Devin Lim

DATA 5322 Written Homework 1: Decision Trees

April 26, 2024

Correlates of Alcohol Consumption Among Youth

Abstract

Youth substance use can lead to various short-term problems and may “develop into lifelong issues such as substance dependence, chronic health problems, and social and financial consequences”[1]. The National Survey on Drug Use and Health (NSDUH), which started in 1971 and became an annual survey in 1990, “is the primary nationally representative source of annual estimates of drug use and mental illness … in the United States”[2]. Using the data from 2020 NSDUH, this paper examined the top factors that are correlated with youth (aged 12 to 17) alcohol use by utilizing various machine learning methods. Results show that there is 1 common predictor across the substance use variables we analyzed: whether the youth’s student peers also drink. Aside from that, the top factors are multifaceted, ranging from violence (when predicting a youth’s age at the time of first drink), to peer influence (when predicting whether a youth has ever drunk), to socioeconomic factors like income and country (when predicting chronic drinking in youth).

Introduction

Alcohol is our youth’s drug of choice; it is consumed more frequently than either tobacco or marijuana. Youth also binge drink more often compared to adults – 90% of alcoholic beverages consumed by youth are consumed during acts of binge drinking. This behavior is linked to increased risk of injuries, death, physical and sexual assault, learning difficulties, and alcohol problems later in life.[3] As such, it is crucial to understand the factors that are correlated with youth drinking so that it may inform our policies and approach to harm reduction.

The dataset used in this analysis is a subset of the 2020 NSDUH[4] that has been processed by Dr. Mendible[5] to include only youth data ages 12 to 17 consisting 79 out of the 2,890 variables. The variables include substance usage statistics for alcohol, marijuana, and cigarettes along with demographic data such as household income, peer influence, and parental involvement.

Various machine learning algorithms and training approaches are compared and utilized to uncover factors most correlated to youth drinking. More specifically, we are looking for factors related to 'iralcage' (the age a youth first consumed alcohol), 'alcflag' (whether a youth has ever consumed alcohol), and 'alcydays' (the number of days a youth has consumed alcohol within the past year). These factors are then discussed to provide us with usable insights. Background information on the data preparation process and machine learning methods is provided below.

Background

Null Model

The Null Model serves as a baseline in both regression and classification problems. In regression, it always predicts the mean of the response variable. In classification, it always predicts the most frequent class. It is important to compare the models we build against the Null Model to ensure that our models provide real benefits over the simplest possible baseline.

Ordinary Least Squares

Ordinary Least Squares is the earliest form of linear regression[6]. This method aims to create a best fit line such that the distance between the points and the line (known as the residuals) is minimized. The best fit line is defined by a set of constants acting as weights to the predictors and an irreducible error. OLS requires many assumptions to be met about the data, such as the linearity between predictor and response and no collinearity between predictors.

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p \ + \ \epsilon$$

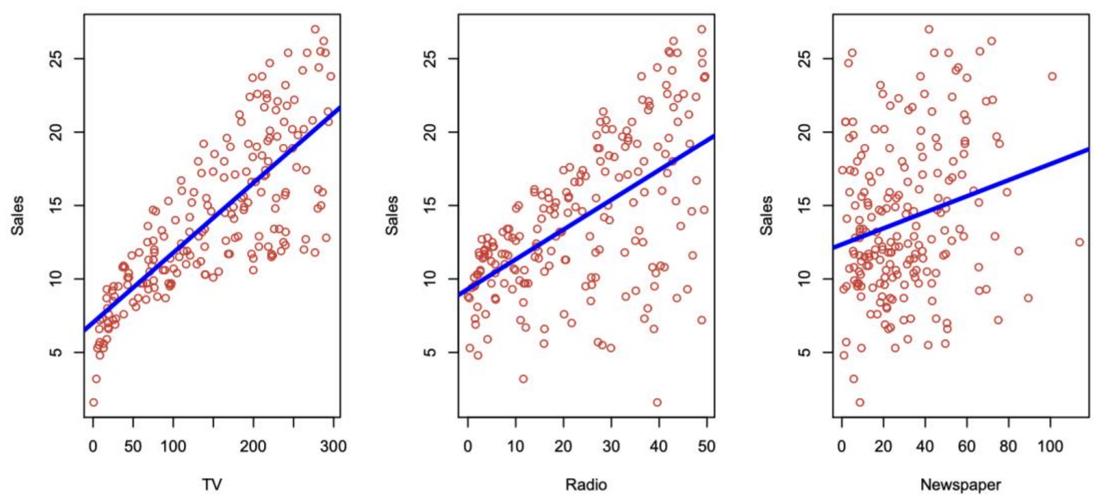


Figure 1. Ordinary Least Squares Regression.[6]

Lasso

The Lasso model adds a penalty term to Ordinary Least Squares. This penalty term is the sum of the absolute values of the coefficients multiplied by the tuning parameter λ. This encourages the model to shrink coefficients as much as possible, often to zero, which improves model interpretability (it performs variable selection). Unlike Ordinary Least Squares, Lasso can handle collinearity between predictors.

$$\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=i}^{p}\beta_j x_{ij}\right)^2 + \lambda\sum_{j=1}^{p}|\beta_j| = RSS + \lambda\sum_{j=1}^{p}|\beta_j|$$

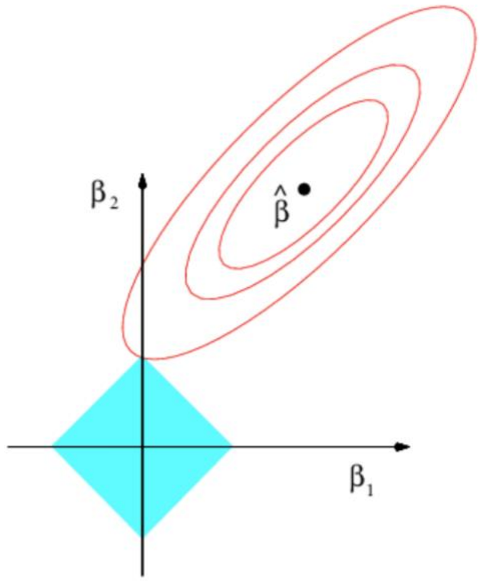


Figure 2. Lasso Regression.[6]

Decision Tree Model

A Decision Tree is a model built through a supervised learning approach that splits recursively on the most important feature. It is a greedy approach, and thus it's deterministic. Decision Trees are flexible; It is applicable to both regression and classification problems with either categorical or numerical features and it can also handle non-linear relationships between the feature and response variables. However, since it's deterministic, it's prone to overfitting.

Compared to later Tree-based models, being a single tree trained on the full training set and features mean that they are much more interpretable. For example, in Figure 3, the most important feature and its direction of correlation is easily understood: if they don't have feathers, they are not a bird species.

Decision Trees can be tuned by adjusting its maximum depth and the minimum number of samples a node must have before it's allowed to split. A shallower tree and a higher minimum number of samples help to prevent overfitting.

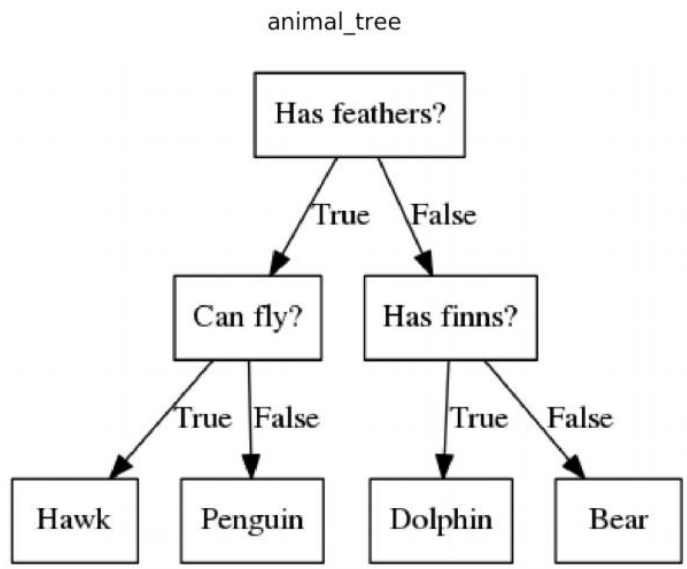


Figure 3. A decision tree.[7]

Bagging Model

A Bagging Model trains multiple decision trees on different samples (obtained through bootstrap sampling) of the dataset then aggregates their predictions to obtain a final prediction (this approach of aggregating multiple models is also known as an Ensemble Method). As a result of random sampling, it is no longer deterministic and reduces overfitting. Being an Ensemble Tree, interpretation is much more difficult as analyzing a single tree will not provide a correct interpretation of the whole model. Feature importance can be calculated through statistical methods but determining the direction of the relationship is difficult.

Bagging Models can be tuned similarly to Decision Trees by adjusting its maximum depth and minimum number of samples, but additionally, we can also adjust the number of trees built for the ensemble. More trees generally further prevent overfitting, but since it is computationally expensive and there are diminishing returns, it is important to test many variations and find a balance.

While no bagging models are used in this analysis directly, later models like the Random Forest builds upon this concept.

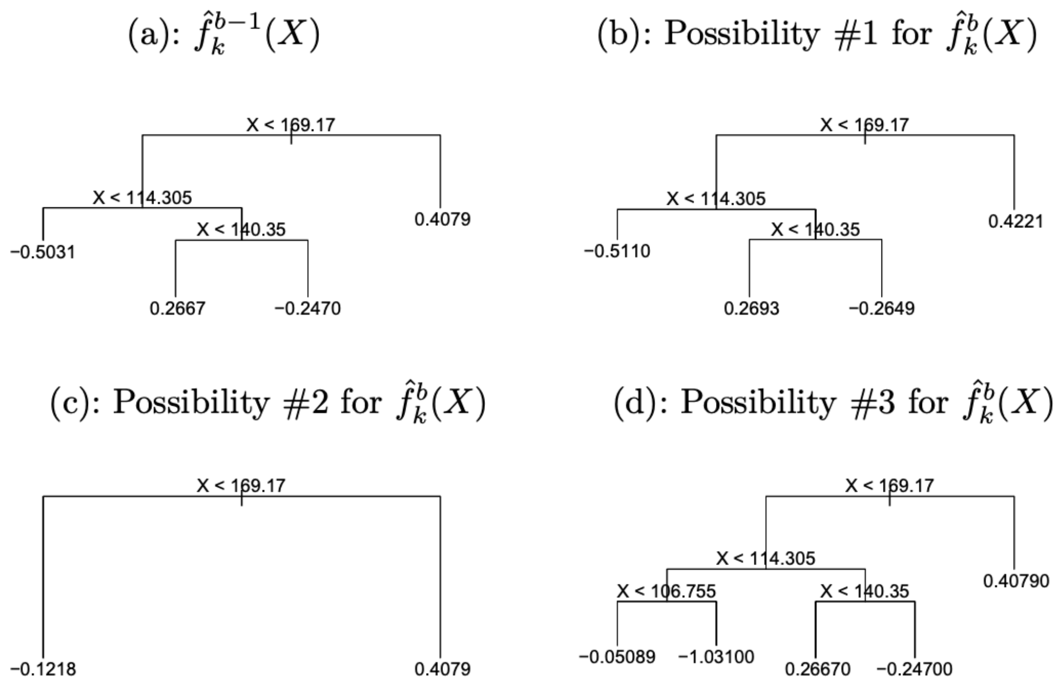


Figure 4. Ensemble Learning Method.[6]

Random Forest

Random Forest learning is an ensemble learning method like bagging but in addition to training multiple models on different samples, it also selects a random subset of features on each split. This allows Random Forest Trees to handle higher dimensional data better than Bagging Trees, further reduces overfitting by reducing the correlation between trees, and generally improve predictions. However, it is even more computationally expensive than Bagging Trees.

Random Forest Models can be tuned by adjusting all the same parameters as Bagging Models, with the addition of the number of features allowed at each split. Lower values reduce variance and reduces overfitting.

Gradient Boost

Gradient Boosted Trees are Ensemble Trees like Random Forest, but rather than training separate, independent trees before finally aggregating their predictions, Gradient Boost learning trains each tree sequentially through correcting mistakes made by previous models (e.g. in a classification problem, it will consider misclassified datapoints to train future models). Gradient Boost often results in better performance, but since it is not parallelizable, it can take significantly longer to train even when compared to Random Forest Trees.

A Gradient Boosted Tree can be tuned by adjusting all the same parameters as a Random Forest Tree, with the addition of the learning rate, which scales the contribution of each new model to the final prediction. A higher learning rate means that the model grows "faster" (i.e. changes more between each tree), allowing the model to fit closer to the training data and therefore increases variance. The learning rate is closely tied to the number of trees in the ensemble – a low learning rate combined with a small number of trees mean that the model will not learn much about the data and therefore may perform worse (i.e. underfitting).

Methodology

Data Preparation

Each variable's value in the NSDUH data is encoded, with their meanings provided in a Codebook[2]. Some variables are numerical with certain values reserved for special meanings (see Figure 4), while others are a mix of categorical and special meanings. Furthermore, some variables were numerical, some were ordinal, and some were nominal.

Due to this, extensive data cleanup was required.

(QD21, QD21DKRE)
EDUSKPCOM   Len : 2   RC - HOW MANY DAYS MISSED SCHOOL FROM SKIPPING (COMBINED)

	Freq	Pct
RANGE = 0 - 30	7219	21.95
94 = DON'T KNOW	22	0.07
97 = REFUSED	13	0.04
98 = BLANK (NO ANSWER)	1666	5.06
99 = LEGITIMATE SKIP	23973	72.88

Figure 5. NSDUH Codebook Example.[2]

The data cleanup process starts with splitting variables into “substance” columns and “demographic” columns, treating “substance” as a pool of response variables to choose from and “demographic” as all our predictor variables. Then we define a list containing values to be treated as a “missing value” so that they may be imputed (such as 94, 97, 98, and 99 for ‘EDUSKPCOM’ as shown in Figure 5). Then variables are categorized as numerical/ordinal/nominal. Finally, we manually special cases such as ‘EDUSCHLGO’ whose value of 11 was merged with 1 (11 means “Yes” with some uncertainty) and ‘EDUSCHGRD2’ which was removed from being a predictor for ‘iralcage’ as it is essentially a proxy for age where the response variable is highly correlated to age.

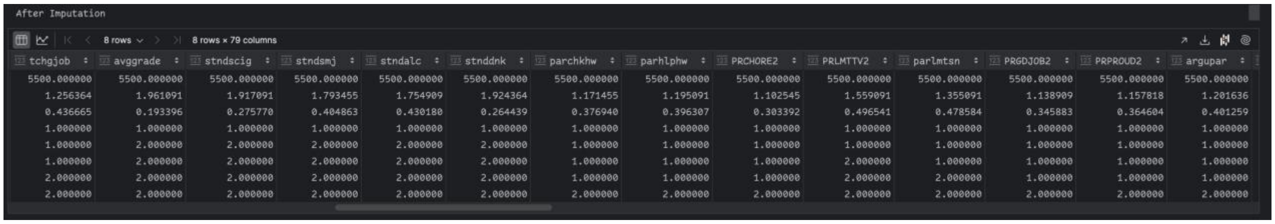
The imputation process uses an Iterative Imputer with a Random Forest Classifier (as it makes few assumptions about the data and the data to be imputed is categorical) estimator to impute missing values for each predictor using all other predictor. This is done to retain as much data as possible since many of our predictors have missing values in different rows. We are also able to confirm that the imputed values are statistically similar to the rest of the data (see Figure 6 and 7). After imputation, some columns were one-hot encoded (as most of the data are already encoded ordinally or are binary).

Before Imputation

8 rows   8 rows × 79 columns

tchgjob	avggrade	stndscig	stndsmj	stndalc	stnddnk	parchkhw	parhlphw	PRCHORE2	PRLMTTV2	parlmtsn	PRGDJOB2	PRPROUD2	argupar
5484.000000	5147.000000	5247.000000	5241.000000	5258.000000	5202.000000	5481.000000	5462.000000	5482.000000	5458.000000	5368.000000	5478.000000	5479.000000	5413.000000
1.256929	1.958422	1.913093	1.784201	1.744960	1.920031	1.171866	1.195899	1.102882	1.560462	1.360469	1.139284	1.158058	1.204877
0.436980	0.199641	0.281725	0.411414	0.435925	0.271272	0.377299	0.396927	0.303833	0.496376	0.480181	0.346275	0.364829	0.403649
1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
1.000000	2.000000	2.000000	2.000000	1.000000	2.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
1.000000	2.000000	2.000000	2.000000	2.000000	2.000000	1.000000	1.000000	1.000000	2.000000	1.000000	1.000000	1.000000	1.000000
2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	1.000000	1.000000	1.000000	2.000000	2.000000	1.000000	1.000000	1.000000
2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000

Figure 6. Statistical summary before imputation



After Imputation

8 rows × 79 columns

tchgjob	avggrade	stndscig	stndsmj	stndalc	stnddnk	parchkhw	parhlphw	PRCHORE2	PRLMTTV2	parlmtsn	PRGDJOB2	PRPROUD2	argupar
5500.000000	5500.000000	5500.000000	5500.000000	5500.000000	5500.000000	5500.000000	5500.000000	5500.000000	5500.000000	5500.000000	5500.000000	5500.000000	5500.000000
1.256364	1.961091	1.917091	1.793455	1.754909	1.924364	1.171455	1.195091	1.102545	1.559091	1.355091	1.138909	1.157818	1.201636
0.436665	0.193396	0.275770	0.404863	0.430180	0.264439	0.376940	0.396307	0.303392	0.496541	0.478584	0.345883	0.364604	0.401259
1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
1.000000	2.000000	2.000000	2.000000	2.000000	2.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
1.000000	2.000000	2.000000	2.000000	2.000000	2.000000	1.000000	1.000000	1.000000	2.000000	1.000000	1.000000	1.000000	1.000000
2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	1.000000	1.000000	1.000000	2.000000	2.000000	1.000000	1.000000	1.000000
2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000

Figure 7. Statistical summary after imputation

Modeling

For each response variable analyzed ('iralcage', 'alcflag', and 'alcydays', we trained a Random Forest model and compared it with a Null Model plus at least 1 other appropriate model (such as Lasso, Decision Trees, and Gradient Boosting). Every model except for the Null Model was trained with all predictors and tuned with Grid Search 5-fold Cross Validation, either sequentially with 1 hyperparameter at a time or exhaustively with multiple hyperparameters at a time. All models were trained and tested on a 70-30 split dataset.

Results

'iralcage' - The age a youth first consumed alcohol

Model	Test MSE
Random Forest Regression	5.41
Null Model	5.91
Ordinary Least Squares	5.42
Lasso Model	5.35

Table 1. Test MSE results predicting 'iralcage' by Model

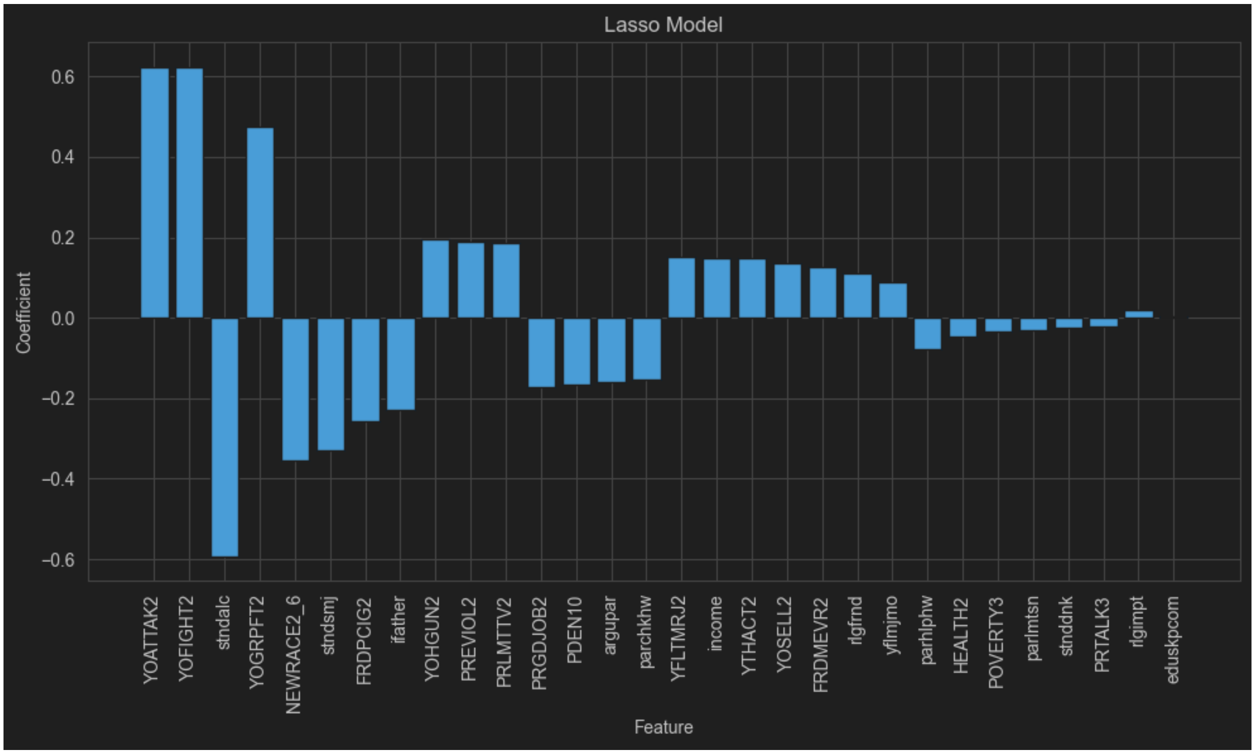


Figure 8. ‘iralcage’ most important features in a Lasso Model.

Predictor	Description	Coefficient
YOATTAK2	Youth attacked with intent to seriously harm	0.625647
YOFIGHT2	Youth had serious fight at school/work	0.623463
STNDALC	Youth’s knowledge of other students’ drinking	-0.594024
YOGRPFT2	Youth fought with group vs other group	0.472173
NEWRACE2_6	Youth of mixed race (non-hispanic)	-0.353626

Table 2. Top 5 predictors for ‘iralcage’.

‘alcflag’ - Whether a youth has ever consumed alcohol

Model	Accuracy Score
Null Model	0.76

Decision Tree	0.8145
Random Forest Classifier	0.8170
Gradient Boosting Classifier	0.8273

Table 3. Accuracy results predicting ‘iralflag’ by Model.

Predictor	Description	Importance
YFLMJMO	Youth’s feelings about peers using marijuana monthly	0.4652
STNDALC	Youth’s knowledge of other students’ drinking	0.2425
EDUSCHGRD2	Youth’s grade in school	0.0615
PRALDLY2	Whether parents strongly disapprove of 1-2 drinks a day	0.0395
FRDMJMON	How youth’s close friends feel about the youth using marijuana monthly	0.0349

Table 4. ‘iralflag’ most important features in a Decision Tree.

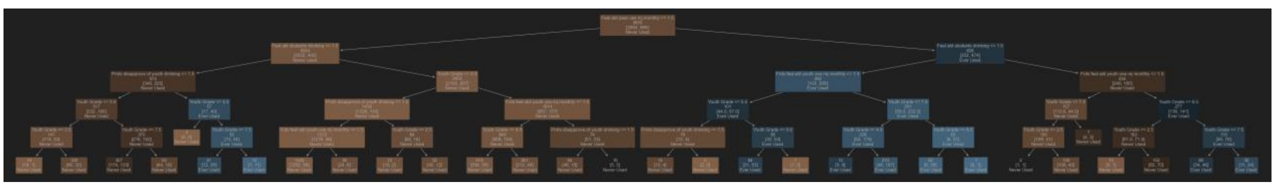


Figure 9. ‘alcflag’ Decision Tree using only top 5 predictors.

‘alcydays’ - Number of days a youth has consumed alcohol within the past year

Model	Accuracy Score
Random Forest Classifier	0.6079
Null Model	0.6049
Gradient Boosting Classifier	0.6049

Table 5. Accuracy results predicting ‘alcydays’ by Model.

Predictor	Description	Importance
EDUSCHGRD2	Youth's grade in school	0.065901
HEALTH2	Youth's health condition	0.058377
COUTYP4	Youth's county's metro/nonmetro status	0.051380
INCOME	Youth's total family income	0.041730
STNDALC	Youth's knowledge of other students' drinking	0.032699

Table 6. 'alcydays' most important features in a Random Forest Tree.

Discussion

Tuning

We were curious whether performing cross-validation one parameter at a time would perform better than performing cross-validation with all parameters at once. The motivation was that if performing cross-validation one parameter at a time perform similarly or better, that would be preferable as the search space would be explored sequentially rather than exhaustively, making it much faster. To this end, for each of the response variables above, we cross-validated a Random Forest Model both by single-parameter tuning and multi-parameter tuning. We find that in all cases the single-parameter tuned model performed similarly or better, and often resulted in a less complex model. This is not a focus of this paper and further exploration may be helpful.

'iralcage' - The age a youth first consumed alcohol

'iralcage' is a numerical column with a possible value of 1-66 and 991 (indicating never used alcohol). We trained various cross-validated models against 'iralcage' using all demographic columns and obtained a result showing that the Lasso Model performed best (see Table 1). Therefore, we used the Lasso Model to determine which factors are most important (see Table 2).

The top four predictors (Youth attacked with intent to seriously harm, Youth had serious fight at school/work, Youth's knowledge of other students' drinking, Youth fought with group vs other group) are binary variables, where 1 is generally a negative (such as youth **has** attacked with intent to seriously harm) and 2 is generally a positive (such as none/few of other students drink alcohol). Since 'iralcage' corresponds to age, a positive correlation means that youth who are not involved in violent acts can be expected to have started drinking later in life. The 5th predictor (Youth of mixed race, non-hispanic) must be treated with caution. It is likely that there are underlying socioeconomic, cultural, or environmental factors associated with this correlation, and we would be good to remember that correlation does not equal causation.

'alcflag' - Whether a youth has ever consumed alcohol

'alcflag' is a binary categorical variable with a possible value of 0 (indicating never used) and 1 (indicating ever used). We trained various cross-validated models against 'alcflag' using all demographic columns and obtained a result showing that the gradient boosted tree performed best (see Table 3). However, for exploration purposes, we will select a basic decision tree to examine.

The top five predictors in order (see Table 4) are: "Youth's feelings about peers using marijuana monthly", "Youth's knowledge of other students' drinking", "Youth's grade in school", "Whether parents strongly disapprove of 1-2 drinks a day", "How youth's close friends feel about the youth using marijuana monthly", which are mostly related to peer/parental influence.

Examining the decision tree (see Figure 9) can tell us the direction of correlation. For example, by tracing the left-most branch, we can see that youth whose peers disagree with monthly marijuana use, whose peers don't drink, whose parents disagree with youth drinking, and in 7th grade or lower is predicted to not have drunk alcohol ever.

‘alcydays’ - Number of days a youth has consumed alcohol within the past year

‘alcydays’ is a multi-class categorical variable with a possible value of 1 (1-11 days), 2 (12-49 days), 3 (50-99 days), 4 (100-299 days), 5 (300-365 days), and 6 (no past year use, though this is removed from the data). We trained various cross-validated models against ‘alcydays’ using all demographic columns and obtained a result showing that Random Forest Classifier performed best (see Table 5). Therefore, we used it to determine which factors are most important (see Table 6). However, unlike the Lasso Model or the Decision Tree, an Ensemble Tree is much harder to interpret as we cannot interpret the model based only on a single tree instance, so we are unable to establish the direction of the correlation.

The top five predictors in order are: “Youth’s grade in school”, “Youth’s health condition”, “Youth’s county’s metro/nonmetro status”, “Youth’s total family income”, “Youth’s knowledge of other students’ drinking”.

Unlike the response variable from previous section, chronic drinking is less (though still) influenced by peer group and more influenced by socioeconomic factors such as income and the county that the youth reside in.

The Importance of Proper Encoding

Under the ‘Methodology’ section it was mentioned that variables were encoded depending on their category as a numerical/ordinal/nominal variable. This is necessary because when dummies are being created, you must not treat an ordinal variable as a binary variable since that implies that each category are independent to one another and will cause misinterpretation or wrong results.

Conclusion

This analysis shows that the factors that correlate with underage drinking are multi-faceted, the only common predictor among them being whether youth’s student peer also drink. Aside from that, the best predictors for a youth’s age at the time of first drink are mostly related to violence, while the best predictors for a youth’s having ever had a drink are mostly related to their peer group, and, perhaps most concerning of all, the best predictors for chronic drinking in youths are socioeconomic factors like income and county.

Future analysis may be useful to uncover more trends between the demographic variables and the substance use variables. There are likely to be similarities between, for example, factors that cause youth to try alcohol and to try marijuana for the first time. Future analysis should also focus on uncovering the underlying causes that affect one racial group disproportionately.

References

1. Substance use/misuse [Internet]. [cited 2024 Apr 15]. Available from: https://youth.gov/youth-topics/substance-abuse

2. 2020 NATIONAL SURVEY ON DRUG USE AND HEALTH [Internet]. [cited 2024 Apr 16]. Available from: https://www.datafiles.samhsa.gov/sites/default/files/field-uploads-protected/studies/NSDUH-2020/NSDUH-2020-datasets/NSDUH-2020-DS0001/NSDUH-2020-DS0001-info/NSDUH-2020-DS0001-info-codebook.pdf

3. Get the facts about underage drinking [Internet]. U.S. Department of Health and Human Services; [cited 2024 Apr 15]. Available from: https://www.niaaa.nih.gov/publications/brochures-and-fact-sheets/underage-drinking

4. National Survey on Drug Use and Health (NSDUH) population data [Internet]. [cited 2024 Apr 15]. Available from: https://www.datafiles.samhsa.gov:443/dataset/national-survey-drug-use-and-health-2020-nsduh-2020-ds0001

5. Mendible A. Youth Data [Internet]. [cited 2024 Apr 15]. Available from: https://github.com/mendible/5322/blob/main/Homework%201/youth_data.csv

6. James G, Witten D, Hastie T, Tibshirani R, Taylor J. An introduction to statistical learning: With applications in Python. Cham, Switzerland: Springer; 2023.

7. Tariverdiyev N. Machine learning algorithms : Decision trees [Internet]. Medium; 2019 [cited 2024 Apr 26]. Available from: https://medium.com/@nadir.tariverdiyev/machine-learning-algorithms-decision-trees-934171552048

Appendix

hw1_code

April 26, 2024

Data exploration

```
[ ]: from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier,␣
     ↪GradientBoostingClassifier
     from sklearn.tree import plot_tree, DecisionTreeClassifier
     import matplotlib.pyplot as plt
     import pandas as pd
     from sklearn.model_selection import train_test_split, GridSearchCV
     from sklearn.linear_model import Lasso
     from sklearn.experimental import enable_iterative_imputer
     from sklearn.impute import IterativeImputer
     from sklearn.metrics import mean_squared_error, confusion_matrix, accuracy_score
     from sklearn.preprocessing import OrdinalEncoder
     from sklearn.dummy import DummyRegressor, DummyClassifier
     import numpy as np
     import seaborn as sns
     sns.set_style("whitegrid")
     from sklearn.linear_model import LinearRegression
```

0.1 Data Preparation

0.1.1 Data Ingestion

```
[ ]: df = pd.read_csv("youth_data.csv")

     substance_dict = {
         "iralcfy": "alcohol frequency past year (1-365)",
         "irmjfy": "marijuana frequency past year (1-365)",
         "ircigfm": "cigarette frequency past month (1-30)",
         "IRSMKLSS30N": "smokeless tobacco frequency past month (1-30)",
         "iralcfm": "alcohol frequency past month (1-30)",
         "irmjfm": "marijuana frequency past month (1-30)",
         "ircigage": "cigarette age of first use (1-55), 991=never used",
         "irsmklsstry": "smokeless tobacco age of first use (1-70), 991=never used",
         "iralcage": "alcohol age of first use (1-66), 991=never used",
         "irmjage": "marijuana age of first use (1-83), 991=never used",
         "mrjflag": "marijuana ever used (0=never, 1=ever)",
         "alcflag": "alcohol ever used (0=never, 1=ever)",
```

```
    "tobflag": "any tobacco ever used (0=never, 1=ever)",
    "alcydays": "number of days of alcohol in past year (1-5 categories,␣
↪6=none)",
    "mrjydays": "number of days of marijuana in past year (1-5 categories,␣
↪6=none)",
    "alcmdays": "number of days of alcohol in past month (1-4 categories,␣
↪5=none)",
    "mrjmdays": "number of days of marijuana in past month (1-4 categories,␣
↪5=none)",
    "cigmdays": "number of days of cigarettes in past month (1-5 categories,␣
↪6=none)",
    "smklsmdays": "number of days of smokeless tobacco in past month (1-4␣
↪categories, 5=none)",
}

# for recorded columns (RC), source variable is commented
demographic_dict = {
    # misc
    "irsex": "binary sex (1=male, 2=female)",
    "NEWRACE2": "RC-RACE/HISPANICITY RECODE", # Unspecified,
    "HEALTH2": "RC-OVERALL HEALTH RECODE", # HEALTH
    "talkprob": "RC-WHO YTH TALKS WITH ABOUT SERIOUS PROBLEMS", # YETLKBGF,␣
↪YETLKNON, YETLKOTA, YETLKPAR, YETLKSOP

    # perception about other youth columns
    "stndscig": "RC-STUDENTS IN YTH GRADE SMOKE CIGARETTES", # YESTSCIG
    "stndsmj": "RC-STUDENTS IN YTH GRADE USE MARIJUANA", # YESTSMJ
    "stndalc": "RC-STUDENTS IN YTH GRADE DRINK ALCOHOLIC BEVERAGES", # YESTSALC
    "stnddnk": "RC-STUDENTS IN YTH GRADE GET DRUNK ONCE/WEEK", # YESTSDNK

    # school columns
    "eduschlgo": "now going to school (1=yes, 2=no)",
    "EDUSCHGRD2": "what grade in now/will be in (11 categories, 98,99= blank/
↪skip)",
    "eduskpcom": "how many days skipped school in past month (0-30, 94/97/98/
↪99=blank/skip)",
    "schfelt": "RC-HOW YTH FELT: ABOUT GOING TO SCHOOL IN PST YR", # YESCHFLT
    "tchgjob": "RC-TEACHER LET YTH KNOW DOING GOOD JOB IN PST YR", # YETCGJOB
    "avggrade": "RC-GRADE AVERAGE FOR LAST GRADING PERIOD COMPLETED", # YELSTGRD

    # parental involvement columns
    "imother": "for youth, mother in household (1=yes, 2=no, 3=don't know,␣
↪4=over 18)",
    "ifather": "for youth, father in household (1=yes, 2=no, 3=don't know,␣
↪4=over 18)",
    "parchkhw": "RC-PARENTS CHECK IF HOMEWORK DONE IN PST YR", # YEPCHKHW
```

```
    "parhlphw": "RC-PARENTS HELP WITH HOMEWORK IN PST YR", # YEPHLPHW
    "PRCHORE2": "RC-PARENTS MAKE YTH DO CHORES AROUND HOUSE IN PST YR", # YEPCHORE
    "PRLMTTV2": "RC-PARENTS LIMIT AMOUNT OF TV IN PST YR", # YEPLMTTV
    "parlmtsn": "RC-PARENTS LIMIT TIME OUT ON SCHOOL NIGHT IN PST YR", # YEPLMTSN
    "PRGDJOB2": "RC-PARENTS TELL YTH HAD DONE GOOD JOB IN PST YR", # YEPGDJOB
    "PRPROUD2": " RC-PARENTS TELL YTH PROUD OF THINGS DONE IN PST YR", # YEPPROUD
    "argupar": "RC-TIMES ARGUED/HAD A FIGHT WITH ONE PARENT IN PST YR", # YEYARGUP
    "PRPKCIG2": "RC-YTH THINK: PARENTS FEEL ABT YTH SMOKE PACK CIG/DAY", # YEPPKCIG
    "PRMJEVR2": " RC-YTH THINK: PARENTS FEEL ABT YTH TRY MARIJUANA", # YEPMJEVR
    "prmjmo": "RC-YTH THINK: PARENTS FEEL ABT YTH USE MARIJUANA MNTHLY", # YEPMJMO
    "PRALDLY2": "RC-YTH THINK: PARNTS FEEL ABT YTH DRK 1-2 ALC BEV/DAY", # YEPALDLY
    "PRTALK3": "RC-TALKED WITH PARENT ABOUT DANGER TOB/ALC/DRG", # YEPRTDNG

    # community program columns
    "PRBSOLV2": "RC-PARTICIPATED IN PRBSLV/COMMSKILL/SELFESTEEM GROUP", # YEPRBSLV
    "PREVIOL2": "RC-PARTICIPATED IN VIOLENCE PREVENTION PROGRAM", # YEVIOPRV
    "PRVDRGO2": "RC-PARTICIPATED IN SUBSTANCE PREV PROGRAM OUTSIDE SCHOOL", # YEDGPRGP
    "GRPCNSL2": "RC-PARTICIPATED IN PROGRAM TO HELP SUBSTANCE USE", # YESLFHLP
    "PREGPGM2": "RC-PARTICIPATED IN PREG/STD PREVENTION PROGRAM", # YEPRGSTD
    "YTHACT2": "RC-YTH PARTICIPATED IN YOUTH ACTIVITIES", # YECOMACT, YEFAIACT, YEOTHACT, YESCHACT
    "DRPRVME3": "RC-YTH SEEN ALC OR DRUG PREVENTION MESSAGE OUTSIDE SCHOOL", # YEPVNTYR
    "ANYEDUC3": "RC-YTH HAD ANY DRUG OR ALCOHOL EDUCATION IN SCHOOL", # YEDECLAS, YEDERGLR, YEDESPCL

    # religious beliefs columns
    "rlgattd": "RC-NUMBER TIMES ATTEND RELIGIOUS SERVICES IN PST YR", # YERLGSVC
    "rlgimpt": "RC-RELIGIOUS BELIEFS VERY IMPORTANT IN LIFE", # YERLGIMP
    "rlgdcsn": "RC-RELIGIOUS BELIEFS INFLUENCE LIFE DECISIONS", # YERLDCSN
    "rlgfrnd": "RC-IMPORTANT FOR FRIENDS TO SHARE RELIGIOUS BELIEFS", # YERLFRND

    # financial situation columns
    "income": "total family income (4 categories)",
    "govtprog": "got gov assistance (1=yes, 2=no)",
    "POVERTY3": "poverty level (4 categories)",
```

```python
    # peer influence columns
    "YFLPKCG2": "RC-HOW YTH FEELS: PEERS SMOKE PACK/DAY CIG", # YEGPKCIG
    "YFLTMRJ2": "RC-HOW YTH FEELS: PEERS TRY MARIJUANA", # YEGMJEVR
    "yflmjmo": "RC-HOW YTH FEELS: PEERS USING MARIJUANA MONTHLY", # YEGMJMO
    "YFLADLY2": "RC-HOW YTH FEELS: PEERS DRNK 1-2 ALC BEV/DAY", # YEGALDLY
    "FRDPCIG2": "RC-YTH THINK: CLSE FRND FEEL ABT YTH SMK 1+ PAC DAILY", # YEFPKCIG
    "FRDMEVR2": "RC-YTH THINK: CLOSE FRNDS FEEL ABT YTH TRY MARIJUANA", # YEFMJEVR
    "frdmjmon": "RC-YTH THINK: CLSE FRNDS FEEL ABT YTH USE MARIJUANA MON", # YEFMJMO
    "FRDADLY2": "RC-YTH THINK: CLSE FRND FEEL ABT YTH HAVE 1-2 ALC/DAY", # YEFALDLY

    # societal environment columns
    "PDEN10": "population density (1= >1M people, 2=<1M people, 3=can't be determined)",
    "COUTYP4": "metro size status (1=large metro, 2=small metro, 3=nonmetro)",

    # violence and crime columns
    "YOFIGHT2": "RC-YOUTH HAD SERIOUS FIGHT AT SCHOOL/WORK", # YOFIGHT2
    "YOGRPFT2": "RC-YOUTH FOUGHT WITH GROUP VS OTHER GROUP", # YEYFGTGP
    "YOHGUN2": "RC-YOUTH CARRIED A HANDGUN", # YEYHGUN
    "YOSELL2": "RC-YOUTH SOLD ILLEGAL DRUGS", # YEYSELL
    "YOSTOLE2": "RC-YOUTH STOLE/TRIED TO STEAL ITEM >$50", # YEYSTOLE
    "YOATTAK2": "RC-YOUTH ATTACKED WITH INTENT TO SERIOUSLY HARM", # YEYATTAK
}

# if a row's columns has these values, either the row must be removed or the values must be imputed. These are our predictors.
impute_cols = {
    "eduschlgo": [85, 94, 97, 98],
    "EDUSCHGRD2": [98, 99],
    "eduskpcom": [94, 97, 98, 99],
    "imother": [3],
    "ifather": [3],
    "PDEN10": [3],
}

# various other filters
numerical_cols = ['eduskpcom']
ordinal_cols = ['income', 'POVERTY3', 'PDEN10', 'COUTYP4', 'HEALTH2', 'EDUSCHGRD2'] # used just to filter the nominal columns, they're already encoded as ordinal
nominal_cols = list(set(demographic_dict.keys()) - set(ordinal_cols) - set(numerical_cols)) # most of them are actually already binary
```

```
nominal_cols_to_encode = ['NEWRACE2'] # these are the only ones that aren't␣
↪binary

# further special handling for values that can be combined
df.loc[df['eduschlgo'] == 11, 'eduschlgo'] = 1
```

0.1.2 Data Imputation

[ ]:
```
# impute all missing or 'bad' values as specified in the filter for our␣
↪predictors, using all other predictors
# set bad values to np.nan so that they are imputed
for col, values in impute_cols.items():
    df[col] = df[col].replace(values, np.nan)
```

[ ]:
```
# before imputation - to show that imputed values are statistically similar to␣
↪the rest of the data
print("Before Imputation")
df.describe()
```

```
Before Imputation
```

[ ]:
```
           iralcfy      irmjfy     ircigfm  IRSMKLSS30N     iralcfm  \
count  5500.000000 5500.000000 5500.000000 5500.000000 5500.000000
mean    798.448545  883.120182   89.746182   90.367818   83.735364
std     387.122577  296.643056   10.675065    7.502499   24.836639
min       1.000000    1.000000    1.000000    1.000000    1.000000
25%     991.000000  991.000000   91.000000   91.000000   91.000000
50%     991.000000  991.000000   91.000000   91.000000   91.000000
75%     991.000000  991.000000   91.000000   91.000000   91.000000
max     993.000000  993.000000   93.000000   93.000000   93.000000

            irmjfm    ircigage  irsmklsstry    iralcage      irmjage  …  \
count  5500.000000 5500.000000 5500.000000 5500.000000 5500.000000  …
mean     85.878818  916.850909  961.495273  748.959273  852.804545  …
std      20.112086  258.904197  167.264944  421.926052  340.492329  …
min       1.000000    5.000000    3.000000    1.000000    1.000000  …
25%      91.000000  991.000000  991.000000  991.000000  991.000000  …
50%      91.000000  991.000000  991.000000  991.000000  991.000000  …
75%      91.000000  991.000000  991.000000  991.000000  991.000000  …
max      93.000000  991.000000  991.000000  991.000000  991.000000  …

         eduschlgo   EDUSCHGRD2   eduskpcom     imother     ifather  \
count  5431.000000 4752.000000 4357.000000 5473.000000 5465.000000
mean      1.122813    5.002104    0.572642    1.071990    1.229460
std       0.328253    1.741684    2.186141    0.258495    0.420524
min       1.000000    1.000000    0.000000    1.000000    1.000000
25%       1.000000    4.000000    0.000000    1.000000    1.000000
```

```
50%       1.000000     5.000000     0.000000     1.000000     1.000000
75%       1.000000     6.000000     0.000000     1.000000     1.000000
max       2.000000    10.000000    30.000000     2.000000     2.000000

              income     govtprog     POVERTY3       PDEN10      COUTYP4
count  5500.000000  5500.000000  5500.000000  5117.000000  5500.000000
mean      3.090545     1.808727     2.512909     1.560485     1.754909
std       1.087791     0.393339     0.745000     0.496377     0.756330
min       1.000000     1.000000     1.000000     1.000000     1.000000
25%       2.000000     2.000000     2.000000     1.000000     1.000000
50%       4.000000     2.000000     3.000000     2.000000     2.000000
75%       4.000000     2.000000     3.000000     2.000000     2.000000
max       4.000000     2.000000     3.000000     2.000000     3.000000

[8 rows x 79 columns]
```

[ ]:
```python
# imputation - use only other predictors to impute the missing values
demographic_data = df[list(demographic_dict.keys())]
imputer = IterativeImputer(estimator=RandomForestClassifier(random_state=333),␣
↪max_iter=5, random_state=333)

# round the imputed values to the nearest integer since they are categorical
df[list(demographic_dict.keys())] = np.round(imputer.
↪fit_transform(df[list(demographic_dict.keys())]))
```

```
/opt/homebrew/anaconda3/envs/py39/lib/python3.9/site-
packages/sklearn/impute/_iterative.py:801: ConvergenceWarning:
[IterativeImputer] Early stopping criterion not reached.
  warnings.warn(
```

[ ]:
```python
# after imputation
print("After Imputation")
df.describe()
```

```
After Imputation
```

[ ]:
```
             iralcfy       irmjfy      ircigfm  IRSMKLSS30N      iralcfm  \
count  5500.000000  5500.000000  5500.000000  5500.000000  5500.000000
mean    798.448545   883.120182    89.746182    90.367818    83.735364
std     387.122577   296.643056    10.675065     7.502499    24.836639
min       1.000000     1.000000     1.000000     1.000000     1.000000
25%     991.000000   991.000000    91.000000    91.000000    91.000000
50%     991.000000   991.000000    91.000000    91.000000    91.000000
75%     991.000000   991.000000    91.000000    91.000000    91.000000
max     993.000000   993.000000    93.000000    93.000000    93.000000

              irmjfm     ircigage  irsmklsstry     iralcage      irmjage  …  \
count  5500.000000  5500.000000  5500.000000  5500.000000  5500.000000  …
```

```
mean     85.878818   916.850909   961.495273   748.959273   852.804545  …
std      20.112086   258.904197   167.264944   421.926052   340.492329  …
min       1.000000     5.000000     3.000000     1.000000     1.000000  …
25%      91.000000   991.000000   991.000000   991.000000   991.000000  …
50%      91.000000   991.000000   991.000000   991.000000   991.000000  …
75%      91.000000   991.000000   991.000000   991.000000   991.000000  …
max      93.000000   991.000000   991.000000   991.000000   991.000000  …

         eduschlgo   EDUSCHGRD2    eduskpcom      imother      ifather  \
count  5500.000000  5500.000000  5500.000000  5500.000000  5500.000000
mean      1.121273     4.990909     0.453818     1.071636     1.228727
std       0.326474     1.714350     1.959550     0.257908     0.420051
min       1.000000     1.000000     0.000000     1.000000     1.000000
25%       1.000000     4.000000     0.000000     1.000000     1.000000
50%       1.000000     5.000000     0.000000     1.000000     1.000000
75%       1.000000     6.000000     0.000000     1.000000     1.000000
max       2.000000    10.000000    30.000000     2.000000     2.000000

            income     govtprog     POVERTY3       PDEN10      COUTYP4
count  5500.000000  5500.000000  5500.000000  5500.000000  5500.000000
mean      3.090545     1.808727     2.512909     1.590909     1.754909
std       1.087791     0.393339     0.745000     0.491711     0.756330
min       1.000000     1.000000     1.000000     1.000000     1.000000
25%       2.000000     2.000000     2.000000     1.000000     1.000000
50%       4.000000     2.000000     3.000000     2.000000     2.000000
75%       4.000000     2.000000     3.000000     2.000000     2.000000
max       4.000000     2.000000     3.000000     2.000000     3.000000

[8 rows x 79 columns]
```

[ ]:
```
# after adding NaNs to a column of type int, it becomes float, so convert it␣
↪back to int
for col in demographic_dict.keys():
    df[col] = df[col].astype('int')

# convert categorical column types, so get_dummies will work properly
for col in ordinal_cols + nominal_cols:
    df[col] = df[col].astype('category')
```

0.1.3 Data Encoding

[ ]:
```
# encode categorical columns.
# in our case, all of our ordinal columns are already encoded. Though most of␣
↪our nominal columns are already binary, we will encode them anyway
df_dummies = pd.get_dummies(df[nominal_cols_to_encode], drop_first=True)
df = pd.concat([df, df_dummies], axis=1)
df = df.drop(columns=nominal_cols_to_encode)
```

0.2 Regression: 'iralcage'

'iralcage' is defined as alcohol age of first use. Since it is a numerical variable, we will use regression models.

0.2.1 Data Preparation

We will remove the predictor 'EDUSCHGRD2' (a youth's grade in school) as it is essentially a proxy for age. It is not insightful as it is self-evident that alcohol use is more likely as a youth grows older.

```
[ ]: # Remove all rows where 'iralcage' == 991 (means never used, we can combine␣
      ↪these findings with alcflag later)
     df_iralcage = df.copy()
     df_iralcage = df_iralcage[df_iralcage['iralcage'] != 991]
     df_iralcage = df_iralcage.drop(columns=['EDUSCHGRD2'])
     X = df_iralcage.drop(substance_dict.keys(), axis=1)
     y = df_iralcage['iralcage']
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
      ↪random_state=333)
```

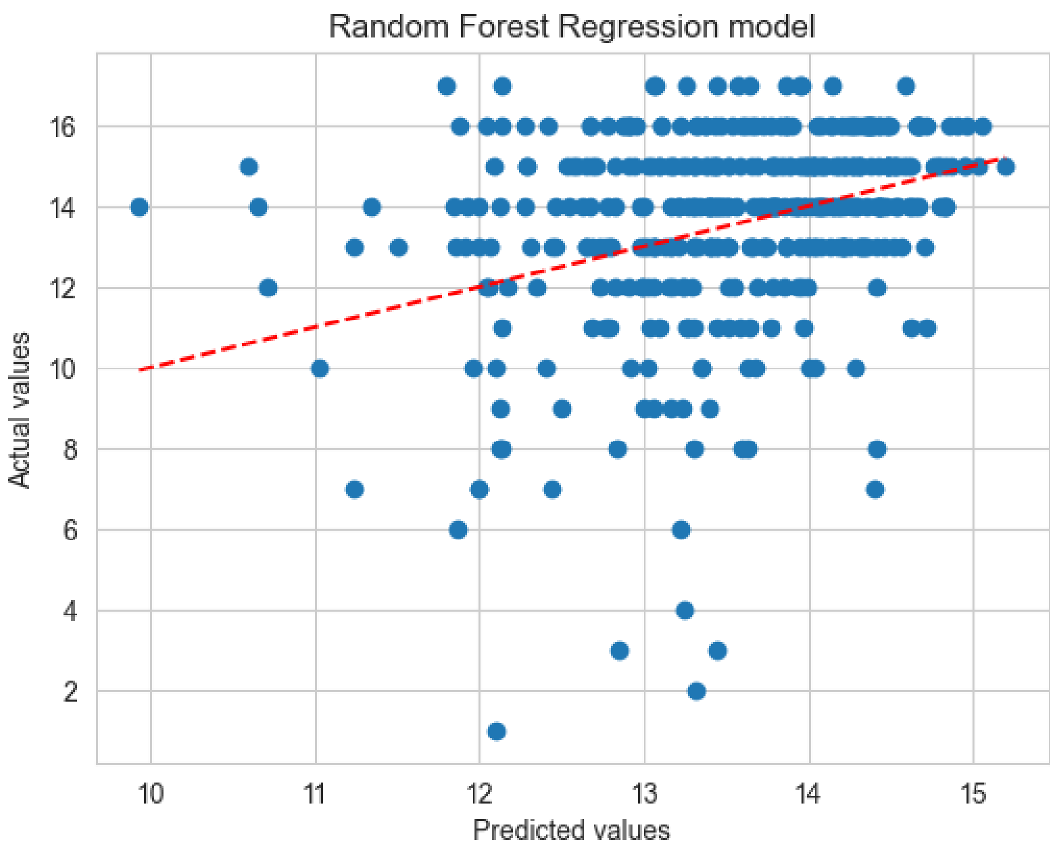
0.2.2 Single-Parameter Tuning

Single-parameter tuning allows us to more easily visualize the effects of each parameter on MSE and choose values that results in a simpler model, but may not give us the "best" values. Furthermore, it is computationally less expensive as the search space is explored sequentially rather than exhaustively.

Baseline Model

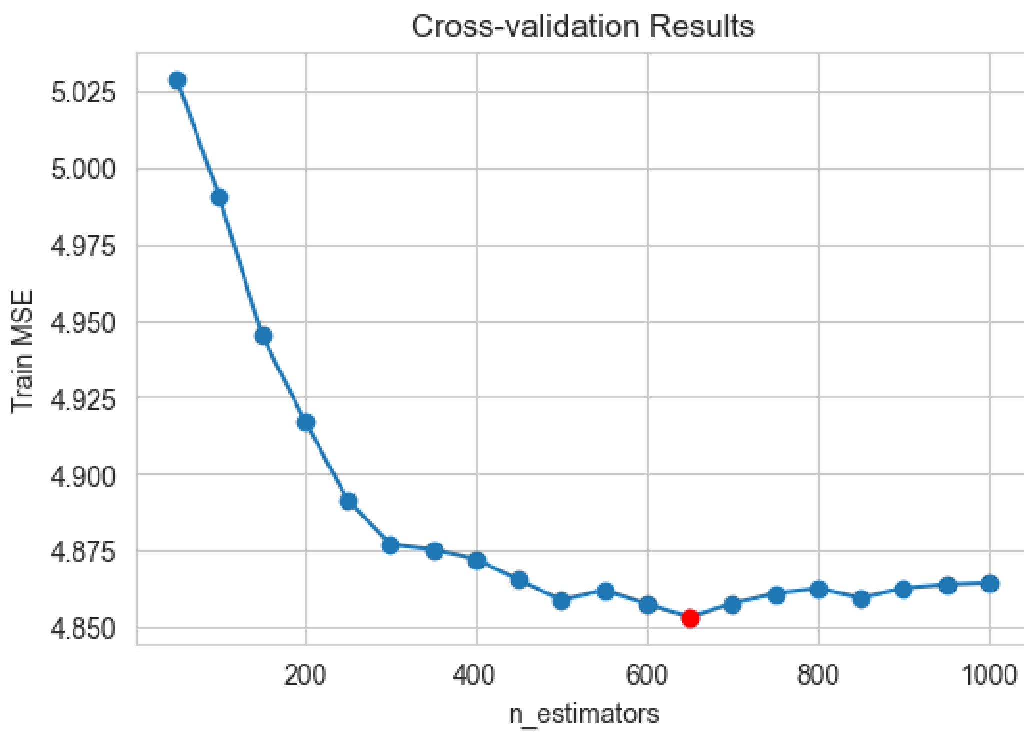
```
[ ]: rf = RandomForestRegressor(max_features='sqrt', n_estimators=100,␣
      ↪random_state=333)
     rf.fit(X_train,y_train)
     y_pred = rf.predict(X_test)

     # plot predicted vs actual values
     plt.scatter(y_pred, y_test)
     plt.plot([min(y_pred), max(y_pred)], [min(y_pred), max(y_pred)], 'r--')
     plt.xlabel('Predicted values')
     plt.ylabel('Actual values')
     plt.title('Random Forest Regression model');

     # mse
     print("Test MSE: {:.2f}".format(mean_squared_error(y_test, y_pred)))
```

```
Test MSE: 5.46
```



Tune 'n_estimators'

```
[ ]: cv_params = {'n_estimators': np.arange(50, 1001, 50)}
cv = GridSearchCV(rf, cv_params, cv=5, scoring='neg_mean_squared_error',␣
 ↪n_jobs=-1) # parallelize with all cores
cv.fit(X_train, y_train);
cv_results = cv.cv_results_
cv_results['mean_test_score'] = -cv_results['mean_test_score']
best_n_estimators = cv.best_params_['n_estimators']
best_score = -cv.best_score_
plt.figure(figsize=(6, 4))
plt.plot(cv_results['param_n_estimators'], cv_results['mean_test_score'], 'o-')
plt.plot(best_n_estimators, best_score, 'ro-')
plt.xlabel('n_estimators')
plt.ylabel('Train MSE')
plt.title('Cross-validation Results')
plt.show()
print(f"best_n_estimators: {best_n_estimators}")
print(f"best_score: {best_score}")
```

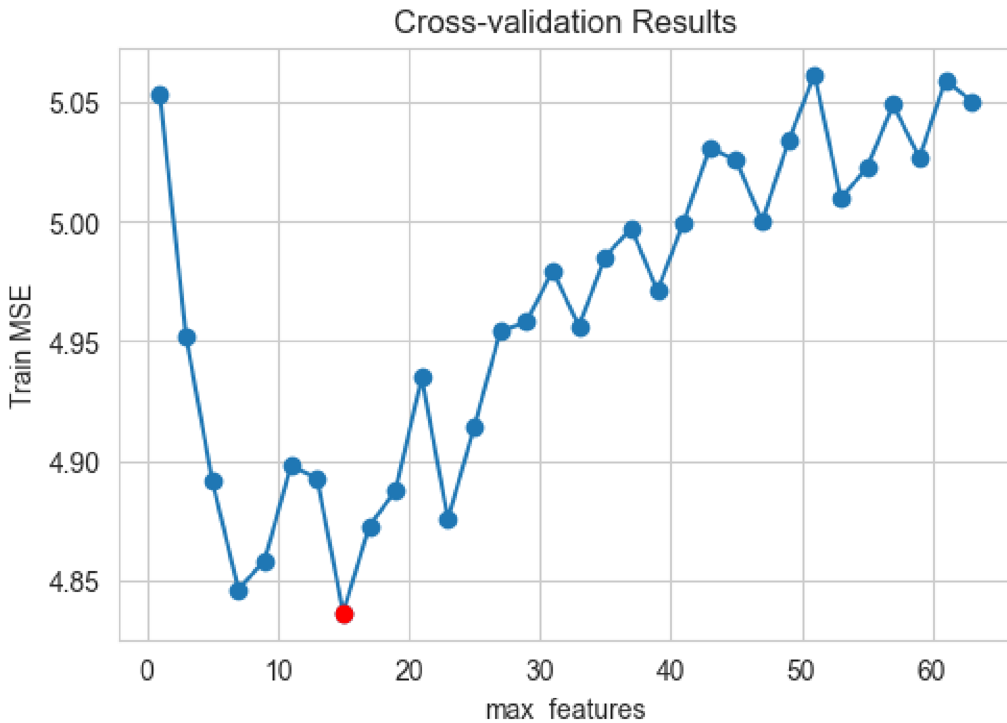



```
best_n_estimators: 650
best_score: 4.853376089541999
```

The best n_estimator is 650 but 500 performs similarly.

Tune 'max_features'

```
[ ]: rf.set_params(n_estimators=500)
cv_params = {'max_features': np.arange(1, X_train.shape[1] + 1, 2)}
cv = GridSearchCV(rf, cv_params, cv=5, scoring='neg_mean_squared_error',␣
 ↪n_jobs=-1) # parallelize with all cores
cv.fit(X_train, y_train)
cv_results = cv.cv_results_
cv_results['mean_test_score'] = -cv_results['mean_test_score']
best_max_features = cv.best_params_['max_features']
best_score = -cv.best_score_
plt.figure(figsize=(6, 4))
plt.plot(cv_results['param_max_features'], cv_results['mean_test_score'], 'o-')
plt.plot(best_max_features, best_score, 'ro-')
plt.xlabel('max_features')
plt.ylabel('Train MSE')
plt.title('Cross-validation Results')
plt.show()
print(f"best_max_features: {best_max_features}")
print(f"best_score: {best_score}")
```



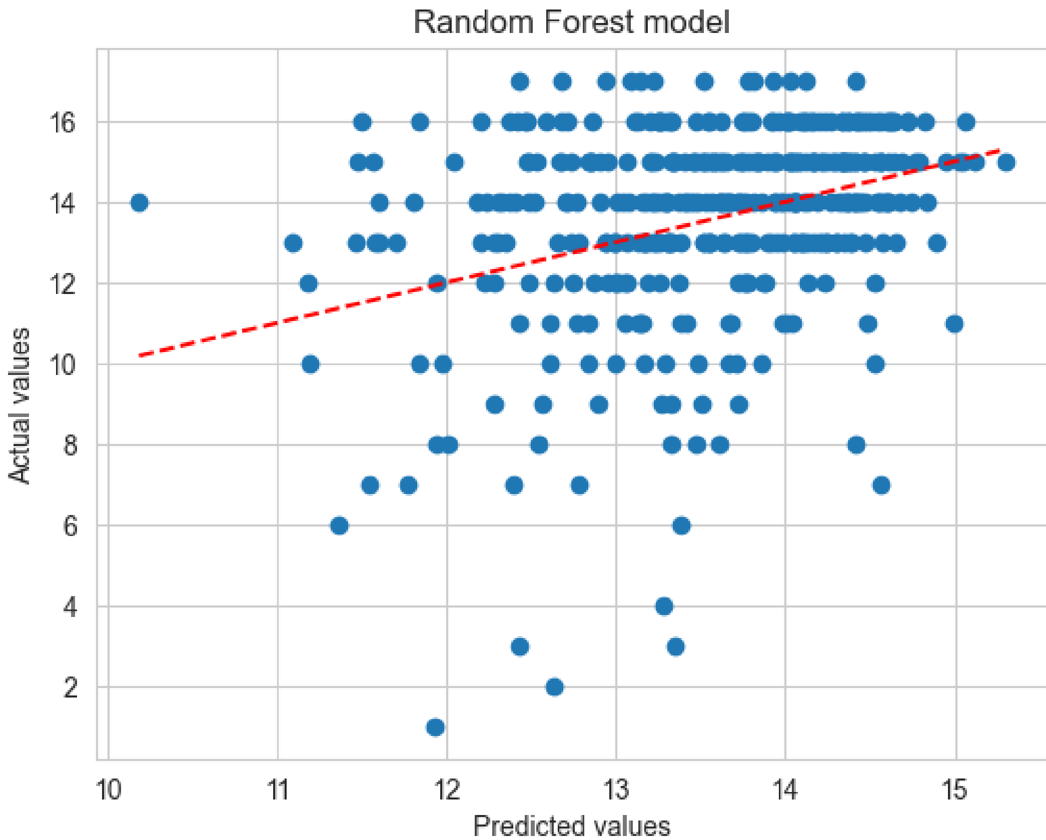

```
best_max_features: 15
best_score: 4.836360621903799
```

The best 'max_features' is 15, but a value of 7 provides similar performance.

Evaluation

[ ]:
```
rf = RandomForestRegressor(n_estimators=300, max_features=7, random_state=333)
rf.fit(X_train,y_train)
y_pred = rf.predict(X_test)

# plot predicted vs actual values
plt.scatter(y_pred, y_test)
plt.plot([min(y_pred), max(y_pred)], [min(y_pred), max(y_pred)], 'r--')
plt.xlabel('Predicted values')
plt.ylabel('Actual values')
plt.title('Random Forest model');

# mse
print("Test MSE: {:.2f}".format(mean_squared_error(y_test, y_pred)))
```

```
Test MSE: 5.38
```



0.2.3 Multi-Parameter Tuning

In contrast, multi-parameter tuning is much more computationally expensive as we test every single permutation, but it may give better results

Tune both 'n_estimators' and 'max_features'

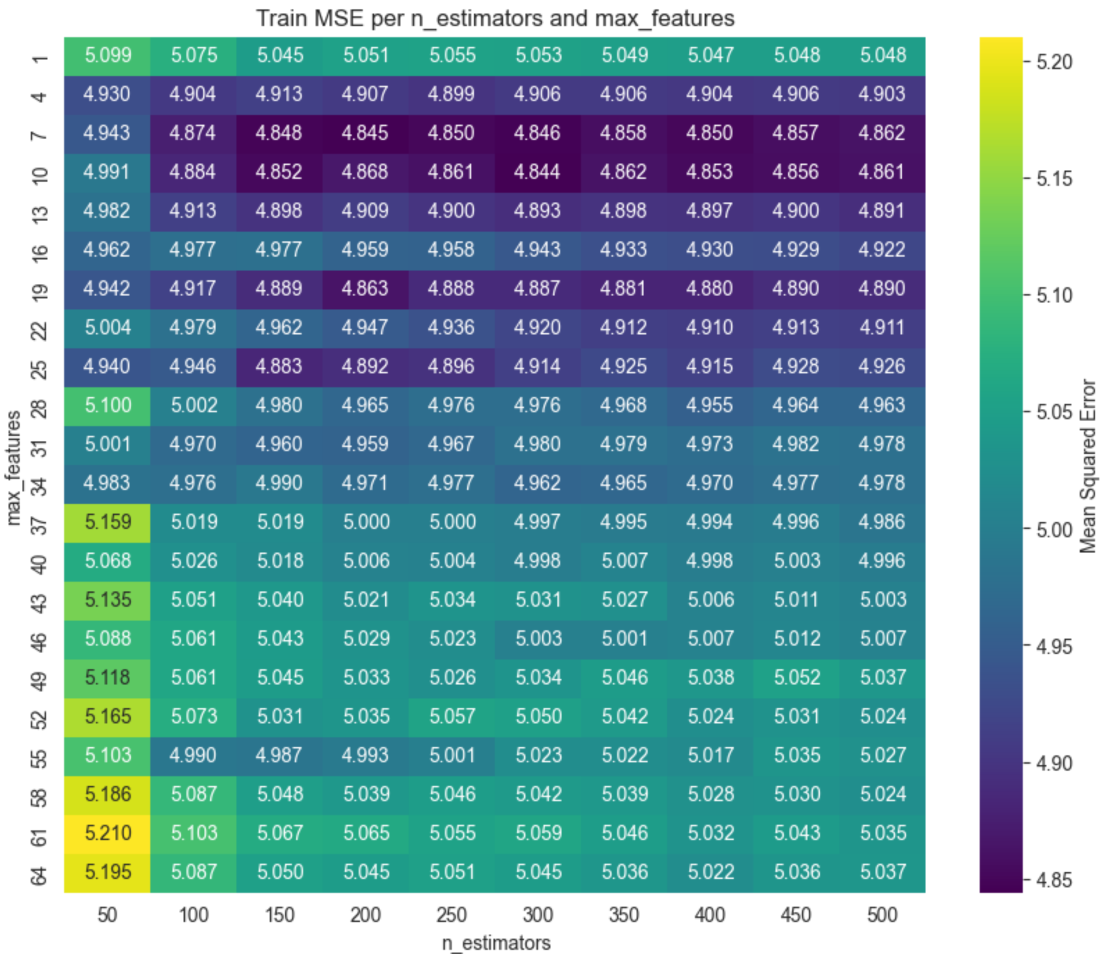
```
[ ]: cv_params = {'n_estimators': np.arange(50, 501, 50), 'max_features': np.
      ↪arange(1, X_train.shape[1] + 1, 3)}
     cv = GridSearchCV(rf, cv_params, cv=5, scoring='neg_mean_squared_error',␣
      ↪n_jobs=-1) # parallelize with all cores
     cv.fit(X_train, y_train);
```

```
[ ]: cv_results = pd.DataFrame(cv.cv_results_)
     cv_results['mean_test_score'] = -cv_results['mean_test_score']  # Convert to␣
      ↪positive mse
     cv_table = cv_results.pivot('param_max_features', 'param_n_estimators',␣
      ↪'mean_test_score')
     best_n_estimators = cv.best_params_['n_estimators']
     best_max_features = cv.best_params_['max_features']
     best_score = -cv.best_score_
```

```
/var/folders/jy/pdgtcrw968d_3_tqfzr3w6y00000gn/T/ipykernel_5030/93146722.py:3:
FutureWarning: In a future version of pandas all arguments of DataFrame.pivot
will be keyword-only.
  cv_table = cv_results.pivot('param_max_features', 'param_n_estimators',
'mean_test_score')
/var/folders/jy/pdgtcrw968d_3_tqfzr3w6y00000gn/T/ipykernel_5030/93146722.py:3:
FutureWarning: In a future version, the Index constructor will not infer numeric
dtypes when passed object-dtype sequences (matching Series behavior)
  cv_table = cv_results.pivot('param_max_features', 'param_n_estimators',
'mean_test_score')
/var/folders/jy/pdgtcrw968d_3_tqfzr3w6y00000gn/T/ipykernel_5030/93146722.py:3:
FutureWarning: In a future version, the Index constructor will not infer numeric
dtypes when passed object-dtype sequences (matching Series behavior)
  cv_table = cv_results.pivot('param_max_features', 'param_n_estimators',
'mean_test_score')
```

[ ]:

```python
plt.figure(figsize=(10, 8))
sns.heatmap(cv_table, annot=True, fmt=".3f", cmap='viridis', cbar_kws={'label':␣
 ↪'Mean Squared Error'})
plt.title('Train MSE per n_estimators and max_features')
plt.xlabel('n_estimators')
plt.ylabel('max_features')
plt.show()

print(f"best_n_estimators: {best_n_estimators}")
print(f"best_max_features: {best_max_features}")
print(f"best_score: {best_score}")
```



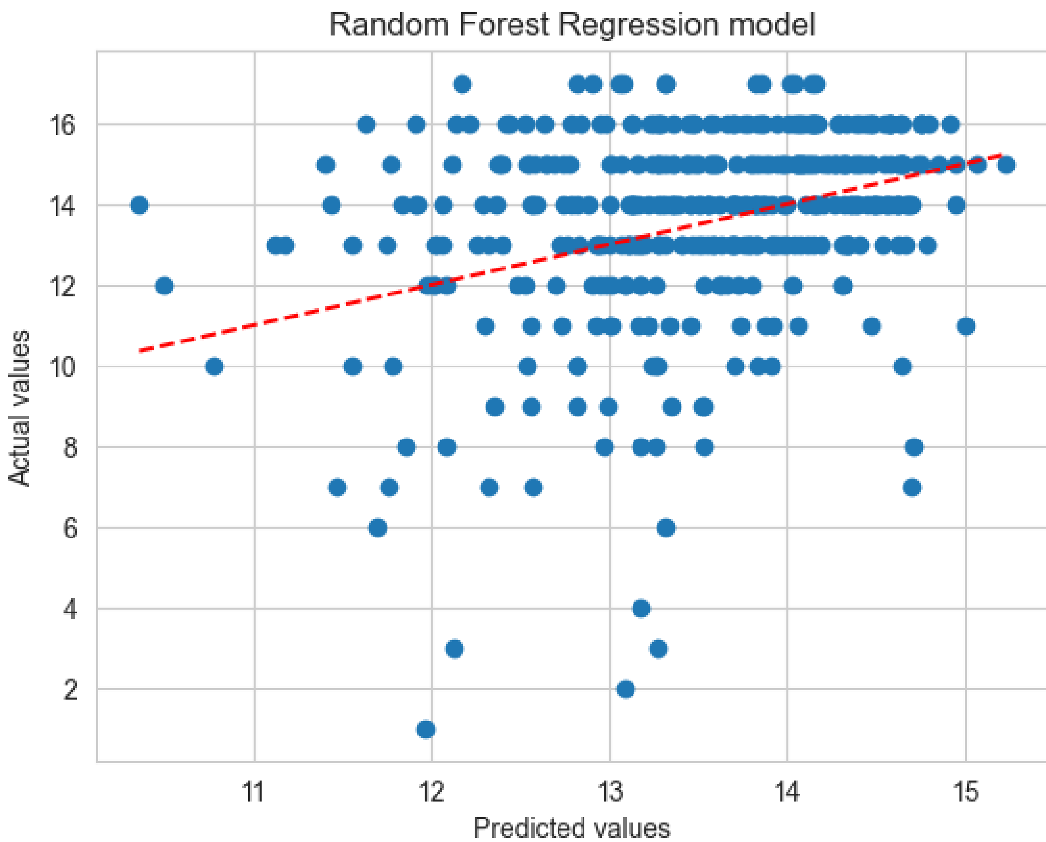

```
best_n_estimators: 300
best_max_features: 10
best_score: 4.844120030256948
```

Evaluation

```
[ ]: rf = RandomForestRegressor(max_features=best_max_features,␣
       ↪n_estimators=best_n_estimators, random_state=333)
     rf.fit(X_train,y_train)
     y_pred = rf.predict(X_test)

     # plot predicted vs actual values
     plt.scatter(y_pred, y_test)
     plt.plot([min(y_pred), max(y_pred)], [min(y_pred), max(y_pred)], 'r--')
     plt.xlabel('Predicted values')
     plt.ylabel('Actual values')
     plt.title('Random Forest Regression model');
```

```
# mse
print("Test MSE: {:.2f}".format(mean_squared_error(y_test, y_pred)))
```

```
Test MSE: 5.41
```



0.2.4 Model Comparisons

Here we will compare our Random Forest Regression against other models

[ ]:
```
# null model
null_model = DummyRegressor(strategy='mean')
null_model.fit(X_train, y_train)
y_pred = null_model.predict(X_test)
print("Null Model Test MSE: {:.2f}".format(mean_squared_error(y_test, y_pred)))
```

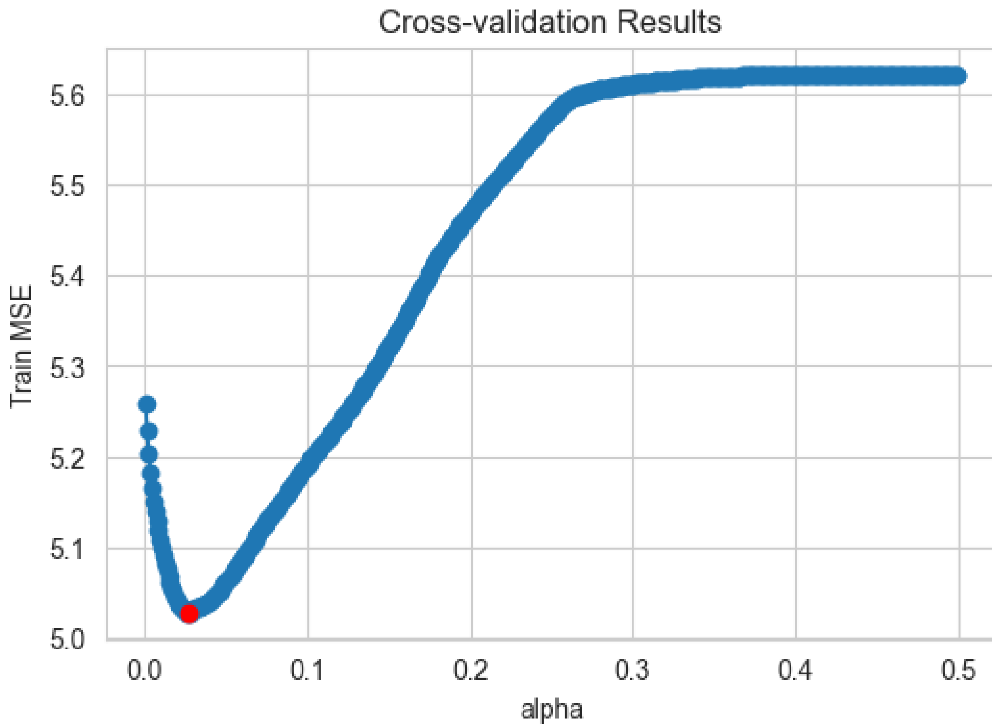
```
Null Model Test MSE: 5.91
```

[ ]:
```
# OLS
lm = LinearRegression()
lm.fit(X_train, y_train)
y_pred = lm.predict(X_test)
```

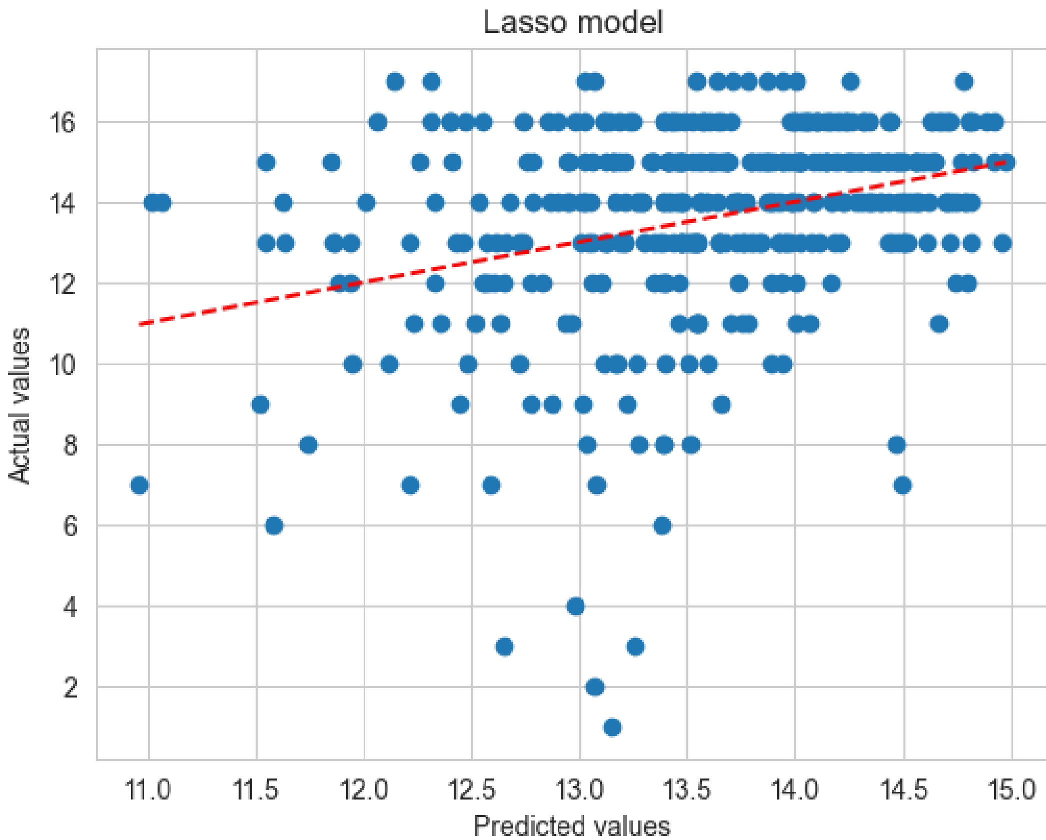
```
print("OLS Test MSE: {:.2f}".format(mean_squared_error(y_test, y_pred)))
```

```
OLS Test MSE: 5.42
```

[ ]:
```
lasso = Lasso(random_state=333)
cv_params = {'alpha': np.arange(0.001, 0.5, 0.001)}
cv = GridSearchCV(lasso, cv_params, cv=5, scoring='neg_mean_squared_error',␣
 ↪n_jobs=-1) # parallelize with all cores
cv.fit(X_train, y_train);
cv_results = cv.cv_results_
cv_results['mean_test_score'] = -cv_results['mean_test_score']
best_alpha = cv.best_params_['alpha']
best_score = -cv.best_score_
plt.figure(figsize=(6, 4))
plt.plot(cv_results['param_alpha'], cv_results['mean_test_score'], 'o-')
plt.plot(best_alpha, best_score, 'ro-')
plt.xlabel('alpha')
plt.ylabel('Train MSE')
plt.title('Cross-validation Results')
plt.show()
print(f"best_alpha: {best_alpha}")
print(f"best_score: {best_score}")

lasso = Lasso(alpha=best_alpha, random_state=333)
lasso.fit(X_train,y_train)
y_pred = lasso.predict(X_test)
plt.scatter(y_pred, y_test)
plt.plot([min(y_pred), max(y_pred)], [min(y_pred), max(y_pred)], 'r--')
plt.xlabel('Predicted values')
plt.ylabel('Actual values')
plt.title('Lasso model');

print("Test MSE: {:.2f}".format(mean_squared_error(y_test, y_pred)))
```




```
best_alpha: 0.027000000000000003
best_score: 5.0278438667324545
Test MSE: 5.35
```

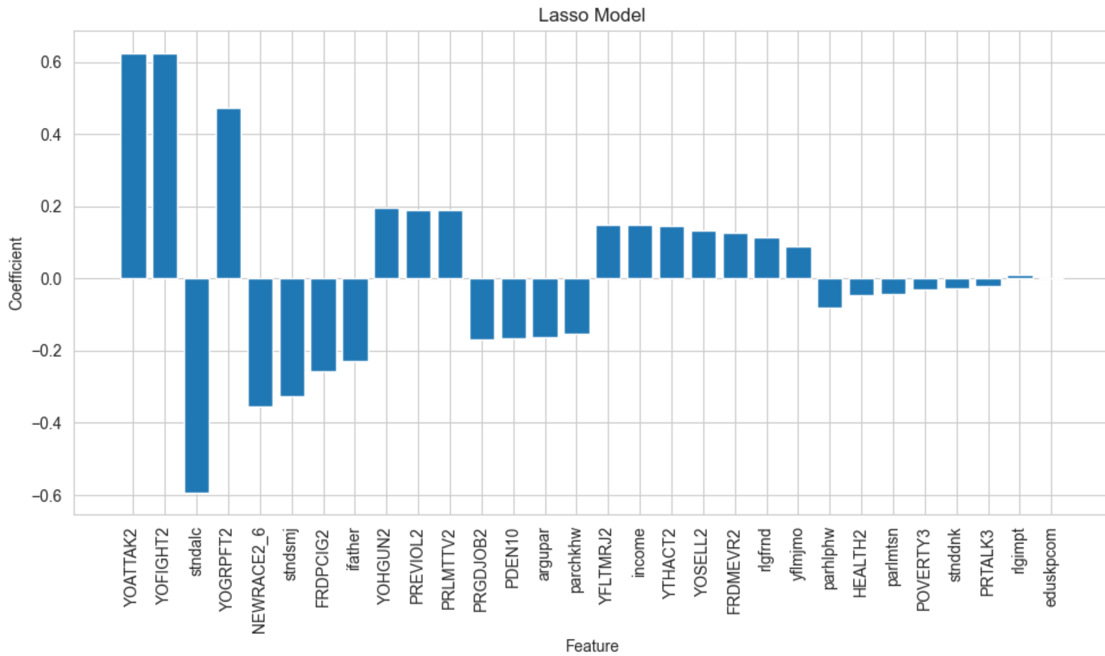


0.2.5 Final Evaluation

The Lasso model performed better than our Random Forest Regression model, hence we will use it to analyze the factors correlated with 'iralcage' as it is more easily interpreted.

```
[ ]: feature_importances = pd.DataFrame({'feature': X.columns, 'coefficient': lasso.
       ↪coef_})
     feature_importances['abs_coefficient'] = abs(feature_importances['coefficient'])
     feature_importances = feature_importances.sort_values('abs_coefficient',␣
       ↪ascending=False)
     feature_importances = feature_importances[feature_importances['coefficient'] !=␣
       ↪0]
     feature_importances = feature_importances.reset_index(drop=True)

     plt.figure(figsize=(10, 6))
     plt.bar(feature_importances['feature'], feature_importances['coefficient'])
     plt.xticks(rotation=90)
     plt.ylabel('Coefficient')
     plt.xlabel('Feature')
     plt.title('Lasso Model')
```

```
plt.tight_layout()
plt.show()

print(feature_importances.head(5))
```




```
     feature  coefficient  abs_coefficient
0   YOATTAK2     0.625647         0.625647
1   YOFIGHT2     0.623463         0.623463
2    stndalc    -0.594024         0.594024
3   YOGRPFT2     0.472173         0.472173
4 NEWRACE2_6    -0.353626         0.353626
```

0.3 Random Forest Binary Classifier: 'alcflag'

'alcflag' is defined as whether a youth have ever used alcohol. It is a binary categorical variable, so we will apply classification techniques.

0.3.1 Data Preparation

[ ]:
```
df_alcflag = df.copy()
X = df_alcflag.drop(substance_dict.keys(), axis=1)
y = df_alcflag['alcflag']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=333)
```

0.3.2 Single-Parameter Tuning

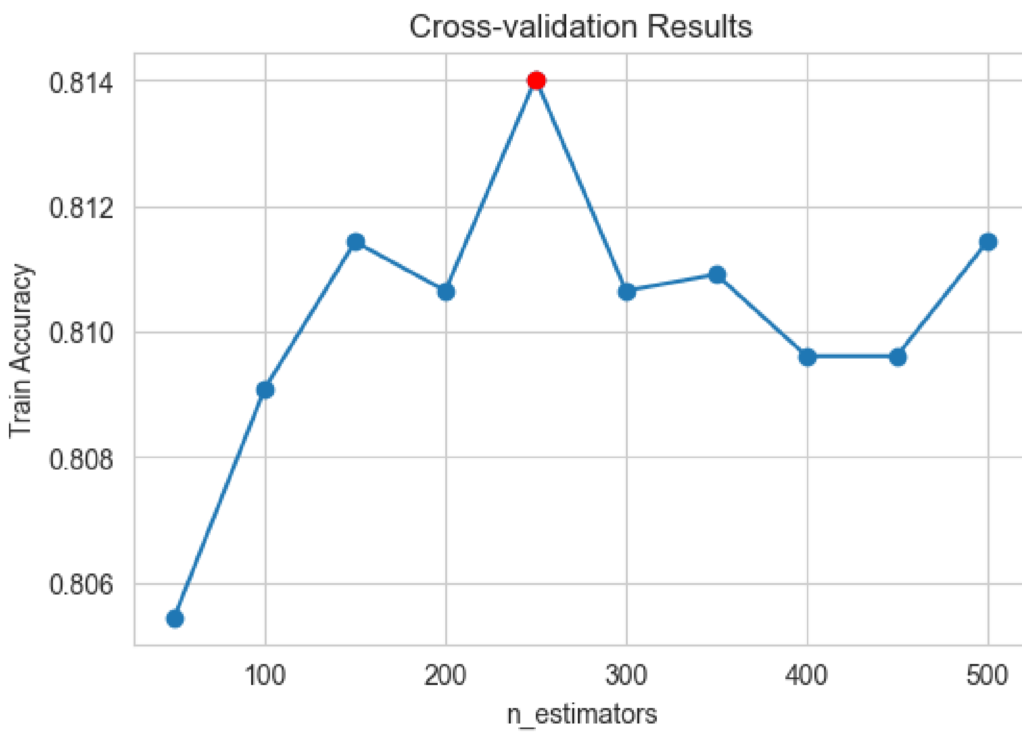
Baseline Model

[ ]:
```
rf = RandomForestClassifier(max_features='sqrt', n_estimators=100,␣
  ↪random_state=333)
rf.fit(X_train,y_train)
y_pred = rf.predict(X_test)

print(confusion_matrix(y_test, y_pred))
print(accuracy_score(y_test, y_pred))
```

```
[[1181   73]
 [ 216  180]]
0.8248484848484848
```

Tune 'n_estimators'

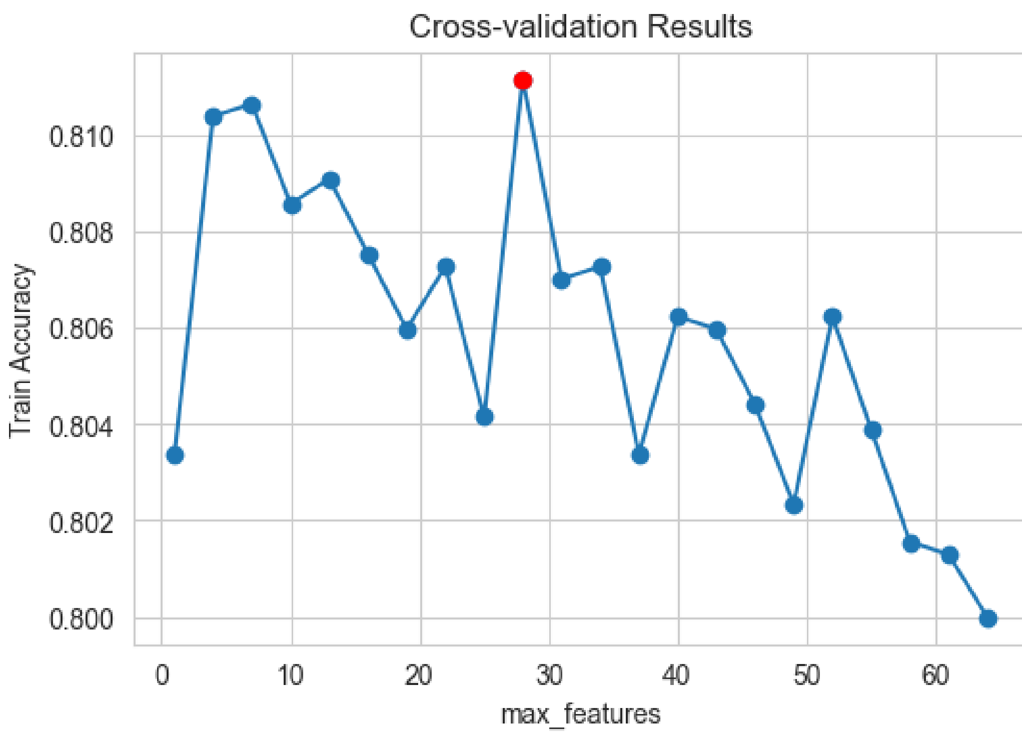
[ ]:
```
cv_params = {'n_estimators': np.arange(50, 501, 50)}
cv = GridSearchCV(rf, cv_params, cv=5, scoring='accuracy', n_jobs=-1) #␣
  ↪parallelize with all cores
cv.fit(X_train, y_train);
cv_results = cv.cv_results_
best_n_estimators = cv.best_params_['n_estimators']
best_score = cv.best_score_
plt.figure(figsize=(6, 4))
plt.plot(cv_results['param_n_estimators'], cv_results['mean_test_score'], 'o-')
plt.plot(best_n_estimators, best_score, 'ro-')
plt.xlabel('n_estimators')
plt.ylabel('Train Accuracy')
plt.title('Cross-validation Results')
plt.show()
print(f"best_n_estimators: {best_n_estimators}")
print(f"best_score: {best_score}")
```




```
best_n_estimators: 250
best_score: 0.814025974025974
```

Tune 'max_features'

[ ]:
```
rf.set_params(n_estimators=250)
cv_params = {'max_features': np.arange(1, X_train.shape[1] + 1, 3)}
cv = GridSearchCV(rf, cv_params, cv=5, scoring='accuracy', n_jobs=-1) #␣
↪parallelize with all cores
cv.fit(X_train, y_train);
cv_results = cv.cv_results_
best_max_features = cv.best_params_['max_features']
best_score = cv.best_score_
plt.figure(figsize=(6, 4))
plt.plot(cv_results['param_max_features'], cv_results['mean_test_score'], 'o-')
plt.plot(best_max_features, best_score, 'ro-')
plt.xlabel('max_features')
plt.ylabel('Train Accuracy')
plt.title('Cross-validation Results')
plt.show()
print(f"best_max_features: {best_max_features}")
print(f"best_score: {best_score}")
```




```
best_max_features: 28
best_score: 0.8111688311688312
```

The best max features is 28 but 4 performs similarly.

Evaluation

[ ]:
```
rf = RandomForestClassifier(n_estimators=250, max_features=4, random_state=333)
rf.fit(X_train,y_train)
y_pred = rf.predict(X_test)

print(confusion_matrix(y_test, y_pred))
print(accuracy_score(y_test, y_pred))
```

```
[[1183   71]
 [ 211  185]]
0.8290909090909091
```

0.3.3 Multi-Parameter Tuning

Tune both 'n_estimators' and 'max_features'

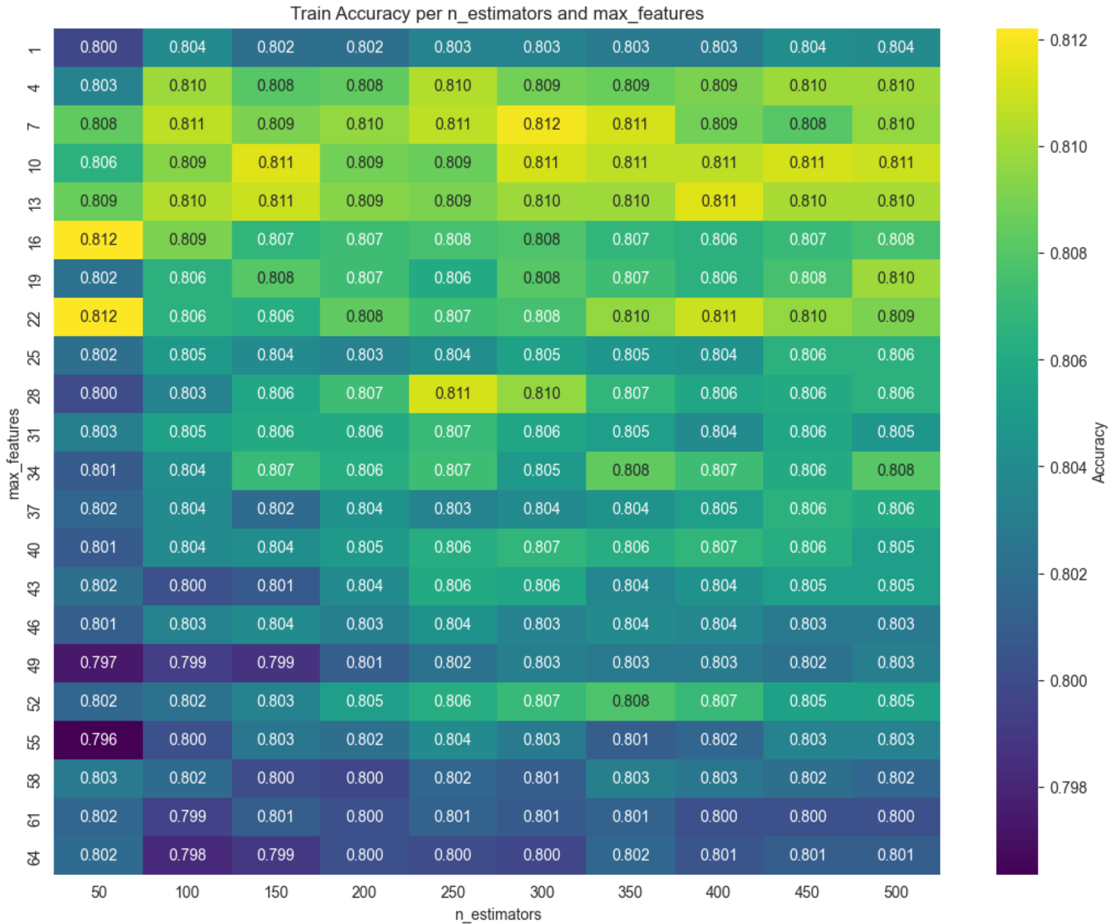
[ ]:
```
cv_params = {'n_estimators': np.arange(50, 501, 50), 'max_features': np.
 ↪arange(1, X_train.shape[1] + 1, 3)}
cv = GridSearchCV(rf, cv_params, cv=5, scoring='accuracy', n_jobs=-1) #␣
 ↪parallelize with all cores
```

```
cv.fit(X_train, y_train);
```

[ ]:
```
cv_results = pd.DataFrame(cv.cv_results_)
cv_table = cv_results.pivot('param_max_features', 'param_n_estimators',␣
↪'mean_test_score')
best_n_estimators = cv.best_params_['n_estimators']
best_max_features = cv.best_params_['max_features']
best_score = cv.best_score_
```

```
/var/folders/jy/pdgtcrw968d_3_tqfzr3w6y00000gn/T/ipykernel_5030/2998427397.py:2:
FutureWarning: In a future version of pandas all arguments of DataFrame.pivot
will be keyword-only.
  cv_table = cv_results.pivot('param_max_features', 'param_n_estimators',
'mean_test_score')
/var/folders/jy/pdgtcrw968d_3_tqfzr3w6y00000gn/T/ipykernel_5030/2998427397.py:2:
FutureWarning: In a future version, the Index constructor will not infer numeric
dtypes when passed object-dtype sequences (matching Series behavior)
  cv_table = cv_results.pivot('param_max_features', 'param_n_estimators',
'mean_test_score')
/var/folders/jy/pdgtcrw968d_3_tqfzr3w6y00000gn/T/ipykernel_5030/2998427397.py:2:
FutureWarning: In a future version, the Index constructor will not infer numeric
dtypes when passed object-dtype sequences (matching Series behavior)
  cv_table = cv_results.pivot('param_max_features', 'param_n_estimators',
'mean_test_score')
```

[ ]:
```
plt.figure(figsize=(13, 10))
sns.heatmap(cv_table, annot=True, fmt=".3f", cmap='viridis', cbar_kws={'label':␣
↪'Accuracy'})
plt.title('Train Accuracy per n_estimators and max_features')
plt.xlabel('n_estimators')
plt.ylabel('max_features')
plt.show()
print(f"best_n_estimators: {best_n_estimators}")
print(f"best_max_features: {best_max_features}")
print(f"best_score: {best_score}")
```




```
best_n_estimators: 50
best_max_features: 16
best_score: 0.8122077922077923
```

Evaluation

```
[ ]: rf = RandomForestClassifier(max_features=best_max_features,␣
         ↪n_estimators=best_n_estimators, random_state=333)
     rf.fit(X_train,y_train)
     y_pred = rf.predict(X_test)

     print(confusion_matrix(y_test, y_pred))
     print(accuracy_score(y_test, y_pred))
```

```
[[1170   84]
 [ 218  178]]
0.8169696969696697
```

The single-parameter tuned model performed better and is less complex compared to the multi-parameter tuned model.

0.3.4 Model Comparisons

[ ]:
```
# basic decision tree
dt = DecisionTreeClassifier(random_state=333)
cv_params = {'max_depth': np.arange(1, 11, 1), 'min_samples_split': np.
 ↪arange(2, 11, 1)}
cv = GridSearchCV(dt, cv_params, cv=5, scoring='accuracy', n_jobs=-1) #␣
 ↪parallelize with all cores
cv.fit(X_train, y_train)
best_max_depth = cv.best_params_['max_depth']
best_min_samples_split = cv.best_params_['min_samples_split']
dt = DecisionTreeClassifier(max_depth=best_max_depth,␣
 ↪min_samples_split=best_min_samples_split, random_state=333)
dt.fit(X_train, y_train)
y_pred = dt.predict(X_test)

print(f"best_max_depth: {best_max_depth}")
print(f"best_min_samples_split: {best_min_samples_split}")
print(confusion_matrix(y_test, y_pred))
print(accuracy_score(y_test, y_pred))
```

```
best_max_depth: 5
best_min_samples_split: 2
[[1170   84]
 [ 222  174]]
0.8145454545454546
```

[ ]:
```
# gradient boost
boost = GradientBoostingClassifier(random_state=333)
cv_params = {'n_estimators': np.arange(50, 301, 50), 'max_features': np.
 ↪arange(1, int(np.round(X_train.shape[1] / 2)), 4), 'learning_rate': np.
 ↪arange(0.001, 0.301, 0.05)}
cv = GridSearchCV(boost, cv_params, cv=5, scoring='accuracy', n_jobs=-1) #␣
 ↪parallelize with all cores
cv.fit(X_train, y_train)
best_n_estimators = cv.best_params_['n_estimators']
best_max_features = cv.best_params_['max_features']
best_learning_rate = cv.best_params_['learning_rate']
boost = GradientBoostingClassifier(n_estimators=best_n_estimators,␣
 ↪max_features=best_max_features, learning_rate=best_learning_rate)
boost.fit(X_train, y_train)
y_pred = boost.predict(X_test)

print(f"best_n_estimators: {best_n_estimators}")
print(f"best_max_features: {best_max_features}")
print(f"best_learning_rate: {best_learning_rate}")
print(confusion_matrix(y_test, y_pred))
```

```
print(accuracy_score(y_test, y_pred))
```

```
best_n_estimators: 250
best_max_features: 5
best_learning_rate: 0.101
[[1174   80]
 [ 205  191]]
0.8272727272727273
```

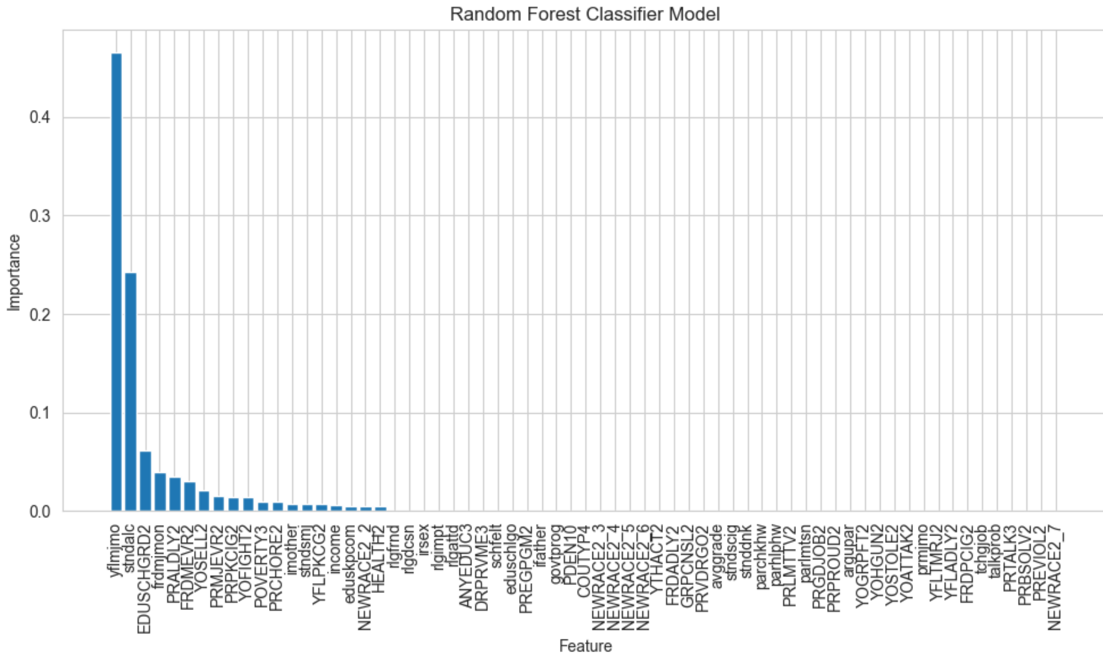
[ ]:
```
# null model
null_model = DummyClassifier()
null_model.fit(X_train, y_train)
y_pred = null_model.predict(X_test)

print(confusion_matrix(y_test, y_pred))
print(accuracy_score(y_test, y_pred))
```

```
[[1254    0]
 [ 396    0]]
0.76
```

0.3.5 Final Evaluation

Our Random Forest Classification model significantly outperformed the null model and performs slightly better than the Gradient Boosting Classifier. We will use it as our final model. Let's check its most important features.

[ ]:
```
feature_importances = pd.DataFrame({'feature': X.columns, 'importance': dt.
 ↪feature_importances_})
feature_importances = feature_importances.sort_values('importance',␣
 ↪ascending=False)
feature_importances = feature_importances.reset_index(drop=True)

plt.figure(figsize=(10, 6))
plt.bar(feature_importances['feature'], feature_importances['importance'])
plt.xticks(rotation=90)
plt.ylabel('Importance')
plt.xlabel('Feature')
plt.title('Random Forest Classifier Model')
plt.tight_layout()
plt.show()

print(feature_importances.head(5))
```

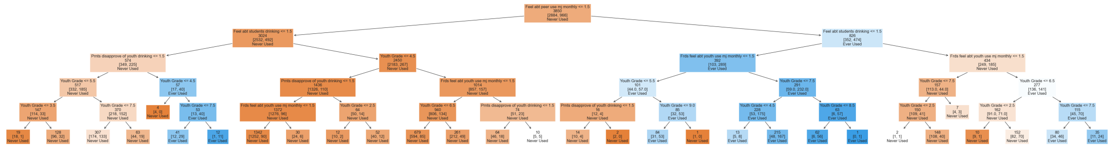


	feature	importance
0	yflmjmo	0.465152
1	stndalc	0.242450
2	EDUSCHGRD2	0.061471
3	frdmjmon	0.039490
4	PRALDLY2	0.034885

0.3.6 Additional Plots

[ ]:
```
# For Discussion
col_mapping = {
    "EDUSCHGRD2": "Youth Grade",
    "stndalc": "Feel abt students drinking",
    "yflmjmo": "Feel abt peer use mj monthly",
    "PRALDLY2": "Prnts disapprove of youth drinking",
    "frdmjmon": "Frds feel abt youth use mj monthly"

}
dt.fit(X_train[col_mapping.keys()],y_train)
plt.figure(figsize=(60, 8))
plot_tree(dt,
          filled=True,
          feature_names=list(col_mapping.values()),
          class_names=['Never Used', 'Ever Used'],
          impurity=False,
          label='none',
```

```
        fontsize=12)
plt.show()
```



0.4 Random Forest Multi Classifier: 'alcydays'

'alcydays' is defined as the number of days a youth have used alcohol in the past year. It is categorized into 6 levels: - 1: 1-11 days - 2: 12-49 days - 3: 50-99 days - 4: 100-299 days - 5: 300-365 days - 6: No past year use

We will again apply classification techniques here.

0.4.1 Data Preparation

[ ]:
```
# Remove all rows where 'alcydays' == 6 (means never used, we can combine these␣
↪findings with alcflag later)
df_alcydays = df.copy()
df_alcydays = df_alcydays[df_alcydays['alcydays'] != 6]
X = df_alcydays.drop(substance_dict.keys(), axis=1)
y = df_alcydays['alcydays']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
↪random_state=333)
```

0.4.2 Single-Parameter Tuning

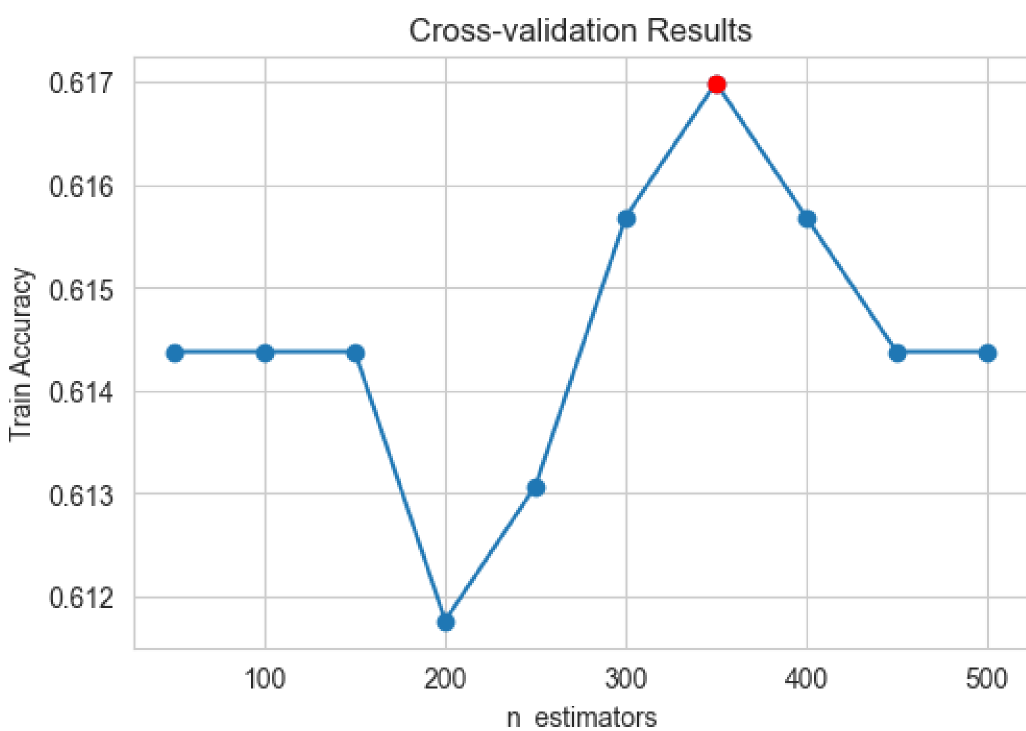
[ ]:
```
rf = RandomForestClassifier(max_features='sqrt', n_estimators=100,␣
↪random_state=333)
rf.fit(X_train,y_train)
y_pred = rf.predict(X_test)

print(confusion_matrix(y_test, y_pred))
print(accuracy_score(y_test, y_pred))
```

```
[[190   9   0   0]
 [ 70  12   0   0]
 [ 21   1   0   0]
 [ 22   4   0   0]]
0.6139817629179332
```

Tune 'n_estimators'

[ ]:
```python
cv_params = {'n_estimators': np.arange(50, 501, 50)}
cv = GridSearchCV(rf, cv_params, cv=5, scoring='accuracy', n_jobs=-1) #␣
 ↪parallelize with all cores
cv.fit(X_train, y_train);
cv_results = cv.cv_results_
best_n_estimators = cv.best_params_['n_estimators']
best_score = cv.best_score_
plt.figure(figsize=(6, 4))
plt.plot(cv_results['param_n_estimators'], cv_results['mean_test_score'], 'o-')
plt.plot(best_n_estimators, best_score, 'ro-')
plt.xlabel('n_estimators')
plt.ylabel('Train Accuracy')
plt.title('Cross-validation Results')
plt.show()
print(f"best_n_estimators: {best_n_estimators}")
print(f"best_score: {best_score}")
```

```
/opt/homebrew/anaconda3/envs/py39/lib/python3.9/site-
packages/sklearn/model_selection/_split.py:737: UserWarning: The least populated
class in y has only 2 members, which is less than n_splits=5.
  warnings.warn(
```

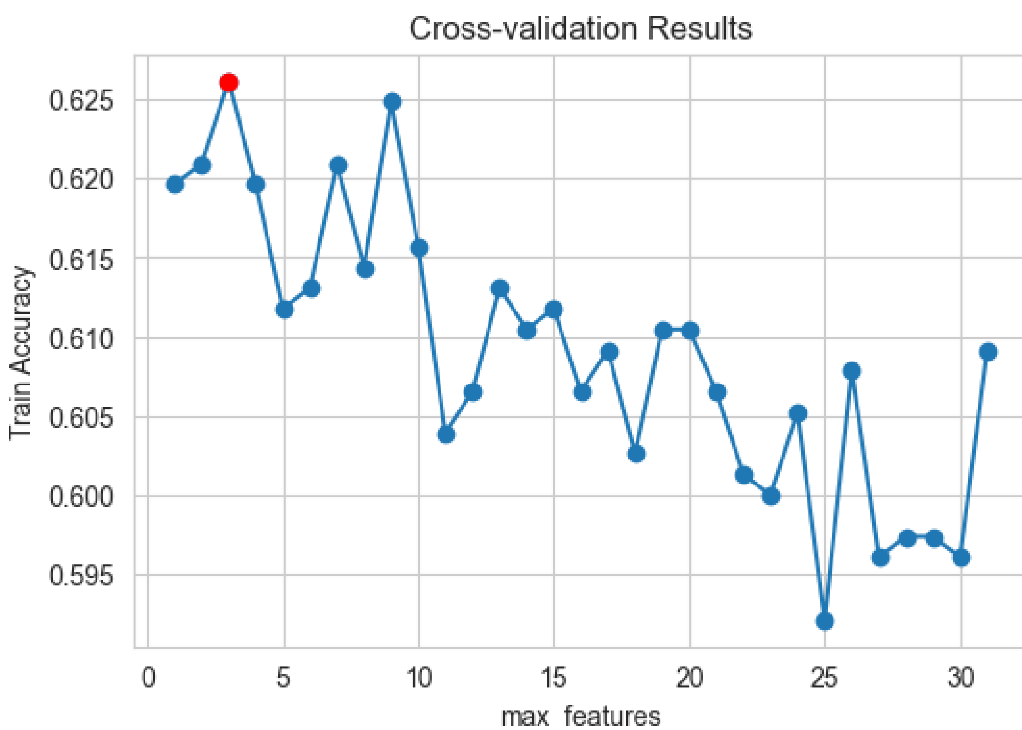



```
best_n_estimators: 350
best_score: 0.6169934640522876
```

Tune 'max__features'

```
[ ]: rf.set_params(n_estimators=150)
cv_params = {'max_features': np.arange(1, int(X_train.shape[1]/2), 1)}
cv = GridSearchCV(rf, cv_params, cv=5, scoring='accuracy', n_jobs=-1) #␣
 ↪parallelize with all cores
cv.fit(X_train, y_train);
cv_results = cv.cv_results_
best_max_features = cv.best_params_['max_features']
best_score = cv.best_score_
plt.figure(figsize=(6, 4))
plt.plot(cv_results['param_max_features'], cv_results['mean_test_score'], 'o-')
plt.plot(best_max_features, best_score, 'ro-')
plt.xlabel('max_features')
plt.ylabel('Train Accuracy')
plt.title('Cross-validation Results')
plt.show()
print(f"best_max_features: {best_max_features}")
print(f"best_score: {best_score}")
```

```
/opt/homebrew/anaconda3/envs/py39/lib/python3.9/site-
packages/sklearn/model_selection/_split.py:737: UserWarning: The least populated
class in y has only 2 members, which is less than n_splits=5.
  warnings.warn(
```



```
best_max_features: 3
best_score: 0.6261437908496732
```

The optimal 'max_features' is 1. There must be a predictor that is so strongly correlated with the response it overshadowed all other predictors.

Evaluation

```
[ ]: rf = RandomForestClassifier(n_estimators=150, max_features=1, random_state=333)
     rf.fit(X_train,y_train)
     y_pred = rf.predict(X_test)

     print(confusion_matrix(y_test, y_pred))
     print(accuracy_score(y_test, y_pred))
```

```
[[199   0   0   0]
 [ 81   1   0   0]
 [ 22   0   0   0]
 [ 26   0   0   0]]
0.60790273556231
```

0.4.3 Multi-Parameter Tuning

Tune both 'n_estimators' and 'max_features'

```
[ ]: cv_params = {'n_estimators': np.arange(50, 301, 50), 'max_features': np.
      ↪arange(1, int(np.round(X_train.shape[1] / 2)), 1)}
     cv = GridSearchCV(rf, cv_params, cv=5, scoring='accuracy', n_jobs=-1) #␣
      ↪parallelize with all cores
     cv.fit(X_train, y_train);
```

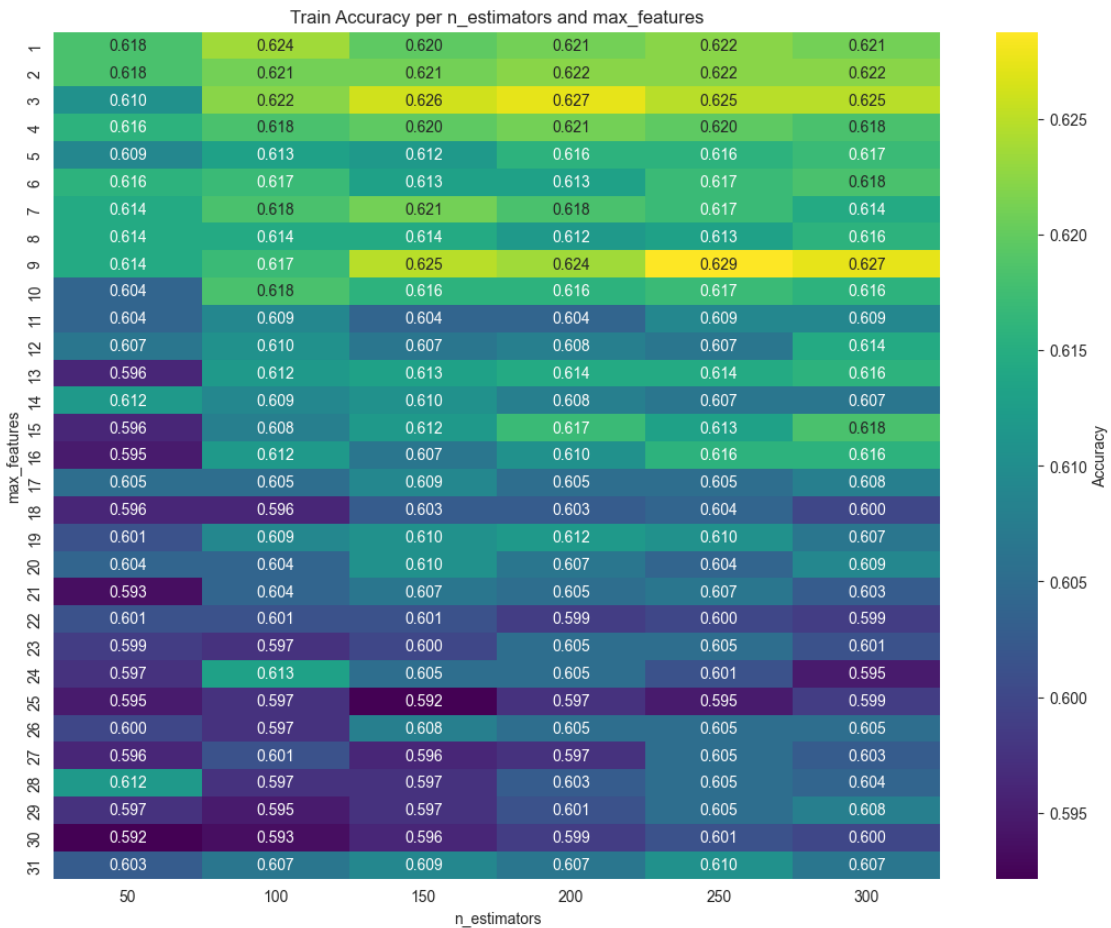
```
/opt/homebrew/anaconda3/envs/py39/lib/python3.9/site-
packages/sklearn/model_selection/_split.py:737: UserWarning: The least populated
class in y has only 2 members, which is less than n_splits=5.
  warnings.warn(
```

```
[ ]: cv_results = pd.DataFrame(cv.cv_results_)
     cv_table = cv_results.pivot('param_max_features', 'param_n_estimators',␣
      ↪'mean_test_score')
     best_n_estimators = cv.best_params_['n_estimators']
     best_max_features = cv.best_params_['max_features']
     best_score = cv.best_score_
```

```
/var/folders/jy/pdgtcrw968d_3_tqfzr3w6y00000gn/T/ipykernel_5030/2998427397.py:2:
FutureWarning: In a future version of pandas all arguments of DataFrame.pivot
will be keyword-only.
  cv_table = cv_results.pivot('param_max_features', 'param_n_estimators',
'mean_test_score')
/var/folders/jy/pdgtcrw968d_3_tqfzr3w6y00000gn/T/ipykernel_5030/2998427397.py:2:
FutureWarning: In a future version, the Index constructor will not infer numeric
dtypes when passed object-dtype sequences (matching Series behavior)
  cv_table = cv_results.pivot('param_max_features', 'param_n_estimators',
```

```
'mean_test_score')
/var/folders/jy/pdgtcrw968d_3_tqfzr3w6y00000gn/T/ipykernel_5030/2998427397.py:2:
FutureWarning: In a future version, the Index constructor will not infer numeric
dtypes when passed object-dtype sequences (matching Series behavior)
  cv_table = cv_results.pivot('param_max_features', 'param_n_estimators',
'mean_test_score')
```

[ ]:
```python
plt.figure(figsize=(13, 10))
sns.heatmap(cv_table, annot=True, fmt=".3f", cmap='viridis', cbar_kws={'label':
 ↪'Accuracy'})
plt.title('Train Accuracy per n_estimators and max_features')
plt.xlabel('n_estimators')
plt.ylabel('max_features')
plt.show()
print(f"best_n_estimators: {best_n_estimators}")
print(f"best_max_features: {best_max_features}")
print(f"best_score: {best_score}")
```




```
best_n_estimators: 250
```

```
best_max_features: 9
best_score: 0.6287581699346405
```

Evaluation

```
[ ]: rf = RandomForestClassifier(max_features=best_max_features,␣
      ↪n_estimators=best_n_estimators, random_state=333)
     rf.fit(X_train,y_train)
     y_pred = rf.predict(X_test)

     print(confusion_matrix(y_test, y_pred))
     print(accuracy_score(y_test, y_pred))
```

```
[[194   5   0   0]
 [ 75   7   0   0]
 [ 21   1   0   0]
 [ 21   5   0   0]]
0.6109422492401215
```

Our single-parameter tuned model performed slightly better, but is much much simpler comprising only of 1 feature with a tree size of 150, versus using 6 features with a tree size of 100.

0.4.4 Model Comparisons

```
[ ]: # gradient boost
     boost = GradientBoostingClassifier(random_state=333)
     cv_params = {'n_estimators': np.arange(1, 2000, 100), 'max_features': np.
      ↪arange(1, int(X_train.shape[1]/2), 3), 'learning_rate': np.arange(0.010, 0.
      ↪101, 0.05)}
     cv = GridSearchCV(boost, cv_params, cv=5, scoring='accuracy', n_jobs=-1) #␣
      ↪parallelize with all cores
     cv.fit(X_train, y_train)
     best_n_estimators = cv.best_params_['n_estimators']
     best_max_features = cv.best_params_['max_features']
     best_learning_rate = cv.best_params_['learning_rate']
     boost = GradientBoostingClassifier(n_estimators=best_n_estimators,␣
      ↪max_features=best_max_features, learning_rate=best_learning_rate)
     boost.fit(X_train, y_train)
     y_pred = boost.predict(X_test)

     print(f"best_n_estimators: {best_n_estimators}")
     print(f"best_max_features: {best_max_features}")
     print(f"best_learning_rate: {best_learning_rate}")
     print(confusion_matrix(y_test, y_pred))
     print(accuracy_score(y_test, y_pred))
```

```
/opt/homebrew/anaconda3/envs/py39/lib/python3.9/site-
packages/sklearn/model_selection/_split.py:737: UserWarning: The least populated
class in y has only 2 members, which is less than n_splits=5.
  warnings.warn(
```

```
best_n_estimators: 401
best_max_features: 1
best_learning_rate: 0.01
[[199   0   0   0]
 [ 82   0   0   0]
 [ 22   0   0   0]
 [ 26   0   0   0]]
0.6048632218844985
```

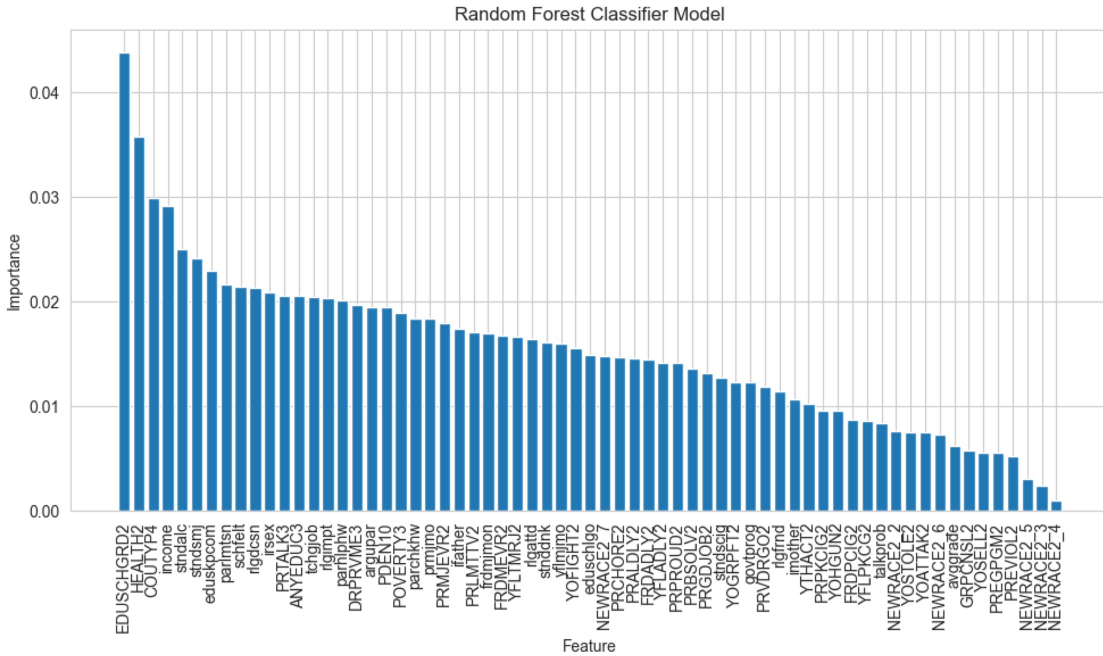
[ ]:
```
# null model
null_model = DummyClassifier()
null_model.fit(X_train, y_train)
y_pred = null_model.predict(X_test)

print(confusion_matrix(y_test, y_pred))
print(accuracy_score(y_test, y_pred))
```

```
[[199   0   0   0]
 [ 82   0   0   0]
 [ 22   0   0   0]
 [ 26   0   0   0]]
0.6048632218844985
```

0.4.5 Final Evaluation

Our Random Forest Classification model slightly outperformed the null model. Now let's check its most important features.

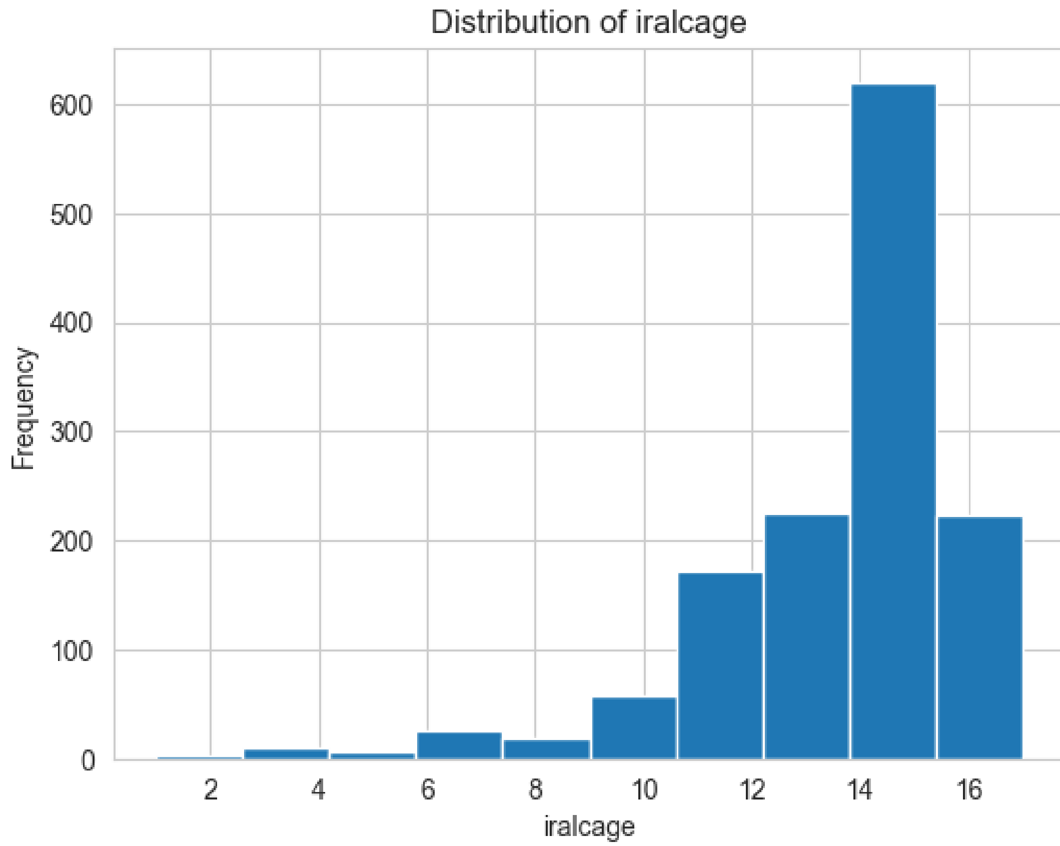
[ ]:
```
rf = RandomForestClassifier(n_estimators=150, max_features=1, random_state=333)
rf.fit(X_train,y_train)
feature_importances = pd.DataFrame({'feature': X.columns, 'importance': rf.
 ↪feature_importances_})
feature_importances = feature_importances.sort_values('importance',␣
 ↪ascending=False)
feature_importances = feature_importances.reset_index(drop=True)

plt.figure(figsize=(10, 6))
plt.bar(feature_importances['feature'], feature_importances['importance'])
plt.xticks(rotation=90)
plt.ylabel('Importance')
plt.xlabel('Feature')
plt.title('Random Forest Classifier Model')
plt.tight_layout()
plt.show()

print(feature_importances.head(5))
```



	feature	importance
0	EDUSCHGRD2	0.043808
1	HEALTH2	0.035768
2	COUTYP4	0.029985
3	income	0.029229
4	stndalc	0.025044

0.5 Additional Plots

[ ]:
```
# Use in intro
plt.hist(df_iralcage['iralcage'])
plt.xlabel('iralcage')
plt.ylabel('Frequency')
plt.title('Distribution of iralcage')
plt.show()
```


Distribution of iralcage
Frequency
iralcage
0
100
200
300
400
500
600
2
4
6
8
10
12
14
16