Investigation of factors correlated with youth drug use

Name: Lakshit Gupta
Course: DATA 5322-01

Abstract

Accurate knowledge of the variables influencing teen substance use is necessary to address the issue of the threat of substance use by teenagers . This study attempts to evaluate the effectiveness of tree-based models in predicting patterns of teen substance use by utilizing data from the National Survey on Drug Use and Health (NSDUH).In order to predict whether teenagers have ever consumed alcohol, the research uses a classification approach. Random forest yields the best results in this regard, with an accuracy of 83.60%. The number of days that marijuana was consumed in the previous month is then divided into different groups according to how frequently it was consumed using a pruned decision tree model, which produces an accuracy of 93.23%. Furthermore, a random forest regressor model with a test MSE of 2.582413 is utilized to forecast the age at which a person begins to use smokeless tobacco.

Introduction

Teenage drug use has a substantial negative influence on the general well-being of society as well as the health of the individual. Our study explores the variables associated with drug use among young people using data from the National Survey on Drug Use and Health (NSDUH). The NSDUH provides a comprehensive dataset that includes information on youth experiences, drug-related behaviors, and demographics.

Our aim is to build predictive models for teenage drug use across a range of instances by utilizing decision tree algorithms, including ensemble methods. In addition to examining the effects of various data types (binary, ordinal, and numerical) on predictive outcomes, our goal is to identify the critical variables linked to drug use among young people. We will also investigate the decision tree model's understanding and identify important variables for drug use prediction.

For the binary classification, I aimed to predict whether an individual has ever used alcohol or not. In the multi-class classification task, I am predicting the number of days an individual has used marijuana in the past month. And for the regression, I am predicting the age of first use of smokeless tobacco.

Theoretical Background

- **Decision Tree Model:**

  Decision tree model is a particularly useful tool that can be applied to both regression and classification tasks. Its operation is based on a recursive procedure known as recursive binary splitting, in which the target variable is generated into uniform subsets by dividing the feature space according to feature values. The model chooses the best splits iteratively to maximize information gain or minimize impurity, starting at the root node. In this case, a decision based on a feature is represented by each internal node, and a predicted result is represented by each leaf node.
  However, decision trees are prone to overfitting, especially when working with large feature datasets. Overfitting happens when the model's ability to generalize to new data fails by capturing noise or unimportant patterns from the training set. In order to address this problem, the model is made simpler and less complex while maintaining predictive performance through the use of tuning strategies like pruning and tree size control.

- **Pruning:**

  Pruning is an approach used to reduce overfitting in decision trees by removing parts of the tree that are superfluous or redundant and do not add much to the improvement of prediction accuracy. This method involves enhancing the model's interpretability, simplifying the model, and cutting off specific branches of the tree. Parameter tuning techniques like cross-validation and cost-complexity pruning are frequently used to determine the ideal tree size and identify the nodes that should be pruned. Cross-validation helps identify the optimal parameter settings to make a balance between bias and variance.

- **Bagging:**

  Bagging is a method that was developed from Bootstrap Aggregating and uses multiple models that were trained on bootstrap samples of the training data to increase the stability and accuracy of decision trees. Bagging generates a variety of datasets for training distinct decision tree models by resampling with replacement to produce multiple bootstrap samples. In the case of regression, the final prediction is obtained by averaging the predictions of each individual tree; in the case of classification, majority voting is used. Bagging contributes to the reduction of the model's variance and can result in improved performance, especially when working with highly variable datasets. Important hyperparameters for the bagging method include n_estimators (number of trees) and max_features (maximum features considered per split). Tuning these parameters is crucial for optimizing performance and preventing over/underfitting.

- **Random Forest:**

  To further improve model performance, Random Forest adds randomness to the feature selection process, building on the ideas of bagging with decision trees. In addition to producing bootstrap samples, Random Forest reduces the correlation between individual trees and minimizes overfitting by taking into account only a random subset of features at each split point. Random Forest is an efficient method for handling high-dimensional datasets with lots of features because it achieves high predictive accuracy and robustness to noisy data by averaging predictions from multiple diverse trees. The hyperparameters that we can tune include n_estimators (number of trees) and mtry (number of features considered per split).

- **Boosting:**

  Boosting is an additional ensemble technique that builds a strong predictive model by combining several weak learners one after the other. Unlike bagging, where every model is trained separately, boosting trains learners in order, with each new learner trying to correct the mistakes made by the ones before it. A well-known boosting algorithm called gradient boosting iteratively optimizes a loss function to produce a strong learner by teaching each tree to predict the residuals (errors) of the previous trees. Boosting is prone to overfitting, however, so optimizing performance requires fine-tuning hyper parameters like learning rate and tree count. Tuning the learning rate (shrinkage) and the number of estimators (trees) are crucial for gradient boosting models. A lower learning rate will require more trees to achieve the same level of performance, but it can help prevent overfitting by making smaller changes to the model. The optimal number of trees depends on the learning rate, with higher learning rates requiring fewer trees and vice versa.

Methodology

Filtering the records and adjusting variables to their proper data types are the first steps in data preprocessing. Defining variable labels as a dictionary was the next step, assigning readable names to the variables as our dataframe columns. I also converted categorical variables into factors, differentiating it between unordered and ordered factor columns to ensure proper handling during analysis. The next step was to remove the missing values in the categorical variables. Furthermore, I removed row values from specific columns that I considered as irrelevant for the dataset and the analysis to be done on that. After completing that I got the preprocessed data with 3168 rows and 79 unique columns.

Trained Models:

- Decision Tree
- Pruned Tree
- Bagging
- Random Forest
- Gradient Boosting

Binary Classification:

I aimed to predict whether an individual teenager has ever consumed alcohol or not. I took ‘alcflag’ as my target variable for predicting that. For getting the same reproducible result I am taking the random state as 123 and then dividing the dataset into training and testing with 70% for the training data and 30% for the testing data. Then I have implemented multiple models like decision tree, pruned decision tree, random forest, bagging and boosting. Then the best model was considered which had the most accuracy with the testing data. Next I performed hyperparameter tuning on the best model to check with what value of tuned parameters the specific model performs the best.

Multi-Class Classification:

In order to forecast the number of days that marijuana was consumed in the preceding month. Certain models for multi-class classification were created like decision tree, pruned decision tree, random forest and some more ensemble methods.
I took ‘mrjmdays’ as my target variable for aiming to predict within which category the person falls who has consumed marijuana.
The target variable is separated into five categories, each of which is described below:

- 1 = 1-2 Days
- 2 = 3-5 Days
- 3 = 6-19 Days
- 4 = 20-30 Days
- 5 = Non User or No Past Month Use

The model with the most accuracy was considered the best in predicting the correct class under which the individual falls. Then further analysis was done by tuning some different parameters like setting up the number of leaf nodes and checking the accuracies on that. Also for checking the performance of the model cross validation was performed on the model.

Regression:

In regression my main aim was to predict the age of first use of smokeless tobacco by an individual. For that I trained 5 different models after taking the target variable as ‘irsmklsstry’. The range of the target variable goes from 1 to 70 and 991 being the never used one. The best model

was considered which had the lowest test MSE value. After selecting the best model then I performed hyperparameter tuning on that model by taking multiple ranges of values of variable features to check with what number of features the model can perform the best.

Computational Results

- Binary Classification: Alcohol ever used

  We used a variety of models in our binary classification analysis to determine whether or not teenagers have ever drank alcohol. The random forest algorithm turned out to be the best of these models. After analysis, it was determined that “youth feeling about their peers using marijuana” (YFLTMRJ2), “student grade in which they drank alcohol” (stndalc) and “youth feeling about their peers using marijuana monthly” (yflmjmo) were important factors affecting the prediction. We used decision tree methodology and optimization pruning techniques to try to improve our predictions as well.

Model	Accuracy
Decision Tree	73.29%
Pruned Decision Tree	79.21%
Bagging	82.44%
Random Forest	83.60%
Boosting	81.70%

From the above table we can see that the Random Forest model performed the best with an accuracy of 83.60% followed by the bagging method which had an accuracy of 82.44%. Carrying on the workings of the random forest model, we used 500 estimators and set the maximum number of features to 30, and for reproducibility we used a random seed of 1.

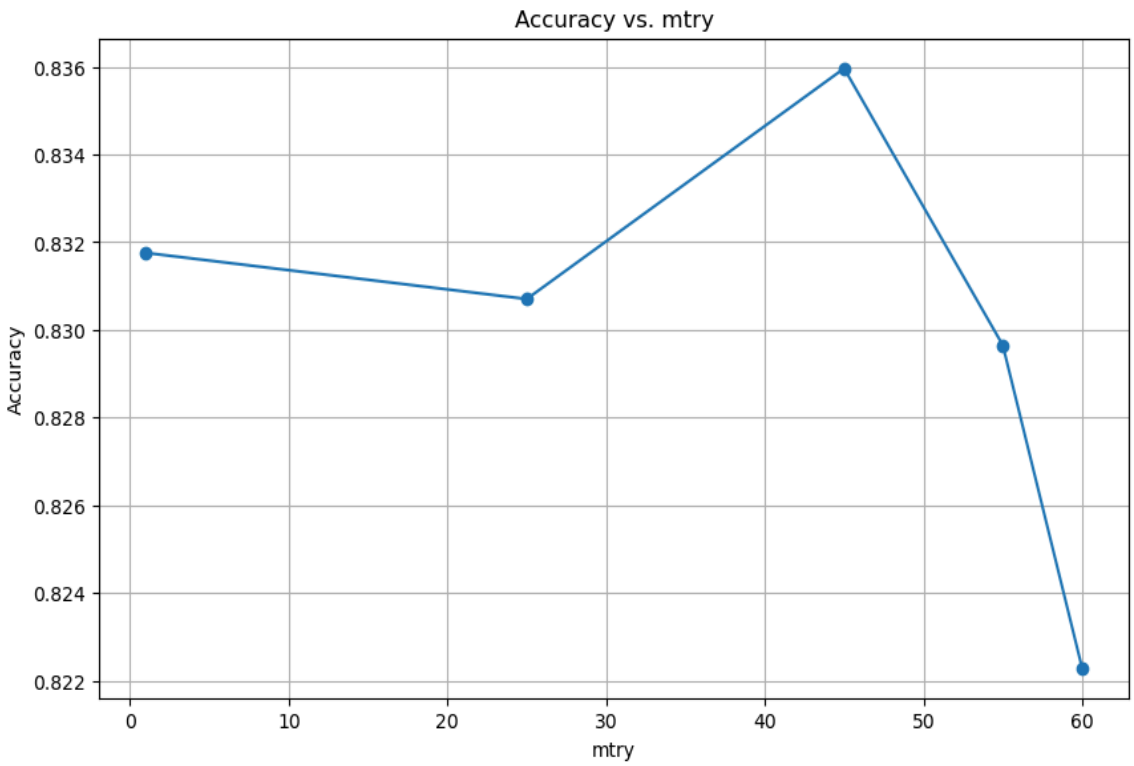

Diagram 1. Plot for Accuracy vs mtry values

Furthermore, as above we conducted experiments with different values of the mtry parameter, which controls the number of features randomly chosen at each split in the random forest algorithm. From the above we can see that the model achieved its highest accuracy of 83.60% when mtry was set to 45, indicating the optimal value for this parameter. This helped us in optimizing the performance and helped us gain some insights about the model.

- Multi-class classification: Number of days of marijuana in past month

  The objective in multi-class classification is to estimate the number of days a teen has used marijuana in the previous month. For this, we used five different models, each with its own performance levels and methods.

  With an accuracy of 93.23%, the pruned decision tree was the best-performing model out of all of them. This suggests that, in comparison to the other models, the decision tree that has been pruned produced the fewest errors in its prediction of the number of days that marijuana was consumed.

  The standard decision tree came in right behind, with an accuracy of 90.89%. Even though it wasn't as precise as the decision tree that had been pruned, it still offered insightful information about the prediction task.

The random forest model performed better in ensemble methods than bagging. In comparison to bagging, which got a marginally an accuracy of 86.53%, the random forest model produced an accuracy of 86.71%, indicating its better predictive accuracy.

Model	Accuracy
Decision Tree	90.89%
Pruned Decision Tree	93.23%
Bagging	86.53%
Random Forest	86.71%
Boosting	85.34%

After each of the models were evaluated, it became clear that the pruned decision tree was the most accurate in estimating how many days a teen would have used marijuana in the previous month. Among the models taken into consideration, its accuracy is higher, indicating that it produced the most accurate predictions.

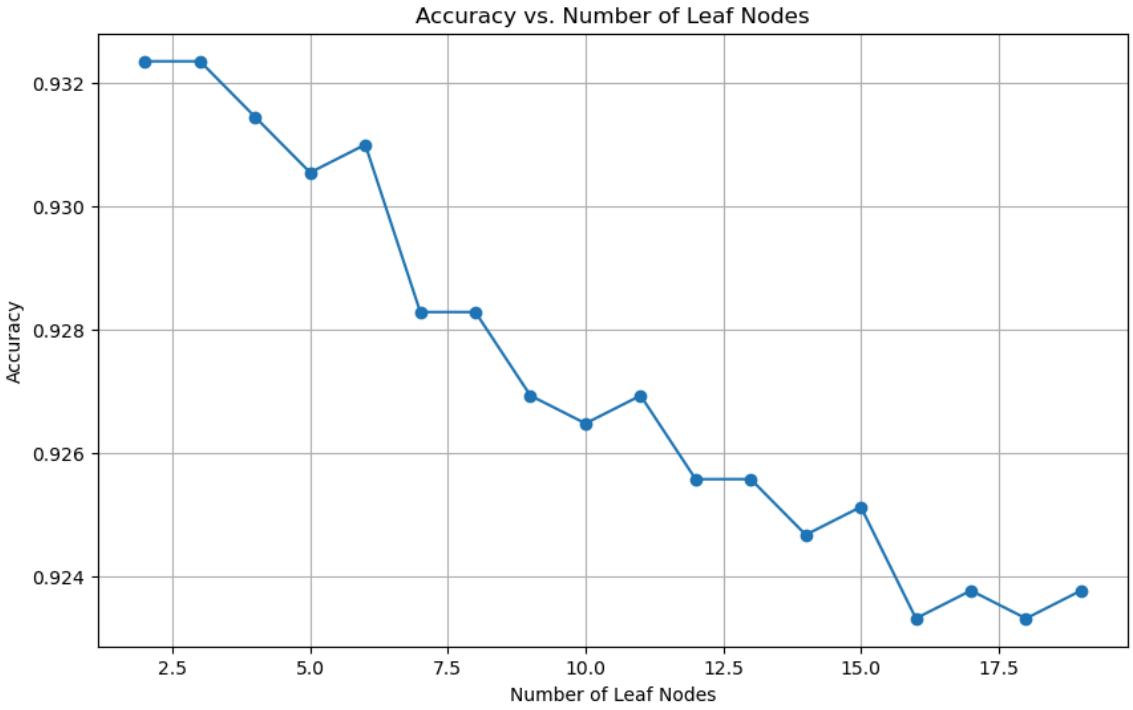

Diagram 2. Plot for Accuracy vs number of leaf nodes

When plotting the accuracy against the number of leaf nodes for the pruned decision tree, we observed a notable trend. The plot tells us that the peak accuracy is 93.23%, with a corresponding number when leaf nodes are just 2. This result signifies the effectiveness of pruning technique. We can also see that with the increase in the number of leaf nodes our accuracy tends to decline with some irregularities in between.

- Regression: Smokeless tobacco age of first use

Five different models were used in our regression analysis to determine how many days a teen had used marijuana in the preceding month: decision tree regressor, bagging, random forest regressor, and others. With a test MSE of 3.57, the decision tree model demonstrated some predictive ability but a comparatively large error margin. After switching to ensemble techniques, bagging produced a test mean squared error (MSE) of 3.24, which was better than the decision tree and suggested that aggregation could improve predictive accuracy. With a test MSE of 2.62, the random forest regressor model, however, performed better than bagging and individual decision trees.

This improved performance demonstrates how well the random forest ensemble method works to reduce prediction error and produce estimates of the number of days of marijuana use in the preceding month that are more precise. All of the models provide predictive insights overall, but the random forest regressor model is the best fit for this regression task because it produces the most accurate predictions.

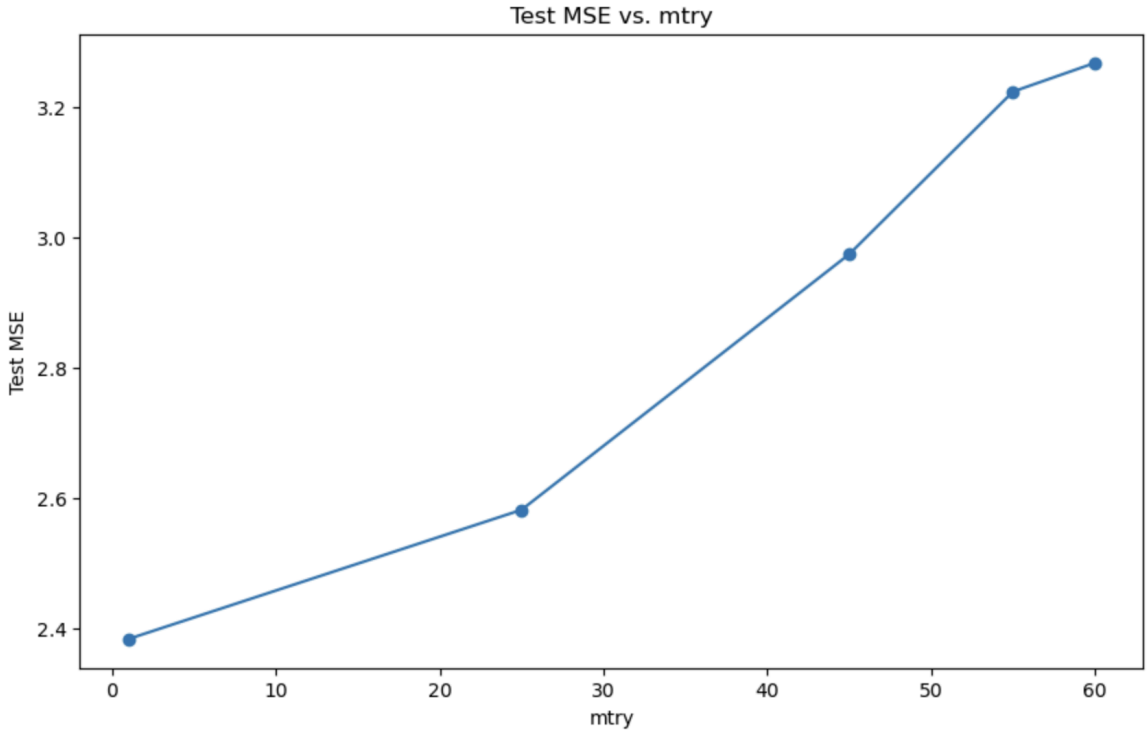

Diagram 3. Plot for Test MSE vs mtry values

The relationship between the test mean squared error (MSE) and the number of predictor variables taken into account (mtry) for a range of values from the minimum to the maximum is depicted in the plot above. Notably, when mtry is set to 1, the minimum test MSE occurs, indicating the possibility of overfitting with such a limited set of predictors. However, it's essential to balance model complexity and predictive accuracy in order to find the ideal value of mtry. The plot in this instance shows that the lowest test MSE is obtained at mtry=25, indicating a value that falls in between the minimum and maximum. This suggests that, in order to avoid the problems of overfitting or underfitting, taking into account a subset of 25 predictor variables yields the most accurate predictions. Consequently, in this case, mtry=25 turns out to be the best option for obtaining the best predictive performance.

Discussion

The NSDUH dataset analysis resulted in helpful findings into teenage drug use patterns. Important results show that pressure from peers, opinions of parents, and educational variables all have an important impact on teens' decisions about alcohol and marijuana use. Social factors are crucial in contributing to teen substance abuse, as shown by the significance of variables such as “peer influence on marijuana use” (YFLTMRJ2) and “student alcohol consumption” (stndalc). In addition, the comparative study of modeling techniques showed that the approaches—like random forests for regression tasks and pruned decision trees for multi-class classification—were better than others. When considered collectively, each model has advantages and disadvantages, but overall, their findings provide a complete knowledge of the detailed structure that encourages teen drug use. It is also important to recognize that there may be constraints that affect how the results are interpreted, such as biases in the data or assumptions made in the model.

- Binary Classification: Alcohol ever used

  In our analysis of teenage alcohol consumption, we found that the Random Forest model stood out as the most accurate predictor, giving an accuracy rate of 83.60%. The most important features we got for this model were 'yfltmrj2', 'stndalc', and 'yflmjmo' to make reliable predictions. 'yfltmrj2' represents how youth feels about their peers marijuana use, telling us about the influence of peer behavior on their own choices. 'stndalc' denotes the grade at which alcohol consumption occurred, providing insights into early patterns among teenagers like when they started consuming alcohol. Additionally, 'yflmjmo' defines youths' perceptions of their peers monthly marijuana use, offering valuable context regarding social norms and substance use behaviors. These results suggest that understanding the interaction between social and individual behaviors is important for addressing teenage alcohol consumption.

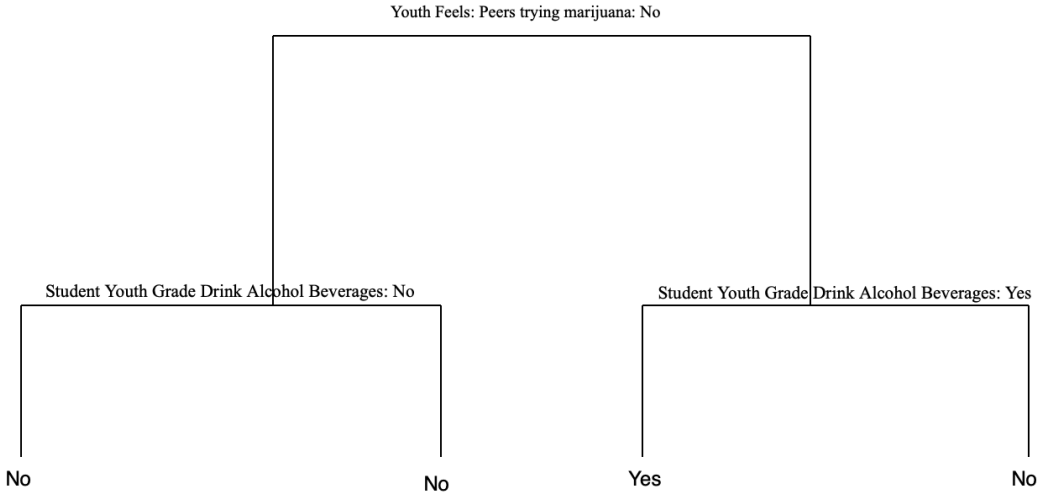

Diagram 4. Pruned tree for Alcohol consumption by a teenager

From the above diagram 4, starting with the youth feeling about their peers using marijuana as the deciding variable for the split of our first node. The left branch and the right goes with the student grade in which they drank alcohol if the deciding node results no or a yes. Following that if the student grade in which they drank alcohol of the left hand side branch is no or a yes it results in that the teen has never consumed alcohol. And now considering the right hand side branch, if the student grade in which they drank alcohol is yes it results in that the teen has consumed alcohol and if it is no it defines that the teen has never consumed alcohol.

In case of Binary Classification binary variables are used which represent two different categories or states by representing data as either 0 or 1. In this case, binary variables represent yes/no answers about drug use or behavioral attitudes. These are useful for predicting outcomes that have a binary nature, like whether a teen has ever drunk alcohol, and also have easy explanations.

- Multi-class classification: Number of days of marijuana in past month

The pruned decision tree emerged as the best performing model during our analysis of predicting the number of days of marijuana in the past month with an accuracy of 93.23%. Upon further analyzing the model, we identified several important features that influenced the prediction outcomes. These features include “how close friends feel about youth using marijuana monthly” (frdmjmon), “students in some grade use marijuana” (stndsmj), “how parents feel about youth using marijuana” (PRMJEVR2), and “whether an individual in youth has ever sold drugs” (YOSELL2). These factors provide details into various aspects of teenage behavior and how people feel related to marijuana use. For example, “how close friends feel about marijuana usage”

(frdmjmon) can influence a teen's choice of engaging in such behavior, as peer influence plays an important role during teenage. Also, knowing “whether students in a particular grade use marijuana” (stndsmj) provides context regarding the existence of substance use within a specific demographic such as school, potentially influencing individual behavior through social forms. Similarly, “parental attitudes toward marijuana use” (PRMJEVR2) can be important for teenage look out and decisions regarding substance use, telling us about family influence. “Involvement in illegal activities such as selling drugs” (YOSELL2) may indicate a higher risk of engaging in marijuana use, reflecting broader behavioral patterns and potential exposure to substance related environments. By considering all these features into our predictive model, we gain insights into different factors contributing to marijuana use among teenagers.

In case of multi class classification, we represent the categorical data. These provide information about different categories in which the data is divided. In our case, predicting the frequency of substance use or attitudes toward substance is considered as ordinal variables. Ordinal variables provides better insights compared to binary variables and are more good for modeling outcomes

- Regression: Smokeless tobacco age of first use

The random forest regressor was the best-performing model for predicting the age of first use of smokeless tobacco, with a test mean squared error (MSE) of 2.62. This model outperformed both bagging and individual decision trees in terms of predictive accuracy. After analyzing the important factors influencing smokeless tobacco start age, we identified the important variables. These include which grade the teen is or will be in (EDUSCHGRD2), telling us about the development stage of the teen and peer influences on them. Also, with “whom they talk about their serious problems” (TALKPROB), this might influence tobacco use behaviors. “How many days the teen has skipped school” (EDUSKPCOM) offers an insight into school attendance trends, which may tell us about the risk taking behaviors. Lastly, “whether the teen has seen substance prevention messages outside of school” (DRPRVME3), giving us an idea of the potential impact of external influences on tobacco usage age. By considering these factors, we gain an important understanding of the factors that lead to smokeless tobacco use among teenagers.

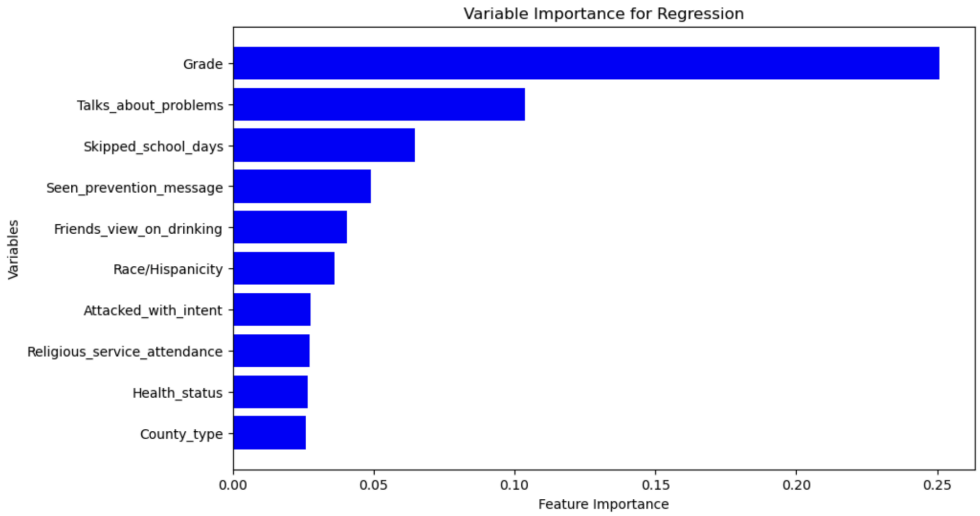

Diagram 5. Plot for variable importances

In our regression problem, numerical variables represent quantitative measurements that provide precise information about the age of first use of smokeless tobacco. These variables may include counts or frequencies related to substance use behaviors, such as “the number of days missed school due to skipping” (EDUSKPCOM).

Conclusion

Our analysis aimed to provide understanding and predicting factors influencing teenage substance use behavior, including alcohol consumption , number of days of marijuana usage and smokeless tobacco initiation age. Through our analysis with various classification and regression models, we identified the random forest as the most effective model for predicting alcohol consumption with an accuracy of 83.60%. This model highlighted crucial features such as how youth feels about their peers marijuana use and the grade at which the alcohol consumption occurred. For the multi-class pruned decision tree performed the best for predicting the number of days of marijuana in the past month with an accuracy of 93.23%. Using this model the most important variables were how close friends feel about youth using marijuana monthly, students in some grade use marijuana and how parents feel about youth using marijuana. Additionally, our regression analysis revealed the random forest regressor as the best model for predicting smokeless tobacco initiation age, with an MSE of 2.62. Important variables in this were like school grade, communication about serious problems with someone, and exposure to substance prevention messages. These findings help us to identify the significance of peer influence, parental guidance, and other interventions in shaping teenage substance use behaviors. Moving forward, more research can build upon these insights to develop support systems aimed at promoting healthier lifestyles among teenagers.

References

https://www.datafiles.samhsa.gov/dataset/national-survey-drug-use-and-health-2020-nsduh-2020-ds0001

https://github.com/mendible/5322/tree/main/Homework%201
https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5026681/
https://www.statlearning.com/

Appendix: Code

```
import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split

from sklearn.tree import DecisionTreeClassifier

import matplotlib.pyplot as plt

from sklearn.tree import plot_tree

from sklearn.metrics import confusion_matrix, accuracy_score

from sklearn.model_selection import GridSearchCV

from sklearn.model_selection import cross_val_score

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy_score

from sklearn.metrics import mean_squared_error

from sklearn.ensemble import GradientBoostingClassifier

from sklearn.tree import DecisionTreeRegressor

from sklearn.ensemble import RandomForestRegressor

from sklearn.ensemble import GradientBoostingRegressor


dfo = pd.read_csv('/Users/lakshitgupta/Library/CloudStorage/OneDrive-SeattleUniversity/Quater3/Machine Learning-2/Written Homeworks/youth_data.csv')

print(dfo.head())


substance_cols = [

    'iralcfy', 'irmjfy', 'ircigfm', 'IRSMKLSS30N', 'iralcfm', 'irmjfm',

    'ircigage', 'irsmklsstry', 'iralcage', 'irmjage',
```

```
    'mrjflag', 'alcflag', 'tobflag',
    'alcydays', 'mrjydays', 'alcmdays', 'mrjmdays', 'cigmdays', 'smklsmdays'
]

demographic_cols = [
    'irsex', 'NEWRACE2', 'HEALTH2', 'eduschlgo', 'EDUSCHGRD2',
    'eduskpcom', 'imother', 'ifather', 'income', 'govtprog',
    'POVERTY3', 'PDEN10', 'COUTYP4'
]

# Load data and select columns of interest
df_youth = dfo.loc[:, 'schfelt':'rlgfrnd']
df_substance = dfo[substance_cols]
df_demog = dfo[demographic_cols]

# Combine into one DataFrame
df = pd.concat([df_substance, df_youth, df_demog], axis=1)

# Fix metadata

# Define unordered and ordered factor columns
unordered_factor_cols = (
    list(df_youth.columns) +
    ['mrjflag', 'alcflag', 'tobflag'] +
    ['irsex', 'NEWRACE2', 'eduschlgo', 'imother', 'ifather', 'govtprog', 'PDEN10', 'COUTYP4']
```

```
)

ordered_factor_cols = ['EDUSCHGRD2', 'HEALTH2', 'POVERTY3', 'income']


# Convert to factors

df[unordered_factor_cols] = df[unordered_factor_cols].astype('category')  # Unordered factors

for col in ordered_factor_cols:

    df[col] = pd.Categorical(df[col], ordered=True)  # Ordered factors


# Define variable labels as a dictionary

variable_labels = {

    'iralcfy': 'Alcohol frequency past year',

    'irmjfy': 'Marijuana frequency past year',

    # and so on...

}


# Assign labels to DataFrame columns

df.columns = pd.Index(variable_labels.get(col, col) for col in df.columns)


# Note: You can access the labeled data using df.columns


youth_experience_cols = df_youth.columns

df.info()


missing_values = df.isna().sum().sum()

print(f"Number of missing values: {missing_values}")
```

```
youth_experience_cols


df_substance.columns


df_demog.columns


df = df.dropna()
missing_values = df.isna().sum().sum()
print(f"Number of missing values after dropping: {missing_values}")


df = df[~df['EDUSCHGRD2'].isin([98, 99])]


# Filter out rows where eduskpcom is 94, 97, 98, or 99
df = df[~df['eduskpcom'].isin([94, 97, 98, 99])]


# Filter out rows where imother is 3 or 4
df = df[~df['imother'].isin([3, 4])]


# Filter out rows where ifather is 3 or 4
df = df[~df['ifather'].isin([3, 4])]


# Filter out rows where PDEN10 is 3
df = df[df['PDEN10'] != 3]
```

```
print(df.shape)
```

Binary Classification
Alcohol - 'alcflag'

Alcohol ever used (0=never, 1=ever)

```
df_youthExp = df[youth_experience_cols]

df_demographic = df[demographic_cols]

ALCFLAG = df[['alcflag']]


# Combining into a new DataFrame

df_New = pd.concat([df_youthExp, df_demographic, ALCFLAG], axis=1)


print(df_New)


X = df_New.drop(columns=['alcflag'])

y = df_New['alcflag']


# Split the dataset into training and test sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=123)



print("Training set dimensions:", X_train.shape)

print("Test set dimensions:", X_test.shape)
```

Decision Tree

```
tree_model = DecisionTreeClassifier(random_state=1)

tree_model.fit(X_train, y_train)


plt.figure(figsize=(70, 100))


# Plotting the decision tree

plot_tree(tree_model,

        filled=True,

        feature_names=X_train.columns.tolist(),

        class_names=['No', 'Yes'],

        label='all',

        fontsize=24)


plt.show()


# Getting feature importances

feature_importance = tree_model.feature_importances_



feature_importance_df = pd.DataFrame({'Feature': X_train.columns, 'Importance': feature_importance})


# Sortting the DataFrame by importance values in descending order

feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=False)
```

```
# Top 10 features
top_10_features = feature_importance_df.head(10)

# Display the top 15 features
print("Top 15 Feature Importance:")
print(top_10_features)

tree_pred = tree_model.predict(X_test)

# Confusion matrix
cm = confusion_matrix(y_test, tree_pred)

# Accuracy
Decaccuracy = accuracy_score(y_test, tree_pred)

print("Confusion Matrix:")
print(cm)

print("Accuracy:", Decaccuracy)
```

Pruning

```
# Defining the parameter grid
param_grid = {'max_leaf_nodes': range(2,20)}
```

```
# Create GridSearchCV

grid_search = GridSearchCV(estimator=tree_model, param_grid=param_grid, cv=5, scoring='accuracy')

grid_search.fit(X_train, y_train)


# Best estimator

best_tree_model = grid_search.best_estimator_


pruned_tree_pred = best_tree_model.predict(X_test)


# Confusion matrix

pruned_cm = confusion_matrix(y_test, pruned_tree_pred)


# Accuracy

pruned_accuracy = accuracy_score(y_test, pruned_tree_pred)


print("max_leaf_nodes:", best_tree_model.max_leaf_nodes)

print("Confusion Matrix after Pruning:")

print(pruned_cm)

print("Accuracy after Pruning:", pruned_accuracy)


feature_names_list = X_train.columns.tolist()


plt.figure(figsize=(20, 10))


# Plotting the pruned decision tree
```

```
plot_tree(best_tree_model, filled=True, feature_names=feature_names_list, class_names=["No", "Yes"], fontsize=10)


plt.show()


max_leaf_nodes_range = range(2, 20)

accuracies = []

num_leaf_nodes = []


for leaf_nodes in max_leaf_nodes_range:

    tree_model = DecisionTreeClassifier(max_leaf_nodes=leaf_nodes, random_state=1)


    accuracy = np.mean(cross_val_score(tree_model, X_train, y_train, cv=5, scoring='accuracy'))


    accuracies.append(accuracy)

    num_leaf_nodes.append(leaf_nodes)


Prunmax_accuracy = max(accuracies)

corresponding_num_leaf_nodes = num_leaf_nodes[accuracies.index(Prunmax_accuracy)]


print('Maximum Accuracy:', Prunmax_accuracy)

print('Corresponding Number of Leaf Nodes:', corresponding_num_leaf_nodes)


plt.figure(figsize=(10, 6))

plt.plot(num_leaf_nodes, accuracies, marker='o', linestyle='-')

plt.xlabel('Number of Leaf Nodes')
```

```
plt.ylabel('Accuracy')

plt.title('Accuracy vs. Number of Leaf Nodes')

plt.grid(True)

plt.show()


feature_importances = best_tree_model.feature_importances_


# Importances along with feature names

feature_importance_df = pd.DataFrame({'Feature': X_train.columns, 'Importance': feature_importances})


feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=False)


top_features = feature_importance_df.head(10)

# Display the top features

print("Top Features:")

print(top_features)
```

Bagging

```
bag_model = RandomForestClassifier(n_estimators=100, max_features=60, random_state=1)


# Fitting the Random Forest classifier

bag_model.fit(X_train, y_train)


yhat_bag = bag_model.predict(X_test)
```

```
# Calculate accuracy

Bagaccuracy = accuracy_score(y_test, yhat_bag)

print('\nAccuracy:', Bagaccuracy)




# Importances from the Random Forest model

feature_importances_bag = bag_model.feature_importances_


feature_importance_bag_df = pd.DataFrame({'Feature': X_train.columns, 'Importance': feature_importances_bag})


feature_importance_bag_df = feature_importance_bag_df.sort_values(by='Importance', ascending=False)


top_features_bag = feature_importance_bag_df.head(10)


plt.figure(figsize=(10, 6))

plt.barh(top_features_bag['Feature'], top_features_bag['Importance'], color='blue')

plt.xlabel('Feature Importance')

plt.ylabel('Variables')

plt.title('Variable Importance for Consumption of Alcohol')

plt.gca().invert_yaxis()

plt.show()
```

Random Forest

```
np.random.seed(123)


# Initialize the Random Forest classifier

random_model = RandomForestClassifier(n_estimators=500, max_features=30, random_state=1)


# Fit the Random Forest classifier to the training data

random_model.fit(X_train, y_train)


# Make predictions on the test set

yhat_rf = random_model.predict(X_test)


# Calculate accuracy

Ranaccuracy = accuracy_score(y_test, yhat_rf)

print('\nAccuracy:', Ranaccuracy)


feature_importance = random_model.feature_importances_


feature_importance_df = pd.DataFrame({'Feature': X_train.columns, 'Importance': feature_importance})


feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=False)
```

```
top_10_features = feature_importance_df.head(10)

print("Top 10 Feature Importance:")
print(top_10_features)

mtry_values = [1, 25, 45, 55, 60]
accuracy_values = []

for mtry in mtry_values:
    # Initializing and fit Random Forest with current mtry value
    rf_model = RandomForestClassifier(n_estimators=500, max_features=mtry, random_state=1)
    rf_model.fit(X_train, y_train)

    yhat_test = rf_model.predict(X_test)

    accuracy = accuracy_score(y_test, yhat_test)
    accuracy_values.append(accuracy)

print('Accuracy Values:', accuracy_values)

# Finding the index of the highest accuracy value
max_accuracy_index = accuracy_values.index(max(accuracy_values))
Ranhighest_accuracy = accuracy_values[max_accuracy_index]
corresponding_mtry = mtry_values[max_accuracy_index]
```

```
print('Highest Accuracy:', Ranhighest_accuracy)

print('Corresponding mtry:', corresponding_mtry)


plt.figure(figsize=(10, 6))

plt.plot(mtry_values, accuracy_values, marker='o', linestyle='-')

plt.xlabel('mtry')

plt.ylabel('Accuracy')

plt.title('Accuracy vs. mtry')

plt.grid(True)

plt.show()
```

Boosting

```
vals = np.arange(0.1, 0.21, 0.02)


# Gradient Boosting model with different shrinkage values

gbm_models = []

accuracy_values = []


for lam1 in vals:

        boosting_model = GradientBoostingClassifier(n_estimators=1000, learning_rate=lam1, random_state=1)

    boosting_model.fit(X_train, y_train)

    yhat_test = boosting_model.predict(X_test)

    accuracy = accuracy_score(y_test, yhat_test)

    gbm_models.append(boosting_model)
```

```
    accuracy_values.append(accuracy)

# Test set accuracy vs. learning rate
plt.plot(vals, accuracy_values, marker='o', linestyle='-')
plt.xlabel('learning_rate')
plt.ylabel('Accuracy')
plt.title('Accuracy vs. learning_rate')
plt.show()

# Finding the highest accuracy and its corresponding learning rate
max_accuracy_index = np.argmax(accuracy_values)
Boosthighest_accuracy = accuracy_values[max_accuracy_index]
corresponding_learning_rate = vals[max_accuracy_index]

print("Highest Accuracy:", Boosthighest_accuracy)
print("Corresponding Learning Rate:", corresponding_learning_rate)

feature_importances = []

for model in gbm_models:
    # Feature importances from Boosting
    importances = model.feature_importances_
    feature_importances.append(importances)
```

```
average_importances = np.mean(feature_importances, axis=0)

importance_df = pd.DataFrame({'Feature': X_train.columns, 'Importance': average_importances})
importance_df = importance_df.sort_values(by='Importance', ascending=False)

top_10_features = importance_df.head(10)

print(top_10_features)

accuracy_data = {
    "Model": ["Decision Tree", "Pruned Decision Tree", "Bagging", "Random Forest", "Boosting"],
            "Accuracy": [Decaccuracy, Prunmax_accuracy, Bagaccuracy, Ranhighest_accuracy, Boosthighest_accuracy]
}

# Create a DataFrame from the accuracy data
accuracy_df = pd.DataFrame(accuracy_data)

# Set the style for the DataFrame
styled_accuracy_df = accuracy_df.style.hide_index().set_caption("Model Accuracy")

# Apply formatting to the accuracy values
styled_accuracy_df = styled_accuracy_df.format({"Accuracy": "{:.2%}"})

# Display the styled accuracy DataFrame
styled_accuracy_df
```

Multi Class Classification

Marijuana - 'mrjmdays'

Number of days of marijuana in past month (1-4 categories, 5=none)

```
df_youthExp1 = df[youth_experience_cols]

df_demographic1 = df[demographic_cols]

MRJMDAYS = df[['mrjmdays']]


df_Multi = pd.concat([df_youthExp1, df_demographic1, MRJMDAYS], axis=1)


X = df_Multi.drop(columns=['mrjmdays'])

y = df_Multi['mrjmdays']


# Training and test sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=123)
```

Decision Tree

```
tree_modelNew = DecisionTreeClassifier(random_state=1)

tree_modelNew.fit(X_train, y_train)


plt.figure(figsize=(70, 100))
```

```
plot_tree(tree_modelNew,
          filled=True,
          feature_names=X_train.columns.tolist(),
          label='all',
          fontsize=24)



plt.show()


feature_importance = tree_modelNew.feature_importances_


# Feature importances
feature_importance_df = pd.DataFrame({'Feature': X_train.columns, 'Importance': feature_importance})


feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=False)


top_10_features = feature_importance_df.head(10)


# Displaying the top 10 features
print("Top 10 Feature Importance:")
print(top_10_features)


tree_predNew = tree_modelNew.predict(X_test)


cm = confusion_matrix(y_test, tree_predNew)
```

```
Decaccuracy1 = accuracy_score(y_test, tree_predNew)


print("Confusion Matrix:")

print(cm)


print("Accuracy:", Decaccuracy1)
```

Pruning

```
# Defining the parameter grid

param_grid = {'max_leaf_nodes': range(2,20)}


grid_search = GridSearchCV(estimator=tree_modelNew, param_grid=param_grid, cv=5,scoring='neg_mean_squared_error')


grid_search.fit(X_train, y_train)


best_tree_modelNew = grid_search.best_estimator_


pruned_tree_predNew = best_tree_modelNew.predict(X_test)

pruned_cmNew = confusion_matrix(y_test, pruned_tree_predNew)

pruned_accuracy_new = accuracy_score(y_test, pruned_tree_predNew)


print("max_leaf_nodes:", best_tree_modelNew.max_leaf_nodes)

print("Confusion Matrix after Pruning:")

print(pruned_cmNew)
```

```
print("Accuracy after Pruning:", pruned_accuracy_new)

feature_names_listNew = X_train.columns.tolist()

plt.figure(figsize=(20, 10))

# Pruned decision tree
plot_tree(best_tree_modelNew, filled=True, feature_names=feature_names_listNew, fontsize=10)

plt.show()

max_leaf_nodes_range = range(2, 20)

accuracies = []
num_leaf_nodes = []

for leaf_nodes in max_leaf_nodes_range:
    tree_modelNew = DecisionTreeClassifier(max_leaf_nodes=leaf_nodes, random_state=1)

        accuracy = np.mean(cross_val_score(tree_modelNew, X_train, y_train, cv=5, scoring='accuracy'))

    accuracies.append(accuracy)
    num_leaf_nodes.append(leaf_nodes)

Prunmax_accuracy1 = max(accuracies)
```

```
corresponding_num_leaf_nodes = num_leaf_nodes[accuracies.index(Prunmax_accuracy1)]

print('Maximum Accuracy:', Prunmax_accuracy1)
print('Corresponding Number of Leaf Nodes:', corresponding_num_leaf_nodes)

plt.figure(figsize=(10, 6))
plt.plot(num_leaf_nodes, accuracies, marker='o', linestyle='-')
plt.xlabel('Number of Leaf Nodes')
plt.ylabel('Accuracy')
plt.title('Accuracy vs. Number of Leaf Nodes')
plt.grid(True)
plt.show()

feature_importances = best_tree_modelNew.feature_importances_

feature_importance_df = pd.DataFrame({'Feature': X_train.columns, 'Importance': feature_importances})

feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=False)

# Top 5 features
top_features = feature_importance_df.head(5)

print("Top Features:")
print(top_features)
```

Bagging

```
bag_model = RandomForestClassifier(n_estimators=100, max_features=60, random_state=1)

bag_model.fit(X_train, y_train)


yhat_bag = bag_model.predict(X_test)


Bagaccuracy1 = accuracy_score(y_test, yhat_bag)

print('\nAccuracy:', Bagaccuracy1)


# Feature importances from the Random Forest model

feature_importances_bag = bag_model.feature_importances_


feature_importance_bag_df = pd.DataFrame({'Feature': X_train.columns, 'Importance': feature_importances_bag})


feature_importance_bag_df = feature_importance_bag_df.sort_values(by='Importance', ascending=False)


# Top 10 variables

top_features_bag = feature_importance_bag_df.head(10)


plt.figure(figsize=(10, 6))

plt.barh(top_features_bag['Feature'], top_features_bag['Importance'], color='blue')

plt.xlabel('Feature Importance')
```

```
plt.ylabel('Variables')

plt.title('Variable Importance for Consumption of marijuana in past month')

plt.gca().invert_yaxis()

plt.show()
```

Random Forest

```
np.random.seed(123)


random_modelNew = RandomForestClassifier(n_estimators=500, max_features=30,
random_state=1)


# Fit the Random Forest classifier to the training data

random_modelNew.fit(X_train, y_train)


yhat_rf = random_modelNew.predict(X_test)


Ranaccuracy1 = accuracy_score(y_test, yhat_rf)

print('\nAccuracy:', Ranaccuracy1)


# Feature importances

feature_importance = random_modelNew.feature_importances_


feature_importance_df = pd.DataFrame({'Feature': X_train.columns, 'Importance':
feature_importance})
```

```
feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=False)

# Top 10 features
top_10_features = feature_importance_df.head(10)

print("Top 10 Feature Importance:")
print(top_10_features)

mtry_values = [1, 25, 45, 55, 60]
accuracy_values = []

for mtry in mtry_values:
    rf_model = RandomForestClassifier(n_estimators=500, max_features=mtry, random_state=1)
    rf_model.fit(X_train, y_train)

    yhat_test = rf_model.predict(X_test)

    accuracy = accuracy_score(y_test, yhat_test)
    accuracy_values.append(accuracy)

# Finding the highest accuracy and its corresponding mtry value
Ranmax_accuracy1 = max(accuracy_values)
corresponding_mtry = mtry_values[accuracy_values.index(Ranmax_accuracy1)]

print('Highest Accuracy:', Ranmax_accuracy1)
```

```
print('Corresponding mtry:', corresponding_mtry)


# Plot Accuracy vs. mtry

plt.figure(figsize=(10, 6))

plt.plot(mtry_values, accuracy_values, marker='o', linestyle='-')

plt.xlabel('mtry')

plt.ylabel('Accuracy')

plt.title('Accuracy vs. mtry')

plt.grid(True)

plt.show()
```

Boosting

```
vals = np.arange(0.1, 0.21, 0.01)


# Gradient Boosting models with different shrinkage values

gbm_models = []

accuracy_values = []


for lam1 in vals:

        boosting_model = GradientBoostingClassifier(n_estimators=1000, learning_rate=lam1, random_state=1)

    boosting_model.fit(X_train, y_train)

    yhat_test = boosting_model.predict(X_test)

    accuracy = accuracy_score(y_test, yhat_test)

    gbm_models.append(boosting_model)
```

```
    accuracy_values.append(accuracy)


# Test set accuracy vs. learning rate
plt.plot(vals, accuracy_values, marker='o', linestyle='-')
plt.xlabel('learning_rate')
plt.ylabel('Accuracy')
plt.title('Accuracy vs. learning_rate')
plt.grid(True)
plt.show()


# Finding the highest accuracy and its corresponding learning rate
Boostmax_accuracy1 = max(accuracy_values)
corresponding_learning_rate = vals[accuracy_values.index(Boostmax_accuracy1)]


print('Highest Accuracy:', Boostmax_accuracy1)
print('Corresponding Learning Rate:', corresponding_learning_rate)


best_boosting_model = GradientBoostingClassifier(n_estimators=1000,
learning_rate=corresponding_learning_rate, random_state=1)
best_boosting_model.fit(X_train, y_train)


# Access feature importances
feature_importance = best_boosting_model.feature_importances_


# Create a DataFrame to store feature importances
```

```
feature_importance_df = pd.DataFrame({'Feature': X_train.columns, 'Importance': feature_importance})

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

# Select the top 10 features
top_10_features = feature_importance_df.head(10)

# Display the top 10 features
print("Top 10 Feature Importance:")
print(top_10_features)

accuracy_data = {
    "Model": ["Decision Tree", "Pruned Decision Tree", "Bagging", "Random Forest", "Boosting"],
        "Accuracy": [Decaccuracy1, Prunmax_accuracy1, Bagaccuracy, Ranmax_accuracy1, Boostmax_accuracy1]
}

# Create a DataFrame from the accuracy data
accuracy_df = pd.DataFrame(accuracy_data)

# Set the style for the DataFrame
styled_accuracy_df = accuracy_df.style.hide_index().set_caption("Model Accuracy")

# Apply formatting to the accuracy values
```

```
styled_accuracy_df = styled_accuracy_df.format({"Accuracy": "{:.2%}"})

# Display the styled accuracy DataFrame
styled_accuracy_df
```

Regression

Tobacco - 'irsmklsstry'

Smokeless tobacco age of first use (1-70), 991=never used

```
df_youthExp = df[youth_experience_cols]
df_demographic = df[demographic_cols]
IRSMKLSSTRY= df[['irsmklsstry']]

df_Reg = pd.concat([df_youthExp, df_demographic, IRSMKLSSTRY], axis=1)
df_Reg = df_Reg[df_Reg['irsmklsstry'] != 991]

print(df_Reg)

X = df_Reg.drop(columns=['irsmklsstry'])
y = df_Reg['irsmklsstry']

# Training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=123)

print("Training set dimensions:", X_train.shape)
```

```
print("Test set dimensions:", X_test.shape)
```

Decision Tree

```
Reg_model = DecisionTreeRegressor(random_state=1)


# Fit the model to the training data

Reg_model.fit(X_train, y_train)


plt.figure(figsize=(70, 100))


plot_tree(Reg_model,

        filled=True,

        feature_names=X_train.columns.tolist(),

        fontsize=24)



plt.show()


feature_importance = Reg_model.feature_importances_


# Feature importances

feature_importance_df = pd.DataFrame({'Feature': X_train.columns, 'Importance': feature_importance})


feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=False)
```

```
# Top 10 features
top_10_features = feature_importance_df.head(10)

print("Top 10 Feature Importance:")
print(top_10_features)

Reg_pred_new = Reg_model.predict(X_test)

# Test error rate
MSE = ((y_test - Reg_pred_new)**2).mean()
print("Test Error Rate:", MSE)
```

Pruning

```
param_grid = {'max_leaf_nodes': range(2, 20)}

grid_search = GridSearchCV(estimator=Reg_model, param_grid=param_grid, cv=5, scoring='neg_mean_squared_error')

grid_search.fit(X_train, y_train)

best_tree_model = grid_search.best_estimator_

# Best max_leaf_nodes value
```

```
print("Best max_leaf_nodes value:", best_tree_model.max_leaf_nodes)


max_leaf_nodes_range = range(2, 20)


num_trees = []

mse_values = []


for max_leaf_nodes in max_leaf_nodes_range:


    regressor = DecisionTreeRegressor(max_leaf_nodes=max_leaf_nodes, random_state=1)

                    mse_scores = -cross_val_score(regressor, X_train, y_train, cv=5,
scoring='neg_mean_squared_error')


    avg_mse = mse_scores.mean()


    num_trees.append(max_leaf_nodes)

    mse_values.append(avg_mse)


prunmin_mse = min(mse_values)

corresponding_num_trees = num_trees[mse_values.index(prunmin_mse)]


print('Minimum MSE:', prunmin_mse)

print('Corresponding Number of Trees (max_leaf_nodes):', corresponding_num_trees)


# MSE vs number of trees

plt.figure(figsize=(10, 6))
```

```
plt.plot(num_trees, mse_values, marker='o')
plt.title('MSE vs Number of Trees')
plt.xlabel('Number of Trees (max_leaf_nodes)')
plt.ylabel('Mean Squared Error (MSE)')
plt.grid(True)
plt.show()

feature_names_listNew = X_train.columns.tolist()

plt.figure(figsize=(20, 10))

plot_tree(best_tree_model, filled=True, feature_names=feature_names_listNew, fontsize=10)

plt.show()
```

Bagging

```
bag_modelReg = RandomForestRegressor(n_estimators=100, max_features=60, random_state=1)

bag_modelReg.fit(X_train, y_train)

yhat_bagReg = bag_modelReg.predict(X_test)

# Calculating the test MSE
Bagtest_mse = mean_squared_error(y_test, yhat_bagReg)
print('\nTest MSE:', Bagtest_mse)
```

```
feature_importances_bagReg = bag_modelReg.feature_importances_


feature_importance_bagReg_df = pd.DataFrame({'Feature': X_train.columns, 'Importance': feature_importances_bagReg})


feature_importance_bagReg_df = feature_importance_bagReg_df.sort_values(by='Importance', ascending=False)


# Select the top 10 variables

top_features_bagReg = feature_importance_bagReg_df.head(10)


# Plot variable importance of different variables

plt.figure(figsize=(10, 6))

plt.barh(top_features_bagReg['Feature'], top_features_bagReg['Importance'], color='blue')

plt.xlabel('Feature Importance')

plt.ylabel('Variables')

plt.title('Variable Importance for Regression')

plt.gca().invert_yaxis()

plt.show()
```

Random Forest

```
random_model = RandomForestRegressor(n_estimators=500, max_features=30, random_state=1)


random_model.fit(X_train, y_train)
```

```
# Test set

yhat_rf = random_model.predict(X_test)


# Test MSE

test_mse = mean_squared_error(y_test, yhat_rf)

print('\nTest MSE:', test_mse)


# Feature importances from the Random Forest model

feature_importance = random_model.feature_importances_


feature_importance_df = pd.DataFrame({'Feature': X_train.columns, 'Importance': feature_importance})


feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=False)


# Top 10 features

top_10_features = feature_importance_df.head(10)


feature_name_mapping = {

    'EDUSCHGRD2': 'Grade',

    'talkprob': 'Talks_about_problems',

    'eduskpcom': 'Skipped_school_days',

    'DRPRVME3': 'Seen_prevention_message',

    'FRDADLY2': 'Friends_view_on_drinking',

    'NEWRACE2': 'Race/Hispanicity',
```

```
    'YOATTAK2': 'Attacked_with_intent',
    'rlgattd': 'Religious_service_attendance',
    'HEALTH2': 'Health_status',
    'COUTYP4': 'County_type'
}

# Convert the original feature names to their descriptive equivalents
readable_features = [feature_name_mapping[f] for f in top_10_features['Feature']]

plt.figure(figsize=(10, 6))
plt.barh(readable_features, top_10_features['Importance'], color='blue')
plt.xlabel('Feature Importance')
plt.ylabel('Variables')
plt.title('Variable Importance for Regression')
plt.gca().invert_yaxis()
plt.show()

mtry_values = [1, 25, 45, 55, 60]
error_rates = []

for mtry in mtry_values:
    rf_model = RandomForestRegressor(n_estimators=500, max_features=mtry, random_state=1)
    rf_model.fit(X_train, y_train)
```

```
    yhat_test = rf_model.predict(X_test)
    test_mse = mean_squared_error(y_test, yhat_test)
    error_rates.append(test_mse)

# Error rates
print('Error Rates:', error_rates)

sorted_error_rates = sorted(error_rates)
second_min_mse = sorted_error_rates[1]

# Find the corresponding mtry value for the second smallest MSE
index_second_min_mse = error_rates.index(second_min_mse)
corresponding_mtry_second_min_mse = mtry_values[index_second_min_mse]

print('Second Smallest MSE:', second_min_mse)
print('Corresponding mtry value for the Second Smallest MSE:', corresponding_mtry_second_min_mse)

# Error rate vs. mtry
plt.figure(figsize=(10, 6))
plt.plot(mtry_values, error_rates, marker='o', linestyle='-')
plt.xlabel('mtry')
plt.ylabel('Test MSE')
plt.title('Test MSE vs. mtry')
plt.show()
```

Boosting

```
vals = np.arange(0.1, 0.21, 0.01)


# Training Gradient Boosting model with different shrinkage values

boosting_models = []

test_error_rates = []


for lam1 in vals:

        boosting_model = GradientBoostingRegressor(n_estimators=1000, learning_rate=lam1, random_state=1)

    boosting_model.fit(X_train, y_train)

    yhat_test = boosting_model.predict(X_test)

    test_mse = mean_squared_error(y_test, yhat_test)


    boosting_models.append(boosting_model)

    test_error_rates.append(test_mse)


Boostmin_test_mse = min(test_error_rates)

corresponding_learning_rate = vals[test_error_rates.index(Boostmin_test_mse)]


print('Minimum Test MSE:', Boostmin_test_mse)

print('Corresponding Learning Rate:', corresponding_learning_rate)


# Error rates vs. learning rate

plt.plot(vals, test_error_rates, marker='o', linestyle='-')
```

```
plt.xlabel('learning_rate')

plt.ylabel('Test MSE')

plt.title('Test MSE vs. learning_rate')

plt.show()


best_learning_rate = vals[np.argmin(test_error_rates)]

best_boosting_model = GradientBoostingRegressor(n_estimators=1000, learning_rate=best_learning_rate, random_state=1)

best_boosting_model.fit(X_train, y_train)


feature_importance = best_boosting_model.feature_importances_


feature_importance_df = pd.DataFrame({'Feature': X_train.columns, 'Importance': feature_importance})


feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=False)


# Top 10 features

top_10_features = feature_importance_df.head(10)


print("Top 10 Feature Importance:")

print(top_10_features)


mse_data = {

    "Model": ["Decision Tree", "Pruned Decision Tree", "Bagging", "Random Forest", "Boosting"],

    "Test MSE": [MSE, prunmin_mse, Bagtest_mse, second_min_mse, Boostmin_test_mse]
```

```
}


mse_df = pd.DataFrame(mse_data)

styled_mse_df = mse_df.style.hide_index().set_caption("Model MSE")

 styled_mse_df
```