accuracy of 83.12% followed by other ensemble methods. The decision tree model performed decently after pruning it.

Binary classification results	
Model	Accuracy
Decision tree classifier	72.24%
Pruned decision tree	79.18%
Random forest classifier	81.39%
Bagging	80.91%
Gradient boosting classifier	83.12%

Table1: Binary classification results

feature_name	Descriptive name	importance
YFLMJMO	Lifetime Marijuana Use	0.575321
STNDALC	Standard Alcohol Consumption	0.242056
YFLTMRJ2	Lifetime Cocaine Use	0.032819
FRDMEVR2	Friend Has Ever Used Drugs	0.031388
POVERTY3	Poverty Level	0.025308

Table2

Figure1 shows the initial decision tree model which shows signs of overfitting since the size of the tree is huge. The confusion matrix for the decision tree in figure2 shows that the model is good at predicting outcome 0 compared to class 1. When cross validation was performed on the decision tree model, we can see a stable increase in the accuracy, which is a good indicator that the model is consistently performing well as shown in figure3. Figure4 and 5 represents the pruned decision tree and its confusion matrix, where a slight improvement was observed. The most important predictors for decision trees can be seen in table2 along with their importance.

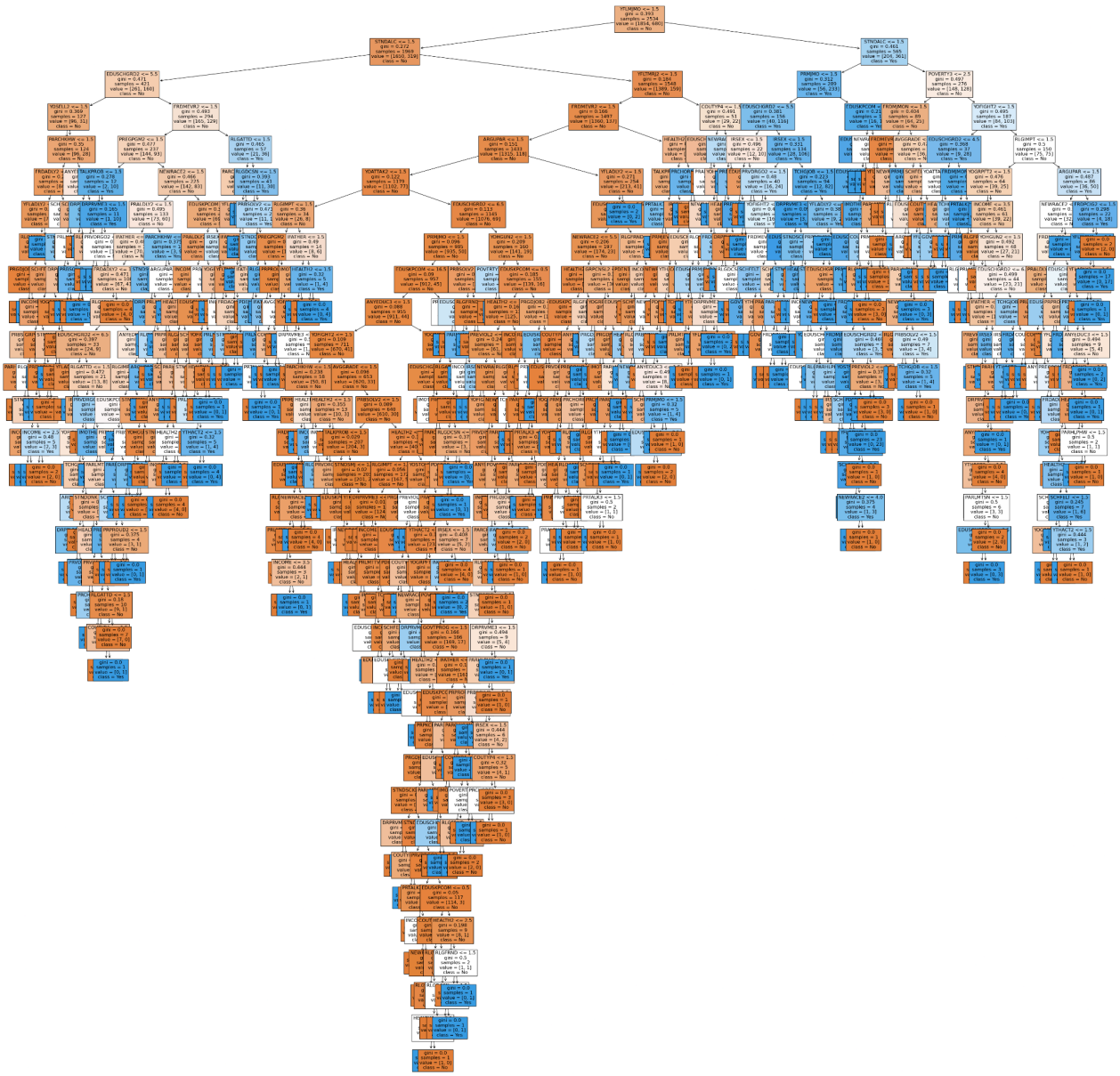


Figure1 : Decision tree

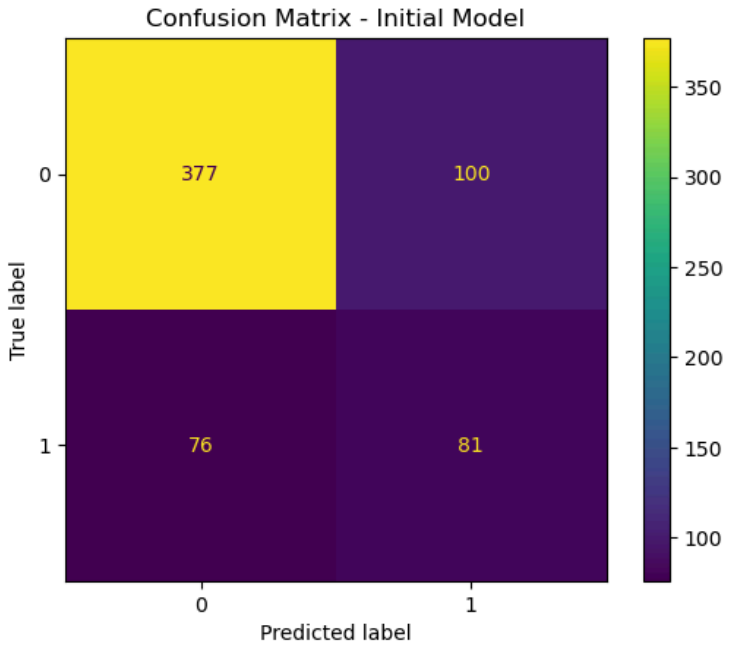


Figure 2: Confusion matrix - decision tree model

Figure3: Cross-validation results to find optimal tree size

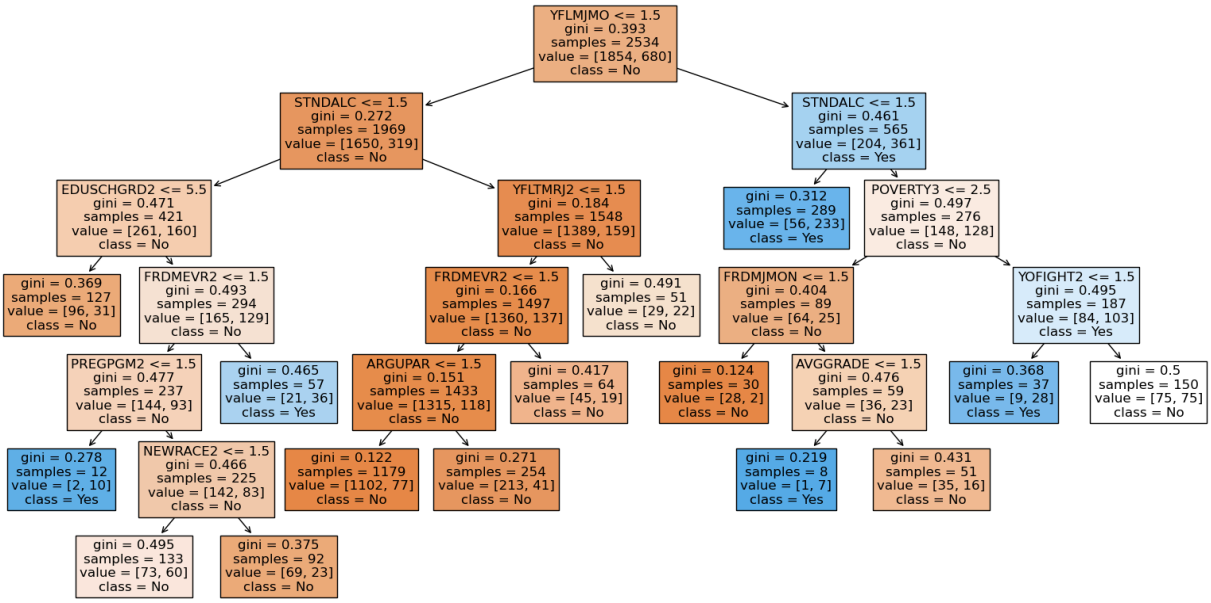


Figure4: Pruned decision tree

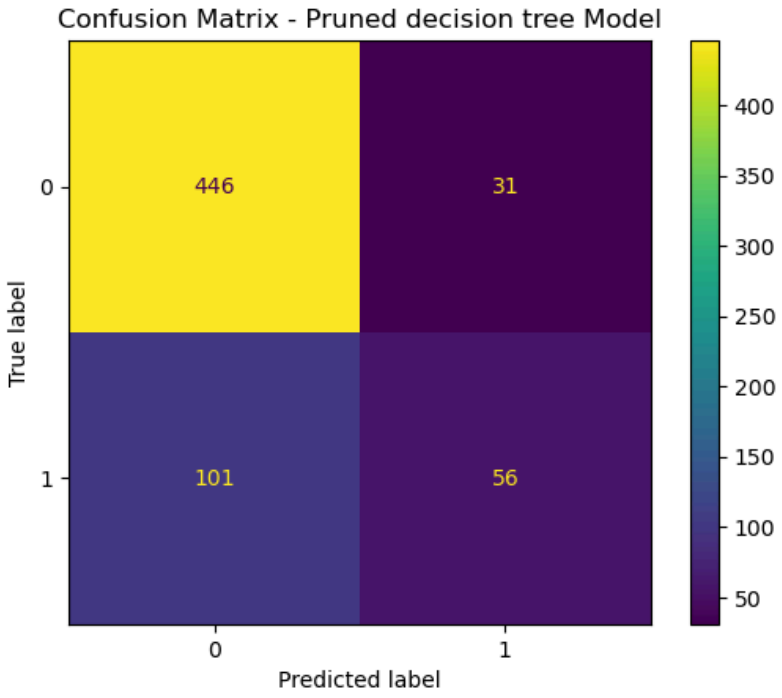


Figure5: Confusion matrix of pruned decision tree model

Random forest and bagging displayed good performance in terms of accuracy but they failed to predict class 1 and showed relatively worse performance for class 1 than decision trees. Although gradient boosting did exceptionally well in predicting both class 0 and 1. This shows

that it was able to learn patterns in data better than the rest of the models. This comparison is displayed in figure 6,7 and 8.

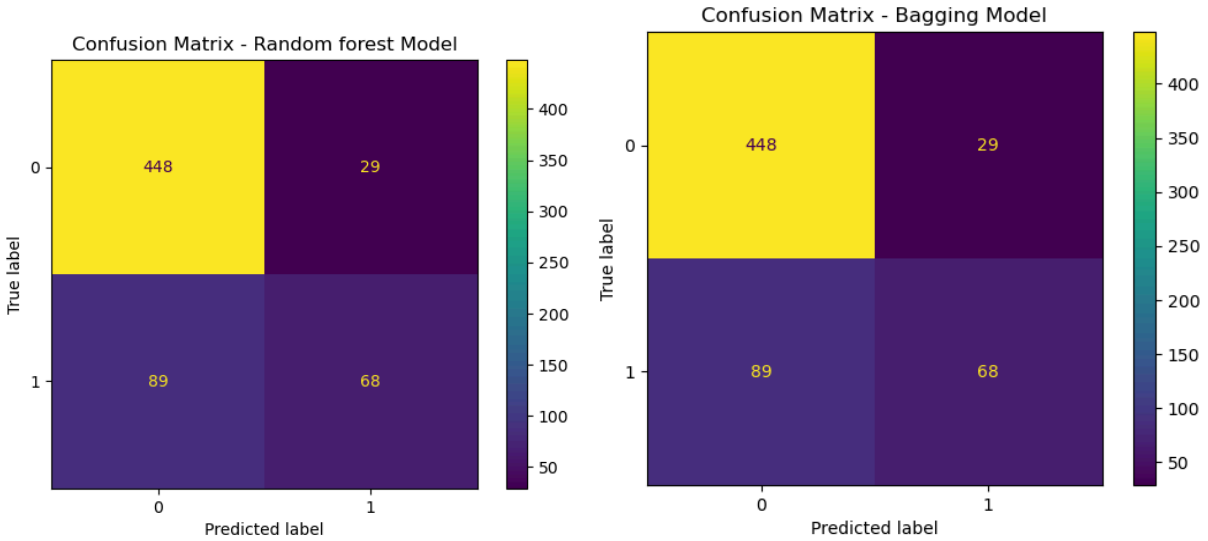


Figure6: Confusion matrix of random forest method, Figure7: Confusion matrix of bagging method

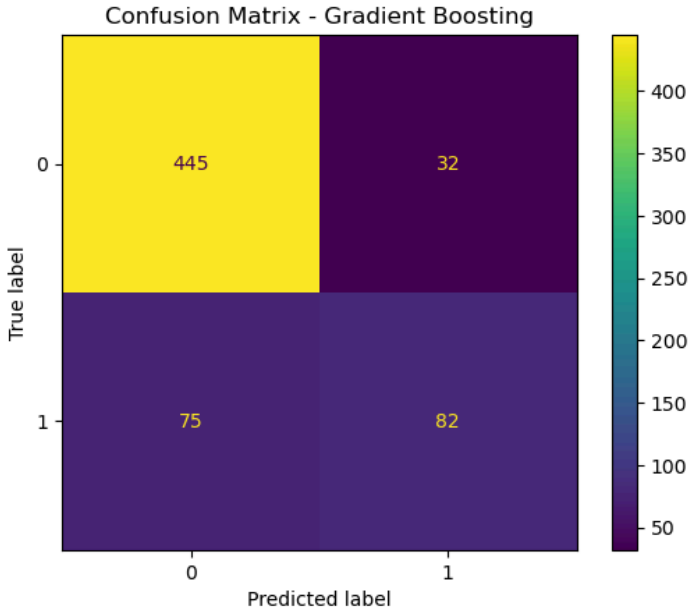


Figure8: Confusion matrix of gradient boosting model

Multi-class classification task:

Multi-class classification results	
Model	Accuracy
Decision tree classifier	68.45%
Pruned decision tree	79.50%
Pruned decision tree with class weights	60.88%
Random forest classifier	79.50%
Gradient boosting classifier	78.08%

Table3 Multi-class classification results

The results for the multi-class classification task are shown in Table3. Contrary to the binary classification task, here the pruned decision tree, random forest and boosting model proved to be better in terms of accuracy. Although when we look at the confusion matrices for each one of these in figure 10,12 ad 13, we can notice that these models were only good at predicting 'never used alcohol in the past year' outcome(class 0). They were extremely bad at capturing the patterns to predict other classes. But when class weights were used for pruned decision trees, we observed slightly better predictions for other classes like class 1 and 2.

Figure 9 CV results for multi-class

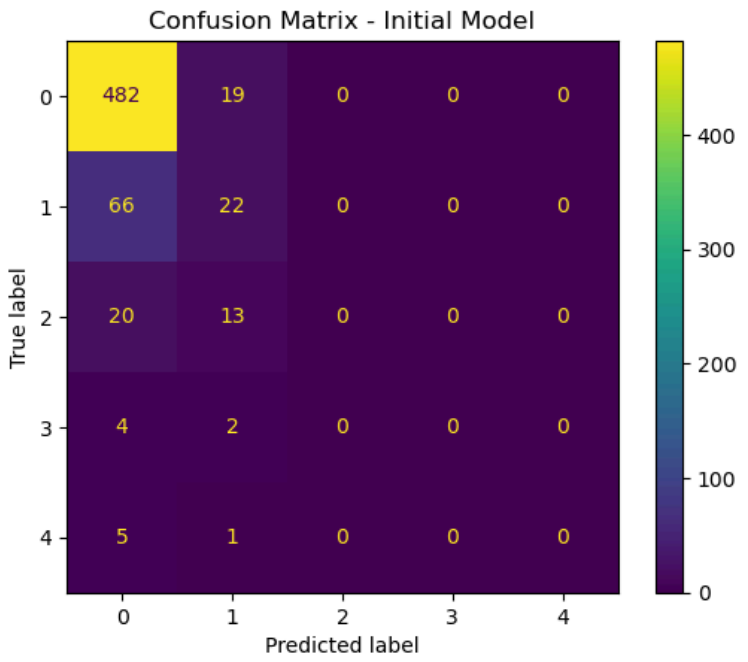


Figure 10 confusion matrix for initial decision tree

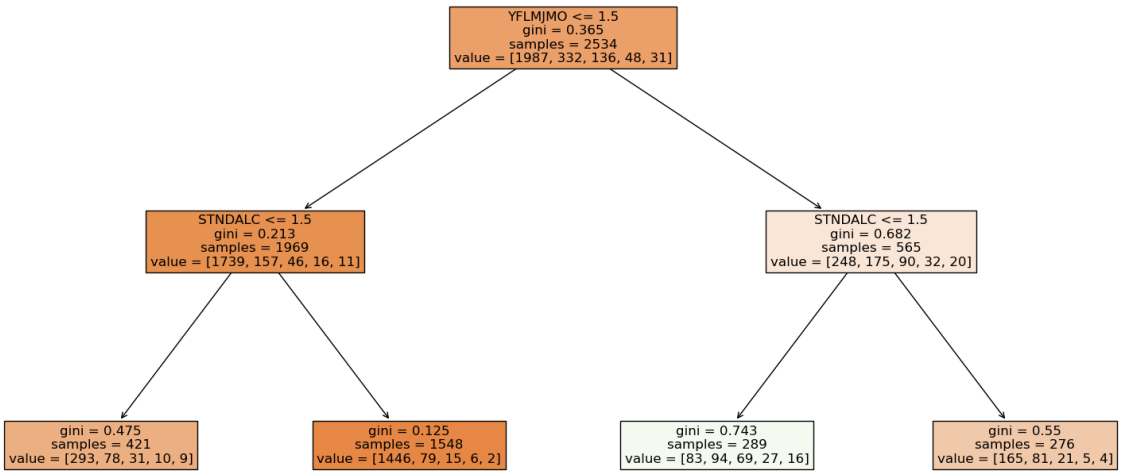


Figure 11 pruned decision tree

The same predictors seem to be important for the multi-class classification using decision trees as we noticed in the binary classification task. However, ensemble methods had different predictors that proved to be significant in the prediction of frequency of alcohol used in the past year as shown in table4.

feature_name	Descriptive name	importance
EDUSCHGRD2	What grade in now/will be in	0.056471
STNDALC	Standard Alcohol Consumption	0.040401
YFLMJMO	Feeling about peers trying marijuana	0.039615
YFLTMRJ2	Lifetime Cocaine Use	0.03828
FRDMEVR2	Friend Has Ever Used Drugs	0.033848

Table4 Feature importances for random forest model

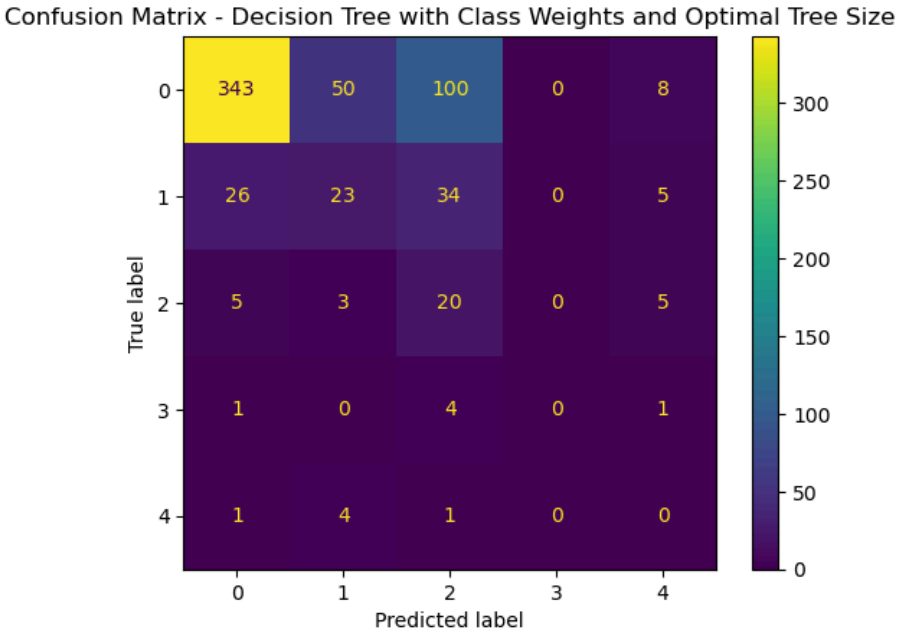


Figure 12 confusion matrix for pruned decision tree with class weights

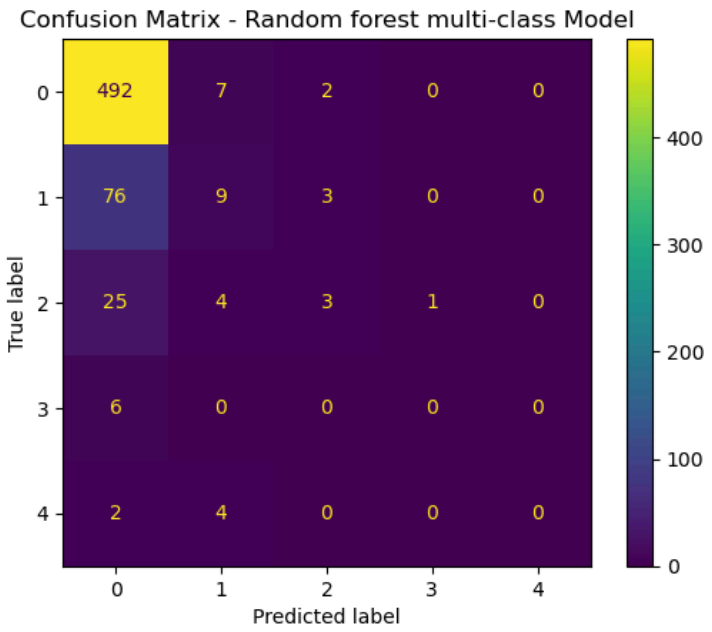


Figure 13 confusion matrix for random forest model

Regression task:

The regression task was performed where the age of initial alcohol consumption was predicted using decision tree, random forest and gradient boosting method. Mean squared error was calculated to find how well the model performed which is shown in table 5 below. The cross validation results are shown in figure 14 for the pruned decision tree. Random forest and decision tree model with optimal tree size showed better performance compared to the rest. The feature importance of random forest is shown in figure 16 with education feature at the top followed by other predictors. The plot in figure 15 shows the actual versus the predicted values by random forest model. The data points don't cluster perfectly around the diagonal line. This suggests that the model is not making very good predictions.

Regression results	
Model	MSE
Decision tree classifier	5.350
Pruned decision tree	2.525
Random forest classifier	2.78
Gradient boosting classifier	3.06

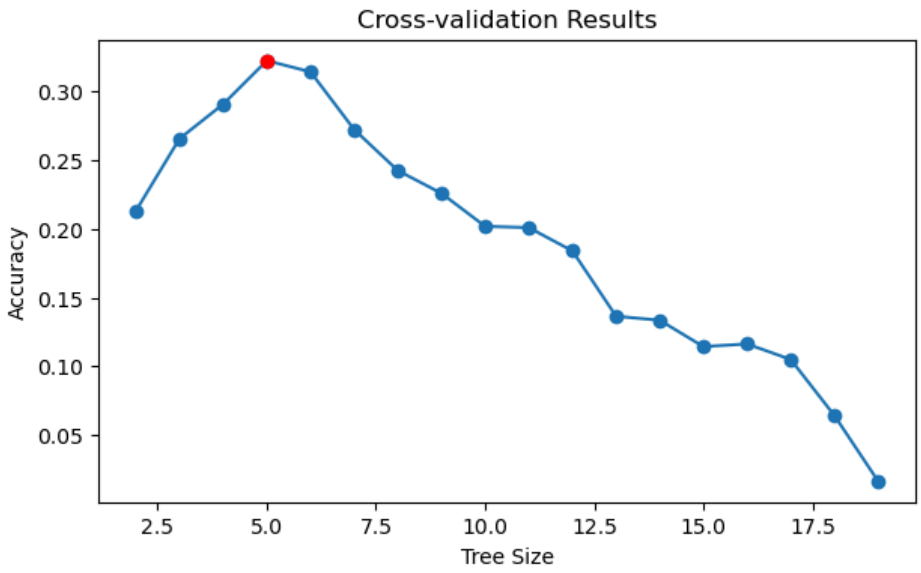


Figure 14 cv results - pruned decision tree

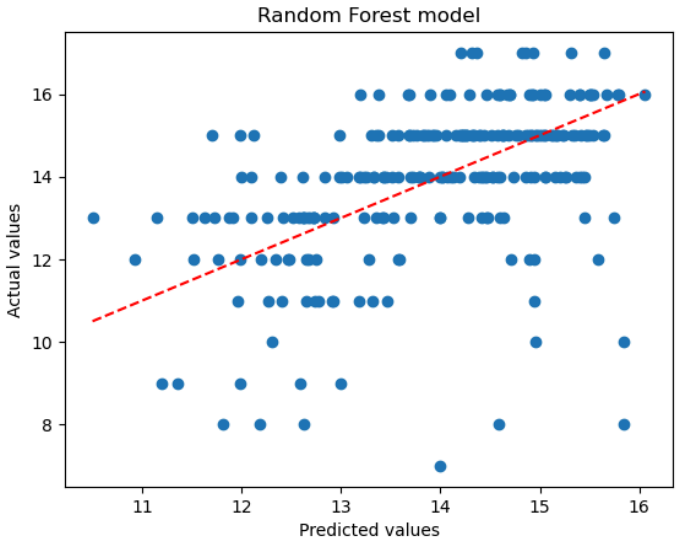


Figure 15 random forest model

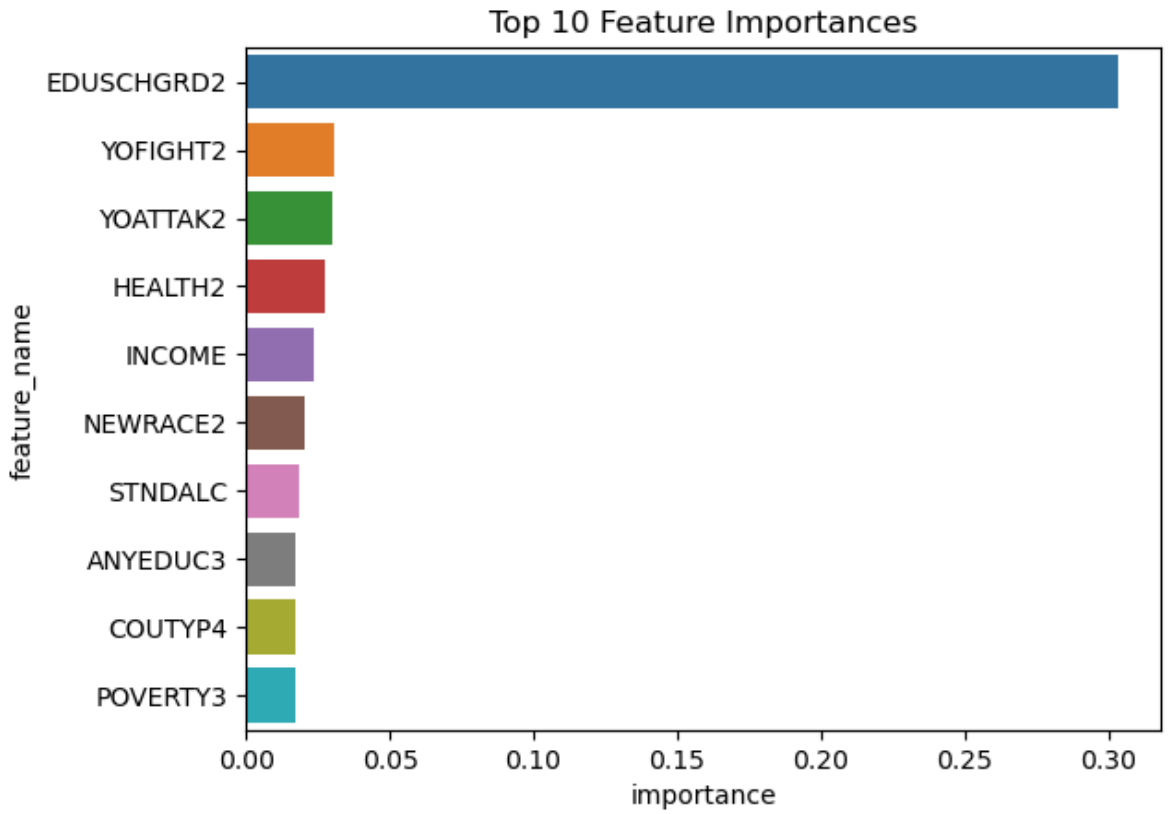


Figure 16 Feature importance - random forest model

Discussion:

The NSDUH dataset presents a comprehensive overview of behavioral and demographic variables that offer valuable insights into substance use patterns. Our analysis revealed that youth's feelings about substance use, exposure to other substances like marijuana, parental communication about substance use, the child's academic achievements, and their attitudes towards school play pivotal roles in influencing their likelihood of consuming alcohol. Specifically, positive school experiences and open conversations about substance risks are linked to a lower likelihood of substance use, underscoring the importance of supportive educational and familial environments.

The gradient boosting model, which was cross-validated to optimize performance, achieved an accuracy rate of 83.12% in predicting whether an individual would consume alcohol. This indicates the model's effectiveness in capturing the complex patterns in factors that contribute to alcohol consumption among youth.

It is not clear why the multi-class or regression task did not performing well given the array of significant predictors, perhaps it might be due to the class imbalance that lead to the models not fully learn other outcomes like in multi class classification, the models performed really well in

predicting the outcome ‘never used alcohol in the past year’ but failed to accurately predict other classes. The data representing the class 0(never used alcohol) was significantly more when compared to other classes. A dataset that’s equally representative of all classes might be a better one for the multi-class and regression task. Although the regression models did well in terms of their MSEs, the plot of actual vs predicted values clearly shows otherwise.

It is also important to acknowledge certain limitations in our study, the reliance on self-reported data might introduce biases, which can affect the accuracy of our predictions. The relatively small sample size could also limit the generalizability of our findings to a broader population.

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

Conclusion:

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

In binary classification tasks, the gradient boosting model notably achieved an accuracy of 83.12%. The multi-class classification task was only good at predicting if an individual has never used alcohol in the past year, the highest accuracy of 79.50% was achieved by pruned decision tree and random forest model. The regression task on the other hand has showed decent performance with least MSE of 2.58 with pruned decision tree model followed by 2.78 for the random forest model.

Our findings are significant as they affirm that both decision tree models and ensemble techniques are robust tools for predicting whether an individual might engage in alcohol consumption. Additional studies could expand on this work by incorporating more diverse datasets or by applying these techniques to predict other types of substance use and behavioral outcomes.

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

References:

1. Substance Abuse and Mental Health Services Administration. (2020). National Survey on Drug Use and Health (NSDUH) 2020. [Codebook]. Retrieved from https://www.datafiles.samhsa.gov/sites/default/files/field-uploads-protected/studies/NSDUH2020/NSDUH-2020-datasets/NSDUH-2020-DS0001/NSDUH-2020-DS0001-info/NSDUH2020-DS0001-info-codebook.pdf.
2. James, G., Witten, D., Hastie, T., Tibshirani, R., & Taylor, J. (2023). An Introduction to Statistical Learning with Applications in Python. (Original work published 2023) https://hastie.su.domains/ISLP/ISLP_website.pdf.download.html

In [285…
```
# @title Loading libraries
import pandas as pd
import numpy as np
# import missingno as msno
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import preprocessing
from sklearn.impute import SimpleImputer
import matplotlib.pyplot as plt

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

In [286…
```
import pandas as pd

df = pd.read_csv('youth_data.csv')
df.columns = df.columns.str.upper()
df.head()
```

Out[286]:

	IRALCFY	IRMJFY	IRCIGFM	IRSMKLSS30N	IRALCFM	IRMJFM	IRCIGAGE	IRSMKLSSTRY	IRAL
0	993	991	91	91	93.0	91.0	991	991	
1	991	991	91	91	91.0	91.0	991	991	
2	993	993	93	91	93.0	93.0	13	991	
3	991	991	91	91	91.0	91.0	991	991	
4	991	991	91	91	91.0	91.0	991	991	

5 rows × 79 columns

In [287…
```
# missing values in each column
missing_values = df.isnull().sum()
# print(missing_values[missing_values > 0])
```

In [288…
```
# dropping rows null values from youth experiences columns.
youth_exp_indicators = ['AVGGRADE', 'ARGUPAR', 'SCHFELT', 'TCHGJOB', 'STNDSCIG', 'S
                    'STNDDNK', 'PARCHKHW', 'PARHLPHW', 'PRCHORE2', 'PRLMTTV2', 'PARL
                    'PRGDJOB2', 'PRPROUD2', 'YOFIGHT2', 'YOGRPFT2', 'YOHGUN2', 'YOSE
                    'YOSTOLE2', 'YOATTAK2', 'PRPKCIG2', 'PRMJEVR2', 'PRMJMO','PRALDL
                      'YFLTMRJ2', 'YFLMJMO', 'YFLADLY2', 'FRDPCIG2', 'FRDMEVR2', 'FF
                      'TALKPROB', 'PRTALK3', 'PRBSOLV2', 'PREVIOL2', 'PRVDRGO2', 'GF
                      'YTHACT2', 'DRPRVME3', 'ANYEDUC3', 'RLGATTD', 'RLGIMPT', 'RLGD

df.dropna(subset=youth_exp_indicators, inplace=True)
df.shape
```

Out[288]: (4269, 79)

In [289…
```
# dropping rows where ifather or imother is 3,4 meaning the youth is either 18 or t
# same with eduschgrd2, eduskcom, droppping rows where response was legitimately sk
# pden10 is fitered to consider responses from densely populated areas or urban are

df = (
    df[~df['IMOTHER'].isin([3, 4])]
    [~df['IFATHER'].isin([3, 4])]
    [~df['EDUSCHGRD2'].isin([98, 99])]
    [~df['EDUSKPCOM'].isin([94, 97, 98, 99])]
    [df['PDEN10'] != 3]
).reset_index(drop=True)

df.shape
```

```
C:\Users\syeda.fatima\AppData\Local\Temp\ipykernel_26524\3931605009.py:6: UserWarn
ing: Boolean Series key will be reindexed to match DataFrame index.
  df[~df['IMOTHER'].isin([3, 4])]
C:\Users\syeda.fatima\AppData\Local\Temp\ipykernel_26524\3931605009.py:6: UserWarn
ing: Boolean Series key will be reindexed to match DataFrame index.
  df[~df['IMOTHER'].isin([3, 4])]
C:\Users\syeda.fatima\AppData\Local\Temp\ipykernel_26524\3931605009.py:6: UserWarn
ing: Boolean Series key will be reindexed to match DataFrame index.
  df[~df['IMOTHER'].isin([3, 4])]
C:\Users\syeda.fatima\AppData\Local\Temp\ipykernel_26524\3931605009.py:6: UserWarn
ing: Boolean Series key will be reindexed to match DataFrame index.
  df[~df['IMOTHER'].isin([3, 4])]
```

Out[289]: (3168, 79)

In [290…
```
demographic_indicators = ['IRSEX', 'NEWRACE2', 'HEALTH2', 'EDUSCHLGO', 'EDUSCHGRD2'
                          'IMOTHER', 'IFATHER', 'INCOME', 'GOVTPROG', 'POVERTY3', 'PDEN1(
```

Binary classification

In [291…
```
required_columns = demographic_indicators + youth_exp_indicators + ['ALCFLAG']
df_b = df[required_columns]

print(df_b.shape)
```

```
(3168, 61)
```

preparing data

In [292…
```
X = df_b.drop('ALCFLAG', axis=1)
y = df_b['ALCFLAG']

# Split the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

decision tree model

In [293…
```
tree = DecisionTreeClassifier(random_state=1)
tree.fit(X_train, y_train)

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

```
Accuracy of decision tree model: 72.24%
```

In [294…

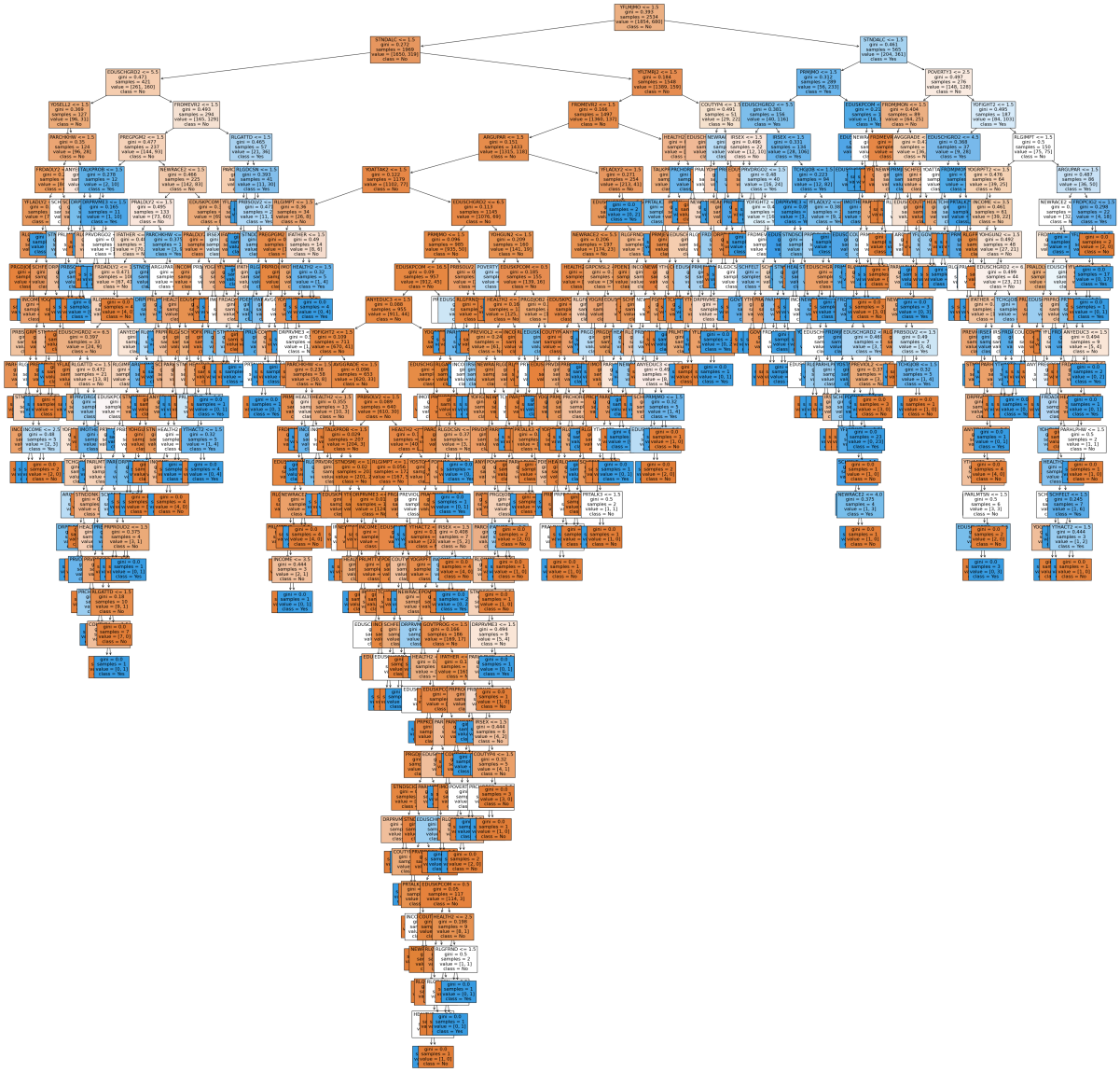
```
features_importance = pd.DataFrame({'feature_name': X_train.columns, 'importance':
features_importance = features_importance.sort_values('importance', ascending=False
features_importance.head(10)
```

Out[294]:

	feature_name	importance
0	YFLMJMO	0.200715
1	STNDALC	0.084447
2	EDUSCHGRD2	0.059864
3	NEWRACE2	0.032205
4	HEALTH2	0.029403
5	INCOME	0.025132
6	IRSEX	0.024186
7	SCHFELT	0.021946
8	EDUSKPCOM	0.021881
9	PRBSOLV2	0.020175

In [295…

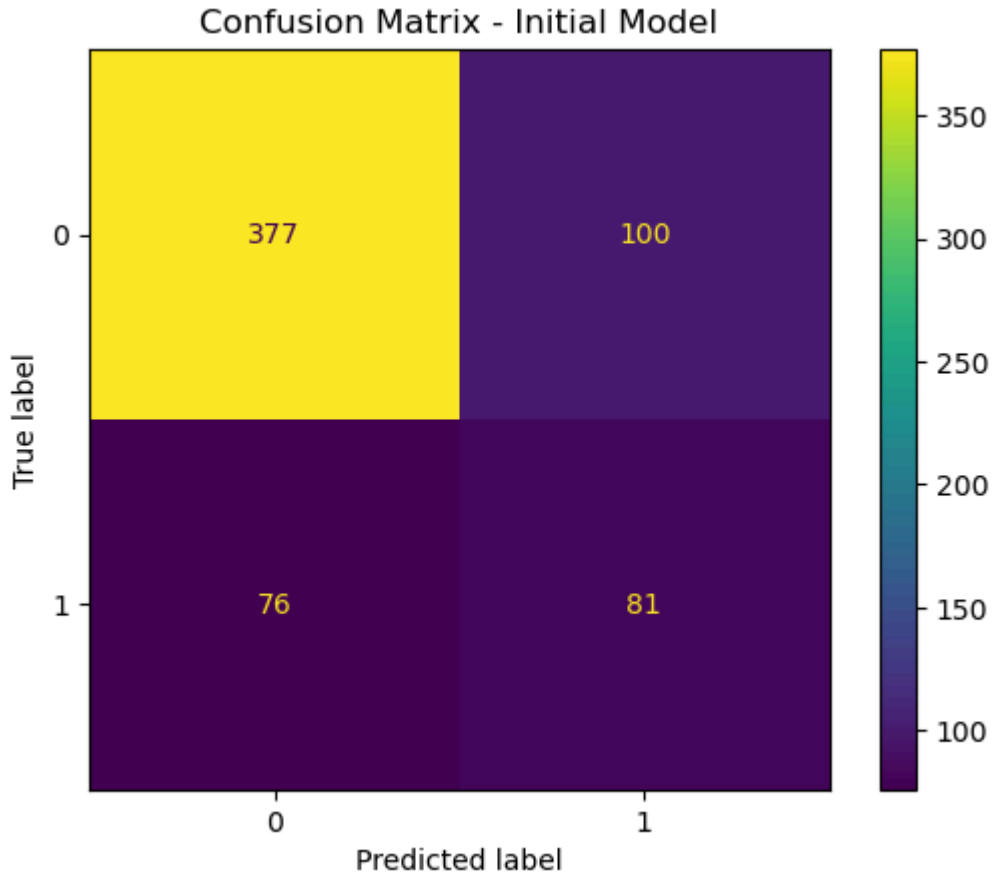
```
plt.figure(figsize=(50,50))
plot_tree(tree
          , filled=True
          , feature_names=X_train.columns
          , class_names=['No', 'Yes']
          , label='all'
          , fontsize=12)
plt.show()
```



The high number of nodes and branches indicates that the model is likely overfitting

In [296…

```
y_pred_tree = tree.predict(X_test)

tree_cm = ConfusionMatrixDisplay(confusion_matrix=confusion_matrix(y_test, y_pred_t
tree_cm.plot()
plt.title("Confusion Matrix - Initial Model")
plt.show()
```



Pruned tree

Let's try pruning by fiding optimal size of the tree

In [97]:
```
tree_ = DecisionTreeClassifier(random_state=1)
tree_.fit(X_train, y_train)

params = {'max_leaf_nodes': range(2, 25)}
cv = GridSearchCV(tree_, params, cv=10)
cv.fit(X_train, y_train)
cv_results = cv.cv_results_
best_size = cv.best_params_['max_leaf_nodes']
best_score = cv.best_score_

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

In [98]:
```
print('Best tree size', best_size)
```

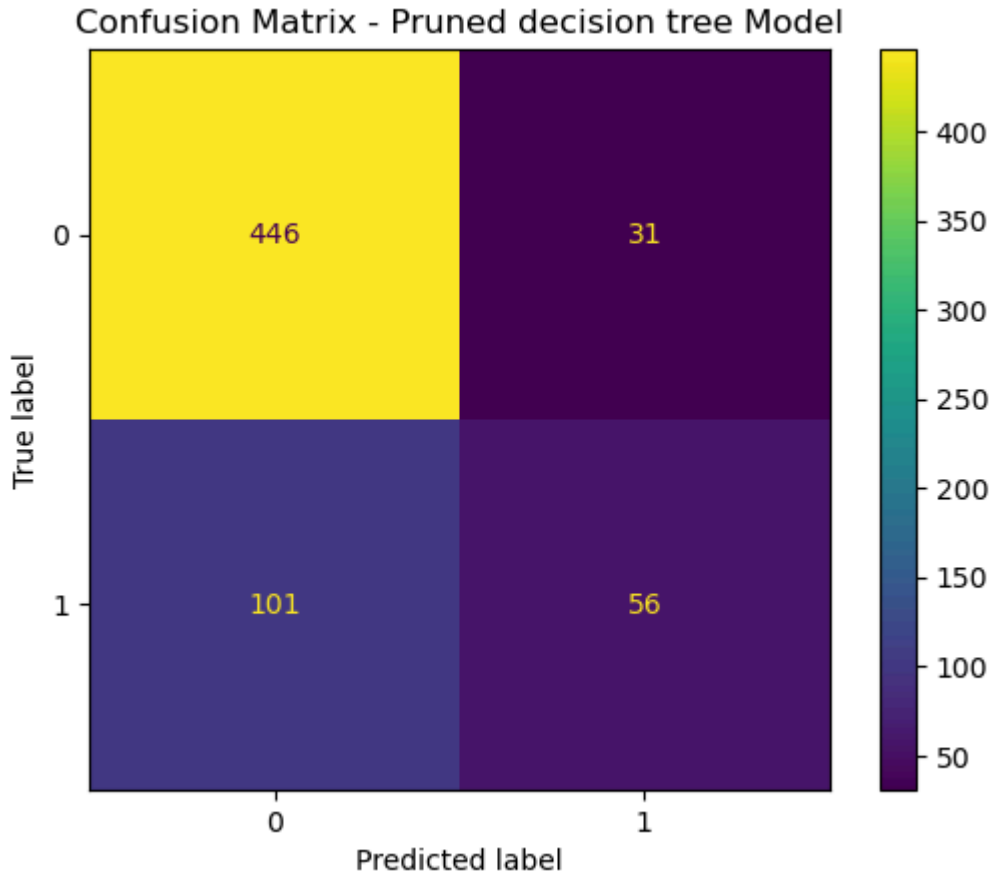
```
Best tree size 16
```

stable and consistent increase in accuracy with tree size around 15-16

In [297…
```
# prune tree with the optimal size
prune_tree = DecisionTreeClassifier(max_leaf_nodes=15, random_state=1)
prune_tree.fit(X_train, y_train)

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

```
Accuracy: 79.18%
```

In [298…
```
y_pred_prune = prune_tree.predict(X_test)
prune_cm = ConfusionMatrixDisplay(confusion_matrix=confusion_matrix(y_test, y_pred_
prune_cm.plot()
plt.title("Confusion Matrix - Pruned decision tree Model")
plt.show()
```

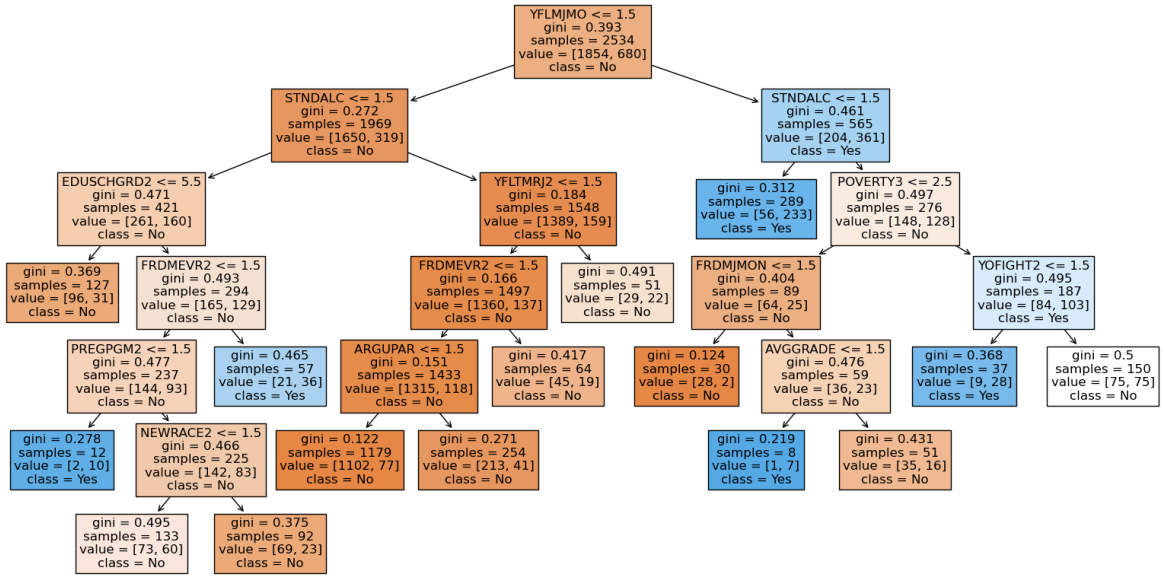


In [109…
```
features_importance = pd.DataFrame({'feature_name': X_train.columns, 'importance':
features_importance = features_importance.sort_values('importance', ascending=False
features_importance.head(10)
```

Out[109]:

	feature_name	importance
0	YFLMJMO	0.575321
1	STNDALC	0.242056
2	YFLTMRJ2	0.032819
3	FRDMEVR2	0.031388
4	POVERTY3	0.025308
5	EDUSCHGRD2	0.019366
6	PREGPGM2	0.014158
7	NEWRACE2	0.012674
8	AVGGRADE	0.012551
9	FRDMJMON	0.011966

In [110…
```
plt.figure(figsize=(20,10))
plt.title('Pruned Tree')
plot_tree(prune_tree
          , filled=True
          , feature_names=X_train.columns
          , class_names=['No', 'Yes']
          , label='all'
          , fontsize=12)
plt.show()
```



In [ ]:
```
tree = ConfusionMatrixDisplay(confusion_matrix=confusion_matrix(y_test, y_pred))
bag.plot()
plt.title("Confusion Matrix - Initial Model")
plt.show()
```

Random Forest

finding optimal number of features to use

In [125…
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

param_grid = {
    'max_features': [5, 10, 15, 20, 'auto', 'sqrt', 'log2']
}

rf = RandomForestClassifier(random_state=1)
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5, scoring='accu
grid_search.fit(X_train, y_train)
best_params = grid_search.best_params_
print("Best max_features parameter:", best_params['max_features'])
```

```
c:\Users\syeda.fatima\anaconda3\Lib\site-packages\sklearn\ensemble\_forest.py:424:
FutureWarning: `max_features='auto'` has been deprecated in 1.1 and will be remove
d in 1.3. To keep the past behaviour, explicitly set `max_features='sqrt'` or remo
ve this parameter as it is also the default value for RandomForestClassifiers and
ExtraTreesClassifiers.
  warn(
c:\Users\syeda.fatima\anaconda3\Lib\site-packages\sklearn\ensemble\_forest.py:424:
FutureWarning: `max_features='auto'` has been deprecated in 1.1 and will be remove
d in 1.3. To keep the past behaviour, explicitly set `max_features='sqrt'` or remo
ve this parameter as it is also the default value for RandomForestClassifiers and
ExtraTreesClassifiers.
  warn(
c:\Users\syeda.fatima\anaconda3\Lib\site-packages\sklearn\ensemble\_forest.py:424:
FutureWarning: `max_features='auto'` has been deprecated in 1.1 and will be remove
d in 1.3. To keep the past behaviour, explicitly set `max_features='sqrt'` or remo
ve this parameter as it is also the default value for RandomForestClassifiers and
ExtraTreesClassifiers.
  warn(
c:\Users\syeda.fatima\anaconda3\Lib\site-packages\sklearn\ensemble\_forest.py:424:
FutureWarning: `max_features='auto'` has been deprecated in 1.1 and will be remove
d in 1.3. To keep the past behaviour, explicitly set `max_features='sqrt'` or remo
ve this parameter as it is also the default value for RandomForestClassifiers and
ExtraTreesClassifiers.
  warn(
c:\Users\syeda.fatima\anaconda3\Lib\site-packages\sklearn\ensemble\_forest.py:424:
FutureWarning: `max_features='auto'` has been deprecated in 1.1 and will be remove
d in 1.3. To keep the past behaviour, explicitly set `max_features='sqrt'` or remo
ve this parameter as it is also the default value for RandomForestClassifiers and
ExtraTreesClassifiers.
  warn(
Best max_features parameter: 5
```

In [299…

```
rf = RandomForestClassifier(max_features=5 , random_state = 1)
rf.fit(X_train,y_train)
```

Out[299]:

```
▾ RandomForestClassifier
RandomForestClassifier(max_features=5, random_state=1)
```

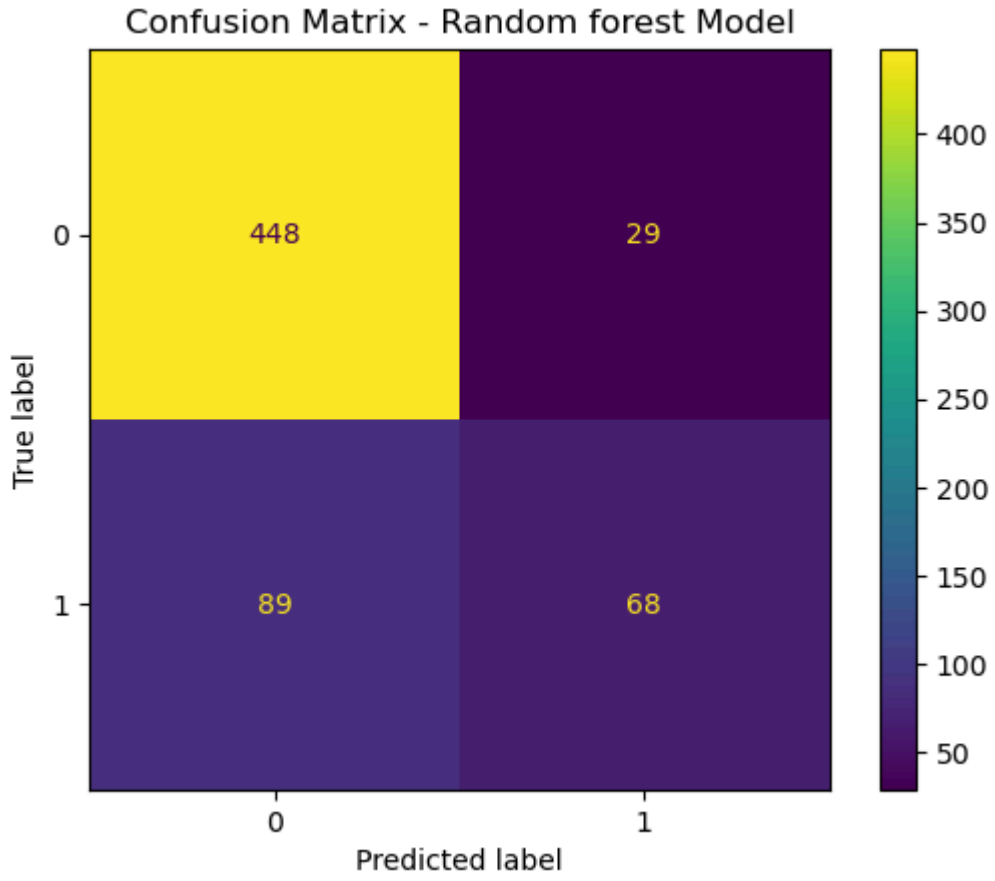
In [300…

```
y_pred_rf = rf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred_rf)
print("Accuracy: {:.2f}%".format(accuracy*100))
```

```
Accuracy: 81.39%
```

In [301…

```
rf_cm = ConfusionMatrixDisplay(confusion_matrix=confusion_matrix(y_test, y_pred_rf)
rf_cm.plot()
plt.title("Confusion Matrix - Random forest Model")
plt.show()
```



In [204…
```
features_importance = pd.DataFrame({'feature_name': X_train.columns, 'importance':
features_importance = features_importance.sort_values('importance', ascending=False
features_importance.head(10)
```

Out[204]:

	feature_name	importance
0	EDUSCHGRD2	0.065209
1	STNDALC	0.061133
2	YFLMJMO	0.052227
3	YFLTMRJ2	0.051950
4	FRDMEVR2	0.038117
5	FRDMJMON	0.033416
6	HEALTH2	0.028782
7	NEWRACE2	0.028071
8	STNDSMJ	0.027839
9	INCOME	0.024851

Bagging

In [132…
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score

# Best max_features and max_depth values found earlier
best_max_features = 20
```

```
best_max_depth = 15

param_grid = {
    'n_estimators': [50, 100, 200],
    'min_samples_split': [2, 5, 10]
}

rf = RandomForestClassifier(max_features=best_max_features, max_depth=best_max_dept

grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5, scoring='accu
grid_search.fit(X_train, y_train)
best_params = grid_search.best_params_
best_score = grid_search.best_score_

print("Best parameters found: ", best_params)
print("Best cross-validation accuracy: {:.2f}%".format(best_score * 100))
```

```
Best parameters found:  {'min_samples_split': 10, 'n_estimators': 100}
Best cross-validation accuracy: 81.02%
Number of trees: 100
Number of features tried at each split: 20
Training score: 93.33%
Test accuracy: 80.91%
```

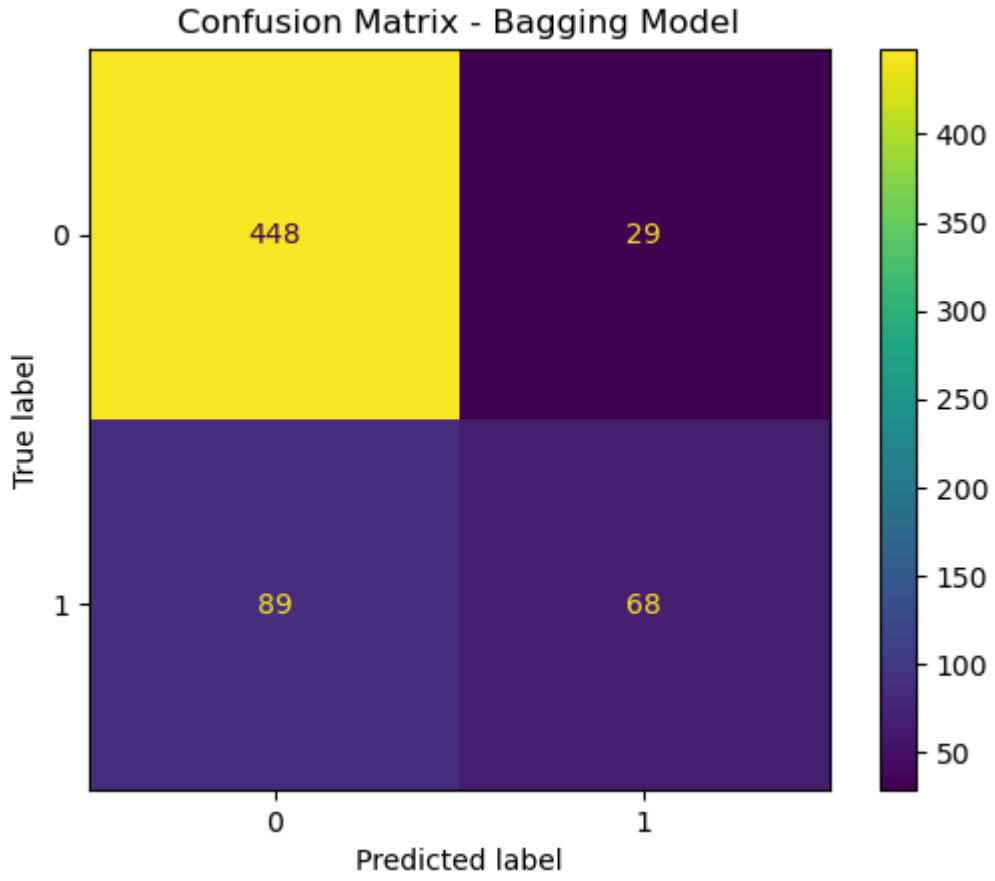
In [302…

```
# Train the model with the best parameters
best_rf = grid_search.best_estimator_
best_rf.fit(X_train, y_train)

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

# Predict on the test data
y_pred = best_rf.predict(X_test)

# Calculate the accuracy of the tuned classifier
accuracy = accuracy_score(y_test, y_pred)
print("Test accuracy: {:.2f}%".format(accuracy * 100))
```

```
Number of trees: 100
Number of features tried at each split: 5
Training score: 100.00%
Test accuracy: 81.39%
```

In [303…

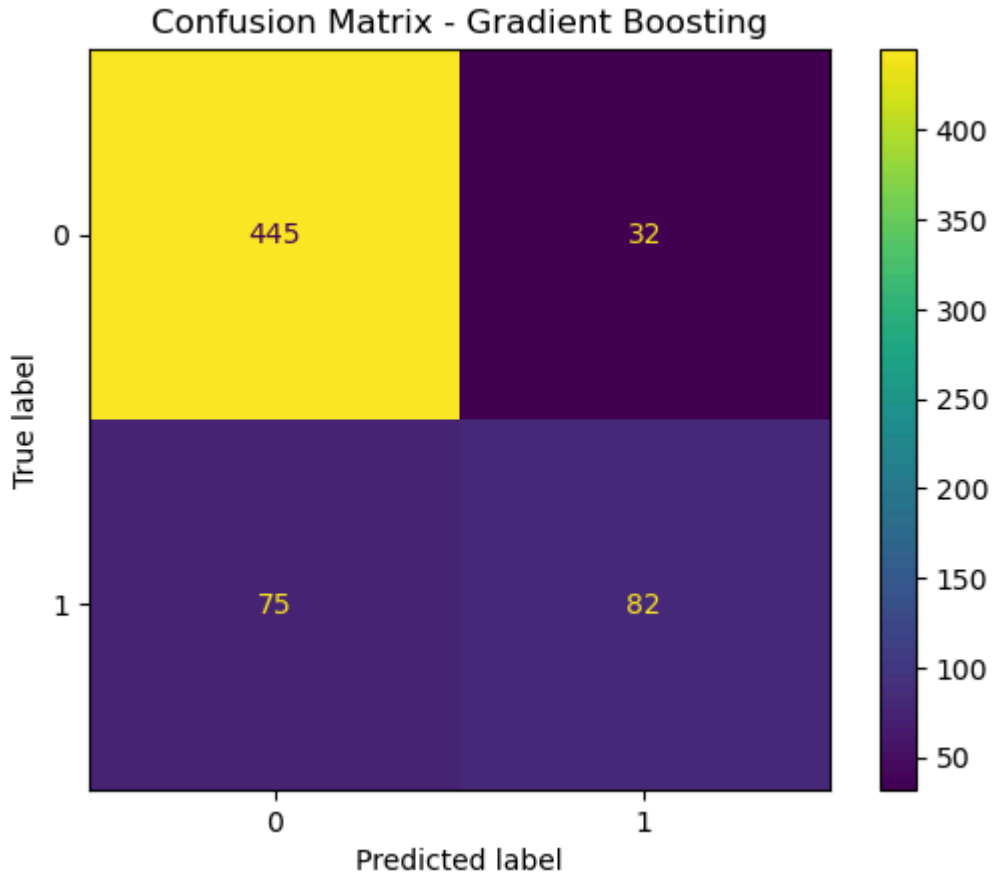
```
bag = ConfusionMatrixDisplay(confusion_matrix=confusion_matrix(y_test, y_pred))
bag.plot()
plt.title("Confusion Matrix - Bagging Model")
plt.show()
```



Boosting

In [148…

```
from sklearn.ensemble import GradientBoostingClassifier

boosting_model = GradientBoostingClassifier(random_state=1)
boosting_model.fit(X_train, y_train)

y_pred_boosting = boosting_model.predict(X_test)

cm_boosting = confusion_matrix(y_test, y_pred_boosting)

boost_cm = ConfusionMatrixDisplay(confusion_matrix=cm_boosting)
boost_cm.plot()
plt.title("Confusion Matrix - Gradient Boosting")
plt.show()

accuracy_boosting = accuracy_score(y_test, y_pred_boosting)
print("Accuracy: {:.2f}%".format(accuracy_boosting * 100))
```




```
Accuracy: 83.12%
```

In [ ]:

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, accuracy_scor
import matplotlib.pyplot as plt

param_grid_boosting = {
    'n_estimators': [50, 100, 200],
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [15],
    'min_samples_split': [10]
}

boosting_model = GradientBoostingClassifier(random_state=1)

grid_search_boosting = GridSearchCV(estimator=boosting_model, param_grid=param_gric
grid_search_boosting.fit(X_train, y_train)

best_params = grid_search_boosting.best_params_
best_score = grid_search_boosting.best_score_

print("Best parameters found: ", best_params)
print("Best cross-validation accuracy: {:.2f}%".format(best_score * 100))

best_boosting = grid_search_boosting.best_estimator_
best_boosting.fit(X_train, y_train)

y_pred_boosting = best_boosting.predict(X_test)

cm_boosting = confusion_matrix(y_test, y_pred_boosting)

disp_boosting = ConfusionMatrixDisplay(confusion_matrix=cm_boosting)
disp_boosting.plot()
plt.title("Confusion Matrix - Tuned Gradient Boosting")
plt.show()
```

```
accuracy_boosting = accuracy_score(y_test, y_pred_boosting)
print("Test accuracy: {:.2f}%".format(accuracy_boosting * 100))
```

Multi Class Classification

In [304…
```
required_columns = demographic_indicators + youth_exp_indicators + ['ALCYDAYS']
df_m = df[required_columns]
print(df_m.shape)
```

```
(3168, 61)
```

In [305…
```
df_m['ALCYDAYS'].value_counts()
```

Out[305]:
```
ALCYDAYS
6    2488
1     420
2     169
3      54
4      37
Name: count, dtype: int64
```

In [306…
```
df_m['ALCYDAYS'] = df_m['ALCYDAYS'].replace({6: 0})

df_m['ALCYDAYS'].value_counts()
```

```
C:\Users\syeda.fatima\AppData\Local\Temp\ipykernel_26524\2676369564.py:1: SettingW
ithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_m['ALCYDAYS'] = df_m['ALCYDAYS'].replace({6: 0})
```

Out[306]:
```
ALCYDAYS
0    2488
1     420
2     169
3      54
4      37
Name: count, dtype: int64
```

In [307…
```
X = df_m.drop('ALCYDAYS', axis=1)
y = df_m['ALCYDAYS']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

decision tree

In [282…
```
tree_m = DecisionTreeClassifier(random_state=1)
tree_m.fit(X_train, y_train)

accuracy = tree_m.score(X_test, y_test)
print("Accuracy of decision tree model: {:.2f}%".format(accuracy*100))
```

```
Accuracy of decision tree model: 68.45%
```

In [165…
```
features_importance = pd.DataFrame({'feature_name': X_train.columns, 'importance':
features_importance = features_importance.sort_values('importance', ascending=False
features_importance.head(10)
```

Out[165]:

	feature_name	importance
0	YFLMJMO	0.129249
1	EDUSCHGRD2	0.053089
2	STNDALC	0.048591
3	EDUSKPCOM	0.039582
4	INCOME	0.029488
5	COUTYP4	0.028077
6	NEWRACE2	0.025900
7	HEALTH2	0.024369
8	TCHGJOB	0.023792
9	RLGATTD	0.021468

In [ ]:
```
plt.figure(figsize=(50,50))
plot_tree(tree
          , filled=True
          , feature_names=X_train.columns
          , class_names=['No', 'Yes']
          , label='all'
          , fontsize=12)
plt.show()
```

pruned

In [179…
```
tree_m = DecisionTreeClassifier(random_state=1)
tree_m.fit(X_train, y_train)

params = {'max_leaf_nodes': range(2, 25)}
cv = GridSearchCV(tree_, params, cv=10)
cv.fit(X_train, y_train)
cv_results = cv.cv_results_
best_size = cv.best_params_['max_leaf_nodes']
best_score = cv.best_score_

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

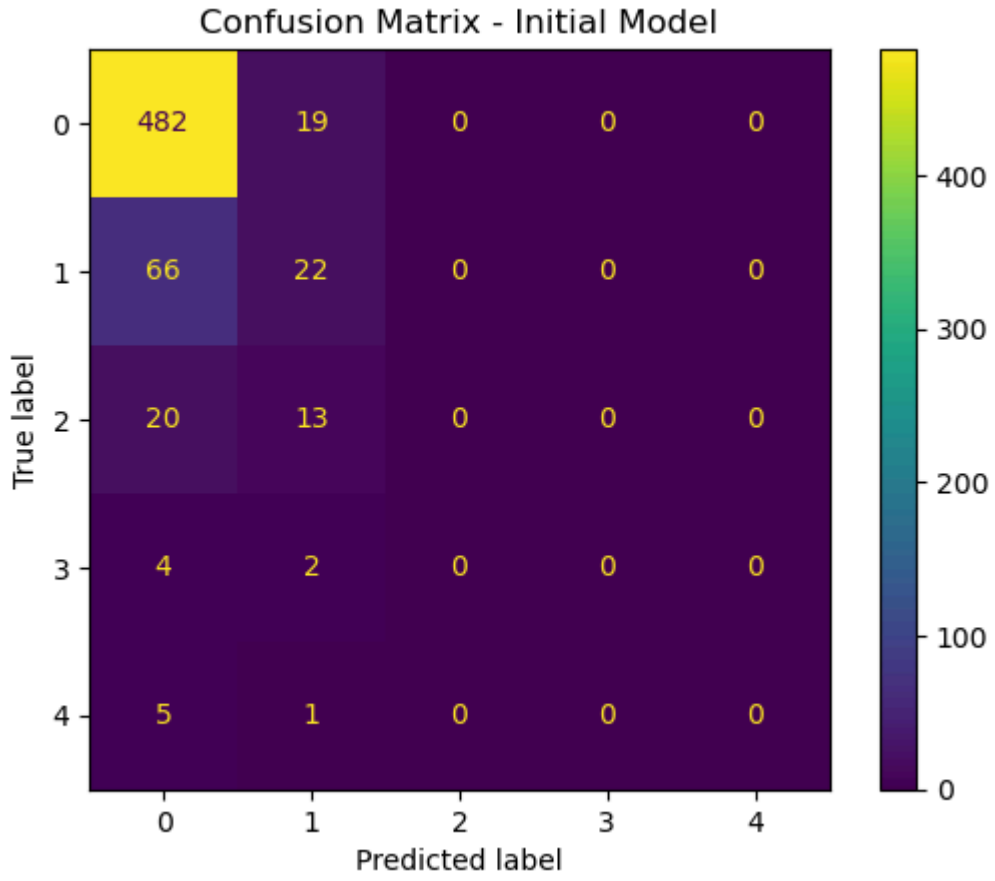
In [195…
```
print('Best tree size', best_size)
```

```
Best tree size 4
```

In [196…
```
# prune tree with the optimal size
prune_tree_m = DecisionTreeClassifier(max_leaf_nodes=4, random_state=1)
prune_tree_m.fit(X_train, y_train)

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

```
Accuracy: 79.50%
```

In [197…
```
y_pred_prune = prune_tree_m.predict(X_test)
prune_cm = ConfusionMatrixDisplay(confusion_matrix=confusion_matrix(y_test, y_pred_
prune_cm.plot()
plt.title("Confusion Matrix - Initial Model")
plt.show()
```

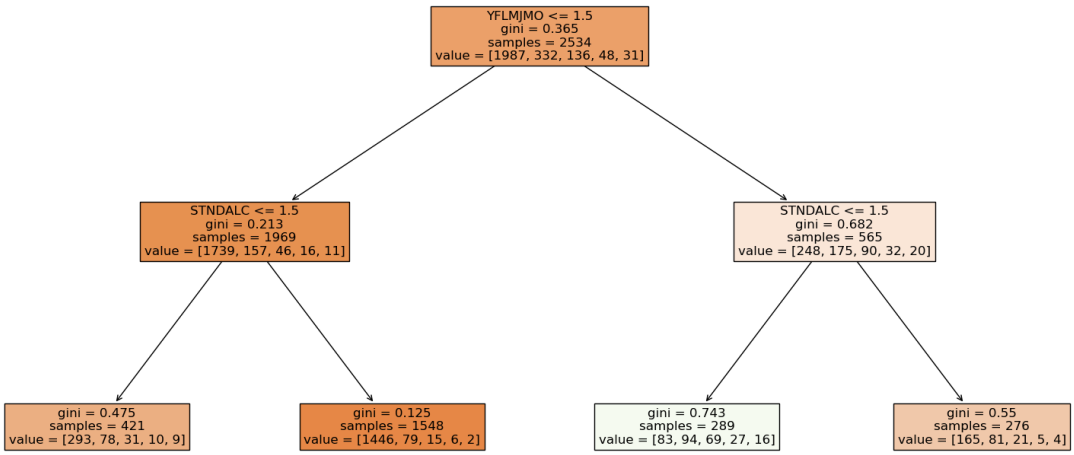


In [198…
```
features_importance = pd.DataFrame({'feature_name': X_train.columns, 'importance':
features_importance = features_importance.sort_values('importance', ascending=False
features_importance.head(10)
```

Out[198]:

	feature_name	importance
0	YFLMJMO	0.72677
1	STNDALC	0.27323
2	IRSEX	0.00000
3	YOSELL2	0.00000
4	YOATTAK2	0.00000
5	PRPKCIG2	0.00000
6	PRMJEVR2	0.00000
7	PRMJMO	0.00000
8	PRALDLY2	0.00000
9	YFLPKCG2	0.00000

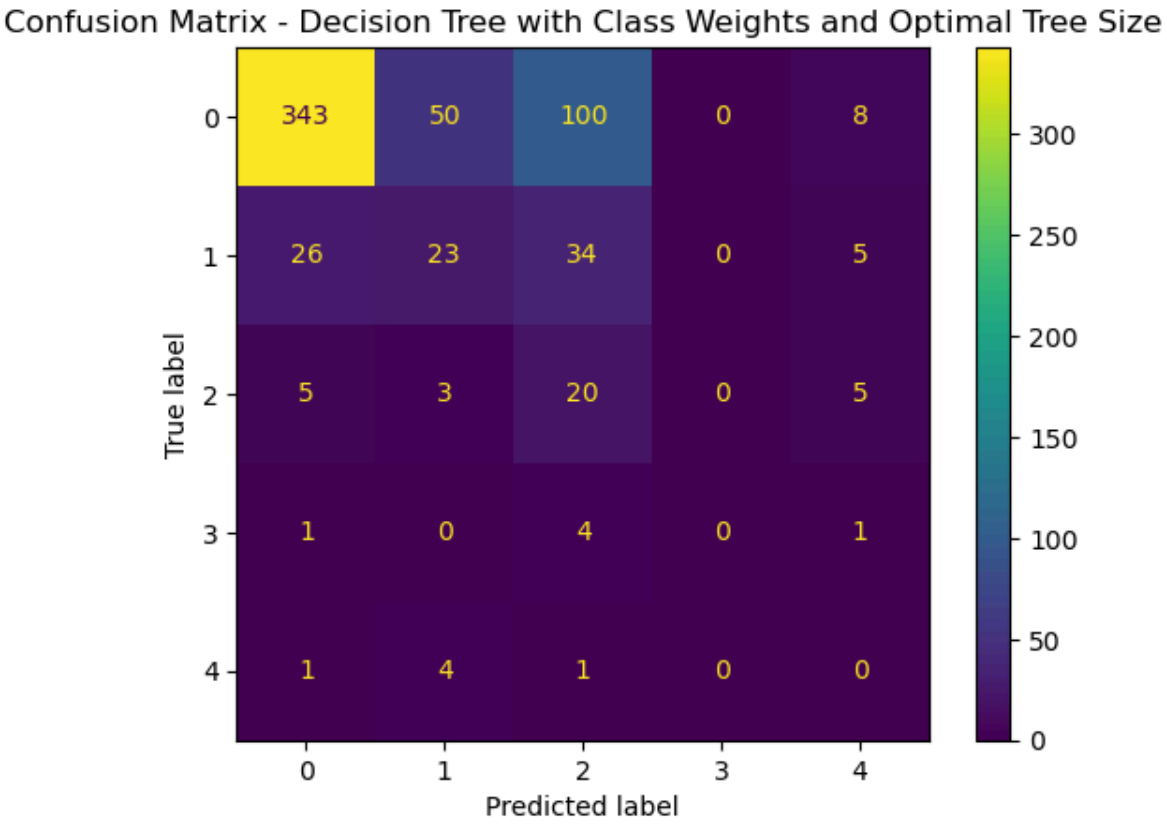
In [199…
```
plt.figure(figsize=(20,10))
plt.title('Pruned Tree')
plot_tree(prune_tree_m
          , filled=True
          , feature_names=X_train.columns
          , label='all'
          , fontsize=12)
plt.show()
```



In [202…

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, ConfusionMatrixDispla
import matplotlib.pyplot as plt

tree_m = DecisionTreeClassifier(class_weight='balanced', max_leaf_nodes=4, random_s
tree_m.fit(X_train, y_train)

y_pred_tree = tree_m.predict(X_test)

accuracy = tree_m.score(X_test, y_test)
print("Accuracy of decision tree model: {:.2f}%".format(accuracy * 100))

cm_tree = confusion_matrix(y_test, y_pred_tree)
disp_tree = ConfusionMatrixDisplay(confusion_matrix=cm_tree)
disp_tree.plot()
plt.title("Confusion Matrix - Decision Tree with Class Weights and Optimal Tree Siz
plt.show()
```

```
Accuracy of decision tree model: 60.88%
```



RandomForest

In [203…

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

param_grid = {
    'max_features': [5, 10, 15, 20, 'auto', 'sqrt', 'log2']
}

rf_m = RandomForestClassifier(random_state=1)
grid_search = GridSearchCV(estimator=rf_m, param_grid=param_grid, cv=5, scoring='ac
grid_search.fit(X_train, y_train)
best_params = grid_search.best_params_
print("Best max_features parameter:", best_params['max_features'])
```

```
c:\Users\syeda.fatima\anaconda3\Lib\site-packages\sklearn\ensemble\_forest.py:424:
FutureWarning: `max_features='auto'` has been deprecated in 1.1 and will be remove
d in 1.3. To keep the past behaviour, explicitly set `max_features='sqrt'` or remo
ve this parameter as it is also the default value for RandomForestClassifiers and
ExtraTreesClassifiers.
  warn(
c:\Users\syeda.fatima\anaconda3\Lib\site-packages\sklearn\ensemble\_forest.py:424:
FutureWarning: `max_features='auto'` has been deprecated in 1.1 and will be remove
d in 1.3. To keep the past behaviour, explicitly set `max_features='sqrt'` or remo
ve this parameter as it is also the default value for RandomForestClassifiers and
ExtraTreesClassifiers.
  warn(
c:\Users\syeda.fatima\anaconda3\Lib\site-packages\sklearn\ensemble\_forest.py:424:
FutureWarning: `max_features='auto'` has been deprecated in 1.1 and will be remove
d in 1.3. To keep the past behaviour, explicitly set `max_features='sqrt'` or remo
ve this parameter as it is also the default value for RandomForestClassifiers and
ExtraTreesClassifiers.
  warn(
c:\Users\syeda.fatima\anaconda3\Lib\site-packages\sklearn\ensemble\_forest.py:424:
FutureWarning: `max_features='auto'` has been deprecated in 1.1 and will be remove
d in 1.3. To keep the past behaviour, explicitly set `max_features='sqrt'` or remo
ve this parameter as it is also the default value for RandomForestClassifiers and
ExtraTreesClassifiers.
  warn(
c:\Users\syeda.fatima\anaconda3\Lib\site-packages\sklearn\ensemble\_forest.py:424:
FutureWarning: `max_features='auto'` has been deprecated in 1.1 and will be remove
d in 1.3. To keep the past behaviour, explicitly set `max_features='sqrt'` or remo
ve this parameter as it is also the default value for RandomForestClassifiers and
ExtraTreesClassifiers.
  warn(
Best max_features parameter: 5
```

In [308…

```
rf_m = RandomForestClassifier(max_features=5 , random_state = 1)
rf_m.fit(X_train,y_train)
```

Out[308]:

```
▾ RandomForestClassifier
RandomForestClassifier(max_features=5, random_state=1)
```

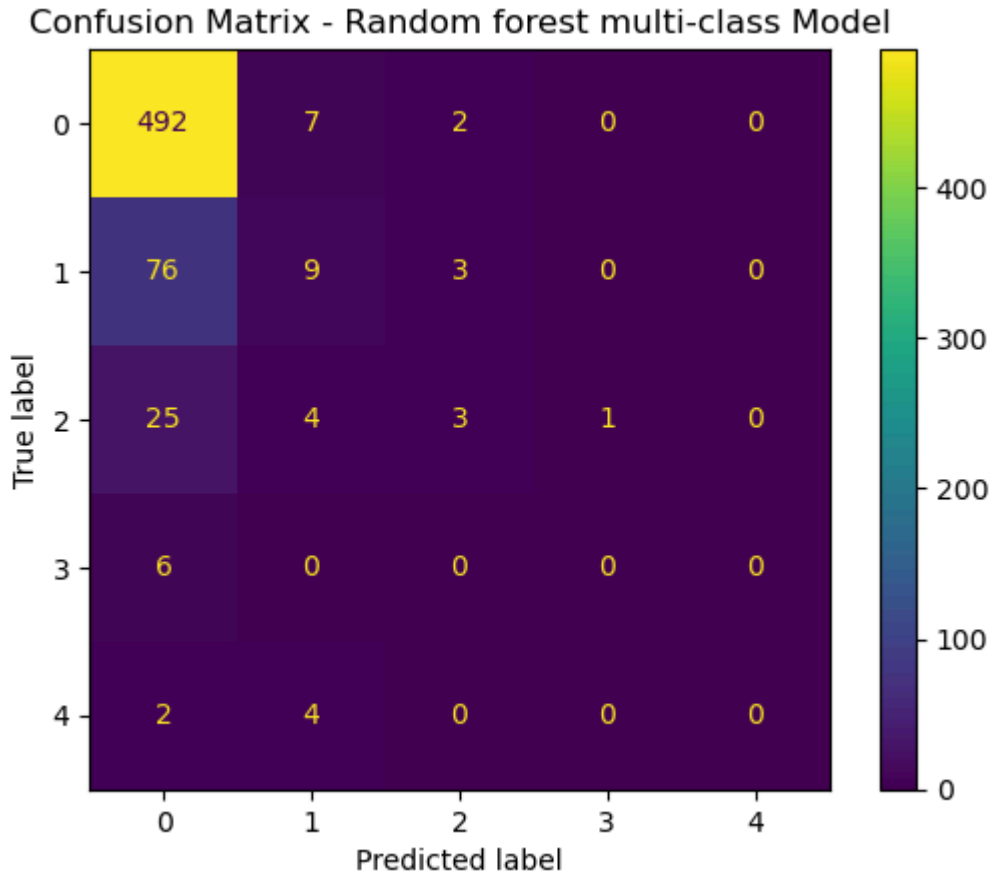
In [309…

```
y_pred_rf = rf_m.predict(X_test)
accuracy = accuracy_score(y_test, y_pred_rf)
print("Accuracy: {:.2f}%".format(accuracy*100))
```

```
Accuracy: 79.50%
```

In [311…

```
rf_m_cm = ConfusionMatrixDisplay(confusion_matrix=confusion_matrix(y_test, y_pred_r
rf_m_cm.plot()
plt.title("Confusion Matrix - Random forest multi-class Model")
plt.show()
```



In [217…
```
features_importance = pd.DataFrame({'feature_name': X_train.columns, 'importance':
features_importance = features_importance.sort_values('importance', ascending=False
features_importance.head(10)
```

Out[217]:

	feature_name	importance
0	EDUSCHGRD2	0.056471
1	STNDALC	0.040401
2	YFLMJMO	0.039615
3	YFLTMRJ2	0.038280
4	FRDMEVR2	0.033848
5	HEALTH2	0.033642
6	NEWRACE2	0.030222
7	FRDMJMON	0.029999
8	INCOME	0.026382
9	EDUSKPCOM	0.025581

In [ ]:
```
features_importance = pd.DataFrame({'feature_name': X_train.columns, 'importance':
features_importance = features_importance.sort_values('importance', ascending=False
features_importance.head(10)
```

boosting

In [218…
```
gb_m = GradientBoostingClassifier(n_estimators=100, learning_rate=0.1, max_depth=4,
```

```
gb_m.fit(X_train, y_train)
```

Out[218]:

GradientBoostingClassifier

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

In [219…

```
y_pred = gb_m.predict(X_test)

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

```
Accuracy: 78.08%
```

In [221…

```
importances = pd.DataFrame({'Feature': X_train.columns, 'Importance': gb_m.feature_
importances= importances.sort_values('Importance', ascending=False).reset_index(drc

importances.head()
```

Out[221]:

	Feature	Importance
0	YFLMJMO	0.168635
1	STNDALC	0.087046
2	EDUSCHGRD2	0.070949
3	YFLTMRJ2	0.041714
4	FRDMJMON	0.038285

Regression

In [226…

```
required_columns = demographic_indicators + youth_exp_indicators + ['IRALCAGE']
df_r = df[required_columns]

print(df_r.shape)
```

```
(3168, 61)
```

In [227…

```
df_r['IRALCAGE'].value_counts()
```

Out[227]:

```
IRALCAGE
991    2331
15      219
14      180
13      137
16      108
12       62
17       36
11       33
10       19
9        12
8        11
7         8
6         7
4         2
2         1
3         1
5         1
Name: count, dtype: int64
```

In [229…
```
df_r = df_r[df_r['IRALCAGE'] >= 7]
df_r.shape
```

Out[229]:
```
(3156, 61)
```

In [231…
```
df_r = df_r[df_r['IRALCAGE'] != 991]
df_r.shape
```

Out[231]:
```
(825, 61)
```

In [252…
```
X = df_r.drop('IRALCAGE', axis=1)
y = df_r['IRALCAGE']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_sta
```

decision tree model

In [253…
```
tree = DecisionTreeRegressor(random_state = 1)
tree.fit(X_train,y_train)

MSE = ((y_test - tree.predict(X_test))**2).mean()
print(MSE)
```

```
5.350806451612903
```

In [254…
```
importances = pd.DataFrame({'feature_name': X_train.columns, 'importance': tree.fea
importances.head()
```

Out[254]:

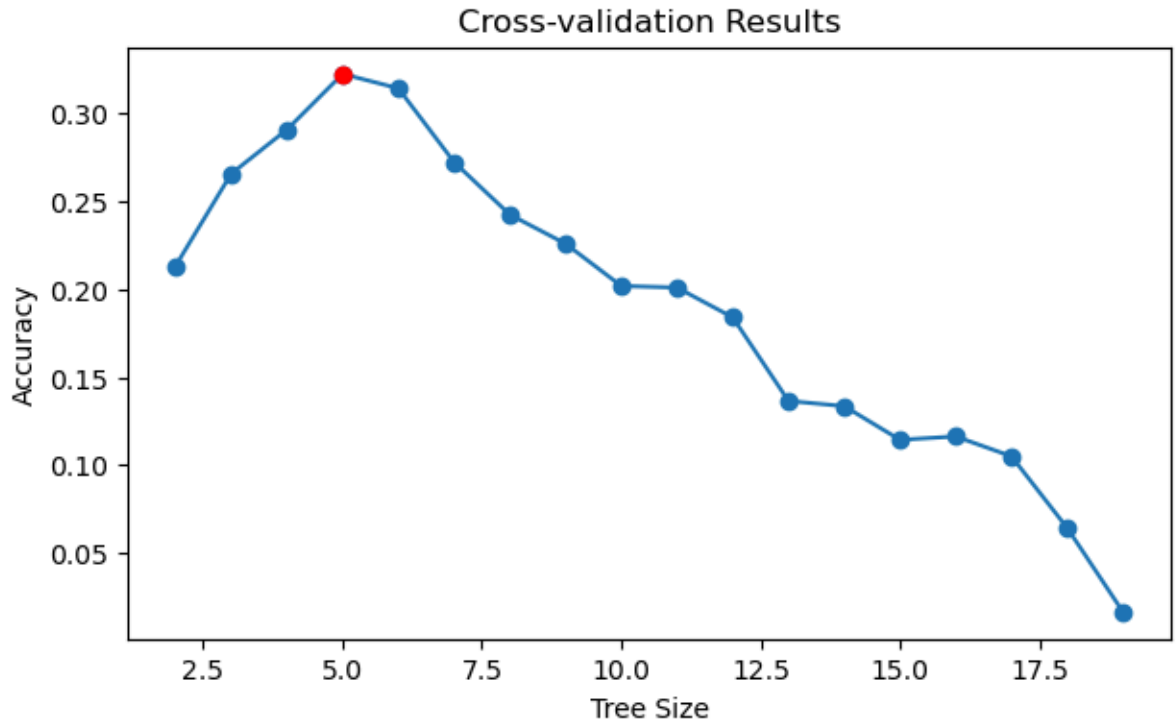
	feature_name	importance
0	EDUSCHGRD2	0.383668
1	PRMJEVR2	0.046209
2	TCHGJOB	0.039820
3	INCOME	0.034177
4	SCHFELT	0.028607

pruning

In [255…
```
tree1 = DecisionTreeRegressor(random_state = 1)
tree1.fit(X_train, y_train)

params = {'max_leaf_nodes': range(2, 20)}
cv_tree1 = GridSearchCV(tree1, params, cv=20)
cv_tree1.fit(X_train, y_train)
cv_results = cv_tree1.cv_results_
best_size = cv_tree1.best_params_['max_leaf_nodes']
best_score = cv_tree1.best_score_

print('Best tree size: ',best_size)

plt.figure(figsize=(7, 4))
plt.plot(cv_results["param_max_leaf_nodes"].data, cv_results["mean_test_score"], 'c
plt.plot(best_size, best_score, 'ro-')
plt.xlabel('Tree Size')
```

```
plt.ylabel('Accuracy')
plt.title('Cross-validation Results');
```

```
Best tree size:  5
```



In [256…

```
prune_tree = DecisionTreeRegressor(random_state = 1, max_leaf_nodes = best_size)
prune_tree.fit(X_train, y_train)

MSE = ((y_test - prune_tree.predict(X_test))**2).mean()
print(MSE)
```

```
2.5254488398977766
```

In [257…

```
importances = pd.DataFrame({'feature_name': X_train.columns, 'importance': prune_tr
importances.head()
```

Out[257]:

	feature_name	importance
0	EDUSCHGRD2	0.929721
1	PRMJEVR2	0.070279
2	IRSEX	0.000000
3	FRDADLY2	0.000000
4	YOATTAK2	0.000000

random forest

In [259…

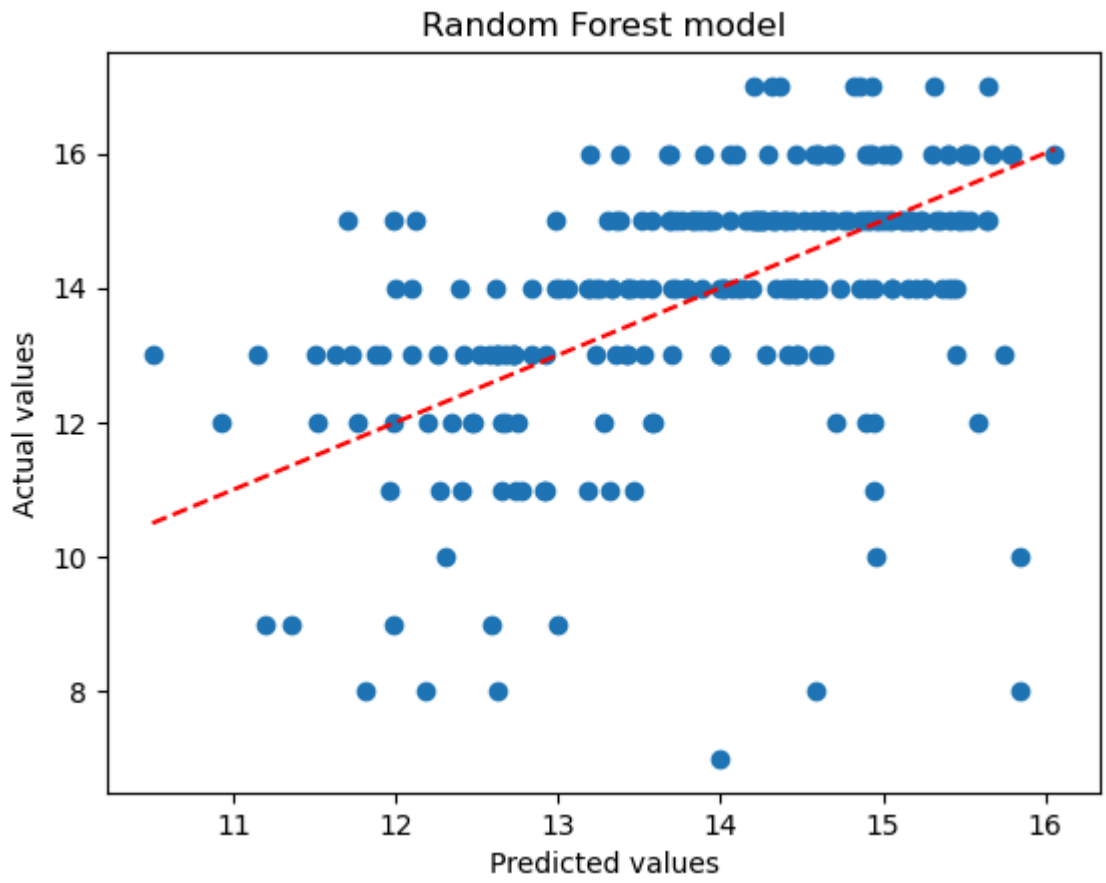
```
rf_reg = RandomForestRegressor(max_features=20 , random_state = 1)
rf_reg.fit(X_train,y_train)
y_pred_rf = rf_reg.predict(X_test)
print("Mean Squared Error: {:.2f}".format(mean_squared_error(y_test, y_pred_rf)))
```

```
Mean Squared Error: 2.78
```

In [260…

```
plt.scatter(y_pred_rf, y_test)
plt.plot([min(y_pred_rf), max(y_pred_rf)], [min(y_pred_rf), max(y_pred_rf)], 'r--')
plt.xlabel('Predicted values')
```

```
plt.ylabel('Actual values')
plt.title('Random Forest model');
```



In [261…

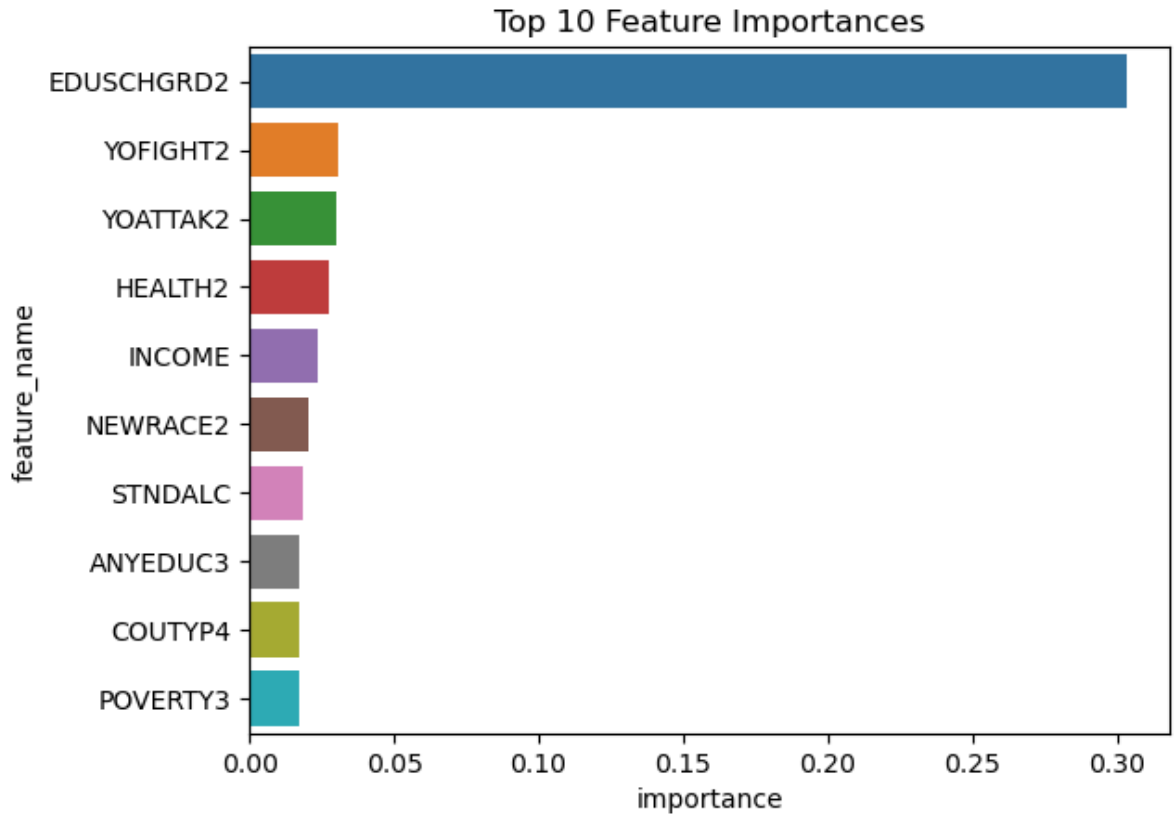
```
importances = pd.DataFrame({'feature_name': X_train.columns, 'importance': rf_reg.f
importances.head()
```

Out[261]:

	feature_name	importance
0	EDUSCHGRD2	0.302927
1	YOFIGHT2	0.030462
2	YOATTAK2	0.030217
3	HEALTH2	0.027298
4	INCOME	0.023916

In [268…

```
rf_importance_features = importances.iloc[:10]

sns.barplot(x='importance', y='feature_name', data=rf_importance_features)
plt.title('Top 10 Feature Importances')
plt.show()
```



boosting

In [271…
```
boost_reg = GradientBoostingRegressor(n_estimators=100, max_depth=4
                                      , random_state=1)
boost_reg.fit(X_train, y_train)

y_pred_boost = boost_reg.predict(X_test)
# find the MSE
print("Mean Squared Error: {:.2f}".format(mean_squared_error(y_test, y_pred_boost))
```

```
Mean Squared Error: 3.06
```

In [273…
```
importances = pd.DataFrame({'feature_name': X_train.columns, 'importance': boost_re
importances.head(10)
```

Out[273]:

	feature_name	importance
0	EDUSCHGRD2	0.438941
1	PRMJEVR2	0.030850
2	YOSELL2	0.027600
3	YOATTAK2	0.022530
4	INCOME	0.021348
5	NEWRACE2	0.020591
6	PARCHKHW	0.019623
7	PRPKCIG2	0.019067
8	YOHGUN2	0.018249
9	PRGDJOB2	0.017008

In [ ]: