Examining and predicting substance use risk through its contributing factors

Team Members: Jose Cordova, Vishal Harihar Ganapathy Krishnan, Shikhar Shukla, Shravani Basanthpur, Pravalika Reddy Arakoti, Pallavi Singh, and Anantha Sai Varma Pericherla.
Indiana University–Purdue University Indianapolis, Indianapolis, Indiana, USA
jfcordov@iu.edu, vganapat@iu.edu, shikshuk@iu.edu, sbasanth@iu.edu, parakoti@iu.edu, singpall@iu.edu, aperich@iu.edu

Introduction

- The UN and WHO recommend treating substance use as a public health issue and to use data to drive policy decision (Volkow et al., 2017).
- In public health, factors that increase the probability of an event occurring are known as risk factors while protective factors reduce the probability of said event occurring (NIDA, 2020).
- Multiple studies have utilized the NSDUH (Center for Behavioral Health Statistics and Quality, 2021) in order to determine the risk factors for drug abuse. Identifying which factors contribute to drug is crucial in order to implement evidence-based prevention programs (Volkow et al., 2017).
- Rosner et al. 2021 paper attempts to identify these factors using the survey's 2021 data to study the risk factors of drug abuse among sexual minorities.
- Waddell uses the 2002-2019 data from this survey to examine the risk of suffering from Alcohol Use Disorder.
- While, Lin et al. (2016) used the 2013 version of this survey to find the factors that contribute to medical and recreational cannabis use.

Aim

Our aim is to determine which factors contribute to substance use and to predict the frequency of said substance use.

Purpose

We seek to determine which factors influence the risk of substance use as well as the frequency of substance use in order to identify risk and protective factors.

In doing so we hope to contribute to the creation of evidence-based prevention programs.

Research hypothesis

Null hypothesis: The examined factors* do not have a relation to substance use in the past year.

Alternative hypothesis: The examined factors* have a relation to substance use in the past year.

**ethnicity, gender, income, government assistance, age, education, depression, population density*

Methodology

1. **DATA COLLECTION:**

   Data Source- National Survey on Drug Use and Health (NSDUH)
   https://www.datafiles.samhsa.gov/dataset/national-survey-drug-use-and-health-2020-nsduh-2020-ds0001

   We have Merged 2015-2020 data and created a master dataset.

2. **DATA CLEANING, EXTRACTION:**

- Finding missing values
- Normality test
- Finding Correlation and Visualizing the data

3. **DEVELOPING A MODEL**

4. **PERFORMANCE ANALYSIS OF THE MODEL**

5. **RESULTS**

Dataset

- “The NSDUH series, formerly the National Household Survey on Drug Abuse, is the leading source of statistical information on the use of illicit drugs, alcohol, and tobacco and mental health issues in the United States” (NSDAH, 2021)

- Using pd.read_table we directly pulled the data from the NSDAH website and selected only the variables included in our analysis.

- We have combined six NSDUH datasets from 2015 to 2020

Description of the dataset variables:

National Survey on Drug Use and Health

Categorical		Quantitative	
Nominal	Ordinal	Discrete	Continuous
Drug use (never/ever used)*, ethnicity, gender, health insurance, receiving government assistance, past year depression	Age group, employment, education, annual household income	Past year frequency of drug use*, level of disability from depression, perception of unmet need for mental health	Population density

*This includes use of the following drugs:alcohol, mariuana, cocaine, crack, heroin, hallucinogens, inhalants and meth.

Descriptive Statistics and Visualization

```
#breakdown of gender
df['IRSEX'].value_counts(normalize=True)*100
```

```
2    52.548145
1    47.451855
Name: IRSEX, dtype: float64
```

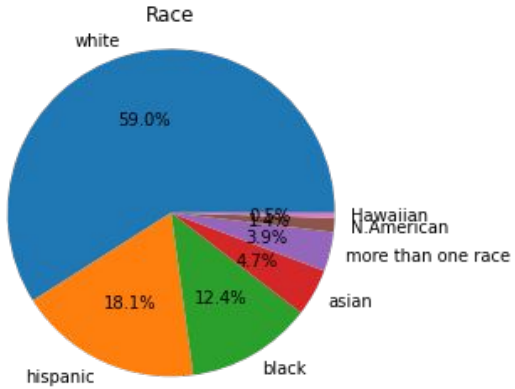
```
#breakdown of age
df['CATAGE'].value_counts(normalize=True)*100
```

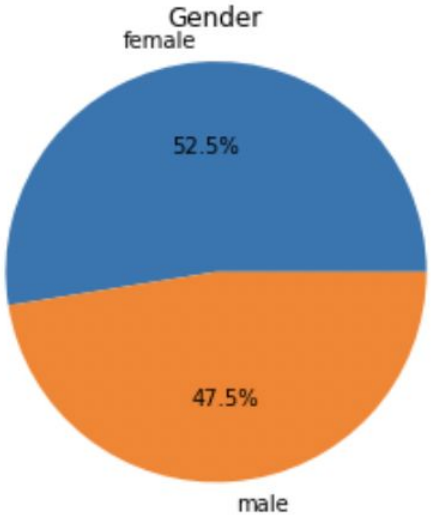
```
35 or older         36.141937
18-25 years old     24.666335
12-17 years old     23.438436
26-34 years old     15.753292
Name: CATAGE, dtype: float64
```

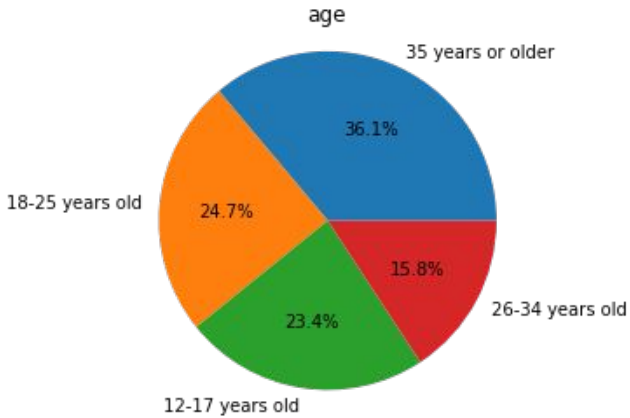
```
#breakdown of race
df['NEWRACE2'].value_counts(normalize=True)*100
```

```
White                                          64.943909
Hispanic                                       14.884626
Black/African American                          9.196486
Asian                                           5.645578
More than one race                              4.040373
Native American/Alaskan Native                  0.884687
Native Hawaiian/Other Pacific Islander          0.404341
Name: NEWRACE2, dtype: float64
```

```python
#descriptive visualization
import matplotlib.pyplot as plt
import numpy as np

my_data = df['IRSEX'].value_counts(normalize=True)*100
my_labels = 'female', 'male'
plt.pie(my_data, labels=my_labels, autopct='%1.1f%%')
plt.title('Gender')
plt.axis('equal')
plt.show()
```







Exploratory Data Analysis (EDA)

```
[29] # Save the file
     # import pandas as pd
     # dv.to_csv('C:\\Users\\gvhec\\Desktop\\Informatics Proj\\Datasets\\6 years combined\\NSDUH_6yrsCombo.csv')

[31] df.shape

     (315661, 40)
```

df.shape shows that we have 315661 rows and 48 columns.

We have converted all the independent variables like Ethnicity, Age, Gender,Employment, Education, Population density, Health Insurance, Annual household income, Government Assistance, Perception of unmet need for mental health, Past year major depression and Level of functioning disability from depression into categorical variables.

We created 2 dependent frequency variables tha is ALCCAT and ALLDGSCAT

ALCCAT is alcohol consumption (Yes -> 1 or No -> 0). '0' when IRALCFY = 0 & '1' when IRALCFY is non-zero

ALLDGSCAT is alcohol consumption (Yes -> 1 or No -> 0). '0' when ALLDGS = 0 & '1' when ALLDGS is non-zero

```
[16] # Convert to "category" type
     df[['ASDSHOM2','ASDSWRK2','ASDSREL2','ASDSSOC2','ASDSOVL2','IRSEX','IREDUHIGHST2','CATAGE','NEWRACE2','IRWRKSTAT','IRPINC3','INCOME','PDEN10','COUTYP4']] =
```

ALCCAT is alcohol consumption (Yes -> 1 or No -> 0). '0' when IRALCFY = 0 & '1' when IRALCFY is non-zero

ALLDGSCAT is alcohol consumption (Yes -> 1 or No -> 0). '0' when ALLDGS = 0 & '1' when ALLDGS is non-zero

```
# Creating ALCCAT and ALLDGSCAT
def mickey(var):
  if var != 0:
    return 1
  else:
    return 0

df['ALCCAT'] = df['IRALCFY']
df['ALLDGSCAT'] = df['ALLDGS']
df['ALCCAT'] = df['ALCCAT'].apply(mickey)
df['ALLDGSCAT'] = df['ALLDGSCAT'].apply(mickey)
```

Exploratory Data Analysis

By using the describe() function from the Python package Pandas we have obtained multiple summary statistics like the average, standard deviation, median, etc.

```
#exploratory statistics of frequency of substance use in a year
frequency=df[['IRMETHAMYFQ', 'IRALCFY',
       'IRMJFY', 'IRCOCFY', 'IRCRKFY', 'IRHERFY', 'IRHALLUCYFQ', 'IRINHALYFQ', 'ALLDGS']].describe(include='all')

frequency.columns = ['Meth', 'Alcohol', 'Marihuana', 'Cocaine', 'Crack', 'Heroin', 'Hallucinogens', 'Inhallants', 'DrugsExceptAlc']

frequency
```

	Meth	Alcohol	Marihuana	Cocaine	Crack	Heroin	Hallucinogens	Inhallants	DrugsExceptAlc
count	315661.000000	315661.000000	315661.000000	315661.000000	315661.000000	315661.000000	315661.000000	315661.000000	315661.000000
mean	0.792093	49.781288	24.286833	0.705865	0.189859	0.453249	0.464378	0.319425	27.211702
std	13.686260	81.377971	77.036602	10.316375	6.127078	10.810214	7.422459	6.569459	86.493220
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	0.000000	6.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
75%	0.000000	60.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
max	365.000000	365.000000	365.000000	365.000000	365.000000	365.000000	365.000000	365.000000	2549.000000

Due to most of our sample not using substances that are not alcohol and marijuana we created a new variable (ALLDGS) which included all these substances.

```
#create all drugs frequency
df["ALLDGS"] = df['IRMETHAMYFQ']+df['IRCOCFY']+df['IRCRKFY']+df['IRHERFY']+ df['IRHALLUCYFQ']+df['IRINHALYFQ']+df['IRMJFY']
```

EDA using crosstab

Percentage of respondents that consumed alcohol (1) or have never used alcohol (0).

```
pd.crosstab(df.CATAGE, df.ALCCAT ,normalize='index')\
  .round(4)*100
```

ALCCAT	0	1
CATAGE		
12-17 years old	77.87	22.13
18-25 years old	26.43	73.57
26-34 years old	20.92	79.08
35 or older	30.64	69.36

ALCCAT	0	1
IRSEX		
Female	39.01	60.99
Male	39.28	60.72

ALCCAT	0	1
INCOME		
$20,000-$49,999	41.70	58.30
$50,000-$74,999	36.52	63.48
$75,000 or more	34.69	65.31
Less than $20,000	45.83	54.17

ALCCAT	0	1
NEWRACE2		
Asian	49.79	50.21
Black/African American	47.29	52.71
Hispanic	46.54	53.46
More than one race	42.95	57.05
Native American/Alaskan Native	46.96	53.04
Native Hawaiian/Other Pacific Islander	50.61	49.39
White	33.77	66.23

ALCCAT	0	1
IREDUHIGHST2		
Associate's degree (AA, AS)	21.86	78.14
College graduate or higher	17.68	82.32
Eighth grade completed	78.33	21.67
Eleventh or Twelfth grade completed, no diploma	47.00	53.00
Fifth grade or less grade completed	85.53	14.47
High school diploma/GED	34.08	65.92
Ninth grade completed	65.51	34.49
Seventh grade completed	89.32	10.68
Sixth grade completed	90.87	9.13
Some college credit, but no degree	23.32	76.68
Tenth grade completed	56.77	43.23

Percentage of respondents that consumed substances (1) or have never used substances (0).

ALLDGSCAT	0	1
CATAGE		
12-17 years old	84.52	15.48
18-25 years old	64.95	35.05
26-34 years old	74.71	25.29
35 or older	87.20	12.80

ALLDGSCAT	0	1
IRSEX		
Female	81.14	18.86
Male	76.88	23.12

ALLDGSCAT	0	1
NEWRACE2		
Asian	89.06	10.94
Black/African American	77.49	22.51
Hispanic	81.41	18.59
More than one race	70.89	29.11
Native American/Alaskan Native	70.15	29.85
Native Hawaiian/Other Pacific Islander	80.73	19.27

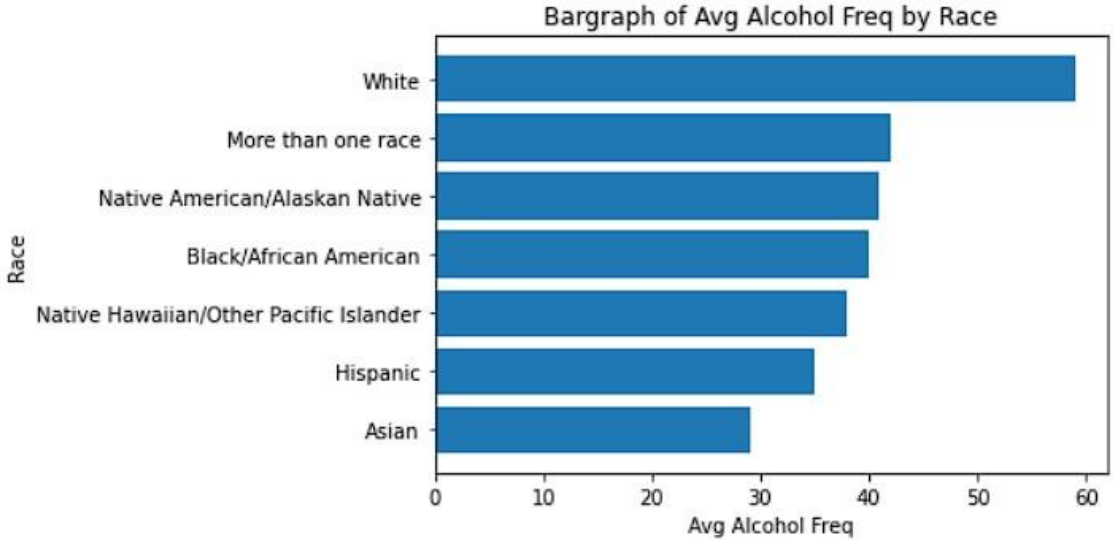
ALLDGSCAT	0	1
INCOME		
$20,000-$49,999	78.02	21.98
$50,000-$74,999	80.46	19.54
$75,000 or more	82.44	17.56
Less than $20,000	73.35	26.65

ALLDGSCAT	0	1
IREDUHIGHST2		
Associate's degree (AA, AS)	80.12	19.88
College graduate or higher	81.88	18.12
Eighth grade completed	86.81	13.19
Eleventh or Twelfth grade completed, no diploma	72.07	27.93
Fifth grade or less grade completed	94.72	5.28
High school diploma/GED	76.22	23.78
Ninth grade completed	80.61	19.39
Seventh grade completed	92.76	7.24
Sixth grade completed	95.53	4.47
Some college credit, but no degree	71.99	28.01
Tenth grade completed	74.94	25.06

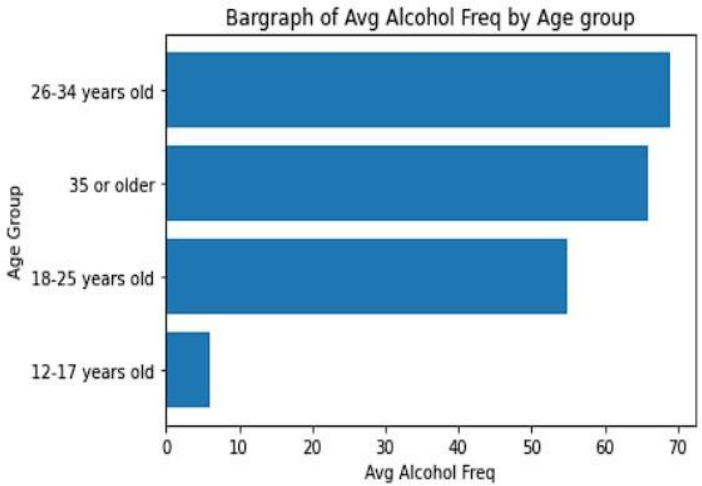
Data Visualization

Visualization code for bar graph comparing Alcohol frequency with different Races

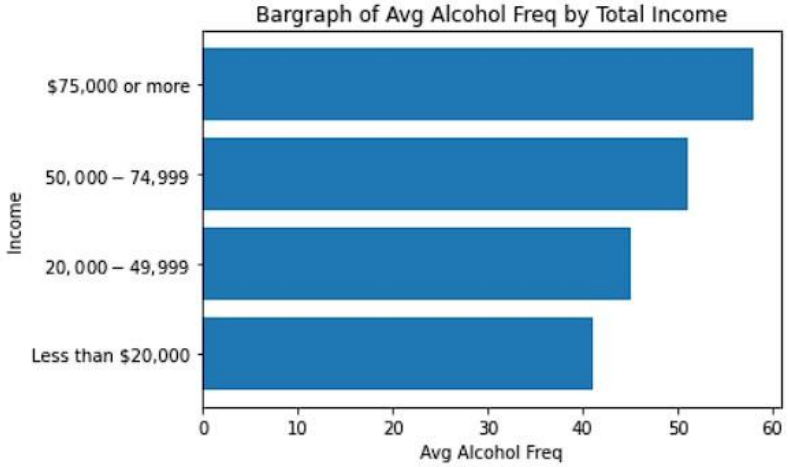
```
#importing the modules
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

#getting the unique values from NEWRACE2 and assigning it to x
x = list(df['NEWRACE2'].unique())
y = []

# Taking average of y-values and rounding-off to nearest number
# Taking sum of y is not preffered, so going for mean (avg)
for i in range(len(x)):
    y.append(int(df['IRALCFY'][df['NEWRACE2'] == x[i]].mean().round()))

df_bar = {"Race": list(x), "Alcfreq": y}
dFrame = pd.DataFrame.from_dict(df_bar)
df_sorted_bar = dFrame.sort_values('Alcfreq',ascending=True)

print(df_sorted_bar.sort_values('Alcfreq',ascending=False))

## Bar Graph (Taking Horizontal bar since regular bar shows an overlap of the Race names)
plt.barh('Race', 'Alcfreq', data=df_sorted_bar)
plt.ylabel('Race')
plt.xlabel('Avg Alcohol Freq')
plt.title('Bargraph of Avg Alcohol Freq by Race')
```

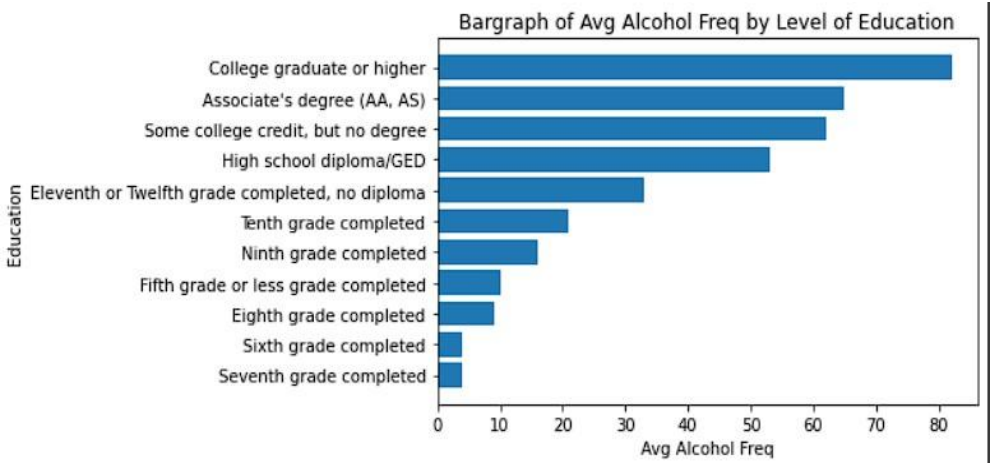


Bar graph showing that White race has the highest alcohol frequency while the Asians have the least.



Bar graph showing Age group of 26 - 34 years have the highest alcohol frequency while the age group 12 - 17 years have the least.



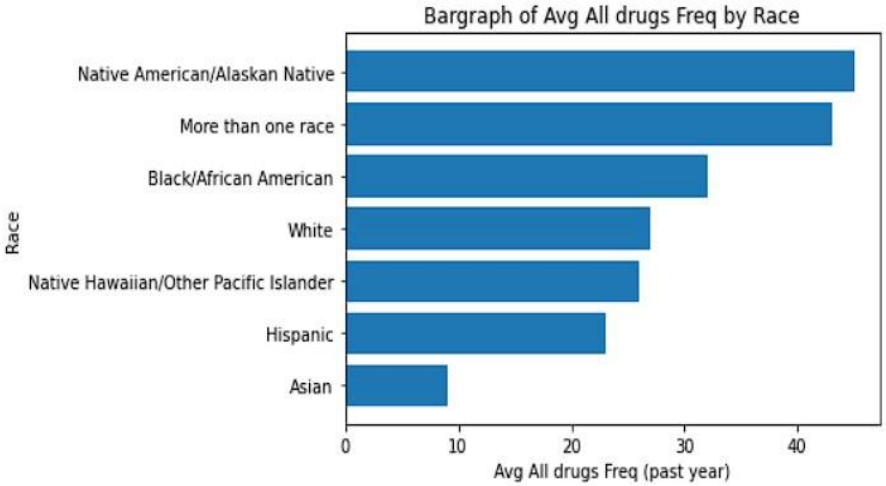


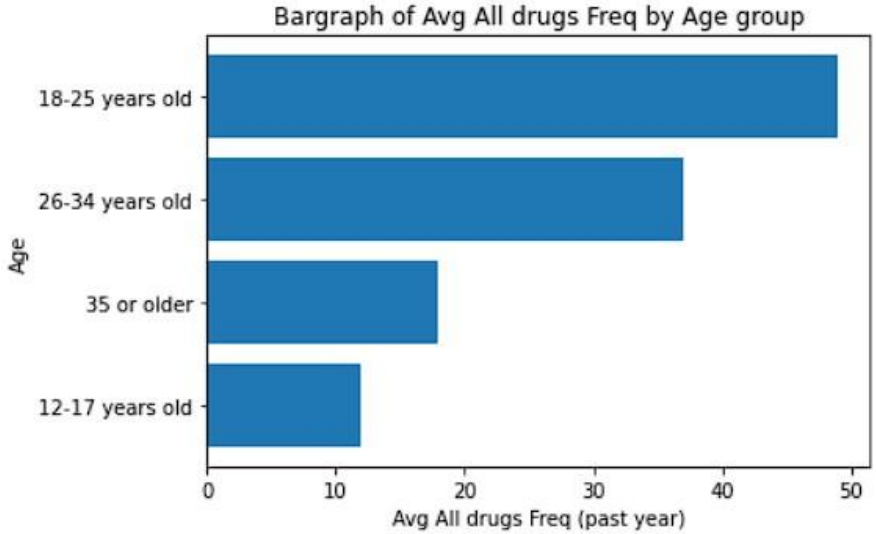
Bar graph showing that People with income 75,000$ or more has the highest alcohol frequency while the people with income less than 20,000$ have the least.

Bar graph showing that College graduates or higher degree holders has the highest alcohol frequency while the Sixth and seventh graders have the least.

Visualization code for bar chart comparing All drugs frequency (except Alcohol) with different Races

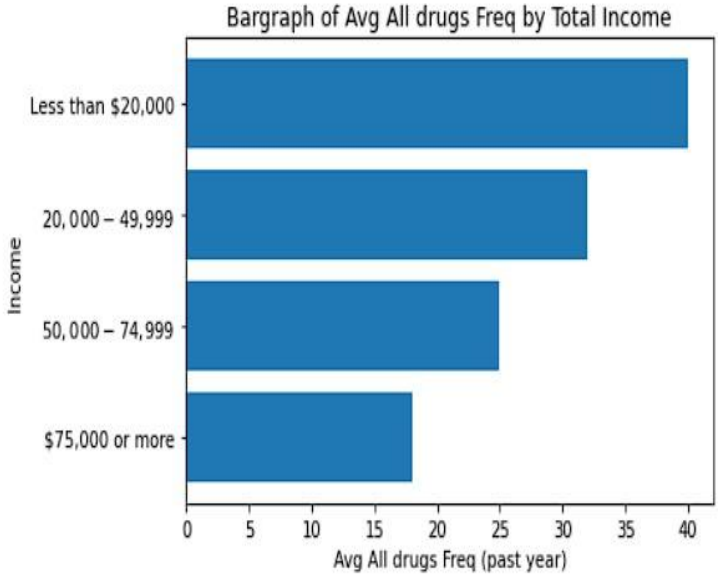
```
#importing the modules
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

#getting the unique values from NEWRACE2 and assigning it to x
x = list(dv['NEWRACE2'].unique())
y = []

# Taking average of y-values and rounding-off to nearest number
# Taking sum of y is not preffered, so going for mean (avg)
for i in range(len(x)):
    y.append(int(dv['ALLDGS'][df['NEWRACE2'] == x[i]].mean().round()))

df_bar = {"Race": list(x), "Alldrugfreq": y}
dFrame = pd.DataFrame.from_dict(df_bar)
df_sorted_bar = dFrame.sort_values('Alldrugfreq',ascending=True)

print(df_sorted_bar.sort_values('Alldrugfreq',ascending=False))

## Bar Graph (Taking Horizontal bar since regular bar shows an overlap of the Race names)
plt.barh('Race', 'Alldrugfreq', data=df_sorted_bar)
plt.ylabel('Race')
plt.xlabel('Avg All drugs Freq (past year)')
plt.title('Bargraph of Avg All drugs Freq by Race')
```

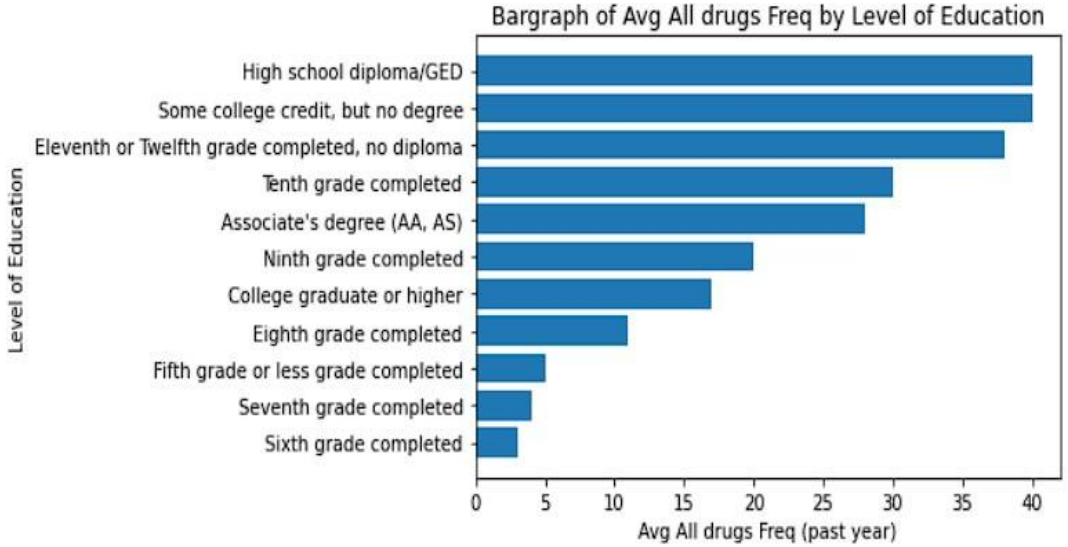




Bar chart shows that Native American/Alaskan Natives have the highest frequency of all drugs (except alcohol) use, while Asians have the least.

Bar chart shows that age group of 18 - 25 years have the highest frequency of all drugs (except alcohol) use, while age group of 12-17 have the least.





Bar chart shows that population with Income group less than 20,000$ have the highest frequency of all drugs (except alcohol) use, while population with Income of 75,000$ or more have the least.

Barchart shows that people with Some college credit but no degree have the highest frequency of all drugs (except alcohol) use, while people with just 6th grade completed have the least but not zero.

Using scipy.stats.normaltest we will test if substance use frequency and alcohol use frequency is normally distributed

```
from scipy import stats
x=df["ALLDGS"]
k2, p = stats.normaltest(x)
alpha = 1e-3
print("p = {:g}".format(p))
p = 8.4713e-19
if p < alpha:  # null hypothesis: x comes from a normal distribution
    print("The null hypothesis can be rejected")
else:
    print("The null hypothesis cannot be rejected")
```

```
p = 0
The null hypothesis can be rejected
```

Substance use frequency is not normally distributed

```
from scipy import stats
x=df["IRALCFY"]
k2, p = stats.normaltest(x)
alpha = 1e-3
print("p = {:g}".format(p))
p = 8.4713e-19
if p < alpha:  # null hypothesis: x comes from a normal distribution
    print("The null hypothesis can be rejected")
else:
    print("The null hypothesis cannot be rejected")
```

```
p = 0
The null hypothesis can be rejected
```

Alcohol use frequency is not normally distributed

Because neither variable is normally distributed we will use a non-parametric test (Kruskal-Wallis) to see if there is a difference between medians of the groups

Hypothesis Testing- Kruskal Wallis Test

Python provides us **kruskal() function from the scipy. stats library** using which we can conduct the Kruskal-Wallis test in Python easily.

For Alcohol and Race

```
x1 = dv['IRALCFY'][dv['NEWRACE2']=='White'].sum()
x2 = dv['IRALCFY'][dv['NEWRACE2']=='Hispanic'].sum()
x3 = dv['IRALCFY'][dv['NEWRACE2']=='Black/African American'].sum()
x4 = dv['IRALCFY'][dv['NEWRACE2']=='Native Hawaiian/Other Pacific Islander'].sum()
x5 = dv['IRALCFY'][dv['NEWRACE2']=='Asian'].sum()
x6 = dv['IRALCFY'][dv['NEWRACE2']=='Native American/Alaskan Native'].sum()
x7 = dv['IRALCFY'][dv['NEWRACE2']=='More than one race'].sum()
x = [x1,x2,x3,x4,x5,x6,x7]
```

```
x
```

```
[10975422, 2008442, 1554597, 58817, 432103, 178837, 505793]
```

```
c1 = dv['NEWRACE2'][dv['NEWRACE2']== 'White'].count()
c2 = dv['NEWRACE2'][dv['NEWRACE2']== 'Hispanic'].count()
c3 = dv['NEWRACE2'][dv['NEWRACE2']== 'Black/African American'].count()
c4 = dv['NEWRACE2'][dv['NEWRACE2']== 'Native Hawaiian/Other Pacific Islander'].count()
c5 = dv['NEWRACE2'][dv['NEWRACE2']== 'Asian'].count()
c6 = dv['NEWRACE2'][dv['NEWRACE2']== 'Native American/Alaskan Native'].count()
c7 = dv['NEWRACE2'][dv['NEWRACE2']== 'More than one race'].count()
c = [c1,c2,c3,c4,c5,c6,c7]
```

```
c
```

```
[186258, 57291, 39123, 1567, 14833, 4406, 12183]
```

```
stats. kruskal(x, c)
```

```
KruskalResult(statistic=8.265306122448976, pvalue=0.004040984683985589)
```

Since, the p-value is 0.004, the null hypothesis that the samples in the race have the same central tendency and therefore come from the same population, is rejected.

For Other Drugs and Race

```
x1 = dv['ALLDGS'][dv['NEWRACE2']=='White'].sum()
x2 = dv['ALLDGS'][dv['NEWRACE2']=='Hispanic'].sum()
x3 = dv['ALLDGS'][dv['NEWRACE2']=='Black/African American'].sum()
x4 = dv['ALLDGS'][dv['NEWRACE2']=='Native Hawaiian/Other Pacific Islander'].sum()
x5 = dv['ALLDGS'][dv['NEWRACE2']=='Asian'].sum()
x6 = dv['ALLDGS'][dv['NEWRACE2']=='Native American/Alaskan Native'].sum()
x7 = dv['ALLDGS'][dv['NEWRACE2']=='More than one race'].sum()
x = [x1,x2,x3,x4,x5,x6,x7]
```

```
[48] x
```

```
[5091892, 1328382, 1271362, 40004, 134709, 200012, 523312]
```

```
c1 = dv['NEWRACE2'][dv['NEWRACE2']== 'White'].count()
c2 = dv['NEWRACE2'][dv['NEWRACE2']== 'Hispanic'].count()
c3 = dv['NEWRACE2'][dv['NEWRACE2']== 'Black/African American'].count()
c4 = dv['NEWRACE2'][dv['NEWRACE2']== 'Native Hawaiian/Other Pacific Islander'].count()
c5 = dv[ int64: c2 dv['NEWRACE2']== 'Asian'].count()
c6 = dv[          dv['NEWRACE2']== 'Native American/Alaskan Native'].count()
c7 = dv[ 57291    dv['NEWRACE2']== 'More than one race'].count()
c = [c1,c2,c3,c4,c5,c6,c7]
```

```
[50] c
```

```
[186258, 57291, 39123, 1567, 14833, 4406, 12183]
```

```
[51] stats. kruskal(x, c)
```

```
KruskalResult(statistic=7.546938775510206, pvalue=0.006011210888904063)
```

Since, the p-value is 0.006, the null hypothesis that the samples in the race have the same central tendency and therefore come from the same population, is rejected.

For Alcohol and Age

Since, the p-value is 0.02, the null hypothesis that the samples in the age category have the same central tendency and therefore come from the same population, is rejected.

For Alcohol and Gender

Since, the p-value is 0.12, the null hypothesis that the samples in the gender category have the same central tendency and therefore come from the same population, is rejected.

For Alcohol and Education

Since, the p-value is 0.0005, the null hypothesis that the samples in the education category have the same central tendency and therefore come from the same population, is rejected.

For Other Drugs and Age

Since, the p-value is 0.02, the null hypothesis that the samples in the age category have the same central tendency and therefore come from the same population, is rejected.

For Other Drugs and Gender

Since, the p-value is 0.12, the null hypothesis that the samples in the gender category have the same central tendency and therefore come from the same population, is rejected.

For Other Drugs and Education

Since, the p-value is 0.0007, the null hypothesis that the samples in the education category have the same central tendency and therefore come from the same population, is rejected.

CHI-SQUARE Test

Performing Chi-Square test to check the correlation between the independent categorical variables and alcohol (ALCCAT) and other drug (ALLDGSCAT) columns.

1. i) To test if there is a statistically significant difference in Population Race and Alcohol consumption, we apply the Chi-Square Test

2. To test if there is a statistically significant difference in Population Race and A

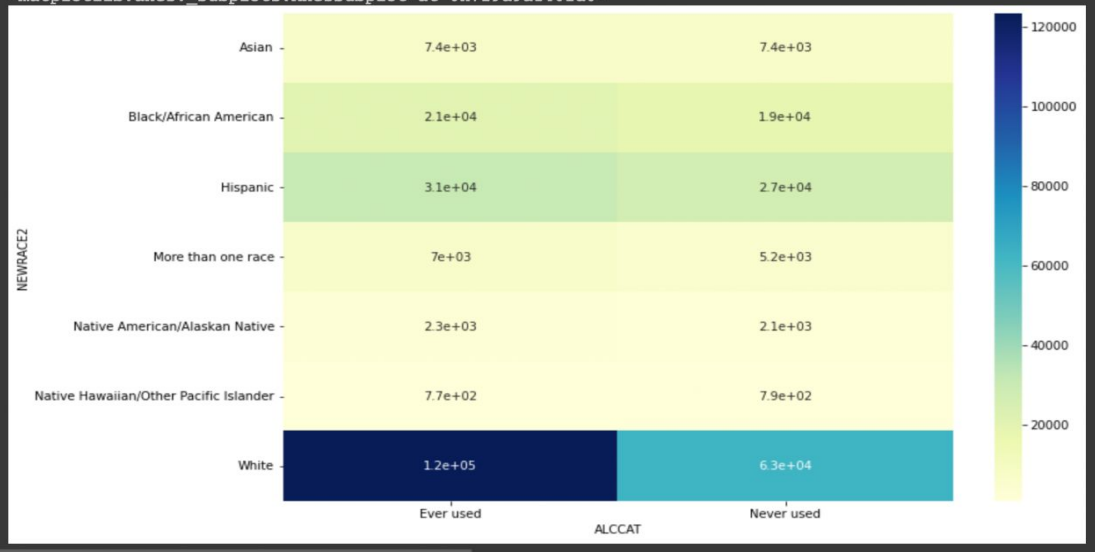
```
import pandas as pd
import numpy as np
from scipy.stats import chi2_contingency

import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

```
contigency= pd.crosstab(dv['NEWRACE2'], dv['ALCCAT'])
contigency
```

NEWRACE2	ALCCAT Ever used	Never used
Asian	7448	7385
Black/African American	20622	18501
Hispanic	30628	26663
More than one race	6950	5233
Native American/Alaskan Native	2337	2069
Native Hawaiian/Other Pacific Islander	774	793
White	123353	62905

```
plt.figure(figsize=(12,8))
sns.heatmap(contigency, annot=True, cmap="YlGnBu")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9a9a140fd0>
```



```
[ ]  # Chi-square test of independence.
     c, p, dof, expected = chi2_contingency(contigency)
     p

     0.0
```

The p-value is 0% which means that we reject the null hypothesis. The null hypothesis was that Alcohol consumption and Race are independent.

The p-value is 0% which means that we reject the null hypothesis. The null hypothesis was that Alcohol consumption and Race are independent.

1. ii) To test if there is a statistically significant difference in <u>Population Race and other drug consumption</u>, we apply the Chi-Square Test

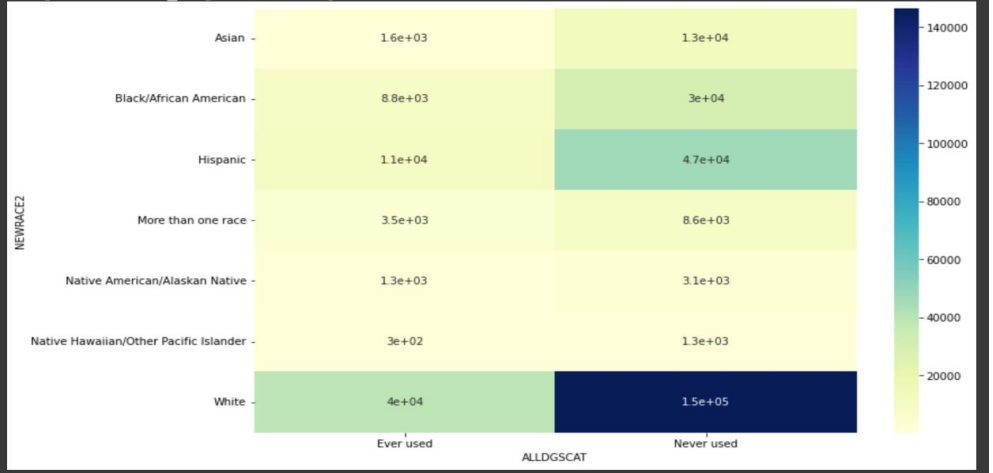
2. To test if there is a statistically significant difference in Population Race and other drug consumption consumption, we apply the Chi-Square Test

```
contigency= pd.crosstab(dv['NEWRACE2'], dv['ALLDGSCAT'])
contigency
```

NEWRACE2	ALLDGSCAT Ever used	Never used
Asian	1623	13210
Black/African American	8806	30317
Hispanic	10649	46642
More than one race	3546	8637
Native American/Alaskan Native	1315	3091
Native Hawaiian/Other Pacific Islander	302	1265
White	39679	146579

```
plt.figure(figsize=(12,8))
sns.heatmap(contigency, annot=True, cmap="YlGnBu")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fef75952bb0>
```



```
# Chi-square test of independence.
c, p, dof, expected = chi2_contingency(contigency)
p
```

```
0.0
```

The p-value is 0% which means that we reject the null hypothesis. The null hypothesis was that other drug consumption and Race are independent.

The p-value is 0% which means that we reject the null hypothesis. The null hypothesis was that other drugs consumption and Race are independent.

2. i) To test if there is a statistically significant difference in Gender and Alcohol consumption, we apply the Chi-Square Test

2. i. To test if there is a statistically significant difference in Genders (M, F) population and Alcohol consumption, we apply the Chi-Square Test

To run the Chi-Square Test, the easiest way is to convert the data into a contingency table with frequencies. We will use the crosstab command from pandas.

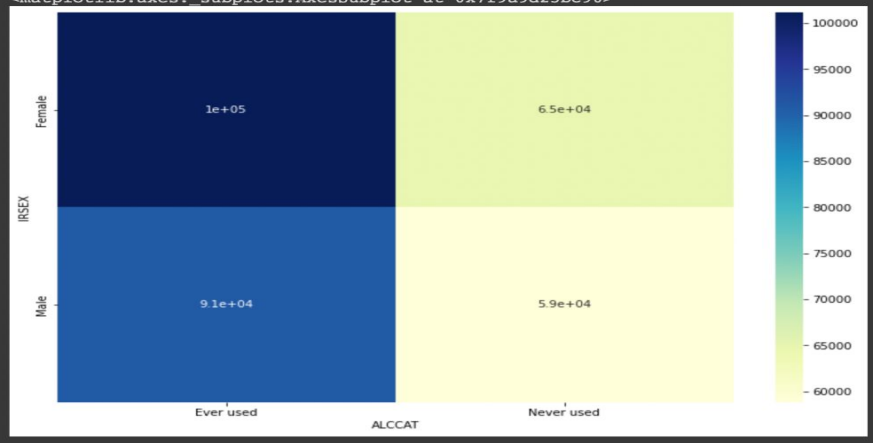
```
contigency= pd.crosstab(dv['IRSEX'], dv['ALCCAT'])
contigency
```

IRSEX	Ever used	Never used
Female	101162	64712
Male	90950	58837

To visualize the contingency tables, we are plotting the heatmaps

```
plt.figure(figsize=(12,8))
sns.heatmap(contigency, annot=True, cmap="YlGnBu")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9a9d25bc90>
```



Now that we have built the contingency table we can pass it to chi2_contingency function from the scipy package which returns the:

chi2: The test statistic p: The p-value of the test dof: Degrees of freedom expected: The expected frequencies, based on the marginal sums of the table

```
[ ] # Chi-square test of independence.
    c, p, dof, expected = chi2_contingency(contigency)
    p
```

```
0.12474173453822644
```

The p-value is 12.47% which means that we do not reject the null hypothesis. The null hypothesis was that Alcohol consumption and Gender are independent.

The p-value is 12.47% which means that we do not reject the null hypothesis. The null hypothesis was that Alcohol consumption and Gender are independent.

2. ii) To test if there is a statistically significant difference in Gender and other drug consumption, we apply the Chi-Square Test

2. ii. To test if there is a statistically significant difference in Genders (M, F) population and other drug consumption, we apply the Chi-Square Test

To run the Chi-Square Test, the easiest way is to convert the data into a contingency table with frequencies. We will use the crosstab command from pandas.

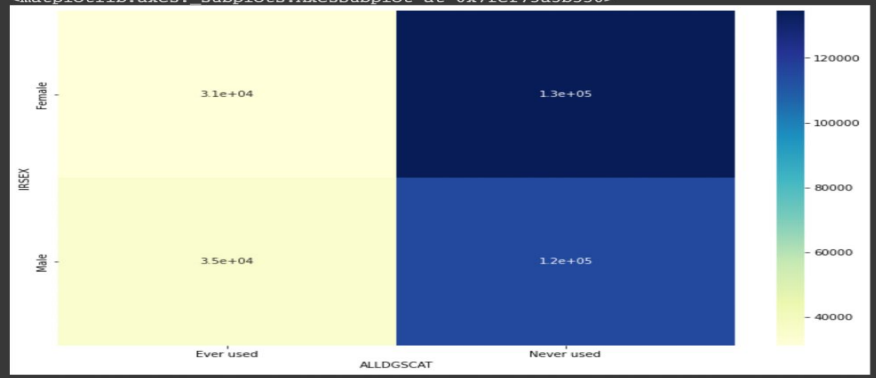
```
[39] contigency= pd.crosstab(dv['IRSEX'], dv['ALLDGSCAT'])
     contigency
```

ALLDGSCAT	Ever used	Never used
IRSEX		
Female	31282	134592
Male	34638	115149

To visualize the contingency tables, we are plotting the heatmaps

```
plt.figure(figsize=(12,8))
sns.heatmap(contigency, annot=True, cmap="YlGnBu")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fef75a3b550>
```



Now that we have built the contingency table we can pass it to chi2_contingency function from the scipy package which returns the:

chi2: The test statistic p: The p-value of the test dof: Degrees of freedom expected: The expected frequencies, based on the marginal sums of the table

```
# Chi-square test of independence.
c, p, dof, expected = chi2_contingency(contigency)
p
```

```
1.7045922058752551e-190
```

The p-value is very small and insignificant which means that we reject the null hypothesis. The null hypothesis was that other drug consumption and Gender are independent.

The p-value is very small and insignificant which means that we reject the null hypothesis. The null hypothesis was that other drug consumption and Gender are independent.

3. i) To test if there is a statistically significant difference in Education and Alcohol consumption, we apply the Chi-Square Test

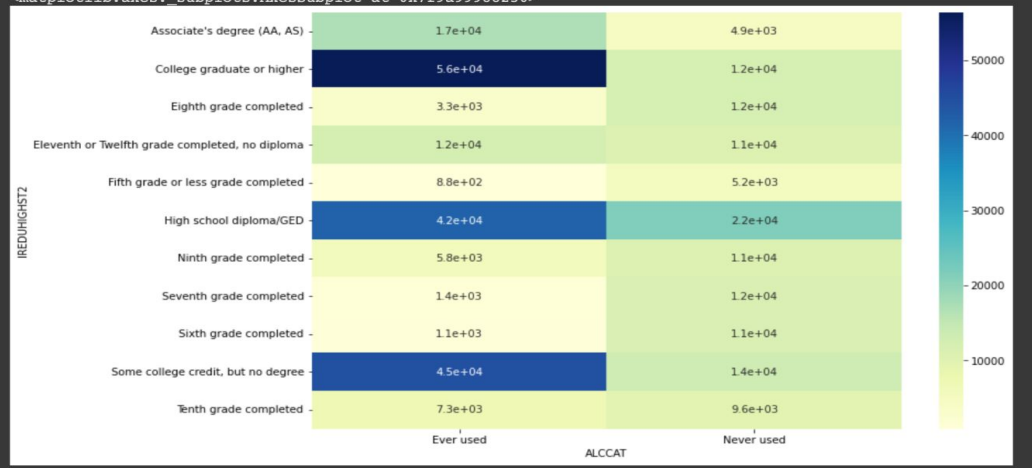
3. i.To test if there is a statistically significant difference in Education and Aclohol consumption, we apply the Chi-Square Test

```
contigency= pd.crosstab(dv['IREDUHIGHST2'], dv['ALCCAT'])
contigency
```

IREDUHIGHST2	ALCCAT Ever used	Never used
Associate's degree (AA, AS)	17347	4853
College graduate or higher	56388	12111
Eighth grade completed	3270	11820
Eleventh or Twelfth grade completed, no diploma	12187	10807
Fifth grade or less grade completed	876	5179
High school diploma/GED	41783	21606
Ninth grade completed	5759	10937
Seventh grade completed	1398	11688
Sixth grade completed	1145	11391
Some college credit, but no degree	44667	13583
Tenth grade completed	7292	9574

```
plt.figure(figsize=(12,8))
sns.heatmap(contigency, annot=True, cmap="YlGnBu")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9a99988250>
```



```
# Chi-square test of independence.
c, p, dof, expected = chi2_contingency(contigency)
p
```

```
0.0
```

The p-value is 0% which means that we reject the null hypothesis. The null hypothesis was that Alcohol consumption and Education are independent.

The p-value is 0% which means that we reject the null hypothesis. The null hypothesis was that Alcohol consumption and Education are independent.

3. i) To test if there is a statistically significant difference in Education and other drug consumption, we apply the Chi-Square Test

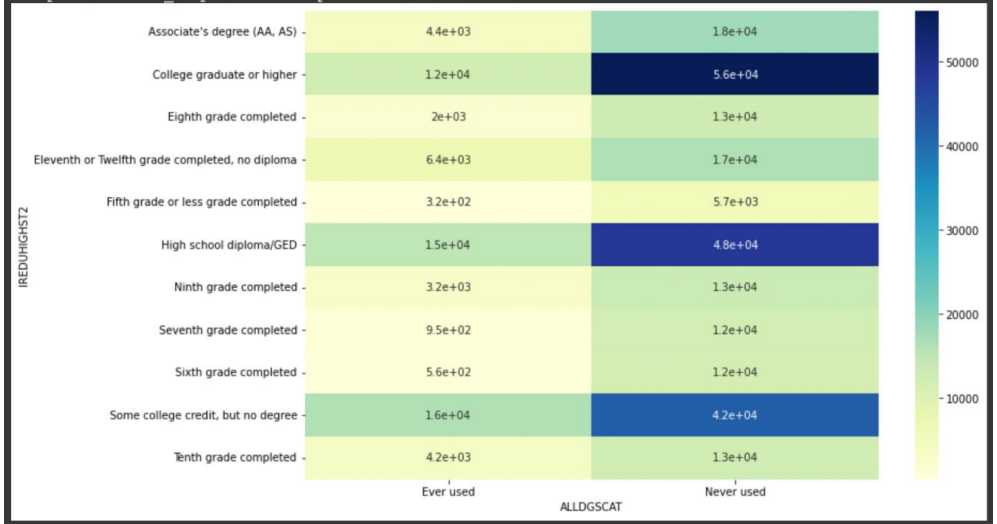
3. ii.To test if there is a statistically significant difference in Education and other drug consumption, we apply the Chi-Square Test

```
contigency= pd.crosstab(dv['IREDUHIGHST2'], dv['ALLDGSCAT'])
contigency
```

IREDUHIGHST2	ALLDGSCAT Ever used	Never used
Associate's degree (AA, AS)	4413	17787
College graduate or higher	12410	56089
Eighth grade completed	1990	13100
Eleventh or Twelfth grade completed, no diploma	6422	16572
Fifth grade or less grade completed	320	5735
High school diploma/GED	15077	48312
Ninth grade completed	3238	13458
Seventh grade completed	948	12138
Sixth grade completed	560	11976
Some college credit, but no degree	16316	41934
Tenth grade completed	4226	12640

```
plt.figure(figsize=(12,8))
sns.heatmap(contigency, annot=True, cmap="YlGnBu")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fef75951ca0>
```



```
[44] # Chi-square test of independence.
     c, p, dof, expected = chi2_contingency(contigency)
     p
```

```
0.0
```

The p-value is 0% which means that we reject the null hypothesis. The null hypothesis was that other drug consumption and Education are independent.

The p-value is 0% which means that we reject the null hypothesis. The null hypothesis was that other drug consumption and Education are independent.

DATA DICTIONARY

NEWRACE2 - Ethnicity

- NonHisp White
- NonHisp Black/Afr Am
- NonHisp Native Am/AK Native
- NonHisp Native HI/Other Pac Isl
- NonHisp Asian
- NonHisp more than one race
- Hispanic

CATAGE- Age

1 = 12-17 Years Old
2 = 18-25 Years Old
3 = 26-34 Years Old
4 = 35 or Older

IRSEX- Gender

- 12-17 Years Old
- 18-25 Years Old
- 26-34 Years Old
- 35 or Older

IRWRKSTAT- Employment

- Employed full time
- Employed part time
- Unemployed
- Other (incl. not in labor force)

= 12-14 year olds

IREDUHIGHST2- Education

- Fifth grade or less grade completed
- Sixth grade completed
- Seventh grade completed
- Eighth grade completed
- Ninth grade completed
- Tenth grade completed
- Eleventh or Twelfth grade completed, no diploma
- High school diploma/GED
- Some college credit, but no degree

= Associate's degree (for example, AA, AS)
= College graduate or higher

PDEN10- Population Density

Segment in a CBSA with 1 million or more persons
Segm. in a CBSA with fewer than 1 million persons
Segment not in a CBSA

COUTYP4- County Metro/non-metro status

Large Metro
Small Metro
Nonmetro

IRPRVHLT- Health Insurance

Yes, R does have private health insurance
No, R does not have private health insurance

IRPINC3- Annual Household Income

Less than $10,000 (Including Loss)
$10,000 - $19,999
$20,000 - $29,999
$30,000 - $39,999
$40,000 - $49,999
$50,000 - $74,999
$75,000 or more

INCOME- Total Family Income

Less than $20,000
$20,000 - $49,999
$50,000 - $74,999
$75,000 or More

IRFAMSOC- Government Assistance

Yes
No

AMHTXND2- Perception for unmet need for mental health

= Unknown/Aged 12-17 (Otherwise)
= Yes (AUUNMTYR=1)
= No (AUUNMTYR=2)

AMDEYR- Past year major depression

Aged 12-17/Unknown (Otherwise)
= Yes (AMDELT=1 & ADPB2WK=1)
= No (AMDELT=2 or [AMDELT=1 & ADPB2WK=2])

ASDSHOM2- Depress feeling role impairment- Home Management

Aged 12-17/Unkn/Legit Skip (Otherwise)
= None (ADPSHMGT=0)
= Mild (ADPSHMGT=1,2,3)
= Moderate (ADPSHMGT=4,5,6)
= Severe (ADPSHMGT=7,8,9)
= Very Severe (ADPSHMGT=10)

ASDSWRK2- Depress feeling role impairment- Ability to work

Aged 12-17/Unkn/Legit Skip (Otherwise)
= None (ADPSWORK=0)
= Mild (ADPSWORK=1,2,3)
= Moderate (ADPSWORK=4,5,6)
= Severe (ADPSWORK=7,8,9)
= Very Severe (ADPSWORK=10)

ASDSREL2- Depress feeling role impairment- Close Relationships

- Aged 12-17/Unkn/Legit Skip (Otherwise)
= None (ADPSRELS=0)
= Mild (ADPSRELS=1,2,3)
= Moderate (ADPSRELS=4,5,6)
= Severe (ADPSRELS=7,8,9)
= Very Severe (ADPSRELS=10)

ASDSSOC2- Depress feeling role impairment- Social Life

Aged 12-17/Unkn/Legit Skip (Otherwise)
None (ADPSSOC=0)
Mild (ADPSSOC=1,2,3)
Moderate (ADPSSOC=4,5,6)
Severe (ADPSSOC=7,8,9)
Very Severe (ADPSSOC=10)

ASDSOVL2- Maximum severity level of depressed feelings- Role Impairment

Aged 12-17/Unknown/Legitimate Skip (Otherwise)
None (See comment above)
Mild (See comment above)
Moderate (See comment above)
Severe (See comment above)
Very Severe (See comment above)

IRALCFY- Alcohol Frequency per year per person- **we will be trying to predict this**.

ALLDGS- This is a combination of frequencies of marijuana, crack, cocaine, meth, heroine, hallucinogens, inhallants per year per person- **We will be trying to predict this**.

ALCCAT- Alcohol Consumers (1) / Non-Consumers (0) - **We will be trying to predict this**.

ALLDGSCAT- All drugs consumers (1) / Non-Consumers (0)- **We will be trying to predict this**.

Data Imputation

```
# Imputing highest mode in AMHTXND2 & convert to int datatype
df['AMHTXND2'] = df['AMHTXND2'].replace([1.,2.],[1,0])
df['AMHTXND2'] = df['AMHTXND2'].fillna(0)
df['AMHTXND2'] = df['AMHTXND2'].astype(int)

# Imputing highest mode in AMDEYR & convert to int datatype
df['AMDEYR'] = df['AMDEYR'].replace([1.,2.],[1,0])
df['AMDEYR'] = df['AMDEYR'].fillna(0)
df['AMDEYR'] = df['AMDEYR'].astype(int)
```

We have imputed the null values for AMHTXND2 (Perception unmet need for mental health) with the highest mode values. Here, ‘0’ (No) has the highest mode value.

We have imputed the null values for AMDEYR (Past year major depression) with the highest mode values. Here, ‘0’ (No) has the highest mode value.

There were many categories in the independent variables that were assigned numerical values like ‘0’ or ‘1’ or ‘1’,’2’, ‘3’, ‘4’, ‘5’, we have converted them into their respected categories with the original names.

ML Models

Classification models:

- Logistic regression
- Random Forest classifier
- CatBoost classifier

Regression models:

- Random Forest regressor
- Linear regression
- CatBoost regressor

ONE-HOT ENCODING (DUMMY ENCODING)

```
#  One-hot encoding (Dummy encoding)

import matplotlib.pyplot as plt

# Dummy encoding on the categorical columns:
# 'ASDSHOM2','ASDSWRK2','ASDSREL2','ASDSSOC2','ASDSOVL2','IRSEX','IREDUHIGHST2','CATAGE','NEWRACE2','IRWRKSTAT','IRPINC3','INCOME','PDEN10' & 'COUTYP4'
dummy1=pd.get_dummies(df['ASDSHOM2'])
dummy2=pd.get_dummies(df['ASDSWRK2'])
dummy3=pd.get_dummies(df['ASDSREL2'])
dummy4=pd.get_dummies(df['ASDSSOC2'])
dummy5=pd.get_dummies(df['ASDSOVL2'])
dummy6=pd.get_dummies(df['IRSEX'])
dummy7=pd.get_dummies(df['IREDUHIGHST2'])
dummy8=pd.get_dummies(df['CATAGE'])
dummy9=pd.get_dummies(df['NEWRACE2'])
dummy10=pd.get_dummies(df['IRWRKSTAT'])
dummy11=pd.get_dummies(df['IRPINC3'])
dummy12=pd.get_dummies(df['INCOME'])
dummy13=pd.get_dummies(df['PDEN10'])
dummy14=pd.get_dummies(df['COUTYP4'])

# Combining df, dummy1 to dummy14. We are adding cols, so axis=1
master = pd.concat([df, dummy1, dummy2, dummy3, dummy4, dummy5, dummy6, dummy7, dummy8, dummy9, dummy10, dummy11, dummy12, dummy13, dummy14],axis=1)
```

```
# Removing QUESTID2, IRALCFY:IRMETHAMYFQ, ALCFLAG:METHAMFLAG, ASDSHOM2:ASDSOVL2 (dummy encoded), IRSEX (dummy enc'd)
# NEWRACE2 (dummy enc'd),'IREDUHIGHST2','CATAGE','NEWRACE2','IRWRKSTAT','IRPINC3','INCOME','PDEN10', 'COUTYP4' (all dummy enc'd), ALCCAT, IRALCFY, ALLDGSFLAG, ALLDGS from Master data
# Also removing dummy encoded variables and variables that were decided to remove after correlation matrix
x = master.drop(master.columns[[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,19,20,21,22,23,24,25,26,27,28,31,32,33,34,35,36,37,38,39]],axis=1)
```

- After prepping up the dataset (df), before feeding data into the ML models, we perform One-hot encoding on the categorical independent variables
- Dummy encoding converts categories within each column into separate columns with 0s & 1s, signifying the presence or absence of that category for a particular row
- After creating dummies, we concatenate all the dummies with our main dataset
- Finally, while declaring the ‘x’ (independent variables), we drop the columns on which we performed the dummy encoding as well as other unwanted columns
- Here we declare, Depression variables (ASDSHOM2/ASDSWRK2/ASDSREL2/ASDSSOC2/ASDSOVL2), IRSEX, IREDUHIGHST2, CATAGE, NEWRACE2, IRWRKSTAT, IRPINC3 (Annual Income category), INCOME (Family Income category), PDEN10, COUTYP4, IRPRVHLT, IRFAMSOC, AMHTXND2, AMDEYR as our Independent variables

Declaring the Dependent (Y) variable for Classification

```
# create target variable
y = df['ALCCAT']
```

Here, we declare ALCCAT (Drank/Never drank Alcohol) as a Dependent variable

```
# create target variable
y = df['ALLDGSCAT']
```

Here, we declare ALLDGSCAT (Consumed/Never consumed Alldrugs) as a Dependent variable

SMOTE

- SMOTE stands for **Synthetic Minority Oversampling TEchnique**
- This technique was described by Nitesh Chawla, et al. in their 2002 paper named for the technique titled "SMOTE: Synthetic Minority Over-sampling Technique."
- SMOTE first selects a minority class instance 'a' at random and finds its k nearest minority class neighbors
- The synthetic instance is then created by choosing one of the k nearest neighbors 'b' at random and connecting 'a' and b to form a line segment in the feature space. The synthetic instances are generated as a convex combination of the two chosen instances 'a' and 'b'
- This means synthetically generated independent variables are created for the minority class
- Firstly, the imblearn package is to be installed through '`!pip install imblearn`' to use SMOTE
- We have seen improved Sensitivity & Specificity of our Classification models after performing SMOTE

```
print(y.value_counts())
print('\n')
print(y.value_counts(dropna=False,normalize=True)*100)
```

```
1    192112
0    123549
Name: ALCCAT, dtype: int64


1    60.860227
0    39.139773
Name: ALCCAT, dtype: float64
```

ALCCAT before SMOTE. Here, '0' is the minority class. As we can see, the proportion of 1's and 0's is 61-39

```
print(y_sm.value_counts())
print('\n')
print(y_sm.value_counts(dropna=False,normalize=True)*100)
```

```
0    192112
1    192112
Name: ALCCAT, dtype: int64


0    50.0
1    50.0
Name: ALCCAT, dtype: float64
```

ALCCAT after SMOTE. As we can see, the proportion of 0's and 1's has become 50-50 after SMOTE (equal)

```
# Takes around 6 minutes to run this code chunk
from imblearn.over_sampling import SMOTE
smote =SMOTE(random_state=42)
x_sm,y_sm=smote.fit_resample(x,y)
```

Code for SMOTE

We perform SMOTE on both ALCCAT & ALLDGSCAT

Train-Test Split

- We feed the SMOTE-transformed independent (x_sm) & dependent (y_sm) variables into the `sklearn.model_selection`.train_test_split
- We split the x_sm & y_sm in the ratio of 80% for training data & 20% for test data

```
# split data into train n test
from sklearn.model_selection import train_test_split

# Outputs 4 variables. First two variables will be 80% (train) & 20% (test) of x and Last two variables are 80% (train) & 20% (test) of y
x_train,x_test,y_train,y_test = train_test_split(x_sm,y_sm,test_size=0.2)

print(x_train.shape,y_train.shape)
print(x_test.shape,y_test.shape)
```

```
(307379, 80) (307379,)
(76845, 80) (76845,)
```

CLASSIFICATION BY LOGISTIC REGRESSION FOR ALCOHOL (ALCCAT)

Model building

```
## fit the model
from sklearn.linear_model import LogisticRegression
logca=LogisticRegression()
logca.fit(x_train,y_train)
```

```
LogisticRegression()
```

TRAINING & TEST MODEL SCORES

```
logca.score(x_train,y_train)
```

```
0.7526961828882259
```

```
logca.score(x_test,y_test)
```

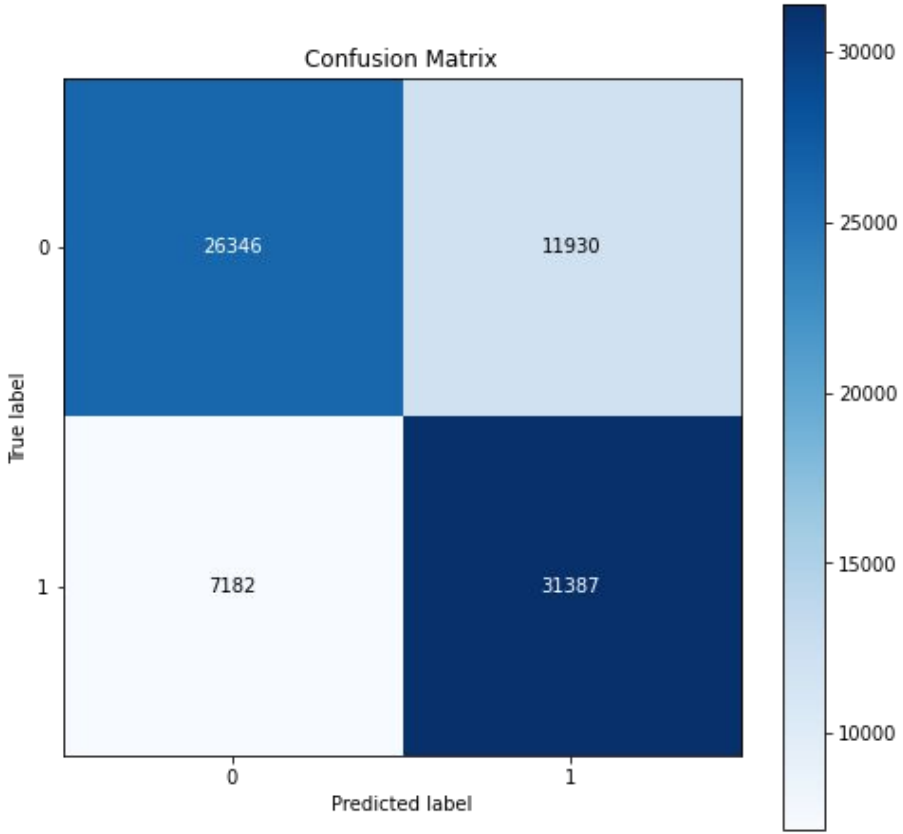
```
0.7558071442514152
```

SENSITIVITY & SPECIFICITY

```
## Confusion Matrix
from sklearn.metrics import confusion_matrix
# y_test is actual and y_pred is predicted
tn,fp,fn,tp=confusion_matrix(y_test,y_pred).ravel()
sensitivity = tp/(tp+fn)
print("sensitivity is ", sensitivity) # Also 'Recall'
specificity = tn/(tn+fp)
print("specificity is ", specificity) # Also 'Precision'
```

```
sensitivity is  0.8137882755580907
specificity is  0.6883164384993207
```

CONFUSION MATRIX




```
import scikitplot as skplt

skplt.metrics.plot_confusion_matrix(
    y_test,
    y_pred,
    figsize=(8,8))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9ccd7cf550>
```

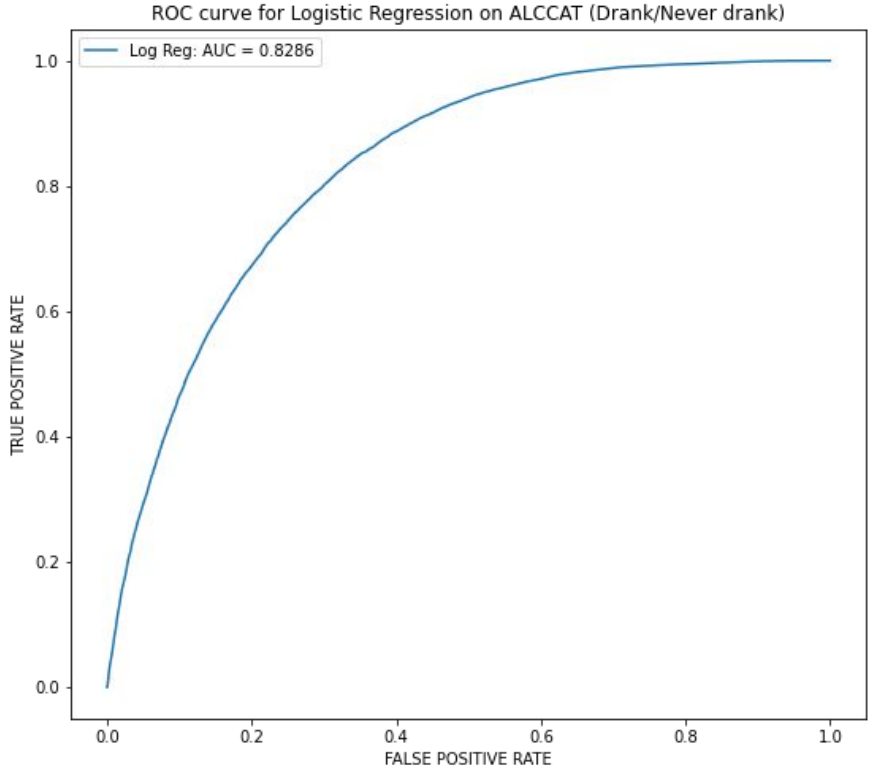
CLASSIFICATION REPORT:

```
# Classification report
import sklearn
print(sklearn.metrics.classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.79      0.69      0.74     38334
           1       0.73      0.82      0.77     38511

    accuracy                           0.76     76845
   macro avg       0.76      0.76      0.75     76845
weighted avg       0.76      0.76      0.75     76845
```

ROC CURVE

```
import sklearn.metrics as metrics
# calculate the fpr and tpr for all thresholds of the classification
probs = logca.predict_proba(x_test)
preds = probs[:,1] # Extracting data under '1'th column of probs
fpr, tpr, threshold = metrics.roc_curve(y_test, preds)
roc_auc = metrics.auc(fpr, tpr)
print('Area under ROC curve:',roc_auc)
print('\n')

import matplotlib.pyplot as plt
plt.plot(fpr,tpr,label="Log Reg: AUC = "+str(round(roc_auc,4)))
plt.rcParams["figure.figsize"] = (10,10)
plt.title('ROC curve for Logistic Regression on ALCCAT (Drank/Never drank)')
plt.xlabel('FALSE POSITIVE RATE')
plt.ylabel('TRUE POSITIVE RATE')
plt.legend(loc='upper left')
plt.show()
```



CLASSIFICATION BY LOGISTIC REGRESSION FOR ALLDRUGS (ALLDGSCAT)

Model building

```
## fit the model
from sklearn.linear_model import LogisticRegression
logca=LogisticRegression()
logca.fit(x_train,y_train)
```

```
LogisticRegression()
```

TRAINING & TEST MODEL SCORES

```
# model score for training data
logcd.score(x_train,y_train)
```

```
0.7255702791646333
```

```
# model score for test data
logcd.score(x_test,y_test)
```

```
0.7255473137331452
```

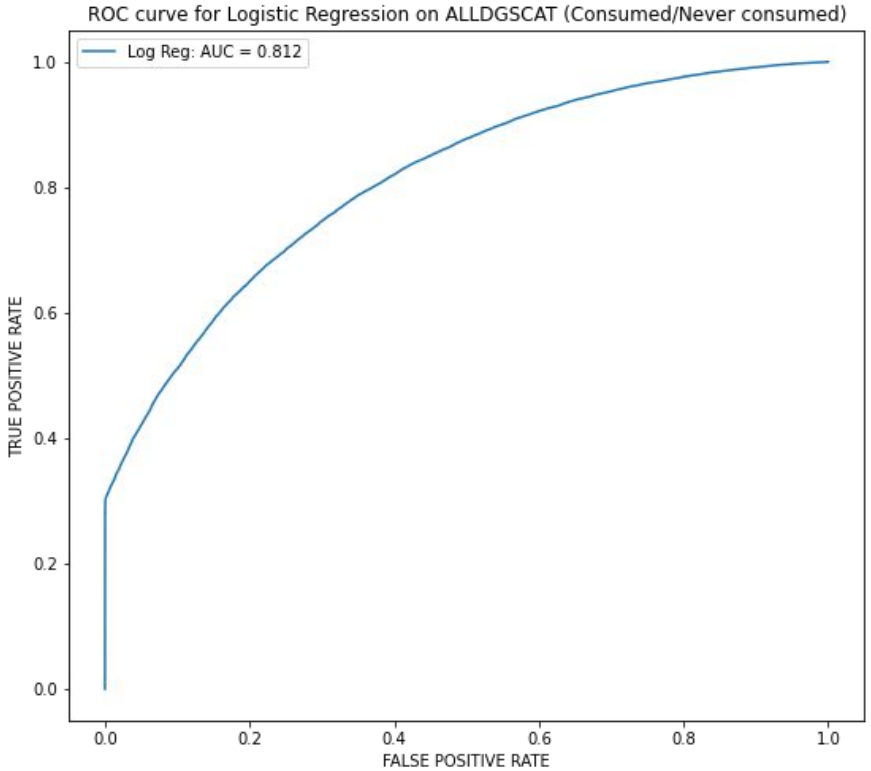
SENSITIVITY & SPECIFICITY

```
## Confusion Matrix
from sklearn.metrics import confusion_matrix
# y_test is actual and y_pred is predicted
tn,fp,fn,tp=confusion_matrix(y_test,y_pred).ravel()
sensitivity = tp/(tp+fn)
print("sensitivity is ", sensitivity) # Also 'Recall'
specificity = tn/(tn+fp)
print("specificity is ", specificity) # Also 'Precision'
```

```
sensitivity is  0.6862909672262191
specificity is  0.7649477505666206
```

ROC CURVE

```
import sklearn.metrics as metrics
# calculate the fpr and tpr for all thresholds of the classification
probs = logcd.predict_proba(x_test)
preds = probs[:,1] # Extracting data under '1'th column of probs
fpr, tpr, threshold = metrics.roc_curve(y_test, preds)
roc_auc = metrics.auc(fpr, tpr)
print('Area under ROC curve:',roc_auc)
print('\n')

import matplotlib.pyplot as plt
plt.plot(fpr,tpr,label="Log Reg: AUC = "+str(round(roc_auc,4)))
plt.rcParams["figure.figsize"] = (9,8)
plt.title('ROC curve for Logistic Regression on ALLDGSCAT (Consumed/Never consumed)')
plt.xlabel('FALSE POSITIVE RATE')
plt.ylabel('TRUE POSITIVE RATE')
plt.legend(loc='upper left')
plt.show()
```



RANDOM FOREST

Step 1: Search space

```
[ ]  import numpy as np
     from sklearn.model_selection import RandomizedSearchCV
     # Number of trees in random forest
     n_estimators = [int(x) for x in np.arange(2,101)]
     # Number of features to consider at every split
     max_features = ['auto', 'sqrt']
     # Maximum number of levels in tree
     max_depth = [int(x) for x in np.arange(2,31,2)]
     max_depth.append(None)
     # Minimum number of samples required to split a node
     min_samples_split = [int(x) for x in np.arange(2,51,2)]
     # Minimum number of samples required at each leaf node
     min_samples_leaf = [int(x) for x in np.arange(2,51,2)]
     # Method of selecting samples for training each tree
     bootstrap = [True, False]
     # Create the random grid
     random_grid = {'n_estimators': n_estimators,
                    'max_features': max_features,
                    'max_depth': max_depth,
                    'min_samples_split': min_samples_split,
                    'min_samples_leaf': min_samples_leaf,
                    'bootstrap': bootstrap}
     print(random_grid)
```

```
{'n_estimators': [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
```

- Random forest is a Supervised Machine Learning Algorithm that is used widely in Classification and Regression problems.
- It is an ensemble of multiple decision trees. Thus a collection of models is used to make predictions rather than an individual model
- It builds decision trees on different samples and takes their majority vote for classification and average in case of regression.
- The Random Forest models that we use for Classification and Regression have been tuned for the Hyperparameters (which are the most optimal settings for every model under which we obtain maximum possible accuracy) using Randomized Search
- Randomized Search picks the parameters randomly unlike Grid Search which uses a Brute-force way of trying all the possible combinations of parameters on our model
- In our model, we have defined a search space with an appropriate range or values for each hyperparameter.
- The Randomized Search technique randomly checks for parameters within the ranges for parameters defined in the search space and returns the most optimal ones defined within the search space

RANDOM FOREST CLASSIFICATION FOR ALCOHOL (ALCCAT)

Hyperparameter Tuning (Randomized Search) & Model building

```
# Use the random grid to search for best hyperparameters
# First create the base model to tune
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rfc_bestfit = RandomizedSearchCV(estimator = rfc, param_distributions = random_grid, n_iter = 100, cv = 3, verbose=2, random_state=42, n_jobs = -1)
# Fit the random search model
rfc_bestfit.fit(x_train, y_train)

rfc_bestfit.best_params_ # These are the parameters that we are setting to our model to get the best possible accuracy
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits
 {'n estimators': 71,
  'min_samples_split': 22,
  'min_samples_leaf': 4,
  'max_features': 'sqrt',
  'max_depth': 26,
  'bootstrap': False}
```

The above mentioned are the best parameters for the Random Forest Classification model for Alcohol

MODEL EVALUATION METRICS

```
# score for training data
rf_tm_ca.score(x_train,y_train)
```

```
0.7745682040737981
```

```
# score for test data
rf_tm_ca.score(x_test,y_test)
```

```
0.7464636606155247
```

```
## The final predicted value. 0 -> Never drank; 1 -> Drank
y_pred = rf_tm_ca.predict(x_test)
```

```
## Confusion Matrix
from sklearn.metrics import confusion_matrix
# y_test is actual and y_pred is predicted
tn,fp,fn,tp=confusion_matrix(y_test,y_pred).ravel()
sensitivity = tp/(tp+fn)
print("sensitivity is ", sensitivity) # Also 'Recall'
specificity = tn/(tn+fp)
print("specificity is ", specificity) # Also 'Precision'
```

```
sensitivity is  0.7920350540589592
specificity is  0.7005434214651479
```

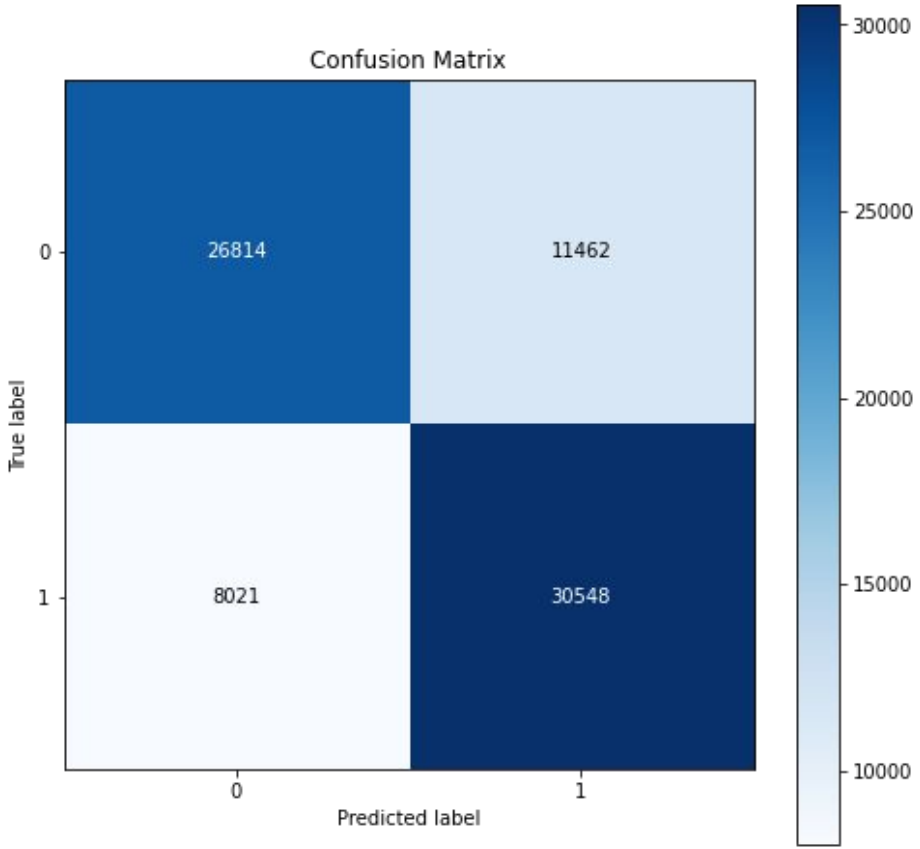
```
# Accuracy
from sklearn import metrics
print('Accuracy of the model is:', metrics.accuracy_score(y_test,y_pred))
```

```
Accuracy of the model is: 0.7464636606155247
```

```
# Classification report
import sklearn
print(sklearn.metrics.classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.77      0.70      0.73     38276
           1       0.73      0.79      0.76     38569

    accuracy                           0.75     76845
   macro avg       0.75      0.75      0.75     76845
weighted avg       0.75      0.75      0.75     76845
```

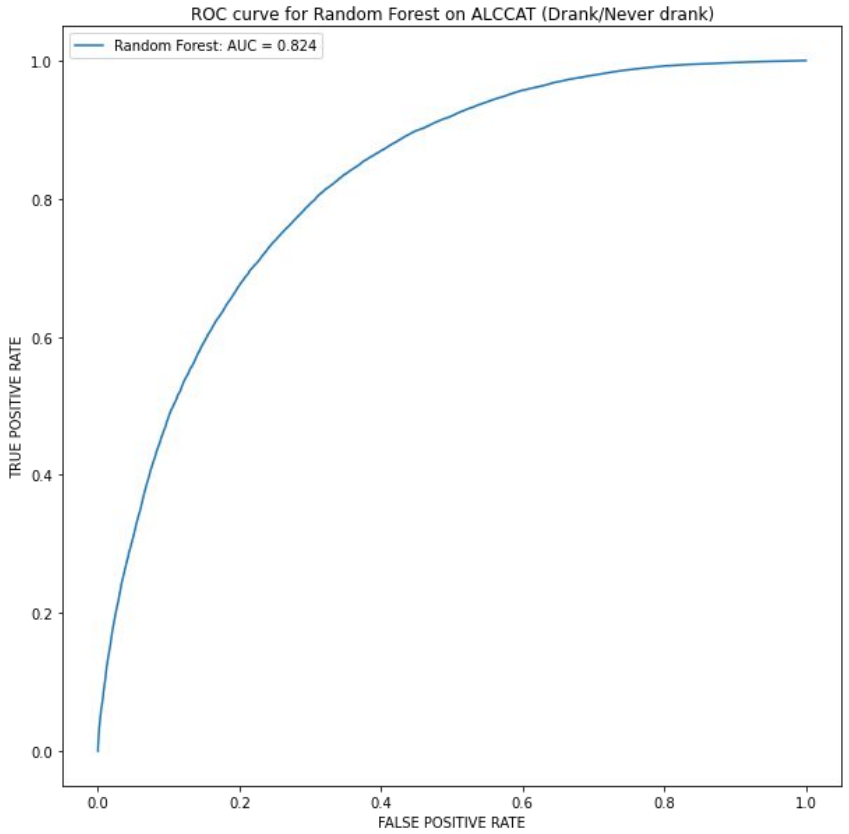
CONFUSION MATRIX




```python
import scikitplot as skplt

skplt.metrics.plot_confusion_matrix(
    y_test,
    y_pred,
    figsize=(8,8))
```

ROC CURVE

```
import sklearn.metrics as metrics
# calculate the fpr and tpr for all thresholds of the classification
probs = rf_tm_ca.predict_proba(x_test)
preds = probs[:,1] # Extracting data under '1'th column of probs
fpr, tpr, threshold = metrics.roc_curve(y_test, preds)
roc_auc = metrics.auc(fpr, tpr)
print('Area under ROC curve:',roc_auc)
print('\n')

import matplotlib.pyplot as plt
plt.plot(fpr,tpr,label="Random Forest: AUC = "+str(round(roc_auc,4)))
plt.rcParams["figure.figsize"] = (10,10)
plt.title('ROC curve for Random Forest on ALCCAT (Drank/Never drank)')
plt.xlabel('FALSE POSITIVE RATE')
plt.ylabel('TRUE POSITIVE RATE')
plt.legend(loc='upper left')
plt.show()
```



RANDOM FOREST CLASSIFICATION FOR ALLDRUGS (ALLDGSCAT)

Hyperparameter Tuning (Randomized Search) & Model building

```
# Use the random grid to search for best hyperparameters
# First create the base model to tune
from sklearn.ensemble import RandomForestClassifier
rfca = RandomForestClassifier()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rfca_bestfit = RandomizedSearchCV(estimator = rfca, param_distributions = random_grid, n_iter = 100, cv = 3, verbose=2, random_state=42, n_jobs = -1)
# Fit the random search model
rfca_bestfit.fit(x_train, y_train)

rfca_bestfit.best_params_ # These are the parameters that we are setting to our model to get the best possible accuracy
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits
```

```
{'n_estimators': 71,
 'min_samples_split': 22,
 'min_samples_leaf': 4,
 'max_features': 'sqrt',
 'max_depth': 26,
 'bootstrap': False}
```

The above mentioned are the best parameters for the Random Forest Classification model for Alldrugs

MODEL EVALUATION METRICS

```
# score for training data
rf_tm_cd.score(x_train,y_train)
```

```
0.7469674787592127
```

```
# score for test data
rf_tm_cd.score(x_test,y_test)
```

```
0.7233550557073786
```

```
## The final predicted value. 0 -> Never used Alcohol; 1 -> Ever used Alcohol
y_pred = rf_tm_cd.predict(x_test)
```

```
## Confusion Matrix
from sklearn.metrics import confusion_matrix
# y_test is actual and y_pred is predicted
tn,fp,fn,tp=confusion_matrix(y_test,y_pred).ravel()
sensitivity = tp/(tp+fn)
print("sensitivity is ", sensitivity)
specificity = tn/(tn+fp)
print("specificity is ", specificity)
```

```
sensitivity is  0.7566746602717825
specificity is  0.6899131516136149
```

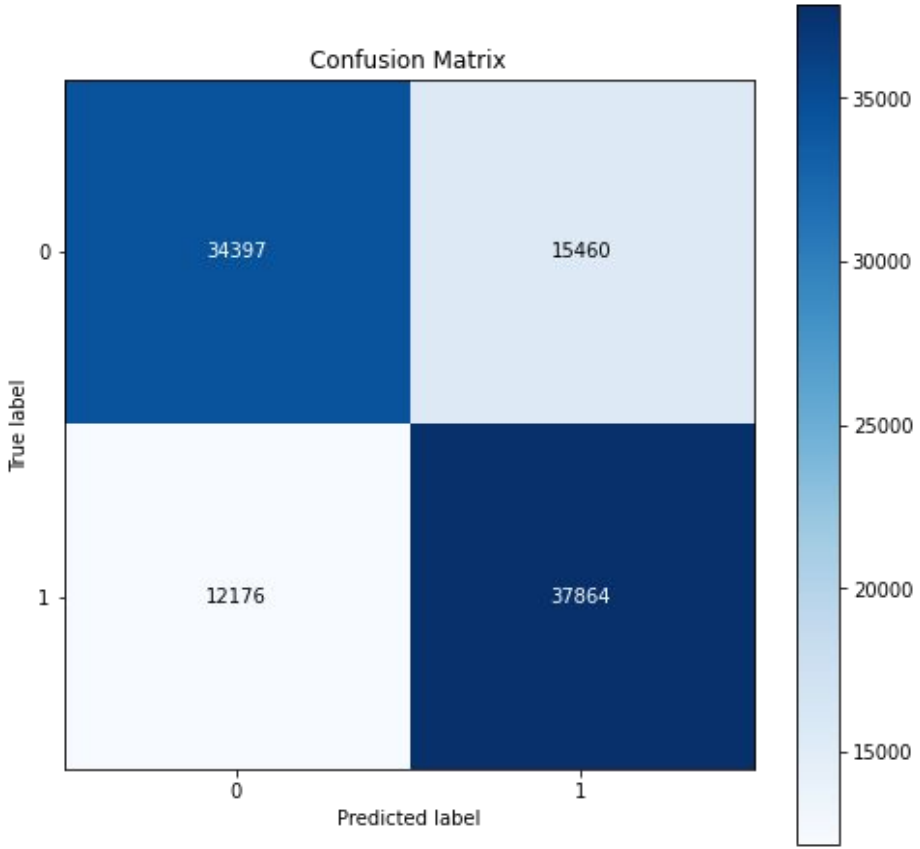
```
# Accuracy
from sklearn import metrics
print('Accuracy of the model is:', metrics.accuracy_score(y_test,y_pred))
```

```
Accuracy of the model is: 0.7233550557073786
```

```
# Classification report
import sklearn
print(sklearn.metrics.classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.74      0.69      0.71     49857
           1       0.71      0.76      0.73     50040

    accuracy                           0.72     99897
   macro avg       0.72      0.72      0.72     99897
weighted avg       0.72      0.72      0.72     99897
```

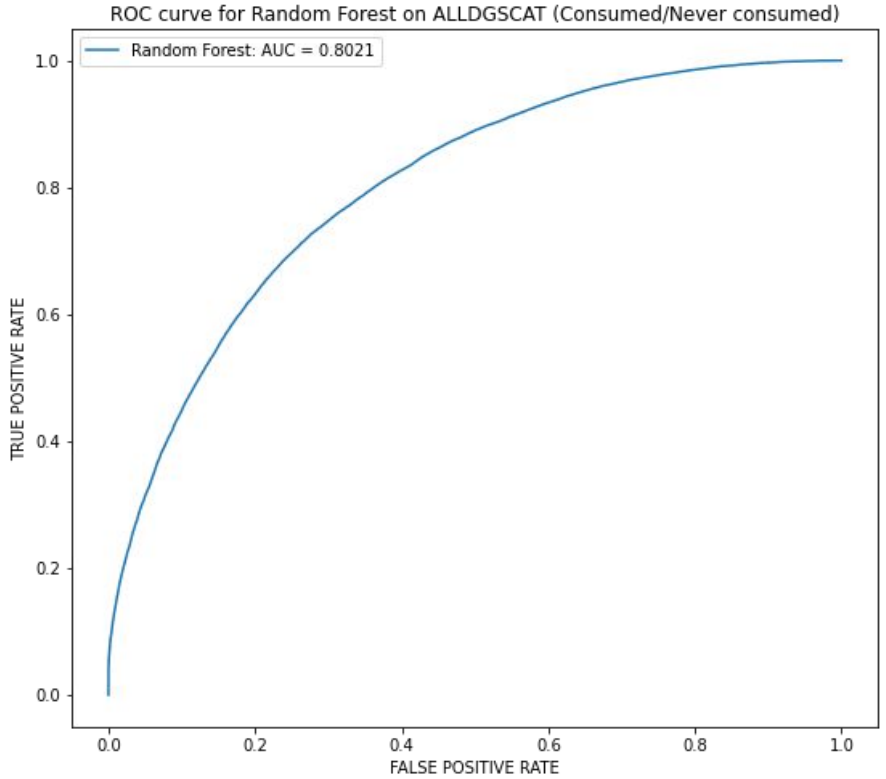
CONFUSION MATRIX




```
import scikitplot as skplt

skplt.metrics.plot_confusion_matrix(
    y_test,
    y_pred,
    figsize=(8,8))
```

ROC CURVE

```
import sklearn.metrics as metrics
# calculate the fpr and tpr for all thresholds of the classification
probs = rfca_bestfit.predict_proba(x_test)
preds = probs[:,1] # Extracting data under '1'th column of probs
fpr, tpr, threshold = metrics.roc_curve(y_test, preds)
roc_auc = metrics.auc(fpr, tpr)
print('Area under ROC curve:',roc_auc)
print('\n')

import matplotlib.pyplot as plt
plt.plot(fpr,tpr)
plt.rcParams["figure.figsize"] = (7,9)
plt.title('ROC curve for random forest on ALLDGSCAT (Consumed/Never consumed)')
plt.xlabel('FALSE POSITIVE RATE')
plt.ylabel('TRUE POSITIVE RATE')
```



CatBoost

- CatBoost stands for ‘Category Boosting’
- It is a Supervised Learning technique & is used for both Classification & Regression
- It is a high performance open source library for gradient boosting on decision trees
- It has the advantage of handling categorical features automatically
- I have used CatBoost since all the features in the independent variables of our dataset are categorical in nature
- For classification, since we had to tune the CatBoost model, we have taken the one-hot encoded form of the independent variables
- Firstly, we have to install scikit_optimize (skopt) and catboost using `!pip install scikit_optimize` and
  `!pip install catboost`
- We make use of Bayes Search (from skopt) and StratifiedKFold to find the hyperparameters
- Bayes Search uses Bayesian Optimization to explore the most promising hyperparameters in the problem-space.
- Bayesian Optimization finds the minimum to an objective function in large problem-spaces
- Bayes Search implements a “fit” and a “score” method
- This also checks for most optimal parameters in a given search space
- The Stratified folds are made by preserving the percentage of samples for each class

CATBOOST CLASSIFICATION FOR ALCOHOL (ALCCAT)

Hyperparameter Tuning (Bayes Search) & Model building

```
# Hyperparameter tuning for Catboost
# Reporting util for different optimizers
def report_perf(optimizer, X, y, title, callbacks=None):
    """
    A wrapper for measuring time and performances of different optmizers

    optimizer = a sklearn or a skopt optimizer
    X = the training set
    y = our target
    title = a string label for the experiment
    """
    start = time()
    if callbacks:
        optimizer.fit(X, y, callback=callbacks)
    else:
        optimizer.fit(X, y)
    d=pd.DataFrame(optimizer.cv_results_)
    best_score = optimizer.best_score_
    best_score_std = d.iloc[optimizer.best_index_].std_test_score
    best_params = optimizer.best_params_
    print((title + " took %.2f seconds,  candidates checked: %d, best CV score: %.3f "
           +u"\u00B1"+" %.3f") % (time() - start,
                                  len(optimizer.cv_results_['params']),
                                  best_score,
                                  best_score_std))
    print('Best parameters:')
    pprint.pprint(best_params)
    print()
    return best_params
```

```
roc_auc = make_scorer(roc_auc_score, greater_is_better=True, needs_threshold=True)
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=0)
```

```
clf = CatBoostClassifier(thread_count=2,
                         loss_function='Logloss',

                         od_type = 'Iter',
                         verbose= False
                        )
```

```
# Defining your search space
search_spaces = {'iterations': Integer(10, 1000),
                 'depth': Integer(1, 8),
                 'learning_rate': Real(0.01, 1.0, 'log-uniform'),
                 'random_strength': Real(1e-9, 10, 'log-uniform'),
                 'bagging_temperature': Real(0.0, 1.0),
                 'border_count': Integer(1, 255),
                 'l2_leaf_reg': Integer(2, 30),
                 'scale_pos_weight':Real(0.01, 1.0, 'uniform')}
```

```
# Setting up BayesSearchCV
# use just 1 job with CatBoost in order to avoid segmentation fault
opt = BayesSearchCV(clf,
                    search_spaces,
                    scoring=roc_auc,
                    cv=skf,
                    n_iter=100,
                    n_jobs=1,
                    return_train_score=False,
                    refit=True,
                    optimizer_kwargs={'base_estimator': 'GP'},
                    random_state=42)
```

```
# Tuning for CatBoost using Baye's Search
best_params_a = report_perf(opt,x_train, y_train,'CatBoost',
                            callbacks=[VerboseCallback(100),
                                       DeadlineStopper(60*10)])
```

Model Building with Optimal Parameters

```
best_params_a # These are the best parameters
```

```
OrderedDict([('bagging_temperature', 0.41010395885331385),
             ('border_count', 186),
             ('depth', 8),
             ('iterations', 323),
             ('l2_leaf_reg', 21),
             ('learning_rate', 0.0673344419215237),
             ('random_strength', 3.230824361824754e-06),
             ('scale_pos_weight', 0.7421091918485163)])
```

The above mentioned are the optimal parameters through Bayes Search

```
# Do this, if you have already run the tuning code
import collections
from collections import OrderedDict
best_params_a = [('bagging_temperature', 0.41010395885331385),
             ('border_count', 186),
             ('depth', 8),
             ('iterations', 323),
             ('l2_leaf_reg', 21),
             ('learning_rate', 0.0673344419215237),
             ('random_strength', 3.230824361824754e-06),
             ('scale_pos_weight', 0.7421091918485163)]

best_params_a = OrderedDict(best_params_a)
```

```
best_params_a['iterations']=1000
```

```
# Building model with optimal parameters
cat_tm_ca = CatBoostClassifier(**best_params_a,verbose=False)
cat_tm_ca = cat_tm_ca.fit(x_train,y_train)
```

```
y_pred = cat_tm_ca.predict(x_test)
```

MODEL EVALUATION METRICS

```
# model score for training data
cat_tm_ca.score(x_train,y_train)
```

```
0.7698639139303596
```

```
# model score for test data
cat_tm_ca.score(x_test,y_test)
```

```
0.7536859912811503
```

```
# Accuracy
from sklearn import metrics
print('Accuracy of the model is:', metrics.accuracy_score(y_test,y_pred))
```

```
Accuracy of the model is: 0.7536859912811503
```

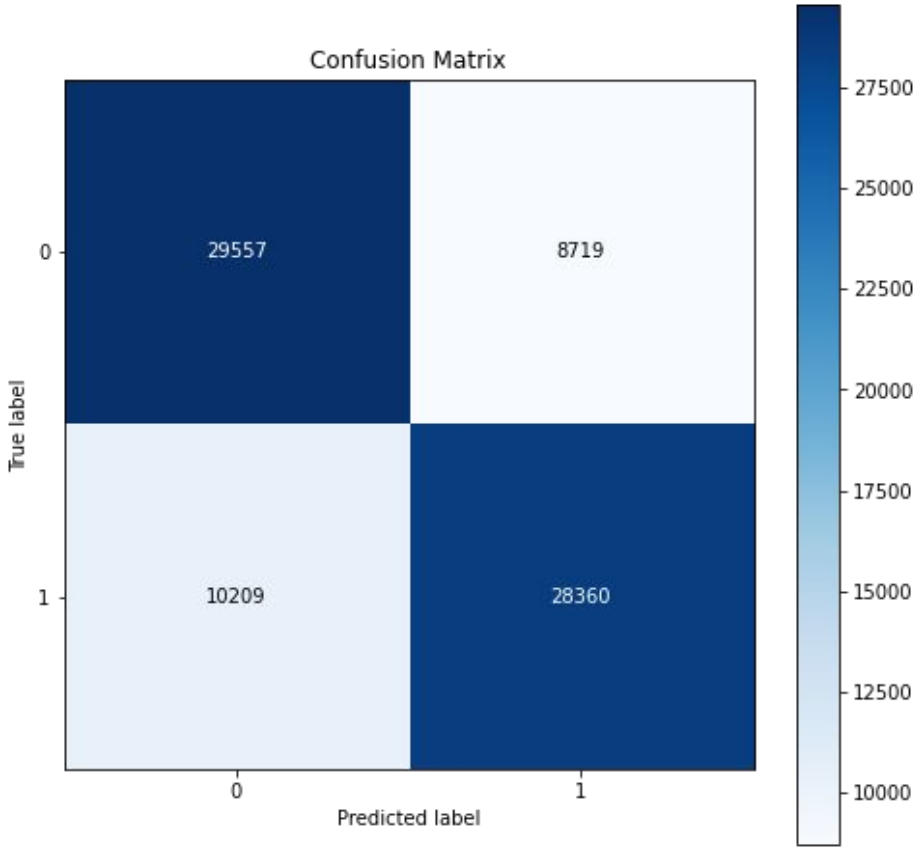
```
## Confusion Matrix
from sklearn.metrics import confusion_matrix
# y_test is actual and y_pred is predicted
tn,fp,fn,tp=confusion_matrix(y_test,y_pred).ravel()
sensitivity = tp/(tp+fn)
print("sensitivity is ", sensitivity) # Also 'Recall'
specificity = tn/(tn+fp)
print("specificity is ", specificity) # Also 'Precision'
```

```
sensitivity is  0.7353055562757655
specificity is  0.7722071271815236
```

```
# Classification report
import sklearn
print(sklearn.metrics.classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.74      0.77      0.76     38276
           1       0.76      0.74      0.75     38569

    accuracy                           0.75     76845
   macro avg       0.75      0.75      0.75     76845
weighted avg       0.75      0.75      0.75     76845
```

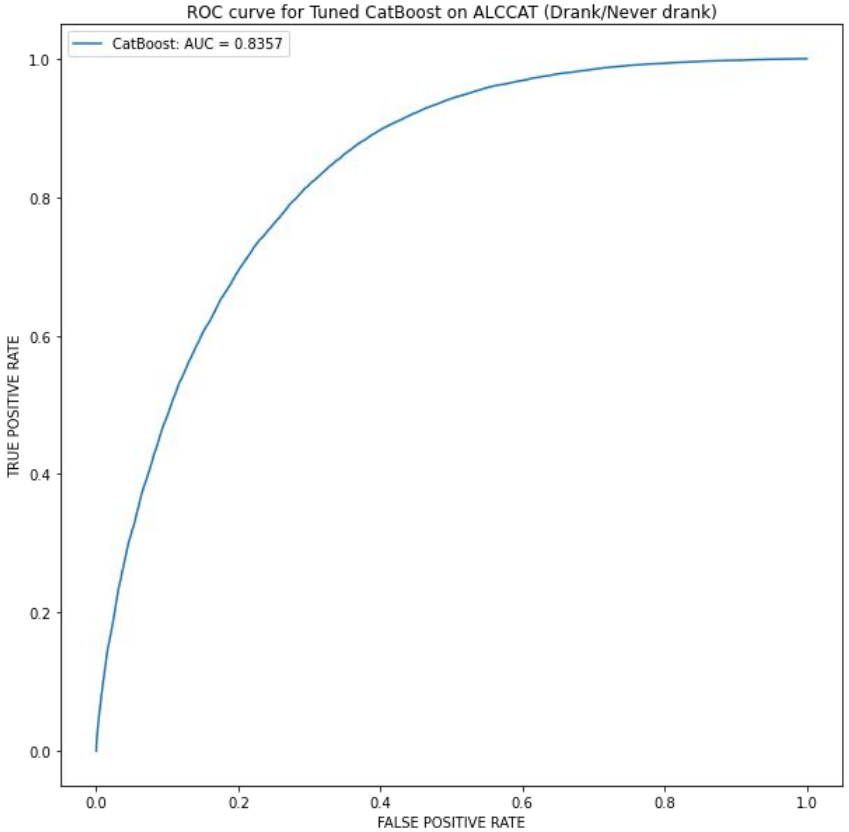
CONFUSION MATRIX




```
import scikitplot as skplt

skplt.metrics.plot_confusion_matrix(
    y_test,
    y_pred,
    figsize=(8,8))
```

ROC CURVE

```
import sklearn.metrics as metrics
# calculate the fpr and tpr for all thresholds of the classification
probs = cat_tm_ca.predict_proba(x_test)
preds = probs[:,1] # Extracting data under '1'th column of probs
fpr, tpr, threshold = metrics.roc_curve(y_test, preds)
roc_auc = metrics.auc(fpr, tpr)
print('Area under ROC curve:',roc_auc)
print('\n')

import matplotlib.pyplot as plt
plt.plot(fpr,tpr,label="CatBoost: AUC = "+str(round(roc_auc,4)))
plt.rcParams["figure.figsize"] = (9,8)
plt.title('ROC curve for Tuned CatBoost on ALCCAT (Drank/Never drank)')
plt.xlabel('FALSE POSITIVE RATE')
plt.ylabel('TRUE POSITIVE RATE')
plt.legend(loc='upper left')
plt.show()
```



CATBOOST CLASSIFICATION FOR ALLDRUGS (ALLDGSCAT)

Hyperparameter Tuning (Bayes Search) & Model building

```
# Hyperparameter tuning for Catboost
# Reporting util for different optimizers
def report_perf(optimizer, X, y, title, callbacks=None):
    """
    A wrapper for measuring time and performances of different optmizers

    optimizer = a sklearn or a skopt optimizer
    X = the training set
    y = our target
    title = a string label for the experiment
    """
    start = time()
    if callbacks:
        optimizer.fit(X, y, callback=callbacks)
    else:
        optimizer.fit(X, y)
    d=pd.DataFrame(optimizer.cv_results_)
    best_score = optimizer.best_score_
    best_score_std = d.iloc[optimizer.best_index_].std_test_score
    best_params = optimizer.best_params_
    print((title + " took %.2f seconds,  candidates checked: %d, best CV score: %.3f "
           +u"\u00B1"+" %.3f") % (time() - start,
                                  len(optimizer.cv_results_['params']),
                                  best_score,
                                  best_score_std))
    print('Best parameters:')
    pprint.pprint(best_params)
    print()
    return best_params
```

```
roc_auc = make_scorer(roc_auc_score, greater_is_better=True, needs_threshold=True)
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=0)
```

```
clf = CatBoostClassifier(thread_count=2,
                         loss_function='Logloss',

                         od_type = 'Iter',
                         verbose= False
                        )
```

```
# Defining your search space
search_spaces = {'iterations': Integer(10, 1000),
                 'depth': Integer(1, 8),
                 'learning_rate': Real(0.01, 1.0, 'log-uniform'),
                 'random_strength': Real(1e-9, 10, 'log-uniform'),
                 'bagging_temperature': Real(0.0, 1.0),
                 'border_count': Integer(1, 255),
                 'l2_leaf_reg': Integer(2, 30),
                 'scale_pos_weight':Real(0.01, 1.0, 'uniform')}
```

```
# Setting up BayesSearchCV
# use just 1 job with CatBoost in order to avoid segmentation fault
opt = BayesSearchCV(clf,
                    search_spaces,
                    scoring=roc_auc,
                    cv=skf,
                    n_iter=100,
                    n_jobs=1,
                    return_train_score=False,
                    refit=True,
                    optimizer_kwargs={'base_estimator': 'GP'},
                    random_state=42)
```

```
# Tuning for CatBoost using Baye's Search
best_params = report_perf(opt, x_train, y_train,'CatBoost',
                          callbacks=[VerboseCallback(100),
                                     DeadlineStopper(60*10)])
```

Model Building with Optimal Parameters

```
best_params # These are the best parameters
```

```
OrderedDict([('bagging_temperature', 0.41010395885331385),
             ('border_count', 186),
             ('depth', 8),
             ('iterations', 1000),
             ('l2_leaf_reg', 21),
             ('learning_rate', 0.0673344419215237),
             ('random_strength', 3.230824361824754e-06),
             ('scale_pos_weight', 0.7421091918485163)])
```

The above mentioned are the optimal parameters through Bayes Search

```
# Do this, if you have already run the tuning code
import collections
from collections import OrderedDict
best_params = [('bagging_temperature', 0.41010395885331385),
             ('border_count', 186),
             ('depth', 8),
             ('iterations', 1000),
             ('l2_leaf_reg', 21),
             ('learning_rate', 0.0673344419215237),
             ('random_strength', 3.230824361824754e-06),
             ('scale_pos_weight', 0.7421091918485163)]

best_params = OrderedDict(best_params)
```

```
best_params['iterations']=1000
```

```
cat_tm_cd = CatBoostClassifier(**best_params,verbose=False)
cat_tm_cd.fit(x_train,y_train)
```

MODEL EVALUATION METRICS

```
# model score for training data
cat_tm_cd.score(x_train,y_train)
```

```
0.747135152720948
```

```
# model score for test data
cat_tm_cd.score(x_test,y_test)
```

```
0.7387909546833238
```

```
## The final predicted value. 0 -> Not consumed Drugs; 1 -> Consumed Drugs
y_pred = cat_tm_cd.predict(x_test)
```

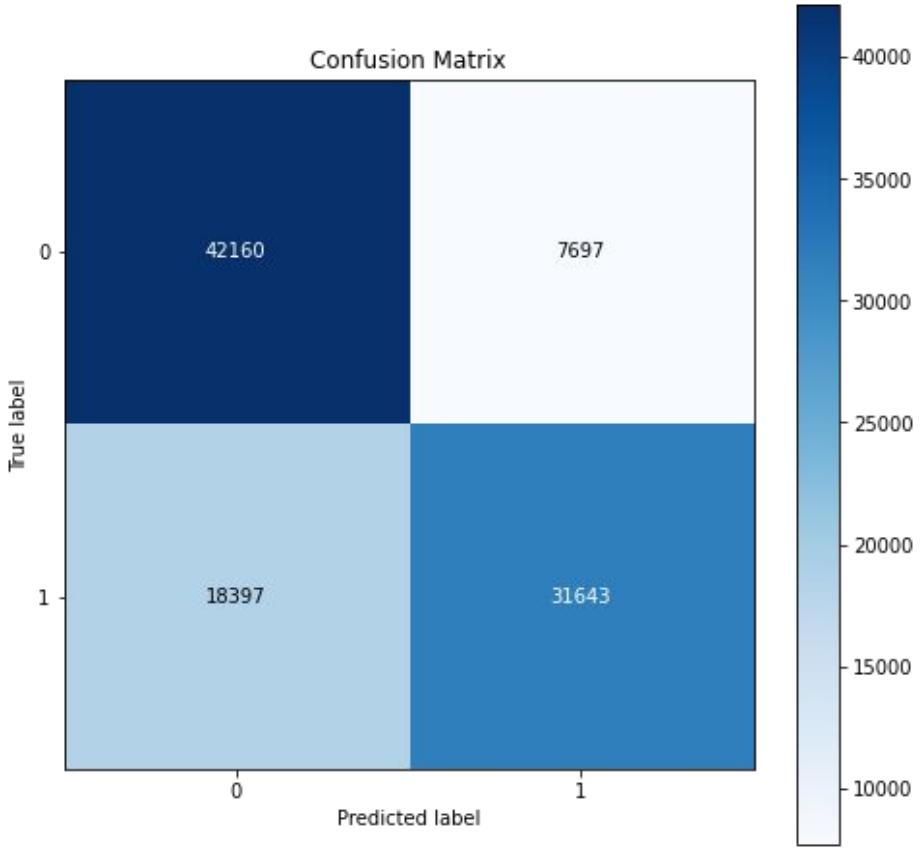
```
## Confusion Matrix
from sklearn.metrics import confusion_matrix
# y_test is actual and y_pred is predicted
tn,fp,fn,tp=confusion_matrix(y_test,y_pred).ravel()
sensitivity = tp/(tp+fn)
print("sensitivity is ", sensitivity) # Also 'Recall'
specificity = tn/(tn+fp)
print("specificity is ", specificity) # Also 'Precision'
```

```
sensitivity is  0.636031258161072
specificity is  0.8419529909413783
```

```
# Classification report
import sklearn
print(sklearn.metrics.classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.70      0.85      0.76     49857
           1       0.80      0.63      0.71     50040

    accuracy                           0.74     99897
   macro avg       0.75      0.74      0.74     99897
weighted avg       0.75      0.74      0.74     99897
```

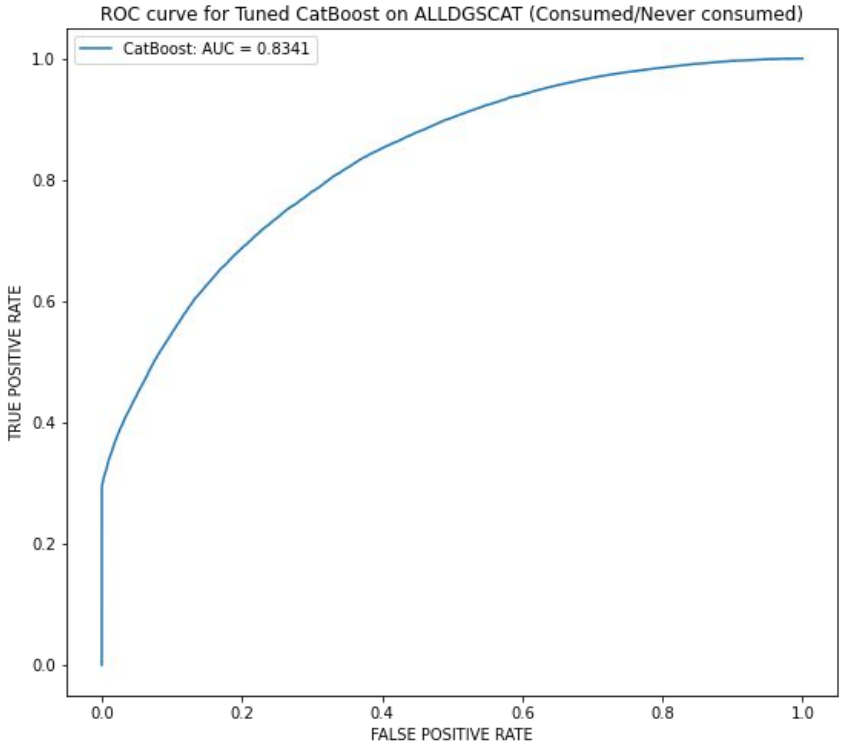
CONFUSION MATRIX




```
import scikitplot as skplt

skplt.metrics.plot_confusion_matrix(
    y_test,
    y_pred,
    figsize=(8,8))
```

ROC CURVE

```
import sklearn.metrics as metrics
# calculate the fpr and tpr for all thresholds of the classification
probs = cat_tm_cd.predict_proba(x_test)
preds = probs[:,1] # Extracting data under '1'th column of probs
fpr, tpr, threshold = metrics.roc_curve(y_test, preds)
roc_auc = metrics.auc(fpr, tpr)
print('Area under ROC curve:',roc_auc)
print('\n')

import matplotlib.pyplot as plt
plt.plot(fpr,tpr,label="CatBoost: AUC = "+str(round(roc_auc,4)))
plt.rcParams["figure.figsize"] = (9,8)
plt.title('ROC curve for Tuned CatBoost on ALLDGSCAT (Consumed/Never consumed)')
plt.xlabel('FALSE POSITIVE RATE')
plt.ylabel('TRUE POSITIVE RATE')
plt.legend(loc='upper left')
plt.show()
```



REMOVING ROWS IN INDEPENDENT VARIABLE WITH ZERO VALUES FOR ALCCAT & ALLDGSCAT

- We have created features like ALCCAT & ALLDGSCAT from the substance frequency variables IRALCFY & ALLDGS.
- When IRALCFY is a non-zero value, then ALCCAT is 1. Else, it is 0
- Likewise, when ALLDGS is a non-zero value, then ALLDGSCAT is 1. Else, it is 0
- Since we already use Classification to distinguish between zero & non-zero values in the frequency variable, we have to leave out the rows in the dataset with zeros in IRALCFY/ALCCAT for Alcohol Frequency prediction & ALLDGS/ALLDGSCAT for Alldrugs frequency prediction before feeding it to the train-test split

```
dk = dk[dk['ALCCAT'] == 1] # deleting rows with 0s in ALCCAT/IRALCFY
```

```
dl = dl[dl['ALLDGSCAT'] == 1] # Deleting rows with 0s in ALLDGSCAT
```

Declaring the target variables for predicting Alcohol Frequency and Alldrugs frequency

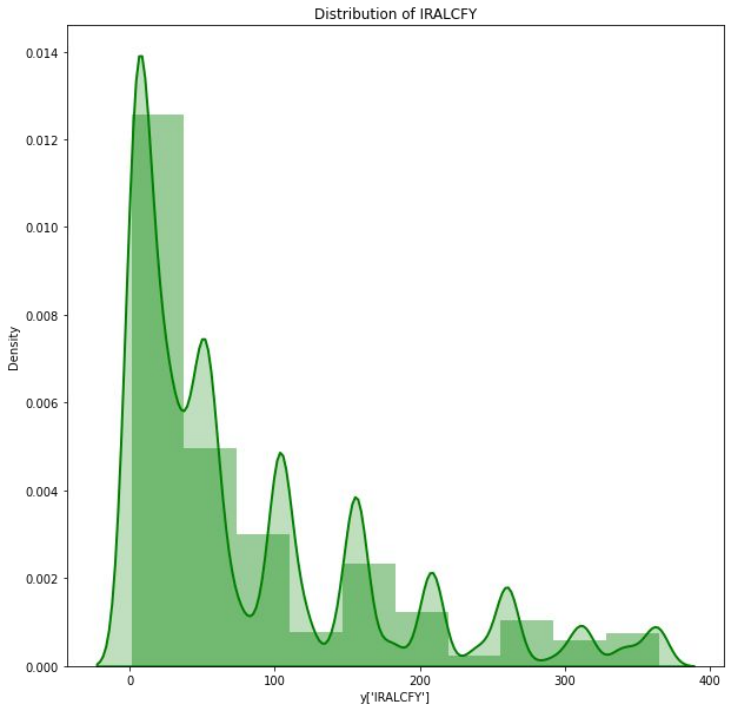
```
# create target variable
y = dk['IRALCFY']
```

```
# create target variable
y = dl['ALLDGS']
```

After that, we perform the train-test split

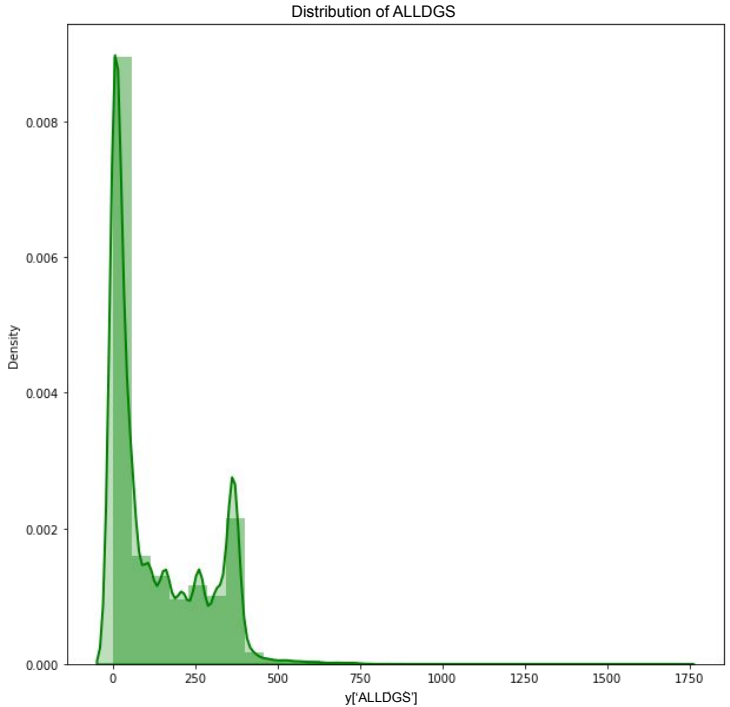
```
# split data into train n test
from sklearn.model_selection import train_test_split

# Outputs 4 variables. First two variables will be 80% (train) & 20% (test) of x and Last two variables are 80% (train) & 20% (test) of y
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2)

print(x_train.shape,y_train.shape)
print(x_test.shape,y_test.shape)
```

DISTRIBUTIONS OF IRALCFY AND ALLDGS VARIABLES




```
# Checking skewness of Y
da = pd.DataFrame(y)
da.IRALCFY.skew()
```

```
1.3947337096300492
```




```
# Checking skewness of Y
da = pd.DataFrame(y)
da.ALLDGS.skew()
```

```
1.3265830536421945
```

Therefore, both IRALCFY and ALLDGS do not follow Normal Distribution. They are positively skewed

LINEAR REGRESSION FOR ALCOHOL (IRALCFY)

```
# Linear Regression Modeling
# Fitting to the model
from sklearn import linear_model
lm = linear_model.LinearRegression()
model = lm.fit(x_train,y_train) # Passing the training data for Predictor & Target variables
prediction = lm.predict(x_test) # The predicted y_test from test data of predicted variable
```

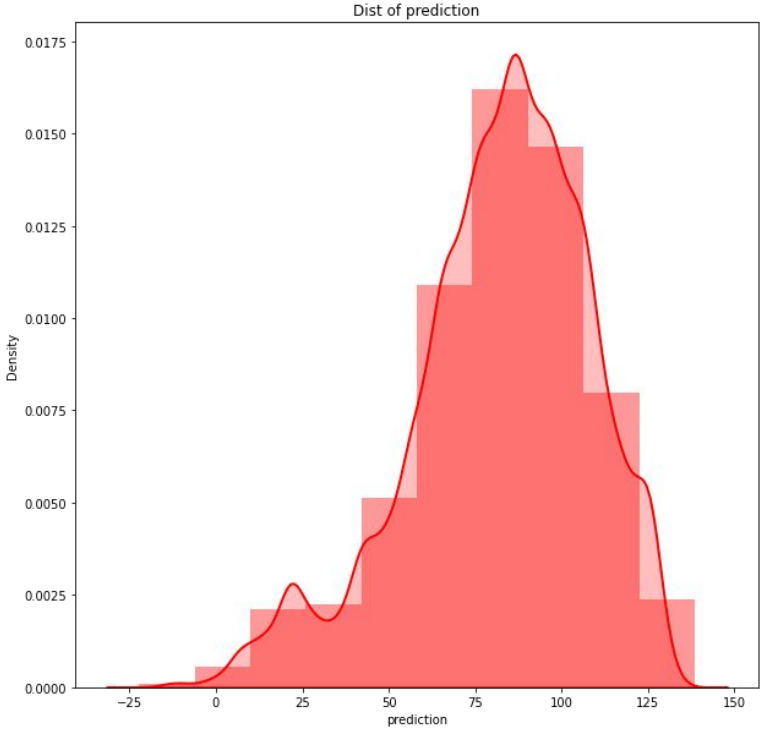
```
# model scores for train and test
import sklearn as sklearn
print(model.score(x_train,y_train))
print(model.score(x_test,y_test))
```

```
0.08226745078017306
0.0773372152438615
```

```
# print the coefficient
# Linear Reg eqn y = B0 + B1.x1 + B2.x2 +...+ BN.xN
print(model.intercept_) # intercept_ is used to find B0 (intercept or c)
print(model.coef_) # other coeffcients (B1, B2...BN)
```

```
83991284538450.58
[ 7.09607835e+00  7.66369933e+00 -1.64819461e+00  3.55468537e+00
 -4.23213688e+13 -4.23213688e+13 -4.23213688e+13 -4.23213688e+13
 -4.23213688e+13 -4.23213688e+13 -2.48092953e+14 -2.48092953e+14
 -2.48092953e+14 -2.48092953e+14 -2.48092953e+14 -2.48092953e+14
 -9.95483457e+13 -9.95483457e+13 -9.95483457e+13 -9.95483457e+13
 -9.95483457e+13 -9.95483457e+13  2.89376308e+14  2.89376308e+14
  2.89376308e+14  2.89376308e+14  2.89376308e+14  2.89376308e+14
 -4.90465655e+14 -4.90465655e+14 -4.90465655e+14 -4.90465655e+14
 -4.90465655e+14 -4.90465655e+14  1.86519327e+14  1.86519327e+14
  3.63529189e+13  3.63529189e+13  3.63529189e+13  3.63529189e+13
  3.63529189e+13  3.63529189e+13  3.63529189e+13  3.63529189e+13
  3.63529189e+13  3.63529189e+13  3.63529189e+13  2.82162053e+14
  2.82162053e+14  2.82162053e+14  2.82162053e+14  4.37455185e+13
  4.37455185e+13  4.37455185e+13  4.37455185e+13  4.37455185e+13
  4.37455185e+13  4.37455185e+13 -3.33649969e+13 -3.33649969e+13
 -3.33649969e+13 -3.33649969e+13 -3.33649969e+13 -1.09500882e+14
 -1.09500882e+14 -1.09500882e+14 -1.09500882e+14 -1.09500882e+14
 -1.09500882e+14 -1.09500882e+14 -2.90717646e+14 -2.90717646e+14
 -2.90717646e+14 -2.90717646e+14 -1.85843183e+14 -1.85843183e+14
 -1.85843183e+14  5.77707621e+14  5.77707621e+14  5.77707621e+14]
```

DISTRIBUTION OF THE PREDICTION



We observe from the graph that the distribution for the prediction using Linear Regression model has values which are negative. We all know that IRALCFY (Alcohol consumption freq) has to be a positive number

EVALUATION METRICS

```
# Evaluation metrics

import sklearn.metrics as metrics
mae = metrics.mean_absolute_error(y_test, prediction)
mse = metrics.mean_squared_error(y_test, prediction)
rmse = np.sqrt(mse) #mse**(0.5)
r2 = metrics.r2_score(y_test,prediction)
mape = metrics.mean_absolute_percentage_error(y_test, prediction)

print("Results of sklearn.metrics:")
print("MAE:",mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R-Squared:", r2)
print("MAPE:",mape)
```

```
Results of sklearn.metrics:
MAE: 67.3325347936132
MSE: 7585.800442891922
RMSE: 87.09650074998376
R-Squared: 0.0773372152438615
MAPE: 6.050438707733569
```

LINEAR REGRESSION FOR ALLDRUGS (ALLDGS)

```
# Linear Regression Modeling
# Fitting to the model
from sklearn import linear_model
lm = linear_model.LinearRegression()
model = lm.fit(x_train,y_train) # Passing the training data for Predictor & Target variables
prediction = lm.predict(x_test) # The predicted y_test from test data of predicted variable
```

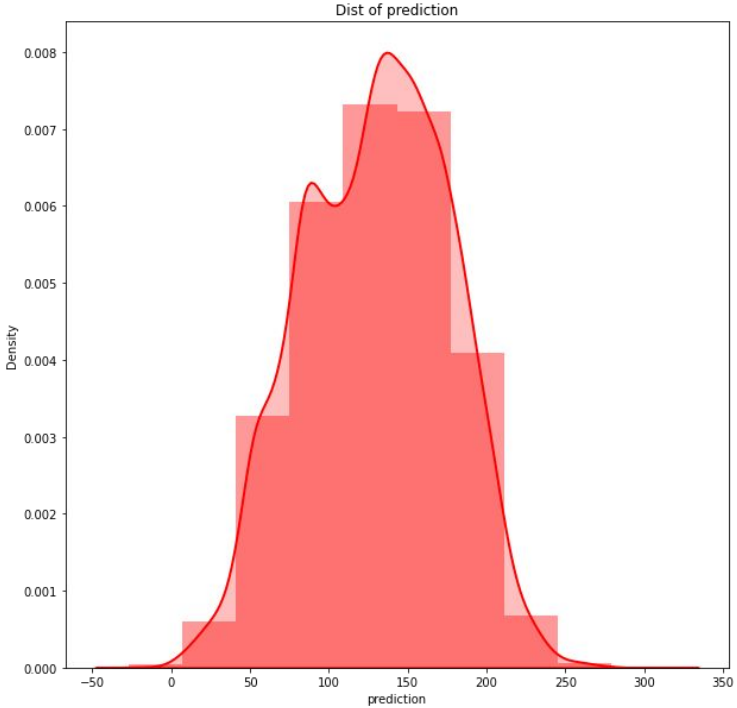
```
# model scores for train and test
import sklearn as sklearn
print(model.score(x_train,y_train))
print(model.score(x_test,y_test))
```

```
0.09436835695801271
0.08648754625047828
```

```
# print the coefficient
# Linear Reg eqn y = B0 + B1.x1 + B2.x2 +...+ BN.xN
print(model.intercept_) # intercept_ is used to find B0 (intercept or c)
print(model.coef_) # other coeffcients (B1, B2...BN)
```

```
-864209604472485.9
[ 1.48021543e+01  6.95576797e+01 -2.13752329e+01  4.21708344e+00
 -8.74559232e+14 -8.74559232e+14 -8.74559232e+14 -8.74559232e+14
 -8.74559232e+14 -8.74559232e+14  2.00070968e+14  2.00070968e+14
  2.00070968e+14  2.00070968e+14  2.00070968e+14  2.00070968e+14
 -8.38725623e+14 -8.38725623e+14 -8.38725623e+14 -8.38725623e+14
 -8.38725623e+14 -8.38725623e+14 -4.98439144e+14 -4.98439144e+14
 -4.98439144e+14 -4.98439144e+14 -4.98439144e+14 -4.98439144e+14
  3.82032808e+13  3.82032808e+13  3.82032808e+13  3.82032808e+13
  3.82032808e+13  3.82032808e+13  7.20687797e+13  7.20687797e+13
  1.23322195e+14  1.23322195e+14  1.23322195e+14  1.23322195e+14
  1.23322195e+14  1.23322195e+14  1.23322195e+14  1.23322195e+14
  1.23322195e+14  1.23322195e+14  1.23322195e+14 -8.25935991e+14
 -8.25935991e+14 -8.25935991e+14 -8.25935991e+14  9.50373893e+14
  9.50373893e+14  9.50373893e+14  9.50373893e+14  9.50373893e+14
  9.50373893e+14  9.50373893e+14 -4.19542660e+14 -4.19542660e+14
 -4.19542660e+14 -4.19542660e+14 -4.19542660e+14 -1.69010055e+15
 -1.69010055e+15 -1.69010055e+15 -1.69010055e+15 -1.69010055e+15
 -1.69010055e+15 -1.69010055e+15  1.63925586e+15  1.63925586e+15
  1.63925586e+15  1.63925586e+15  1.50757367e+15  1.50757367e+15
  1.50757367e+15  1.48064416e+15  1.48064416e+15  1.48064416e+15]
```

DISTRIBUTION OF THE PREDICTION



We observe from the graph that the distribution for the prediction using Linear Regression model has values which are negative. We all know that ALLDGS (Alldrugs consumption freq) has to be a positive number

EVALUATION METRICS

```
# Evaluation metrics

import sklearn.metrics as metrics
mae = metrics.mean_absolute_error(y_test, prediction)
mse = metrics.mean_squared_error(y_test, prediction)
rmse = np.sqrt(mse) #mse**(0.5)
r2 = metrics.r2_score(y_test,prediction)
mape = metrics.mean_absolute_percentage_error(y_test, prediction)

print("Results of sklearn.metrics:")
print("MAE:",mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R-Squared:", r2)
print("MAPE:",mape)
```

```
Results of sklearn.metrics:
MAE: 116.80163455703884
MSE: 20683.889513330552
RMSE: 143.818946990063
R-Squared: 0.08648754625047828
MAPE: 15.828576233565963
```

RANDOM FOREST REGRESSION FOR ALCOHOL (IRALCFY)

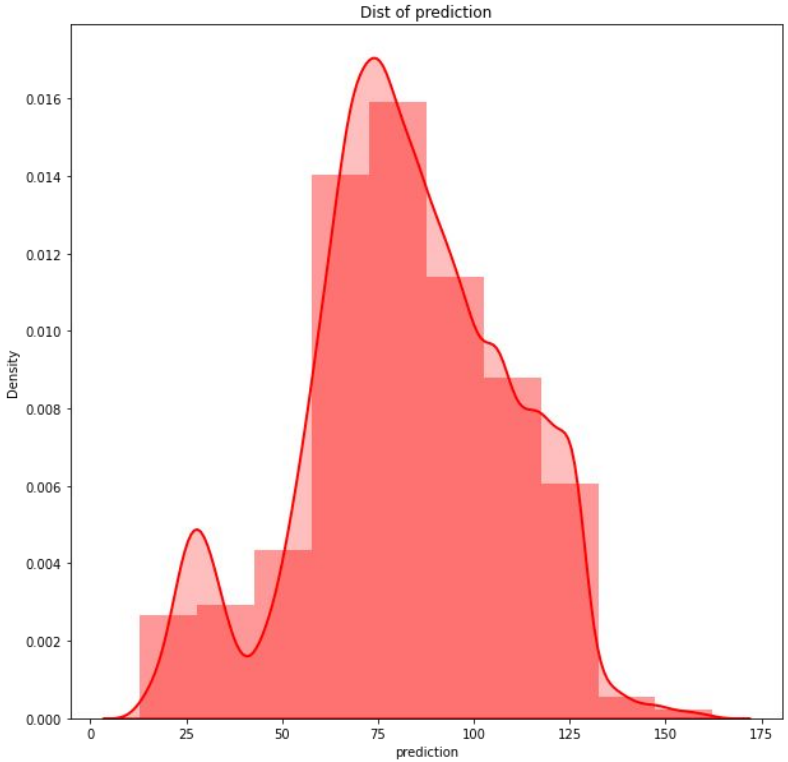
Hyperparameter Tuning (Randomized Search) & Model building

```
# Use the random grid to search for best hyperparameters
# First create the base model to tune
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_bestfit = RandomizedSearchCV(estimator = rf, param_distributions = random_grid, n_iter = 100, cv = 3, verbose=2, random_state=42, n_jobs = -1)
# Fit the random search model
rf_bestfit.fit(x_train, y_train)

rf_bestfit.best_params_ # These are the parameters that we are setting to our model to get the best possible accuracy
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits
{'n_estimators': 43,
 'min_samples_split': 48,
 'min_samples_leaf': 18,
 'max_features': 'sqrt',
 'max_depth': 28,
 'bootstrap': True}
```

```
# Run this, if you have already tuned the model
from sklearn.ensemble import RandomForestRegressor
rf_bestfit = RandomForestRegressor(n_estimators= 43,
 min_samples_split= 48,
 min_samples_leaf= 18,
 max_features= 'sqrt',
 max_depth= 28,
 bootstrap= True,random_state=42)
rf_bestfit = rf_bestfit.fit(x_train,y_train)
prediction = rf_bestfit.predict(x_test)
```

DISTRIBUTION OF THE PREDICTION




```
# Checking the distribution of prediction
import seaborn as sns
fig, ax = plt.subplots(1,1,figsize=(10, 10))

# plotting the original data(non-normal) and
# fitted data (normal)
sns.distplot(prediction, hist = True, kde = True,
             kde_kws = {'shade': True, 'linewidth': 2},
             label = "Non-Normal", color ="red", ax = ax,bins=10)
ax.title.set_text('Dist of prediction')
ax.set_xlabel('prediction')
```

```
# score on Training data
print(rf_bestfit.score(x_train,y_train))
```

```
0.11502089672821669
```

```
# score on Test data
print(rf_bestfit.score(x_test,y_test))
```

```
0.09366934372894598
```

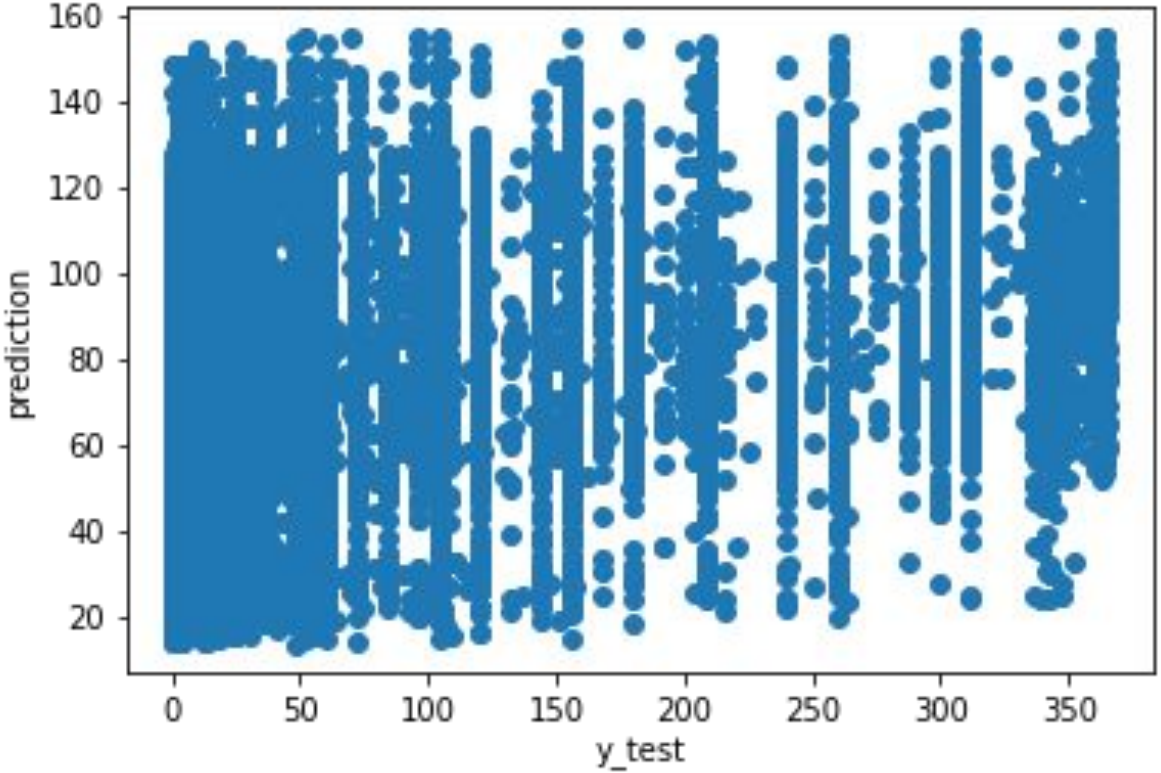
We observe from the graph that the distribution for the prediction using Random Forest Regression model does not have values which are negative.

MODEL EVALUATION METRICS

```
# Evaluation metrics

import sklearn.metrics as metrics
mae = metrics.mean_absolute_error(y_test, prediction)
mse = metrics.mean_squared_error(y_test, prediction)
rmse = np.sqrt(mse) #mse**(0.5)
r2 = metrics.r2_score(y_test,prediction)
mape = metrics.mean_absolute_percentage_error(y_test, prediction)

print("Results of sklearn.metrics:")
print("MAE:",mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R-Squared:", r2)
print("MAPE:",mape)
```

```
Results of sklearn.metrics:
MAE: 66.65218863624317
MSE: 7501.849392723051
RMSE: 86.61321719416183
R-Squared: 0.09366934372894598
MAPE: 5.851723056694891
```

Test data vs Prediction scatterplot for IRALCFY



RANDOM FOREST REGRESSION FOR ALLDRUGS (ALLDGS)

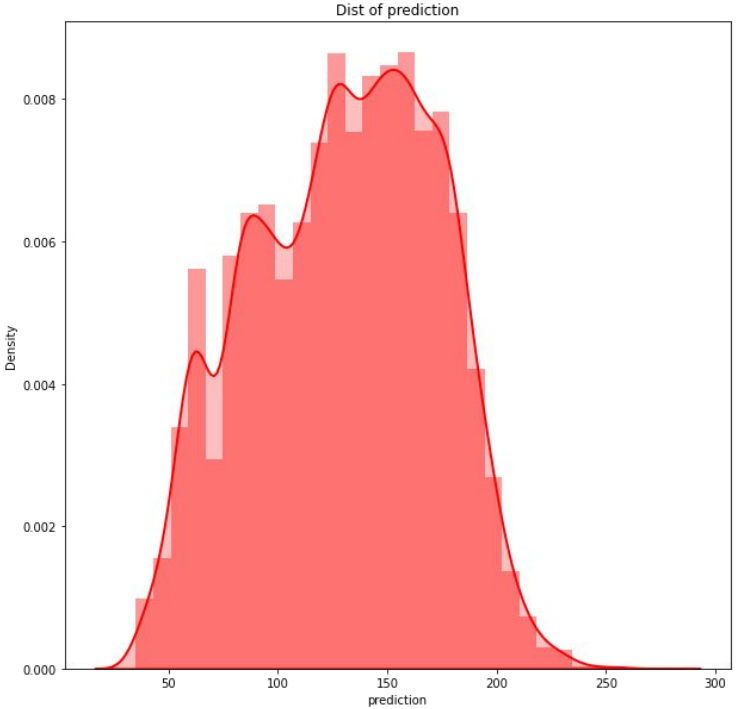
Hyperparameter Tuning (Randomized Search) & Model building

```
# Use the random grid to search for best hyperparameters
# First create the base model to tune
from sklearn.ensemble import RandomForestRegressor
rfa = RandomForestRegressor()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rfa_bestfit = RandomizedSearchCV(estimator = rfa, param_distributions = random_grid, n_iter = 100, cv = 3, verbose=2, random_state=42, n_jobs = -1)
# Fit the random search model
rfa_bestfit.fit(x_train, y_train)

rfa_bestfit.best_params_ # These are the parameters that we are setting to our model to get the best possible accuracy
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits
{'n_estimators': 52,
 'min_samples_split': 26,
 'min_samples_leaf': 10,
 'max_features': 'sqrt',
 'max_depth': 14,
 'bootstrap': False}
```

```
# model build
from sklearn.ensemble import RandomForestRegressor
rfa_bestfit = RandomForestRegressor(n_estimators= 52,
 min_samples_split= 26,
 min_samples_leaf= 10,
 max_features= 'sqrt',
 max_depth= 14,
 bootstrap= False, random_state=42)
rfa_bestfit = rfa_bestfit.fit(x_train,y_train)
prediction = rfa_bestfit.predict(x_test) # Prediction
```

DISTRIBUTION OF THE PREDICTION




```
print(min(np.array(prediction)))
```

```
34.98647937554737
```

```
# Checking the distribution of prediction
import seaborn as sns
fig, ax = plt.subplots(1,1,figsize=(10, 10))

# plotting the original data(non-normal) and
# fitted data (normal)
sns.distplot(prediction, hist = True, kde = True,
             kde_kws = {'shade': True, 'linewidth': 2},
             label = "Non-Normal", color ="red", ax = ax,bins=30)
ax.title.set_text('Dist of prediction')
ax.set_xlabel('prediction')
```

```
# score on Training data
print(rfa_bestfit.score(x_train,y_train))
```

```
0.14587565491390486
```

```
# score on Test data
print(rfa_bestfit.score(x_test,y_test))
```

```
0.0887552559157
```

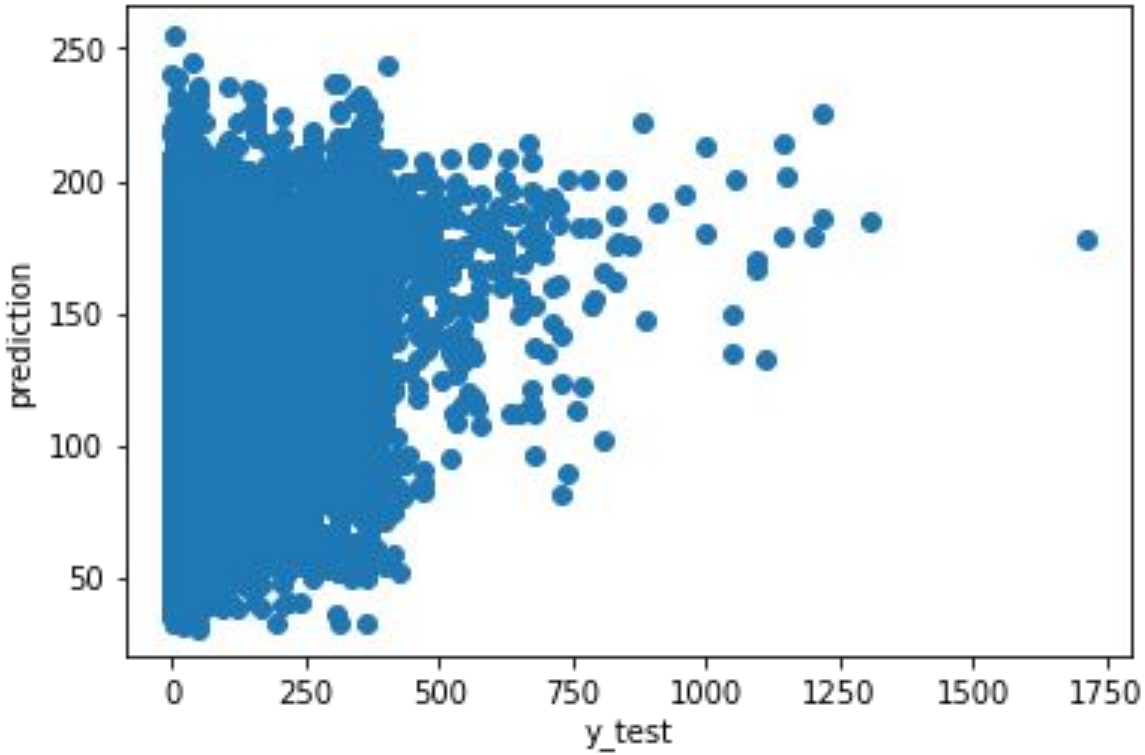
We observe from the graph that the distribution for the prediction using RF Regression model does not have values which are negative.

MODEL EVALUATION METRICS

```
# Evaluation metrics

import sklearn.metrics as metrics
mae = metrics.mean_absolute_error(y_test, prediction)
mse = metrics.mean_squared_error(y_test, prediction)
rmse = np.sqrt(mse) #mse**(0.5)
r2 = metrics.r2_score(y_test,prediction)
mape = metrics.mean_absolute_percentage_error(y_test, prediction)

print("Results of sklearn.metrics:")
print("MAE:",mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R-Squared:", r2)
print("MAPE:",mape)
```

```
Results of sklearn.metrics:
MAE: 117.2241321299863
MSE: 20632.543682224215
RMSE: 143.64032749274912
R-Squared: 0.0887552559157
MAPE: 16.076025282912205
```

Test data vs Prediction scatterplot for ALLDGS



CATBOOST REGRESSOR

```
# Declaring Categorical features
cat_features = list(range(0, X.shape[1]))
print(cat_features)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17]
```

In CatBoost, it is possible to declare which variables in your dataset are categorical in nature. This is exactly what we have done for the CatBoost Regressor on IRALCFY & ALLDGS

```
# Train test split
from sklearn.model_selection import train_test_split

X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=0)
```

CATBOOST REGRESSION FOR ALCOHOL (IRALCFY)

```
# catboost implementation
from catboost import CatBoostRegressor

clf = CatBoostRegressor(
    iterations=5,
    learning_rate=0.1,
    #loss_function='CrossEntropy'
)


clf.fit(X_train, y_train,
        cat_features=cat_features,
        eval_set=(X_val, y_val),
        verbose=False
)

print('CatBoost model is fitted: ' + str(clf.is_fitted()))
print('CatBoost model parameters:')
print(clf.get_params())
```

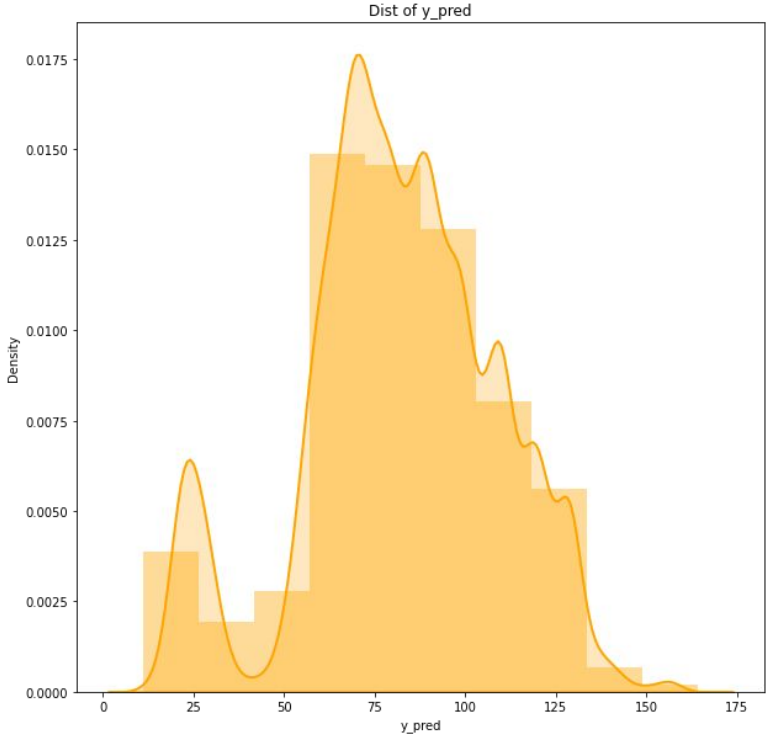
```
CatBoost model is fitted: True
CatBoost model parameters:
{'iterations': 5, 'learning_rate': 0.1, 'loss_function': 'RMSE'}
```

```
# Stdout of the training
from catboost import CatBoostRegressor
clf = CatBoostRegressor(
    iterations=10,
#     verbose=5,
)

clf.fit(
    X_train, y_train,
    cat_features=cat_features,
    eval_set=(X_val, y_val),
)
```

```
Learning rate set to 0.5
0:      learn: 88.7743962       test: 88.6560109       best: 88.6560109 (0)    total: 167ms    remaining: 1.5s
1:      learn: 87.9530420       test: 87.8639898       best: 87.8639898 (1)    total: 293ms    remaining: 1.17s
2:      learn: 87.3975179       test: 87.3454853       best: 87.3454853 (2)    total: 422ms    remaining: 985ms
3:      learn: 87.1109220       test: 87.0725560       best: 87.0725560 (3)    total: 562ms    remaining: 844ms
4:      learn: 86.9632323       test: 86.9541484       best: 86.9541484 (4)    total: 695ms    remaining: 695ms
5:      learn: 86.9252421       test: 86.9243924       best: 86.9243924 (5)    total: 806ms    remaining: 537ms
6:      learn: 86.8476882       test: 86.8616067       best: 86.8616067 (6)    total: 923ms    remaining: 395ms
7:      learn: 86.8005121       test: 86.8154670       best: 86.8154670 (7)    total: 1.04s    remaining: 260ms
8:      learn: 86.7357402       test: 86.7608915       best: 86.7608915 (8)    total: 1.16s    remaining: 129ms
9:      learn: 86.6889754       test: 86.7444333       best: 86.7444333 (9)    total: 1.28s    remaining: 0us

bestTest = 86.74443326
bestIteration = 9

<catboost.core.CatBoostRegressor at 0x7f4f1165bfd0>
```

```
y_pred = clf.predict(X_val)
```

DISTRIBUTION OF PREDICTION




```
import seaborn as sns
fig, ax = plt.subplots(1,1,figsize=(10, 10))

# plotting the original data(non-normal) and
# fitted data (normal)
sns.distplot(y_pred, hist = True, kde = True,
             kde_kws = {'shade': True, 'linewidth': 2},
             label = "Non-Normal", color ="orange", ax = ax,bins=10)
ax.title.set_text('Dist of y_pred')
ax.set_xlabel('y_pred')
```

```
# score on Training data
print(clf.score(X_train,y_train))
```

```
0.0913498690706609
```

```
# score on Test data
print(clf.score(X_val,y_val))
```

```
0.08847903238038313
```

We observe from the graph that the distribution for the prediction using CatBoost Regression model does not have values which are negative.

MODEL EVALUATION METRICS

```
import sklearn.metrics as metrics
mae = metrics.mean_absolute_error(y_val, y_pred)
mse = metrics.mean_squared_error(y_val, y_pred)
rmse = np.sqrt(mse) #mse**(0.5)
r2 = metrics.r2_score(y_val,y_pred)
mape = metrics.mean_absolute_percentage_error(y_val, y_pred)

print("Results of sklearn.metrics:")
print("MAE:",mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R-Squared:", r2)
print("MAPE:",mape)
```

```
Results of sklearn.metrics:
MAE: 67.0744796879123
MSE: 7524.596701757585
RMSE: 86.74443326091644
R-Squared: 0.08847903238038313
MAPE: 5.963419228648019
```

CATBOOST REGRESSION FOR ALLDRUGS (ALLDGS)

```
# catboost implementation
from catboost import CatBoostRegressor

clf = CatBoostRegressor(
    iterations=5,
    learning_rate=0.1,
    #loss_function='CrossEntropy'
)


clf.fit(X_train, y_train,
        cat_features=cat_features,
        eval_set=(X_val, y_val),
        verbose=False
)

print('CatBoost model is fitted: ' + str(clf.is_fitted()))
print('CatBoost model parameters:')
print(clf.get_params())
```

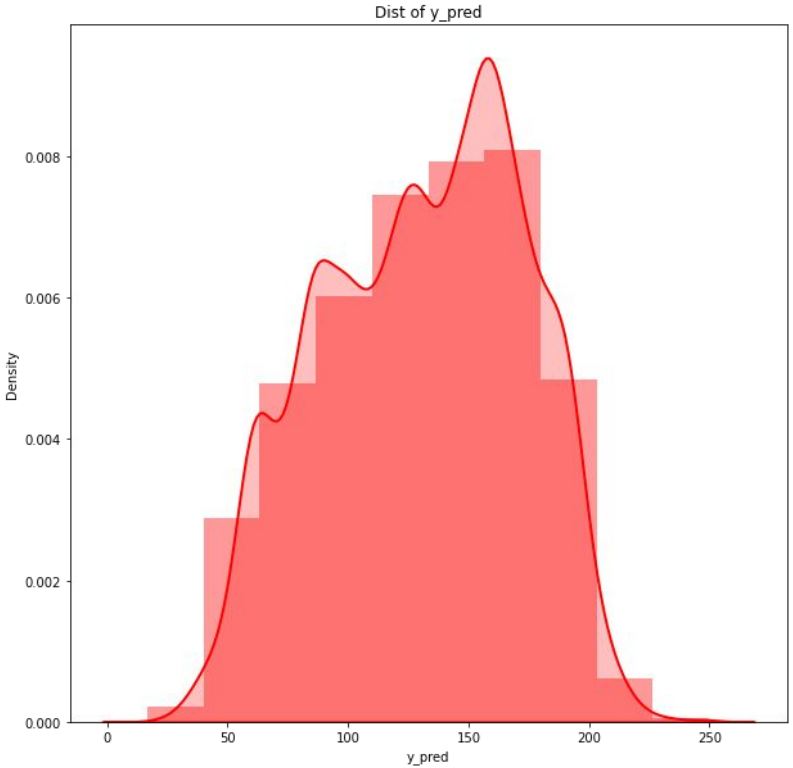
```
CatBoost model is fitted: True
CatBoost model parameters:
{'iterations': 5, 'learning_rate': 0.1, 'loss_function': 'RMSE'}
```

```
# Stdout of the training
from catboost import CatBoostRegressor
clf = CatBoostRegressor(
    iterations=10,
#     verbose=5,
)

clf.fit(
    X_train, y_train,
    cat_features=cat_features,
    eval_set=(X_val, y_val),
)
```

```
Learning rate set to 0.5
0:      learn: 145.5364664      test: 148.6019577      best: 148.6019577 (0)   total: 113ms    remaining: 1.02s
1:      learn: 143.8200433      test: 146.8313184      best: 146.8313184 (1)   total: 236ms    remaining: 943ms
2:      learn: 143.3409063      test: 146.3004773      best: 146.3004773 (2)   total: 328ms    remaining: 766ms
3:      learn: 143.1069057      test: 146.0582593      best: 146.0582593 (3)   total: 375ms    remaining: 563ms
4:      learn: 142.7957502      test: 145.6702745      best: 145.6702745 (4)   total: 471ms    remaining: 471ms
5:      learn: 142.6750708      test: 145.4848649      best: 145.4848649 (5)   total: 554ms    remaining: 369ms
6:      learn: 142.5517910      test: 145.3794689      best: 145.3794689 (6)   total: 671ms    remaining: 288ms
7:      learn: 142.3937752      test: 145.1993597      best: 145.1993597 (7)   total: 789ms    remaining: 197ms
8:      learn: 142.2264626      test: 145.0177022      best: 145.0177022 (8)   total: 853ms    remaining: 94.8ms
9:      learn: 142.0290909      test: 144.8759360      best: 144.8759360 (9)   total: 940ms    remaining: 0us

bestTest = 144.875936
bestIteration = 9

<catboost.core.CatBoostRegressor at 0x7f3fcd1c0790>
```

```
y_pred = clf.predict(X_val)
```

DISTRIBUTION OF PREDICTION




```
# Checking the distribution of prediction
import seaborn as sns
fig, ax = plt.subplots(1,1,figsize=(10, 10))

# plotting the original data(non-normal) and
# fitted data (normal)
sns.distplot(y_pred, hist = True, kde = True,
             kde_kws = {'shade': True, 'linewidth': 2},
             label = "Non-Normal", color ="red", ax = ax,bins=10)
ax.title.set_text('Dist of y_pred')
ax.set_xlabel('y_pred')
```

```
# score on Training data
print(clf.score(X_train,y_train))
```

```
0.09153999456372541
```

```
# score on Test data
print(clf.score(X_val,y_val))
```

```
0.09357765755734104
```

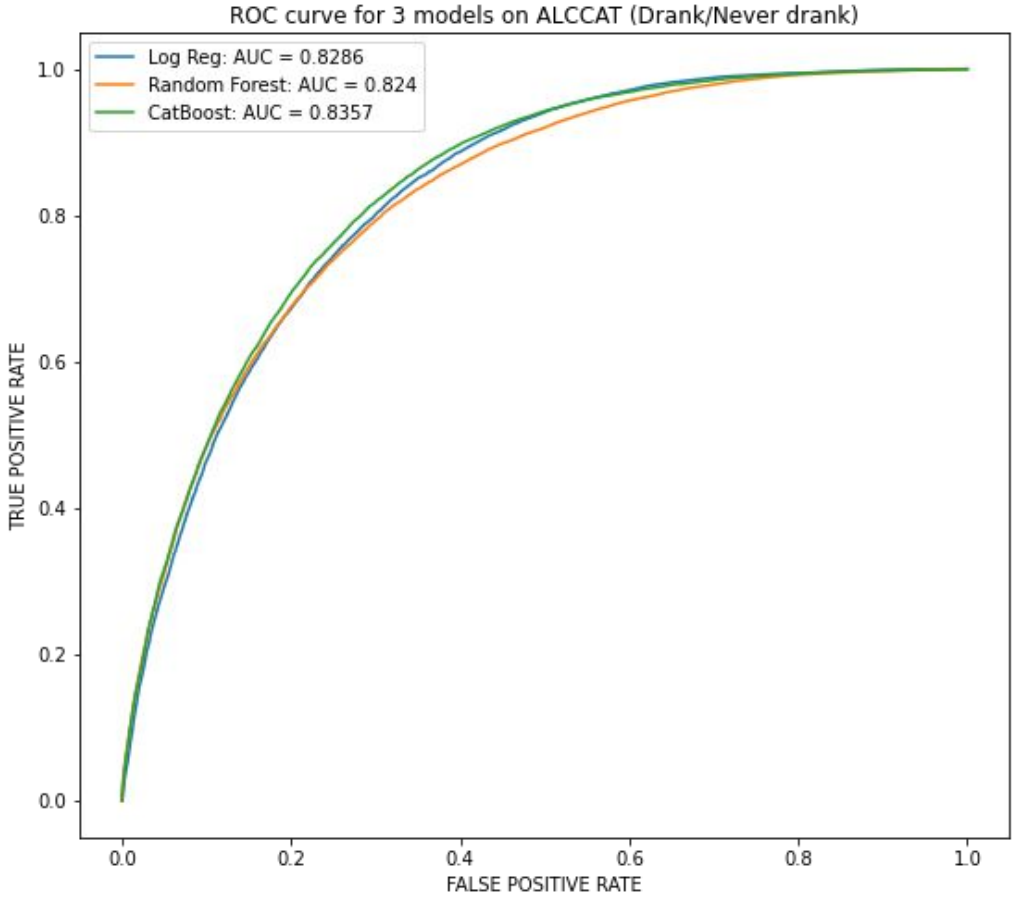
We observe from the graph that the distribution for the prediction using CatBoost Regression model does not have values which are negative.

MODEL EVALUATION METRICS

```
# Evaluation Metrics

import sklearn.metrics as metrics
mae = metrics.mean_absolute_error(y_val, y_pred)
mse = metrics.mean_squared_error(y_val, y_pred)
rmse = np.sqrt(mse) #mse**(0.5)
r2 = metrics.r2_score(y_val,y_pred)
mape = metrics.mean_absolute_percentage_error(y_val, y_pred)

print("Results of sklearn.metrics:")
print("MAE:",mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R-Squared:", r2)
print("MAPE:",mape)
```

```
Results of sklearn.metrics:
MAE: 118.36996281804136
MSE: 20989.036830604855
RMSE: 144.87593599561265
R-Squared: 0.09357765755734104
MAPE: 16.57605053667785
```

CLASSIFICATION MODEL SELECTION FOR ALCOHOL (ALCCAT)

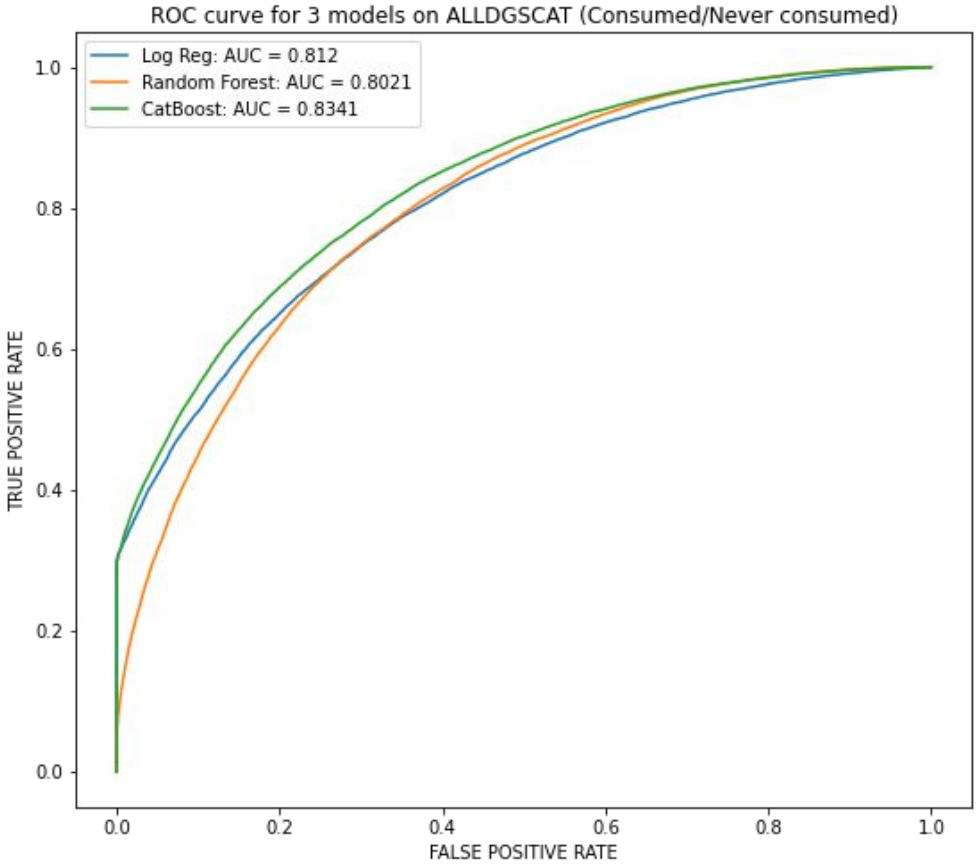


We see a close competition between the three models

We observe from the graph that CatBoost Model has the highest value for Area Under the Curve with a value of 0.8357

This makes **CatBoost Classifier** most suitable for predicting ALCCAT
(Drank/Never Drank Alcohol)

CLASSIFICATION MODEL SELECTION FOR ALLDRUGS (ALLDGSCAT)



We see a close competition between the three models

We observe from the graph that CatBoost Model has the highest value for Area Under the Curve with a value of 0.8341

This makes **CatBoost Classifier** most suitable for predicting ALLDGSCAT (Consumed/Never consumed Alldrugs)

REGRESSION MODEL SELECTION FOR ALCOHOL (IRALCFY)

Evaluation Metrics	Linear Regression	Random Forest	CatBoost
MAE	67.3325347936132	66.65218863624317	67.0744796879123
MSE	7585.800442891922	7501.849392723051	7524.596701757585
RMSE	87.09650074998376	**86.61321719416183**	86.74443326091644
R-Squared	0.0773372152438615	0.09366934372894598	0.08847903238038313
MAPE	6.050438707733569	**5.851723056694891**	5.963419228648019

From the table above, it is evident that **Random Forest Regressor** has the lowest values for Root Mean Square Error (RMSE) as well as the Mean Absolute Percentage Error (MAPE)

Therefore, this makes the Random Forest Regressor model most suitable for predicting the Alcohol consumption frequency (IRALCFY)

REGRESSION MODEL SELECTION FOR ALLDRUGS (ALLDGS)

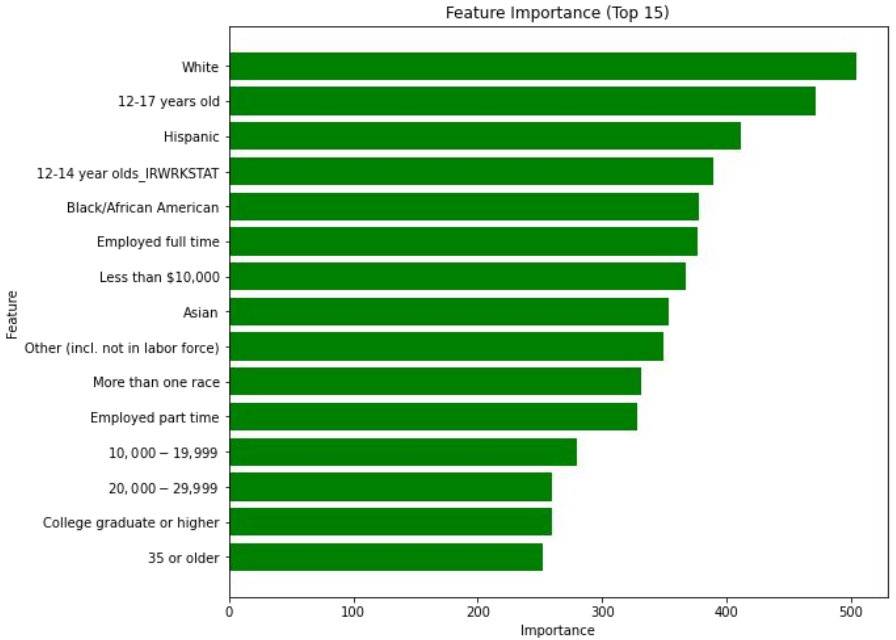
Evaluation Metrics	Linear Regression	Random Forest	CatBoost
MAE	116.80163455703884	117.2241321299863	118.36996281804136
MSE	20683.889513330552	20632.543682224215	20989.036830604855
RMSE	143.818946990063	**143.64032749274912**	144.87593599561265
R-Squared	0.0864875462504782	0.0887552559157	0.09357765755734104
MAPE	15.828576233565963	16.076025282912205	16.57605053667785

From the table above, it is evident that **Random Forest Regressor** has the lowest value for Root Mean Square Error (RMSE)

Therefore, this makes the Random Forest Regressor model most suitable for predicting the Alldrugs consumption frequency (ALLDGS)

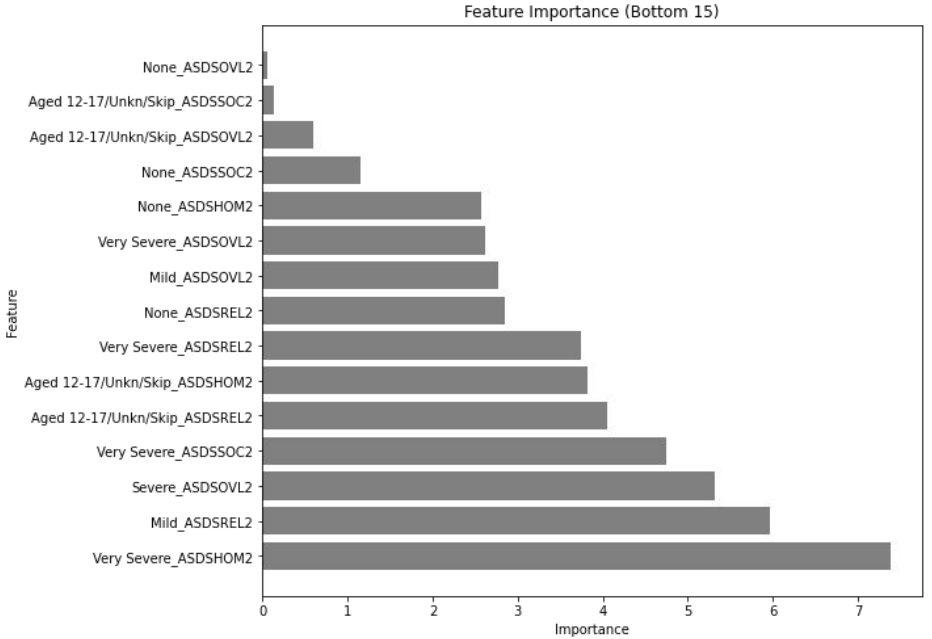
FEATURE IMPORTANCE FOR CATBOOST CLASSIFIER ON ALCOHOL (ALCCAT)

Top 15 Features



From the bargraph, it is evident that different categories of features like Race (NEWRACE2), Age (CATAGE), Employment (IRWRKSTAT), Annual household income (IRPINC3), Education (IREDUHIGHST2) have high predictability of ALCCAT (Drank/Never Drank Alcohol)

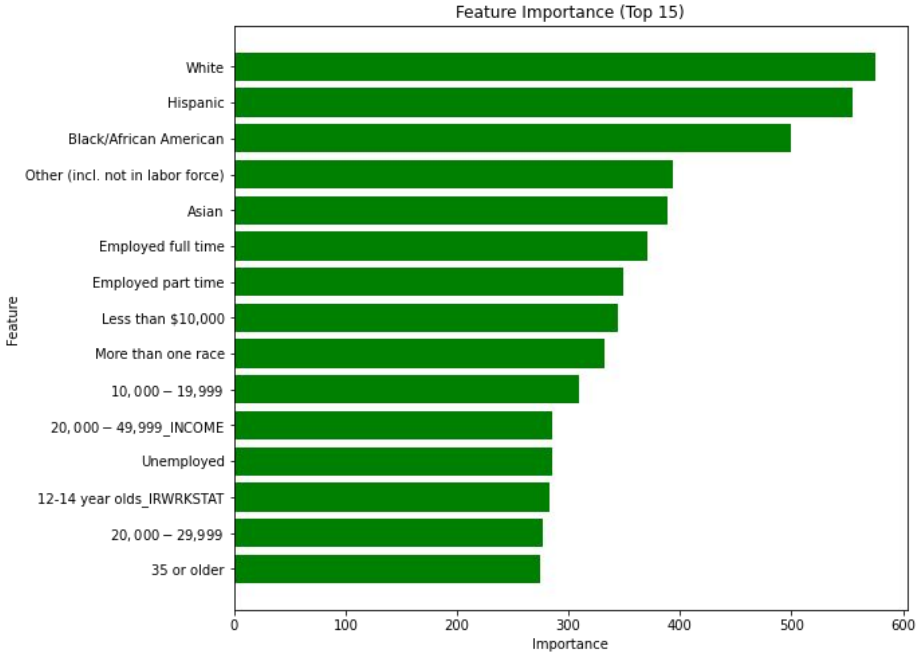
Bottom 15 Features



From the bargraph, it is evident that different categories of Depression features like ASDSOVL2, ASDSSOC2, ASDSREL2, ASDSHOM2 have low predictability of ALCCAT (Drank/Never Drank Alcohol)

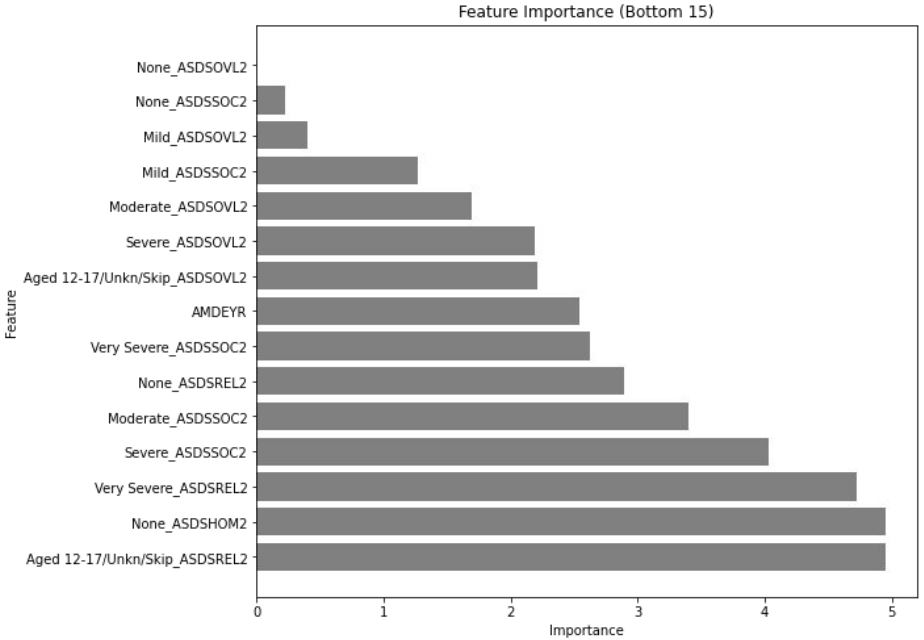
FEATURE IMPORTANCE FOR CATBOOST CLASSIFIER ON ALLDRUGS (ALLDGSCAT)

Top 15 Features



From the bargraph, it is evident that different categories of features like Race (NEWRACE2), Age (CATAGE), Employment (IRWRKSTAT), Annual household income (IRPINC3), Total Family Income (INCOME) have high predictability of ALLDGSCAT (Consumed/Never consumed Alldrugs)

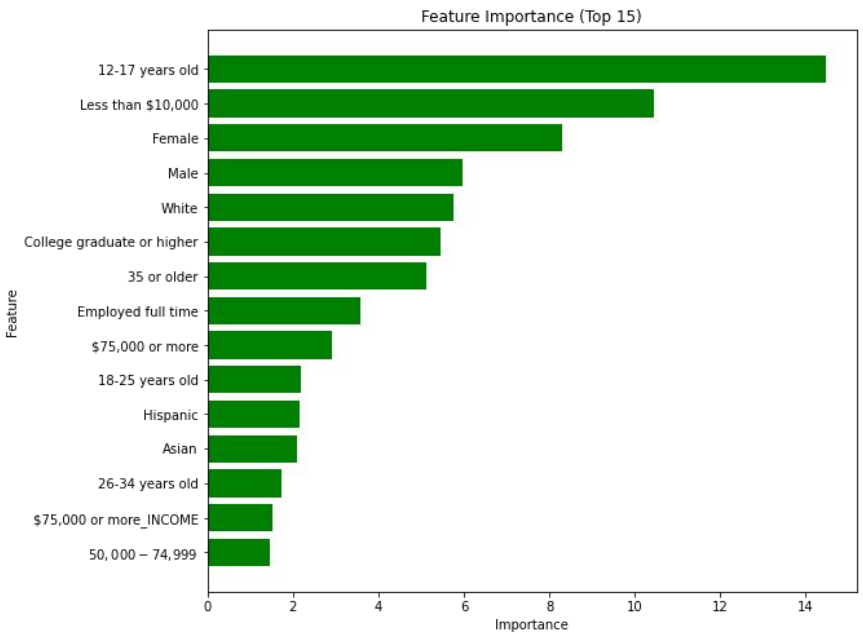
Bottom 15 Features



From the bargraph, it is evident that different categories of Depression features like ASDSOVL2, ASDSSOC2, ASDSREL2, ASDSHOM2, AMDEYR have low predictability of ALLDGSCAT (Consumed/Never consumed Alldrugs)

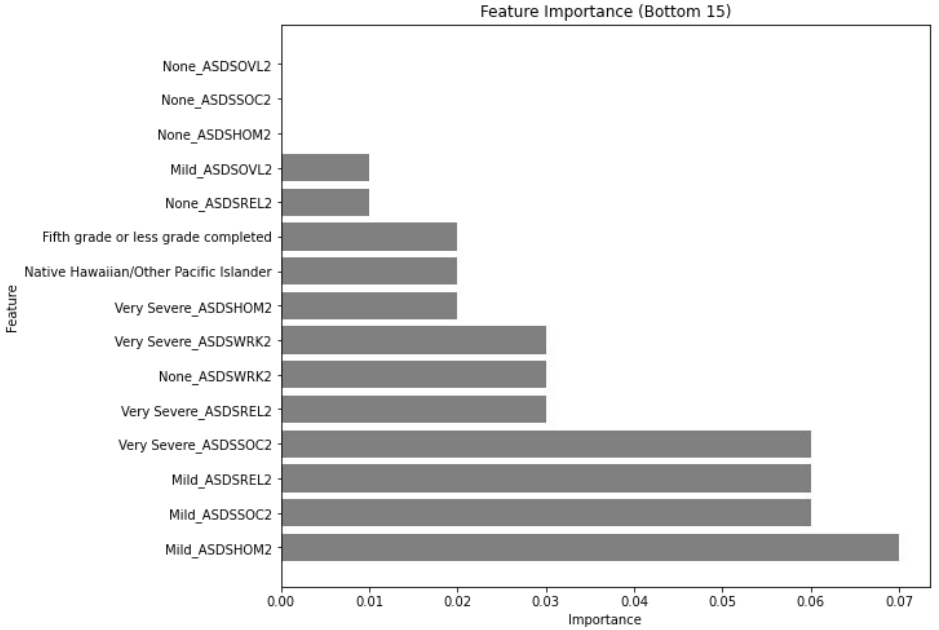
FEATURE IMPORTANCE FOR RANDOM FOREST REGRESSOR ON ALCOHOL (IRALCFY)

Top 15 Features



From the bargraph, it is evident that different categories of features like Gender (IRSEX), Race (NEWRACE2), Age (CATAGE), Education (IREDUHIGHST2), Employment (IRWRKSTAT), Annual household income (IRPINC3) have high predictability of Alcohol consumption frequency (IRALCFY)

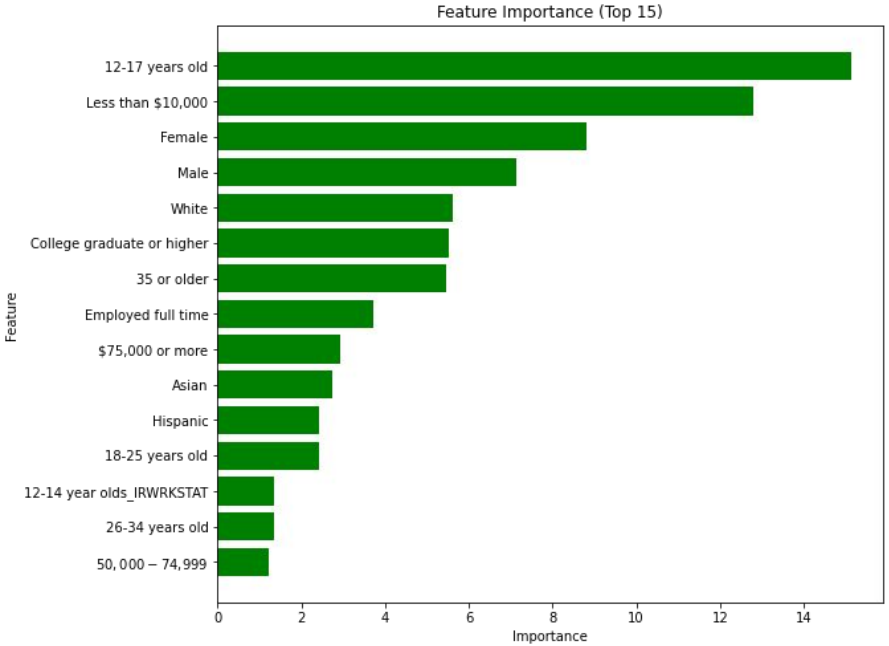
Bottom 15 Features



From the bargraph, it is evident that different categories of Depression features like ASDSOVL2, ASDSSOC2, ASDSREL2, ASDSHOM2, ‘Native Hawaiian’ race category, ‘Fifth grade or less’ education have low predictability of Alcohol consumption frequency (IRALCFY)

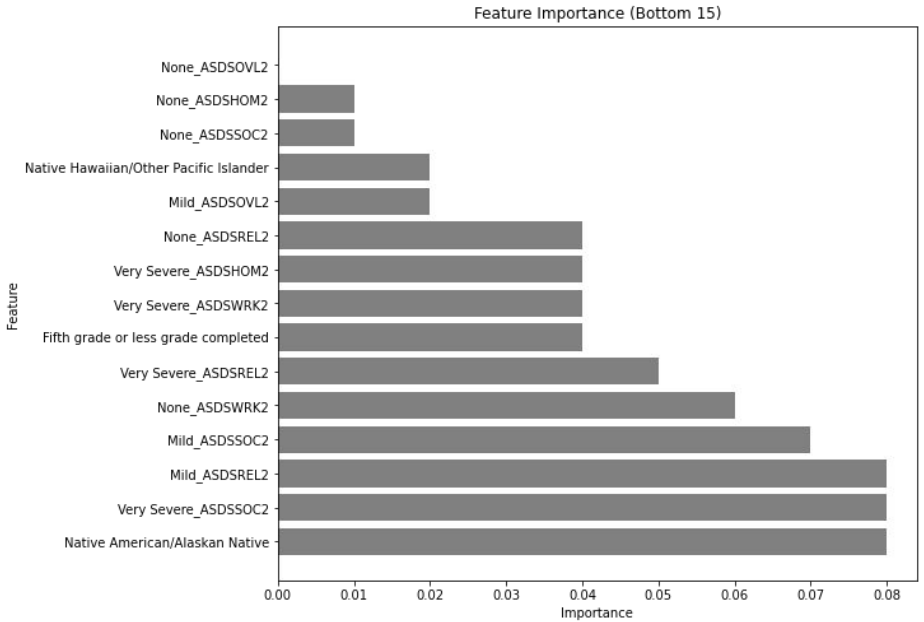
FEATURE IMPORTANCE FOR RANDOM FOREST REGRESSOR ON ALLDRUGS (ALLDGS)

Top 15 Features



From the bargraph, it is evident that different categories of features like Gender (IRSEX), Race (NEWRACE2), Age (CATAGE), Education (IREDUHIGHST2), Employment (IRWRKSTAT), Annual household income (IRPINC3), Total Family income (INCOME) have high predictability of Alldrugs consumption frequency (ALLDGS)

Bottom 15 Features



From the bargraph, it is evident that different categories of Depression features like ASDSOVL2, ASDSSOC2, ASDSREL2, ASDSHOM2, ‘Native Hawaiian’ & ‘Native American’ race categories, ‘Fifth grade or less’ education have low predictability of Alldrugs consumption frequency (ALLDGS)

PREDICTIONS

```
[117] # sample input data1
      data1 = [0, 0,  1,  0,  1,  0,  0,  0,  0,  0,
               1, 0,  0,  0,  0,  0,  1,  0,  0,  0,
               0, 0,  1,  0,  0,  0,  0,  0,  1,  0,
               0, 0,  0,  0,  0,  1,  0,  0,  0,  0,
               0, 0,  0,  0,  0,  1,  0,  0,  0,  0,
               1, 0,  0,  0,  0,  0,  0,  1,  0,  1,
               0, 0,  0,  0,  0,  0,  1,  0,  0,  0,
               0, 1,  0,  0,  1,  0,  0,  0,  0,  1]

      list1 = data1

      # Code to predict
      y_pred_alc = cat_tm_ca.predict([list1])
      y_pred_drug = cat_tm_cd.predict([list1])
      if y_pred_alc == 0:
        print('Does not consume Alcohol')
      else:
        print('Consumed Alcohol')

      if y_pred_drug == 0:
        print('Does not consume Drugs')
      else:
        print('Consumed Drugs')

      if y_pred_alc == 1:
        print('Avg Alcohol consumption per year:', round(rf_bestfit.predict([list1])[0]))

      if y_pred_drug == 1:
        print('Avg Alldrugs consumption per year:', round(rfa_bestfit.predict([list1])[0]))
```

```
Consumed Alcohol
Does not consume Drugs
Avg Alcohol consumption per year: 108
```

```
[118] # sample input data2
      data2 = [0, 0,  1,  0,  1,  0,  0,  0,  0,  0,  1,  0,  0,  0,  0,  0,  1,  0,  0,
               0, 0,  0,  1,  0,  0,  0,  0,  0,  1,  0,  0,  0,  0,  0,  0,  1,  0,  0,
               1, 0,  0,  0,  0,  0,  0,  0,  0,  1,  0,  0,  0,  0,  0,  0,  1,  0,  0,
               0, 0,  0,  0,  1,  0,  0,  0,  0,  0,  0,  0,  1,  1,  0,  0,  0,  1,  0,
               0, 0,  0,  1]

      list2 = data2

      # Code to predict
      y_pred_alc = cat_tm_ca.predict([list2])
      y_pred_drug = cat_tm_cd.predict([list2])
      if y_pred_alc == 0:
        print('Does not consume Alcohol')
      else:
        print('Consumed Alcohol')

      if y_pred_drug == 0:
        print('Does not consume Drugs')
      else:
        print('Consumed Drugs')

      if y_pred_alc == 1:
        print('\n')
        print('Avg Alcohol consumption per year:', round(rf_bestfit.predict([list1])[0]))

      if y_pred_drug == 1:
        print('\n')
        print('Avg Alldrugs consumption per year:', round(rfa_bestfit.predict([list1])[0]))
```

```
Does not consume Alcohol
Does not consume Drugs
```

PREDICTIONS

```
# sample input data3
data3 = [0, 0, 1, 0, 1, 0, 0, 0, 0, 0,
         1, 0, 0, 0, 0, 0, 1, 0, 0, 0,
         0, 0, 1, 0, 0, 0, 0, 0, 1, 0,
         0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
         0, 1, 0, 0, 0, 0, 0, 0, 1, 0,
         0, 0, 0, 0, 0, 0, 0, 1, 0, 1,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
         0, 0, 1, 0, 0, 1, 0, 1, 0, 0]

list1 = data3

# Code to predict
y_pred_alc = cat_tm_ca.predict([list1])
y_pred_drug = cat_tm_cd.predict([list1])
if y_pred_alc == 0:
  print('Does not consume Alcohol')
else:
  print('Consumed Alcohol')

if y_pred_drug == 0:
  print('Does not consume Drugs')
else:
  print('Consumed Drugs')

if y_pred_alc == 1:
  print('Avg Alcohol consumption per year:', round(rf_bestfit.predict([list1])[0]))

if y_pred_drug == 1:
  print('Avg Alldrugs consumption per year:', round(rfa_bestfit.predict([list1])[0]))
```

```
Consumed Alcohol
Consumed Drugs
Avg Alcohol consumption per year: 68
Avg Alldrugs consumption per year: 142
```

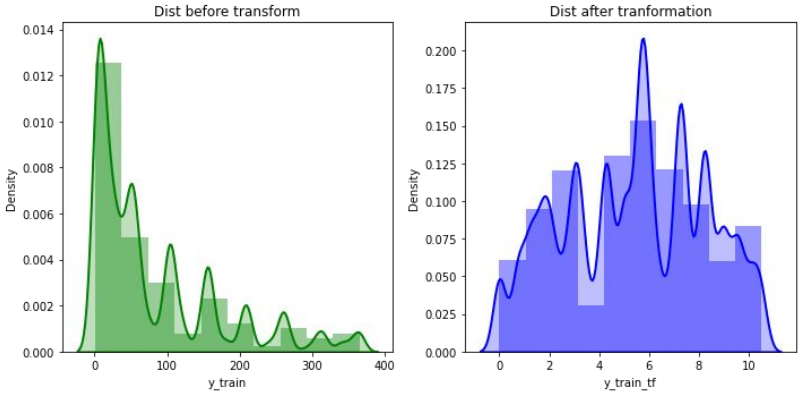
Hypothesis Results

We can reject the null hypothesis:

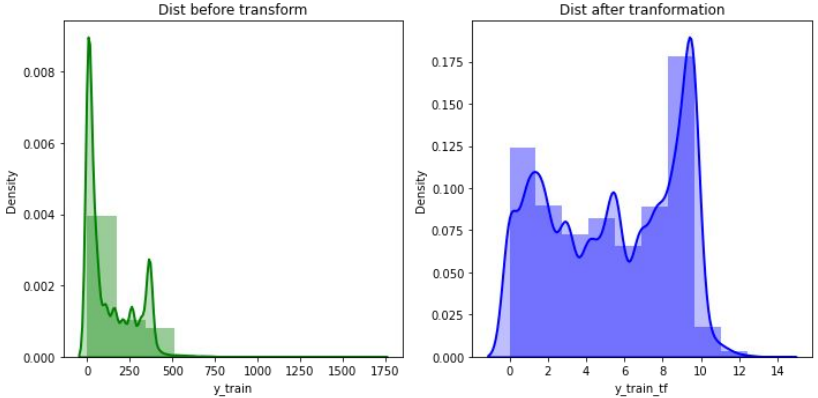
- The examined factors have a relation to substance use in the past year.

Limitations

- Most of our independent variables were categorical, this created scaling issues with our quantitative dependent variables.
- The frequency of drug use (IRALCFY) was skewed, a large amount of our sample had not consumed any substances with the exception of alcohol.
- We wanted to use Box-Cox transform to get the distribution of the target variable close to Gaussian, but the issue we were facing was that the scale of the transformed variable was different when compared to the original one.



IRALCFY before & after transform



ALLDGS before & after transform

- For Regression, before removing rows of data having ALCCAT=0 for Alcohol freq prediction & ALLDGSCAT=0 for Alldrugs freq prediction, we were thinking of different ways to deal with Zero Inflation (i.e Data with a lot of zeros and fewer non-zero values). We wanted to explore the possibility of applying ‘Hurdle Models’ to this problem. Ways to deal with Zero-Inflation is mentioned here. More info on ‘Hurdle models’ is explained here

References

Center for Behavioral Health Statistics and Quality. (2021). 2020 National Survey on substance Use and Health Public Use File Codebook, Substance Abuse and Mental Health Services Administration, Rockville, MD
https://www.samhsa.gov/data/data-we-collect/nsduh-national-survey-substance-use-and-health
Center for Disease Control and Prevention. (2022). High Risk Substance Use in Youth. https://www.cdc.gov/healthyyouth/substance-use/index.htm

NIDA. 2020, May 25. What are risk factors and protective factors?. Retrieved from
https://nida.nih.gov/publications/preventing-substance-use-among-children-adolescents/chapter-1-risk-factors-protective-factors/what-are-risk-factors, 2022, September 27

Rosner, B., Neicun, J., Yang, J. C., & Roman-Urrestarazu, A. (2021). Substance use among sexual minorities in the US – Linked to inequalities and unmet need for mental health treatment? Results from the National Survey on substance Use and Health (NSDUH). Journal of Psychiatric Research, 135, 107–118. https://doi.org/10.1016/j.jpsychires.2020.12.023

Lin, L. A., Ilgen, M. A., Jannausch, M., & Bohnert, K. M. (2016). Comparing adults who use cannabis medically with those who use recreationally: Results from a national sample. Addictive Behaviors, 61, 99–103.
https://doi.org/10.1016/j.addbeh.2016.05.015

Waddell, J. T. (2021). Between- and within-group effects of alcohol and cannabis co-use on AUD/CUD in the NSDUH 2002–2019. substance and Alcohol Dependence, 225, 108768. https://doi.org/10.1016/j.substancealcdep.2021.108768

Volkow, N. D., Poznyak, V., Saxena, S., & Gerra, G. (2017). substance use disorders: impact of a public health rather than a criminal justice approach. World Psychiatry, 16(2), 213–214. https://doi.org/10.1002/wps.20428
NIDA. 2020, May 25. What are risk factors and protective factors?. Retrieved from
https://nida.nih.gov/publications/preventing-drug-use-among-children-adolescents/chapter-1-risk-factors-protective-factors/what-are-risk-factors, 2022, September 27

Appendix

Code:

https://colab.research.google.com/drive/1s9qCyuMkCxiiLFYdN29G7Yz4VEob5a7K?usp=sharing

CODE APPENDIX

#Importing Data

```
import warnings
warnings.filterwarnings('ignore')

import pandas as pd
import numpy as np

# Read TSV file into DataFrame (Import only the variables we are going
to analyze, refer to 'variables and analysis' file)
link0 = 'https://www.datafiles.samhsa.gov/sites/default/files/field-
uploads-protected/studies/NSDUH-2020/NSDUH-2020-datasets/NSDUH-2020-
DS0001/NSDUH-2020-DS0001-bundles-with-study-info/NSDUH-2020-DS0001-
bndl-data-tsv_v1.zip'
link1 = 'https://www.datafiles.samhsa.gov/sites/default/files/field-
uploads-protected/studies/NSDUH-2019/NSDUH-2019-datasets/NSDUH-2019-
DS0001/NSDUH-2019-DS0001-bundles-with-study-info/NSDUH-2019-DS0001-
bndl-data-tsv.zip'
link2 = 'https://www.datafiles.samhsa.gov/sites/default/files/field-
uploads-protected/studies/NSDUH-2018/NSDUH-2018-datasets/NSDUH-2018-
DS0001/NSDUH-2018-DS0001-bundles-with-study-info/NSDUH-2018-DS0001-
bndl-data-tsv.zip'
link3 = 'https://www.datafiles.samhsa.gov/sites/default/files/field-
uploads-protected/studies/NSDUH-2017/NSDUH-2017-datasets/NSDUH-2017-
DS0001/NSDUH-2017-DS0001-bundles-with-study-info/NSDUH-2017-DS0001-
bndl-data-tsv.zip'
link4 = 'https://www.datafiles.samhsa.gov/sites/default/files/field-
uploads-protected/studies/NSDUH-2016/NSDUH-2016-datasets/NSDUH-2016-
DS0001/NSDUH-2016-DS0001-bundles-with-study-info/NSDUH-2016-DS0001-
bndl-data-tsv.zip'
link5 = 'https://www.datafiles.samhsa.gov/sites/default/files/field-
uploads-protected/studies/NSDUH-2015/NSDUH-2015-datasets/NSDUH-2015-
DS0001/NSDUH-2015-DS0001-bundles-with-study-info/NSDUH-2015-DS0001-
bndl-data-tsv.zip'

df0 = pd.read_table(link0, usecols =['QUESTID2','IRALCFY', 'IRMJFY',
'IRCOCFY','IRCRKFY', 'IRHERFY', 'IRHALLUCYFQ', 'IRINHALYFQ',
'IRMETHAMYFQ', 'ALCFLAG', 'MRJFLAG', 'COCYR', 'CRKFLAG', 'HERFLAG',
'HALLUCFLAG', 'INHALFLAG', 'METHAMFLAG', 'NEWRACE2', 'CATAGE',
'IRSEX', 'IRWRKSTAT', 'IREDUHIGHST2', 'PDEN10', 'COUTYP4', 'IRPRVHLT',
'IRPINC3', 'INCOME', 'IRFAMSOC', 'AMHTXND2', 'AMDEYR', 'ASDSHOM2',
'ASDSWRK2', 'ASDSREL2', 'ASDSSOC2', 'ASDSOVL2'])

df1 = pd.read_table(link1, usecols =['QUESTID2','IRALCFY', 'IRMJFY',
'IRCOCFY','IRCRKFY', 'IRHERFY', 'IRHALLUCYFQ', 'IRINHALYFQ',
'IRMETHAMYFQ', 'ALCFLAG', 'MRJFLAG', 'COCYR', 'CRKFLAG', 'HERFLAG',
'HALLUCFLAG', 'INHALFLAG', 'METHAMFLAG', 'NEWRACE2', 'CATAGE',
'IRSEX', 'IRWRKSTAT', 'IREDUHIGHST2', 'PDEN10', 'COUTYP4', 'IRPRVHLT',
'IRPINC3', 'INCOME', 'IRFAMSOC', 'AMHTXND2', 'AMDEYR', 'ASDSHOM2',
'ASDSWRK2', 'ASDSREL2', 'ASDSSOC2', 'ASDSOVL2'])
```

```
df2 = pd.read_table(link2, usecols =['QUESTID2','IRALCFY', 'IRMJFY',
'IRCOCFY','IRCRKFY', 'IRHERFY', 'IRHALLUCYFQ', 'IRINHALYFQ',
'IRMETHAMYFQ', 'ALCFLAG', 'MRJFLAG', 'COCYR', 'CRKFLAG', 'HERFLAG',
'HALLUCFLAG', 'INHALFLAG', 'METHAMFLAG', 'NEWRACE2', 'CATAGE',
'IRSEX', 'IRWRKSTAT', 'IREDUHIGHST2', 'PDEN10', 'COUTYP4', 'IRPRVHLT',
'IRPINC3', 'INCOME', 'IRFAMSOC', 'AMHTXND2', 'AMDEYR', 'ASDSHOM2',
'ASDSWRK2', 'ASDSREL2', 'ASDSSOC2', 'ASDSOVL2'])

df3 = pd.read_table(link3, usecols =['QUESTID2','IRALCFY', 'IRMJFY',
'IRCOCFY','IRCRKFY', 'IRHERFY', 'IRHALLUCYFQ', 'IRINHALYFQ',
'IRMETHAMYFQ', 'ALCFLAG', 'MRJFLAG', 'COCYR', 'CRKFLAG', 'HERFLAG',
'HALLUCFLAG', 'INHALFLAG', 'METHAMFLAG', 'NEWRACE2', 'CATAGE',
'IRSEX', 'IRWRKSTAT', 'IREDUHIGHST2', 'PDEN10', 'COUTYP4', 'IRPRVHLT',
'IRPINC3', 'INCOME', 'IRFAMSOC', 'AMHTXND2', 'AMDEYR', 'ASDSHOM2',
'ASDSWRK2', 'ASDSREL2', 'ASDSSOC2', 'ASDSOVL2'])

df4 = pd.read_table(link4, usecols =['QUESTID2','IRALCFY', 'IRMJFY',
'IRCOCFY','IRCRKFY', 'IRHERFY', 'IRHALLUCYFQ', 'IRINHALYFQ',
'IRMETHAMYFQ', 'ALCFLAG', 'MRJFLAG', 'COCYR', 'CRKFLAG', 'HERFLAG',
'HALLUCFLAG', 'INHALFLAG', 'METHAMFLAG', 'NEWRACE2', 'CATAGE',
'IRSEX', 'IRWRKSTAT', 'IREDUHIGHST2', 'PDEN10', 'COUTYP4', 'IRPRVHLT',
'IRPINC3', 'INCOME', 'IRFAMSOC', 'AMHTXND2', 'AMDEYR', 'ASDSHOM2',
'ASDSWRK2', 'ASDSREL2', 'ASDSSOC2', 'ASDSOVL2'])

df5 = pd.read_table(link5, usecols =['QUESTID2','IRALCFY', 'IRMJFY',
'IRCOCFY','IRCRKFY', 'IRHERFY', 'IRHALLUCYFQ', 'IRINHALYFQ',
'IRMETHAMYFQ', 'ALCFLAG', 'MRJFLAG', 'COCYR', 'CRKFLAG', 'HERFLAG',
'HALLUCFLAG', 'INHALFLAG', 'METHAMFLAG', 'NEWRACE2', 'CATAGE',
'IRSEX', 'IRWRKSTAT', 'IREDUHIGHST2', 'PDEN10', 'COUTYP4', 'IRPRVHLT',
'IRPINC3', 'INCOME', 'IRFAMSOC', 'AMHTXND2', 'AMDEYR', 'ASDSHOM2',
'ASDSWRK2', 'ASDSREL2', 'ASDSSOC2', 'ASDSOVL2'])


df0['Year'] = 2020
df1['Year'] = 2019
df2['Year'] = 2018
df3['Year'] = 2017
df4['Year'] = 2016
df5['Year'] = 2015

df = pd.concat([df0,df1,df2,df3,df4,df5])
```

#Recoding Data

```
def alldrugsBclean(var):
    if var>=1:
      return 1
    elif var==0:
      return 0
```

```
    else:
      return var



df["ALLDGSFLAG"] = df['COCYR']+df['CRKFLAG']+df['HERFLAG']
+df['HALLUCFLAG']+ df['INHALFLAG']+df['METHAMFLAG']+df['MRJFLAG']

#recode alldrugsB into binary
df['ALLDGSFLAG']=df['ALLDGSFLAG'].apply(alldrugsBclean)



#function that recodes frequency variables
def freqvarclean(var):
    if var== 991:
      return 0
    elif var==993:
      return 0
    else:
      return var

#function that recodes frequency variables
def binaryvarclean(var):
    if var== 0:
      return 'Never used'
    elif var==1:
      return 'Ever used'
    else:
      return None

#functions that recode independent variables (that need to be recoded)
def independentvarclean(var):
    if var== 99:
      return None
    elif var== '.':
      return None
    else:
      return var


#function that recodes race variable
def racevarclean(var):
    if var== 1:
      return 'White'
    elif var== 2:
      return 'Black/African American'
    elif var== 3:
      return 'Native American/Alaskan Native'
```

```
    elif var== 4:
      return 'Native Hawaiian/Other Pacific Islander'
    elif var== 5:
      return 'Asian'
    elif var== 6:
      return 'More than one race'
    elif var==7:
      return 'Hispanic'
    else:
      return 'Null'

#function that recodes age variable
def agevarclean(var):
    if var== 1:
      return '12-17 years old'
    elif var== 2:
      return '18-25 years old'
    elif var== 3:
      return '26-34 years old'
    elif var== 4:
      return '35 or older'
    else:
      return None


#function that recodes age variable
def gendervarclean(var):
    if var== 1:
      return 'Male'
    elif var== 2:
      return 'Female'
    else:
      return 'null'

def deppclean(var):
    if var== 1.:
      return 'None'
    elif var== 2.:
      return 'Mild'
    elif var== 3.:
      return 'Moderate'
    elif var== 4.:
      return 'Severe'
    elif var== 5.:
      return 'Very Severe'
    else:
      return 'Aged 12-17/Unkn/Skip'

def irpinc(var):
     if var== 1:
```

```
        return 'Less than $10,000'
      elif var== 2:
        return '$10,000-$19,999'
      elif var== 3:
        return '$20,000-$29,999'
      elif var== 4:
        return '$30,000-$39,999'
      elif var== 5:
        return '$40,000-$49,999'
      elif var== 6:
        return '$50,000-$74,999'
      elif var== 7:
        return '$75,000 or more'

def ireduhigh(var):
      if var== 1:
        return 'Fifth grade or less grade completed'
      elif var== 2:
        return 'Sixth grade completed'
      elif var== 3:
        return 'Seventh grade completed'
      elif var== 4:
        return 'Eighth grade completed'
      elif var== 5:
        return 'Ninth grade completed'
      elif var== 6:
        return 'Tenth grade completed'
      elif var== 7:
        return 'Eleventh or Twelfth grade completed, no diploma'
      elif var== 8:
        return 'High school diploma/GED'
      elif var== 9:
        return 'Some college credit, but no degree'
      elif var== 10:
        return "Associate's degree (AA, AS)"
      elif var== 11:
        return 'College graduate or higher'

def pden(var):
      if var== 1:
        return 'Segment in a CBSA with 1 million or more persons'
      elif var== 2:
        return 'Segm in a CBSA with fewer  than 1 million persons'
      elif var== 3:
        return 'Segment not in CBSA'

def coutyp(var):
      if var== 1:
        return 'Large Metro'
      elif var== 2:
```

```
        return 'Small Metro'
      elif var== 3:
        return 'Nonmetro'

def wrkstatclean(var):
    if var== 1:
      return 'Employed full time'
    elif var== 2:
      return 'Employed part time'
    elif var== 3:
      return 'Unemployed'
    elif var== 4:
      return 'Other (incl. not in labor force)'
    else:
      return '12-14 year olds'

def incomex(var):
     if var== 1:
       return 'Less than $20,000'
     elif var== 2:
       return '$20,000-$49,999'
     elif var== 3:
       return '$50,000-$74,999'
     elif var== 4:
       return '$75,000 or more'


#apply frequency variable coding
df['IRALCFY'] = df['IRALCFY'].apply(freqvarclean)
df['IRMJFY'] = df['IRMJFY'].apply(freqvarclean)
df['IRCOCFY'] = df['IRCOCFY'].apply(freqvarclean)
df['IRHERFY'] = df['IRHERFY'].apply(freqvarclean)
df['IRCRKFY'] = df['IRCRKFY'].apply(freqvarclean)
df['IRHALLUCYFQ'] = df['IRHALLUCYFQ'].apply(freqvarclean)
df['IRINHALYFQ'] = df['IRINHALYFQ'].apply(freqvarclean)
df['IRMETHAMYFQ'] = df['IRMETHAMYFQ'].apply(freqvarclean)

#apply binary variable coding
df['ALCFLAG'] = df['ALCFLAG'].apply(binaryvarclean)
df['MRJFLAG'] = df['MRJFLAG'].apply(binaryvarclean)
df['COCYR'] = df['COCYR'].apply(binaryvarclean)
df['CRKFLAG'] = df['CRKFLAG'].apply(binaryvarclean)
df['HERFLAG'] = df['HERFLAG'].apply(binaryvarclean)
df['HALLUCFLAG'] = df['HALLUCFLAG'].apply(binaryvarclean)
df['INHALFLAG'] = df['INHALFLAG'].apply(binaryvarclean)
df['METHAMFLAG'] = df['METHAMFLAG'].apply(binaryvarclean)
df['ALLDGSFLAG'] = df['ALLDGSFLAG'].apply(binaryvarclean)

#apply independent variables recoding
df['IRWRKSTAT'] = df['IRWRKSTAT'].apply(independentvarclean)
```

```
df['AMHTXND2'] = df['AMHTXND2'].apply(independentvarclean)
df['AMDEYR'] = df['AMDEYR'].apply(independentvarclean)
df['ASDSHOM2'] = df['ASDSHOM2'].apply(independentvarclean)
df['ASDSWRK2'] = df['ASDSWRK2'].apply(independentvarclean)
df['ASDSREL2'] = df['ASDSREL2'].apply(independentvarclean)
df['ASDSSOC2'] = df['ASDSSOC2'].apply(independentvarclean)
df['ASDSOVL2'] = df['ASDSOVL2'].apply(independentvarclean)

#apply recoding
df['NEWRACE2'] = df['NEWRACE2'].apply(racevarclean)
df['CATAGE'] = df['CATAGE'].apply(agevarclean)
df['IRSEX'] = df['IRSEX'].apply(gendervarclean)
df['IRWRKSTAT'] = df['IRWRKSTAT'].apply(wrkstatclean)
df['ASDSHOM2'] = df['ASDSHOM2'].apply(deppclean)
df['ASDSWRK2'] = df['ASDSWRK2'].apply(deppclean)
df['ASDSREL2'] = df['ASDSREL2'].apply(deppclean)
df['ASDSSOC2'] = df['ASDSSOC2'].apply(deppclean)
df['ASDSOVL2'] = df['ASDSOVL2'].apply(deppclean)
df['IRPINC3'] = df['IRPINC3'].apply(irpinc)
df['INCOME'] = df['INCOME'].apply(incomex)
df['COUTYP4'] = df['COUTYP4'].apply(coutyp)
df['PDEN10'] = df['PDEN10'].apply(pden)
df['IREDUHIGHST2'] = df['IREDUHIGHST2'].apply(ireduhigh)

#Create ALLDGS
df["ALLDGS"] = df['IRMETHAMYFQ']+df['IRCOCFY']+df['IRCRKFY']
+df['IRHERFY']+ df['IRHALLUCYFQ']+df['IRINHALYFQ']+df['IRMJFY']

# Converting ALCFLAG to METHAMFLAG to binary
df.iloc[:,9:17] = df.iloc[:,9:17].replace(['Never used', 'Ever used'],
[0,1])

# Change ALLDGSFLAG to binary
df['ALLDGSFLAG'] = df['ALLDGSFLAG'].replace(['Ever used', 'Never
used'],[1,0])

# Converting IRFAMSOC & IRPRVHLT to binary
df['IRFAMSOC'] = df['IRFAMSOC'].replace([2,1],[0,1])
df['IRPRVHLT'] = df['IRPRVHLT'].replace([1,2],[1,0])

# Imputing highest mode in AMHTXND2 & convert to int datatype
df['AMHTXND2'] = df['AMHTXND2'].replace([1.,2.],[1,0])
df['AMHTXND2'] = df['AMHTXND2'].fillna(0)
df['AMHTXND2'] = df['AMHTXND2'].astype(int)

# Imputing highest mode in AMDEYR & convert to int datatype
df['AMDEYR'] = df['AMDEYR'].replace([1.,2.],[1,0])
df['AMDEYR'] = df['AMDEYR'].fillna(0)
df['AMDEYR'] = df['AMDEYR'].astype(int)
```

```
df[['ASDSHOM2','ASDSWRK2','ASDSREL2','ASDSSOC2','ASDSOVL2','IRSEX','IR
EDUHIGHST2','CATAGE','NEWRACE2','IRWRKSTAT','IRPINC3','INCOME','PDEN10
','COUTYP4']] =
df[['ASDSHOM2','ASDSWRK2','ASDSREL2','ASDSSOC2','ASDSOVL2','IRSEX','IR
EDUHIGHST2','CATAGE','NEWRACE2','IRWRKSTAT','IRPINC3','INCOME','PDEN10
','COUTYP4']].astype('category')


# Creating ALCCAT and ALLDGSCAT
def mickey(var):
  if var != 0:
    return 1
  else:
    return 0

df['ALCCAT'] = df['IRALCFY']
df['ALLDGSCAT'] = df['ALLDGS']
df['ALCCAT'] = df['ALCCAT'].apply(mickey)
df['ALLDGSCAT'] = df['ALLDGSCAT'].apply(mickey)


# Forming dv
dv = df.copy()

# Converting IRFAMSOC & IRPRVHLT to categorical (for visualization)
dv['IRFAMSOC'] = dv['IRFAMSOC'].replace([0,1],['No','Yes'])
dv['IRPRVHLT'] = dv['IRPRVHLT'].replace([1,0],['Yes','No'])

# Converting AMHTXND2 & AMDEYR to categorical (for visualization)
dv['AMHTXND2'] = dv['AMHTXND2'].replace([1,0],['Yes','No'])
dv['AMDEYR'] = dv['AMDEYR'].replace([1,0],['Yes','No'])

# Convert ALCFLAG to categorical  (for visualization)
dv['ALCFLAG'] = dv['ALCFLAG'].replace([1,0],['Ever used','Never
used'])

# Convert ALLDGSFLAG to categorical (for visualization)
dv['ALLDGSFLAG'] = dv['ALLDGSFLAG'].replace([1,0],['Ever used','Never
used'])

# Convert ALCCAT to categorical (for visualization)
dv['ALCCAT'] = dv['ALCCAT'].replace([1,0],['Ever used','Never used'])

# Convert ALLDGSCAT to categorical (for visualization)
dv['ALLDGSCAT'] = dv['ALLDGSCAT'].replace([1,0],['Ever used','Never
used'])

# Use dv for visualization instead of df
```

```
# Forming dv
dv = df.copy()


dv[['IRFAMSOC','IRPRVHLT','AMHTXND2','AMDEYR','ALCFLAG','ALLDGSFLAG','
ALCCAT','ALLDGSCAT','Year']] =
dv[['IRFAMSOC','IRPRVHLT','AMHTXND2','AMDEYR','ALCFLAG','ALLDGSFLAG','
ALCCAT','ALLDGSCAT','Year']].astype('category')
```

#Descriptive and Exploratory stats

```
#breakdown of gender
df['IRSEX'].value_counts(normalize=True)*100
```

```
Female    52.548145
Male      47.451855
Name: IRSEX, dtype: float64
```

```
pd.crosstab(df.CATAGE, df.ALCCAT ,normalize='index')\
  .round(4)*100
```

```
ALCCAT               0      1
CATAGE
12-17 years old  77.87  22.13
18-25 years old  26.43  73.57
26-34 years old  20.92  79.08
35 or older      30.64  69.36
```

```
#breakdown of race
df['NEWRACE2'].value_counts(normalize=True)*100
```

```
White                                          64.943909
Hispanic                                       14.884626
Black/African American                          9.196486
Asian                                           5.645578
More than one race                              4.040373
Native American/Alaskan Native                  0.884687
Native Hawaiian/Other Pacific Islander          0.404341
Name: NEWRACE2, dtype: float64
```

```
#breakdown of age
df['CATAGE'].value_counts(normalize=True)*100
```

```
35 or older        41.081689
18-25 years old    24.157115
12-17 years old    17.398839
26-34 years old    17.362357
Name: CATAGE, dtype: float64
```

```
#Proportion of respondents who have used alcohol
df['ALCFLAG'].value_counts(normalize=True)*100
```

```
Ever used     74.520415
Never used    25.479585
Name: ALCFLAG, dtype: float64
```

```
#proportion of respondents who haveused any drugs except alcohol
df['alldrugsB'].value_counts(normalize=True)*100
```

```
Never used    51.980665
Ever used     48.019335
Name: alldrugsB, dtype: float64
```

```
#Proportion of respondents who have used marihuana
df['MRJFLAG'].value_counts(normalize=True)*100
```

```
Never used    54.877938
Ever used     45.122062
Name: MRJFLAG, dtype: float64
```

```
#Proportion of respondents who have used cocaine
df['COCYR'].value_counts(normalize=True)*100
```

```
Never used    97.814125
Ever used      2.185875
Name: COCYR, dtype: float64
```

```
#Proportion of respondents who have used crack
df['CRKFLAG'].value_counts(normalize=True)*100
```

```
Never used    97.525309
Ever used      2.474691
Name: CRKFLAG, dtype: float64
```

```
#Proportion of respondents who have used heroin
df['HERFLAG'].value_counts(normalize=True)*100
```

```
Never used    98.242787
Ever used      1.757213
Name: HERFLAG, dtype: float64
```

```
#Proportion of respondents who have used hallucinogens
df['HALLUCFLAG'].value_counts(normalize=True)*100
```

```
Never used    84.373575
Ever used     15.626425
Name: HALLUCFLAG, dtype: float64
```

```
#Proportion of respondents who have used inhalants
df['INHALFLAG'].value_counts(normalize=True)*100
```

```
Never used    89.739458
Ever used     10.260542
Name: INHALFLAG, dtype: float64
```

```
#Proportion of respondents who have used meth
df['METHAMFLAG'].value_counts(normalize=True)*100
```

```
Never used    95.728574
Ever used      4.271426
Name: METHAMFLAG, dtype: float64
```

```
#exploratory statistics of frequency of substance use in a year
frequency=df[['IRMETHAMYFQ', 'IRALCFY',
       'IRMJFY', 'IRCOCFY', 'IRCRKFY', 'IRHERFY', 'IRHALLUCYFQ',
'IRINHALYFQ', 'ALLDGS']].describe(include='all')
```

```
frequency.columns = ['Meth', 'Alcohol', 'Marihuana', 'Cocaine',
'Crack', 'Heroin', 'Hallucinogens', 'Inhallants', 'DrugsExceptAlc']
```

```
frequency
```

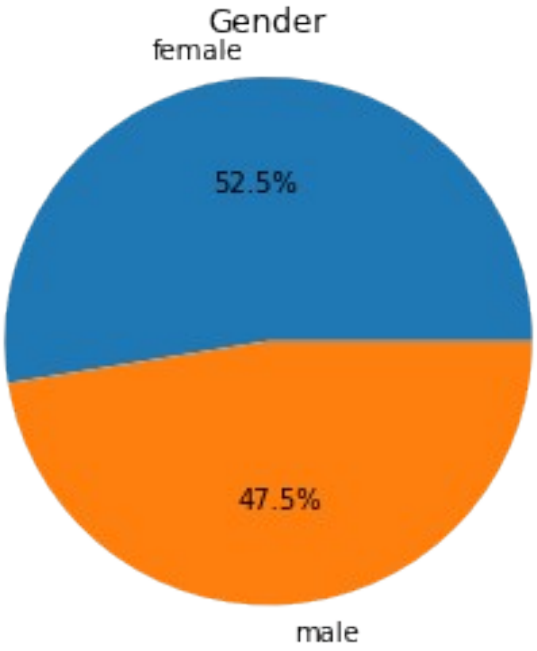
	Meth	Alcohol	Marihuana	Cocaine
count	315661.000000	315661.000000	315661.000000	315661.000000
mean	0.792093	49.781288	24.286833	0.705865
std	13.686260	81.377971	77.036602	10.316375
min	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000
50%	0.000000	6.000000	0.000000	0.000000
75%	0.000000	60.000000	0.000000	0.000000
max	365.000000	365.000000	365.000000	365.000000

	Crack	Heroin	Hallucinogens	Inhallants
count	315661.000000	315661.000000	315661.000000	315661.000000
mean	0.189859	0.453249	0.464378	0.319425
std	6.127078	10.810214	7.422459	6.569459
min	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000	0.000000
75%	0.000000	0.000000	0.000000	0.000000
max	365.000000	365.000000	365.000000	365.000000

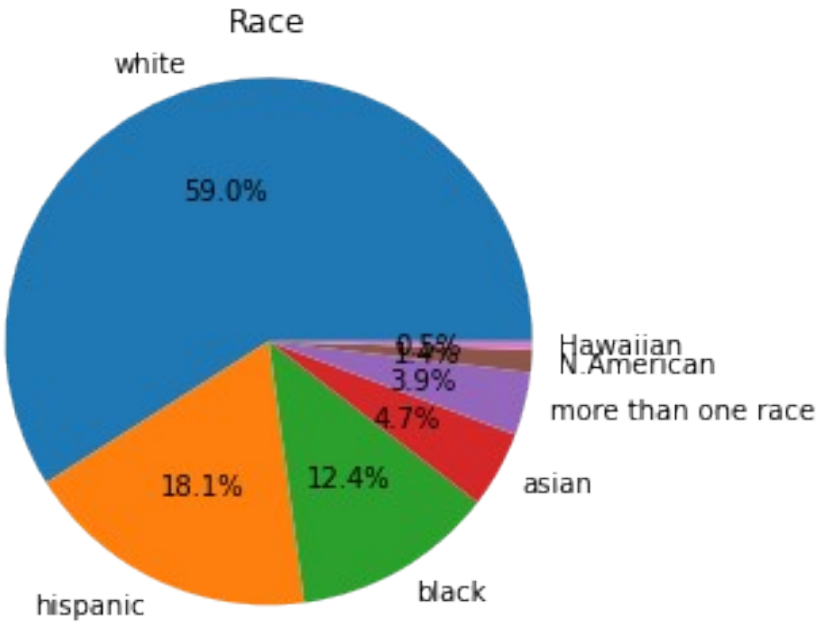
	DrugsExceptAlc
count	315661.000000
mean	27.211702
std	86.493220
min	0.000000
25%	0.000000
50%	0.000000
75%	0.000000
max	2549.000000

```
#descriptive visualization
import matplotlib.pyplot as plt
import numpy as np
```

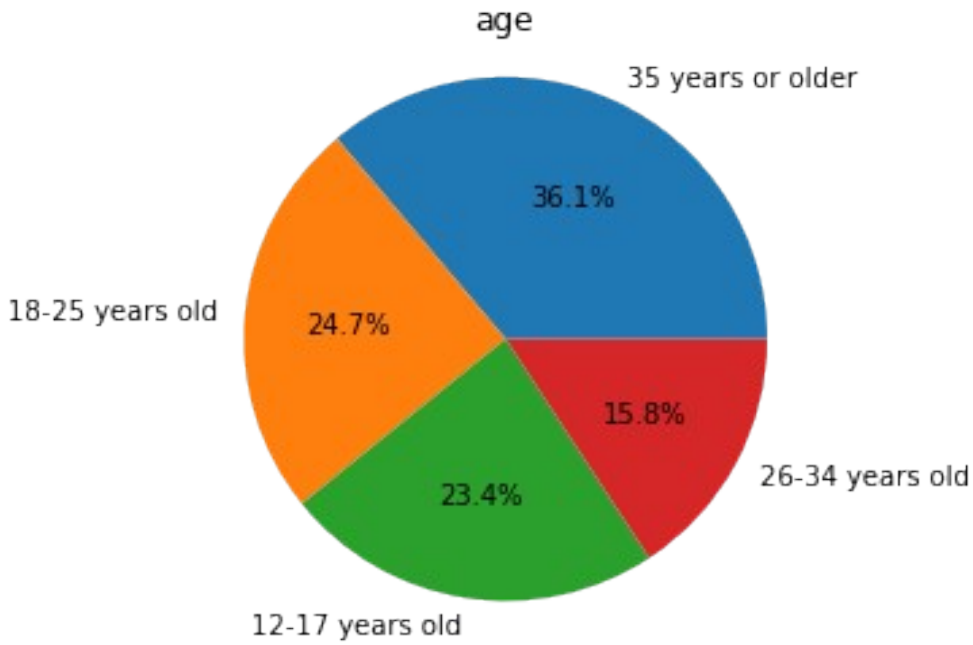
```
my_data = df['IRSEX'].value_counts(normalize=True)*100
my_labels = 'female', 'male'
plt.pie(my_data, labels=my_labels, autopct='%1.1f%%')
plt.title('Gender')
plt.axis('equal')
plt.show()
```




```
my_data = df['NEWRACE2'].value_counts(normalize=True)*100
my_labels = 'white', 'hispanic', 'black', 'asian', 'more than one
race', 'N.American', 'Hawaiian'
plt.pie(my_data, labels=my_labels, autopct='%1.1f%%')
plt.title('Race')
plt.axis('equal')
plt.show()
```




```
my_data = df['CATAGE'].value_counts(normalize=True)*100
my_labels = '35 years or older', '18-25 years old', '12-17 years old',
'26-34 years old'
plt.pie(my_data, labels=my_labels, autopct='%1.1f%%')
plt.title('age')
plt.axis('equal')
plt.show()
```



```
pd.crosstab(df.CATAGE, df.ALCCAT ,normalize='index')\
  .round(4)*100
```

```
ALCCAT               0      1
CATAGE
12-17 years old  77.87  22.13
18-25 years old  26.43  73.57
26-34 years old  20.92  79.08
35 or older      30.64  69.36
```

```
pd.crosstab(df.NEWRACE2, df.ALCCAT ,normalize='index')\
  .round(4)*100
```

```
ALCCAT                                        0      1
NEWRACE2
Asian                                     49.79  50.21
Black/African American                    47.29  52.71
Hispanic                                  46.54  53.46
More than one race                        42.95  57.05
Native American/Alaskan Native            46.96  53.04
Native Hawaiian/Other Pacific Islander    50.61  49.39
White                                     33.77  66.23
```

```
pd.crosstab(df.IRSEX, df.ALCCAT ,normalize='index')\
  .round(4)*100
```

```
ALCCAT      0      1
IRSEX
Female  39.01  60.99
Male    39.28  60.72
```

```
pd.crosstab(df.NEWRACE2, df.ALCCAT ,normalize='index')\
  .round(4)*100
```

```
ALCCAT                                        0      1
NEWRACE2
Asian                                     49.79  50.21
Black/African American                    47.29  52.71
Hispanic                                  46.54  53.46
More than one race                        42.95  57.05
Native American/Alaskan Native            46.96  53.04
Native Hawaiian/Other Pacific Islander    50.61  49.39
White                                     33.77  66.23
```

```
pd.crosstab(df.IRSEX, df.ALCCAT ,normalize='index')\
  .round(4)*100
```

```
ALCCAT      0      1
IRSEX
Female  39.01  60.99
Male    39.28  60.72
```

```
pd.crosstab(df.INCOME, df.ALCCAT ,normalize='index')\
  .round(4)*100
```

```
ALCCAT                 0      1
INCOME
$20,000-$49,999    41.70  58.30
$50,000-$74,999    36.52  63.48
$75,000 or more    34.69  65.31
Less than $20,000  45.83  54.17
```

```
pd.crosstab(df.IREDUHIGHST2, df.ALCCAT ,normalize='index')\
  .round(4)*100
```

```
ALCCAT                                                0      1
IREDUHIGHST2
Associate's degree (AA, AS)                       21.86  78.14
College graduate or higher                        17.68  82.32
Eighth grade completed                            78.33  21.67
Eleventh or Twelfth grade completed, no diploma   47.00  53.00
Fifth grade or less grade completed               85.53  14.47
High school diploma/GED                           34.08  65.92
Ninth grade completed                             65.51  34.49
Seventh grade completed                           89.32  10.68
Sixth grade completed                             90.87   9.13
Some college credit, but no degree                23.32  76.68
Tenth grade completed                             56.77  43.23
```

```
pd.crosstab(df.CATAGE, df.ALLDGSCAT ,normalize='index')\
  .round(4)*100
```

```
ALLDGSCAT            0      1
CATAGE
12-17 years old  84.52  15.48
18-25 years old  64.95  35.05
26-34 years old  74.71  25.29
35 or older      87.20  12.80
```

```
pd.crosstab(df.NEWRACE2, df.ALLDGSCAT ,normalize='index')\
  .round(4)*100
```

```
ALLDGSCAT                                    0      1
NEWRACE2
Asian                                    89.06  10.94
Black/African American                   77.49  22.51
Hispanic                                 81.41  18.59
More than one race                       70.89  29.11
Native American/Alaskan Native           70.15  29.85
Native Hawaiian/Other Pacific Islander   80.73  19.27
White                                    78.70  21.30
```

```
pd.crosstab(df.IRSEX, df.ALLDGSCAT ,normalize='index')\
  .round(4)*100
```

```
ALLDGSCAT      0      1
IRSEX
Female     81.14  18.86
Male       76.88  23.12
```

```
pd.crosstab(df.INCOME, df.ALLDGSCAT ,normalize='index')\
  .round(4)*100
```

```
ALLDGSCAT              0      1
INCOME
$20,000-$49,999    78.02  21.98
$50,000-$74,999    80.46  19.54
$75,000 or more    82.44  17.56
Less than $20,000  73.35  26.65
```

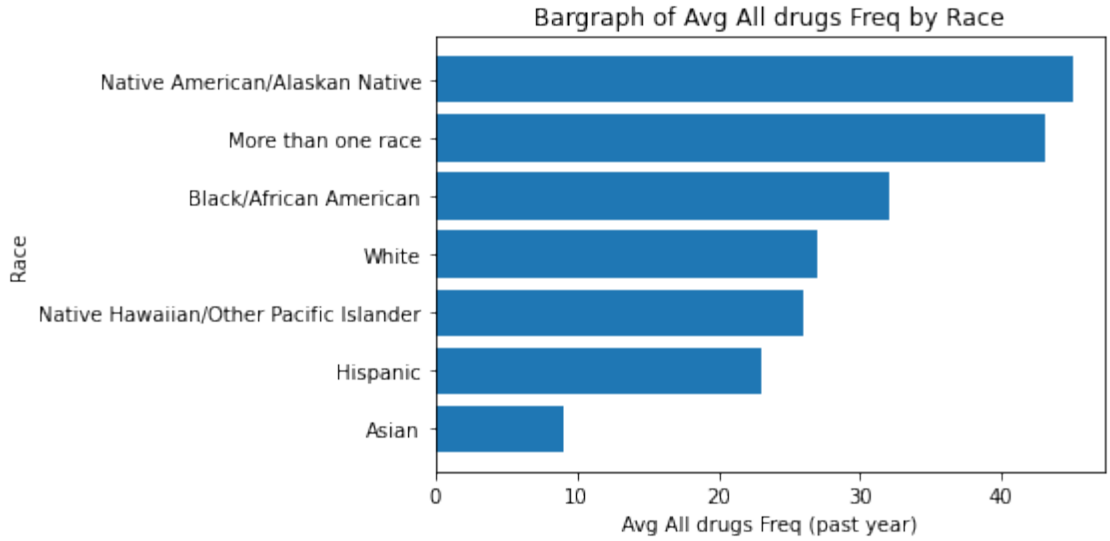
```
pd.crosstab(df.IREDUHIGHST2, df.ALLDGSCAT ,normalize='index')\
  .round(4)*100
```

```
ALLDGSCAT                                          0      1
IREDUHIGHST2
Associate's degree (AA, AS)                    80.12  19.88
College graduate or higher                     81.88  18.12
Eighth grade completed                         86.81  13.19
Eleventh or Twelfth grade completed, no diploma  72.07  27.93
Fifth grade or less grade completed            94.72   5.28
High school diploma/GED                        76.22  23.78
Ninth grade completed                          80.61  19.39
Seventh grade completed                        92.76   7.24
Sixth grade completed                          95.53   4.47
Some college credit, but no degree             71.99  28.01
Tenth grade completed                          74.94  25.06
```

```
#importing the modules
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

#getting the unique values from NEWRACE2 and assigning it to x
x = list(dv['NEWRACE2'].unique())
y = []

# Taking average of y-values and rounding-off to nearest number
# Taking sum of y is not preffered, so going for mean (avg)
for i in range(len(x)):
    y.append(int(dv['ALLDGS'][df['NEWRACE2'] == x[i]].mean().round()))


df_bar = {"Race": list(x), "Alldrugfreq": y}
```

```
dFrame = pd.DataFrame.from_dict(df_bar)
df_sorted_bar = dFrame.sort_values('Alldrugfreq',ascending=True)

print(df_sorted_bar.sort_values('Alldrugfreq',ascending=False))

## Bar Graph (Taking Horizontal bar since regular bar shows an overlap
of the Race names)
plt.barh('Race', 'Alldrugfreq', data=df_sorted_bar)
plt.ylabel('Race')
plt.xlabel('Avg All drugs Freq (past year)')
plt.title('Bargraph of Avg All drugs Freq by Race')
```

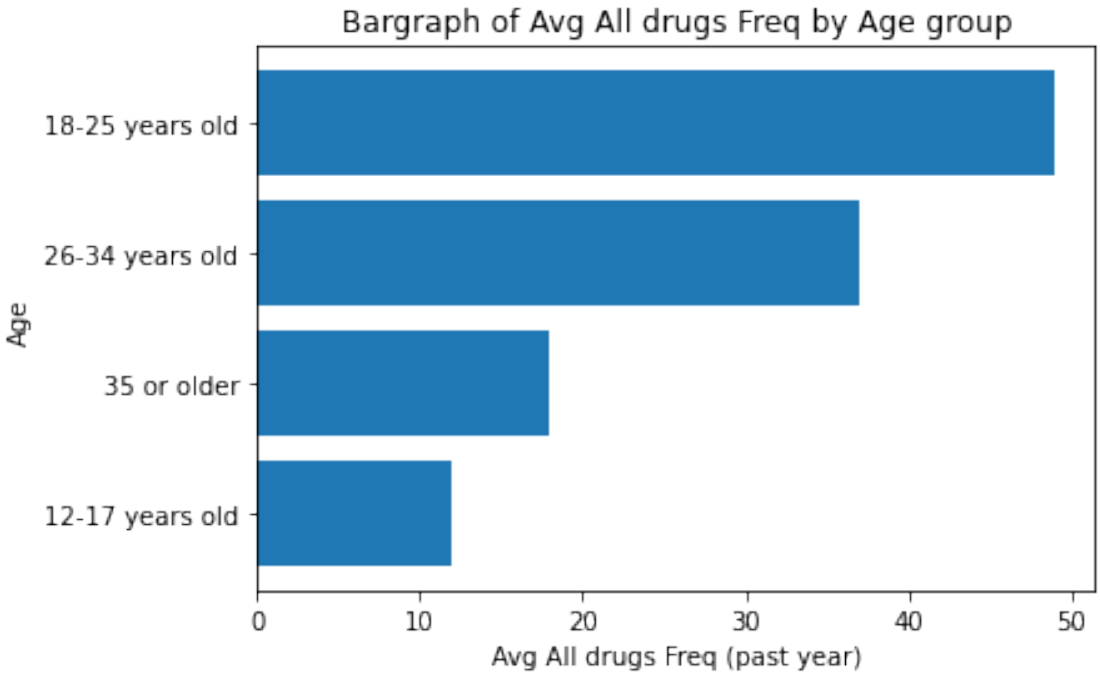
```
                                    Race  Alldrugfreq
6          Native American/Alaskan Native           45
1                    More than one race           43
3                 Black/African American           32
0                                  White           27
4  Native Hawaiian/Other Pacific Islander           26
2                               Hispanic           23
5                                  Asian            9

Text(0.5, 1.0, 'Bargraph of Avg All drugs Freq by Race')
```




```
#importing the modules
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

#getting the unique values from NEWRACE2 and assigning it to x
x = list(dv['CATAGE'].unique())
y = []

# Taking average of y-values and rounding-off to nearest number
```

```
# Taking sum of y is not preffered, so going for mean (avg)
for i in range(len(x)):
    y.append(int(dv['ALLDGS'][df['CATAGE'] == x[i]].mean().round()))

df_bar = {"Age": list(x), "Alldrugfreq": y}
dFrame = pd.DataFrame.from_dict(df_bar)
df_sorted_bar = dFrame.sort_values('Alldrugfreq',ascending=True)

print(df_sorted_bar.sort_values('Alldrugfreq',ascending=False))

## Bar Graph (Taking Horizontal bar since regular bar shows an overlap
of the Race names)
plt.barh('Age', 'Alldrugfreq', data=df_sorted_bar)
plt.ylabel('Age')
plt.xlabel('Avg All drugs Freq (past year)')
plt.title('Bargraph of Avg All drugs Freq by Age group')
```

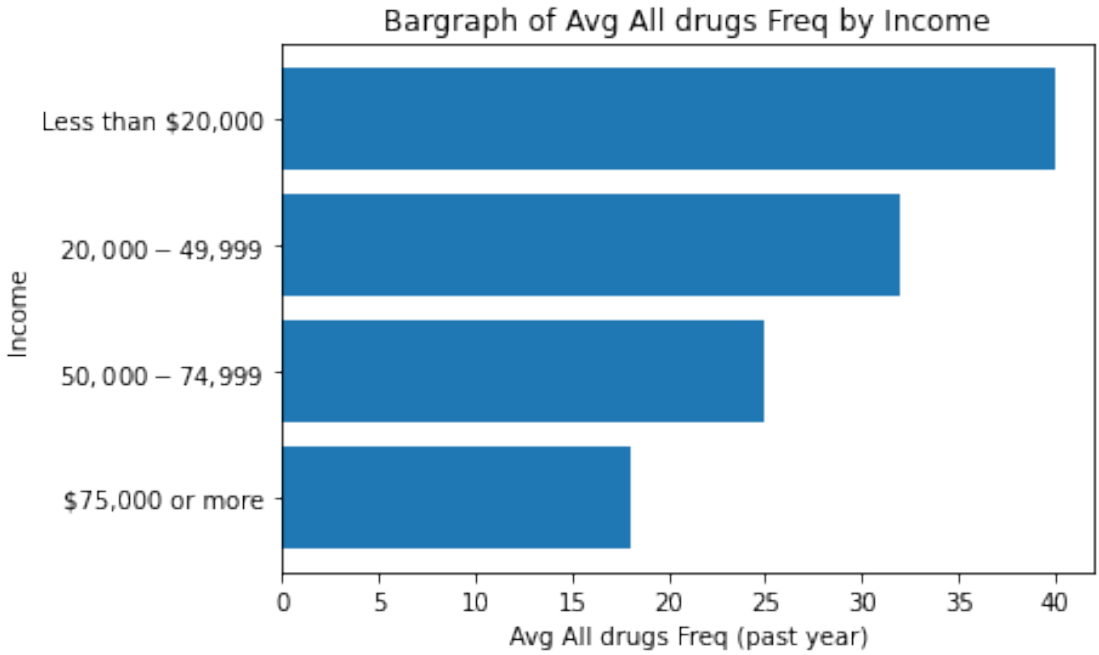
```
                Age  Alldrugfreq
3  18-25 years old           49
1  26-34 years old           37
0      35 or older           18
2  12-17 years old           12

Text(0.5, 1.0, 'Bargraph of Avg All drugs Freq by Age group')
```




```
#importing the modules
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
```

```
#getting the unique values from NEWRACE2 and assigning it to x
x = list(dv['INCOME'].unique())
y = []

# Taking average of y-values and rounding-off to nearest number
# Taking sum of y is not preffered, so going for mean (avg)
for i in range(len(x)):
    y.append(int(dv['ALLDGS'][df['INCOME'] == x[i]].mean().round()))

df_bar = {"Income": list(x), "Alldrugfreq": y}
dFrame = pd.DataFrame.from_dict(df_bar)
df_sorted_bar = dFrame.sort_values('Alldrugfreq',ascending=True)

print(df_sorted_bar.sort_values('Alldrugfreq',ascending=False))

## Bar Graph (Taking Horizontal bar since regular bar shows an overlap
of the Race names)
plt.barh('Income', 'Alldrugfreq', data=df_sorted_bar)
plt.ylabel('Income')
plt.xlabel('Avg All drugs Freq (past year)')
plt.title('Bargraph of Avg All drugs Freq by Income')
```

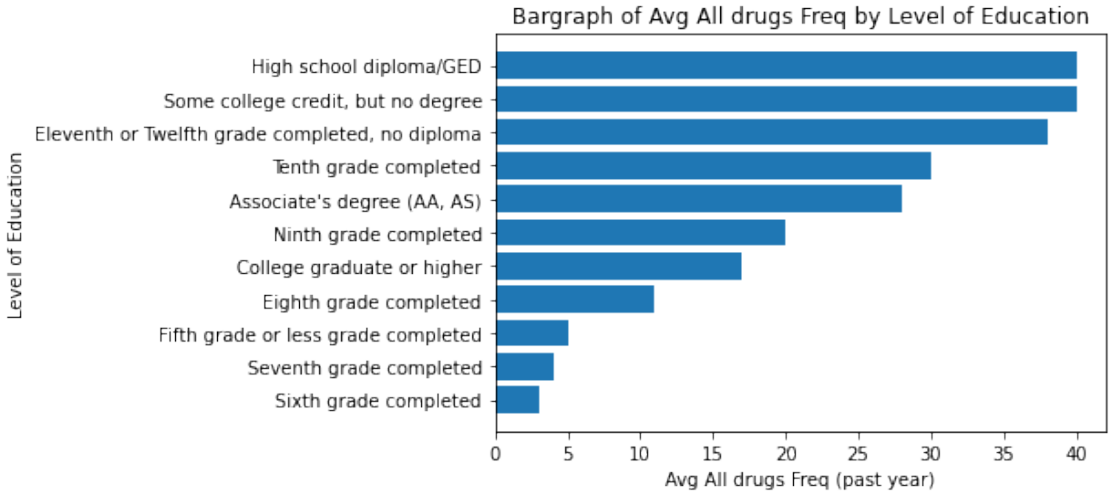
```
              Income  Alldrugfreq
3  Less than $20,000           40
2    $20,000-$49,999           32
1    $50,000-$74,999           25
0    $75,000 or more           18

Text(0.5, 1.0, 'Bargraph of Avg All drugs Freq by Income')
```



```
#importing the modules
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

#getting the unique values from NEWRACE2 and assigning it to x
x = list(dv['IREDUHIGHST2'].unique())
y = []

# Taking average of y-values and rounding-off to nearest number
# Taking sum of y is not preffered, so going for mean (avg)
for i in range(len(x)):
    y.append(int(dv['ALLDGS'][df['IREDUHIGHST2'] ==
x[i]].mean().round()))

df_bar = {"Level of Education": list(x), "Alldrugfreq": y}
dFrame = pd.DataFrame.from_dict(df_bar)
df_sorted_bar = dFrame.sort_values('Alldrugfreq',ascending=True)

print(df_sorted_bar.sort_values('Alldrugfreq',ascending=False))

## Bar Graph (Taking Horizontal bar since regular bar shows an overlap
of the Race names)
plt.barh('Level of Education', 'Alldrugfreq', data=df_sorted_bar)
plt.ylabel('Level of Education')
plt.xlabel('Avg All drugs Freq (past year)')
plt.title('Bargraph of Avg All drugs Freq by Level of Education')
```

	Level of Education	Alldrugfreq
0	Some college credit, but no degree	40
5	High school diploma/GED	40
8	Eleventh or Twelfth grade completed, no diploma	38
4	Tenth grade completed	30
3	Associate's degree (AA, AS)	28
9	Ninth grade completed	20
1	College graduate or higher	17
2	Eighth grade completed	11
10	Fifth grade or less grade completed	5
6	Seventh grade completed	4
7	Sixth grade completed	3

```
Text(0.5, 1.0, 'Bargraph of Avg All drugs Freq by Level of Education')
```

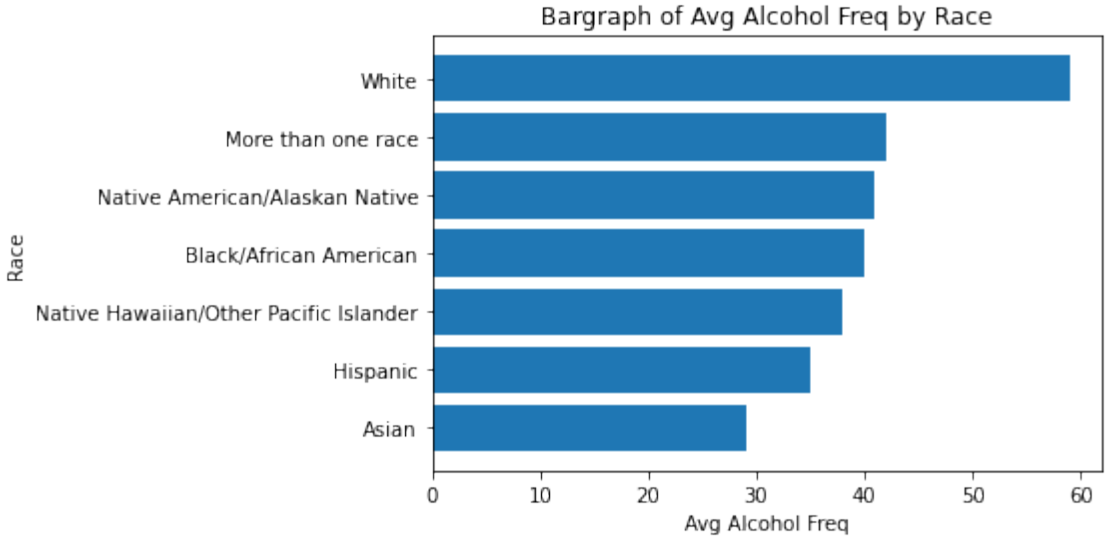



```
#importing the modules
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

#getting the unique values from NEWRACE2 and assigning it to x
x = list(df['NEWRACE2'].unique())
y = []

# Taking average of y-values and rounding-off to nearest number
# Taking sum of y is not preffered, so going for mean (avg)
for i in range(len(x)):
    y.append(int(df['IRALCFY'][df['NEWRACE2'] ==
x[i]].mean().round()))

df_bar = {"Race": list(x), "Alcfreq": y}
dFrame = pd.DataFrame.from_dict(df_bar)
df_sorted_bar = dFrame.sort_values('Alcfreq',ascending=True)

print(df_sorted_bar.sort_values('Alcfreq',ascending=False))

## Bar Graph (Taking Horizontal bar since regular bar shows an overlap
of the Race names)
plt.barh('Race', 'Alcfreq', data=df_sorted_bar)
plt.ylabel('Race')
plt.xlabel('Avg Alcohol Freq')
plt.title('Bargraph of Avg Alcohol Freq by Race')
```

```
                                    Race  Alcfreq
0                                  White       59
1                   More than one race       42
6        Native American/Alaskan Native       41
3                 Black/African American       40
4  Native Hawaiian/Other Pacific Islander       38
```

```
2                                  Hispanic       35
5                                     Asian       29
```

```
Text(0.5, 1.0, 'Bargraph of Avg Alcohol Freq by Race')
```

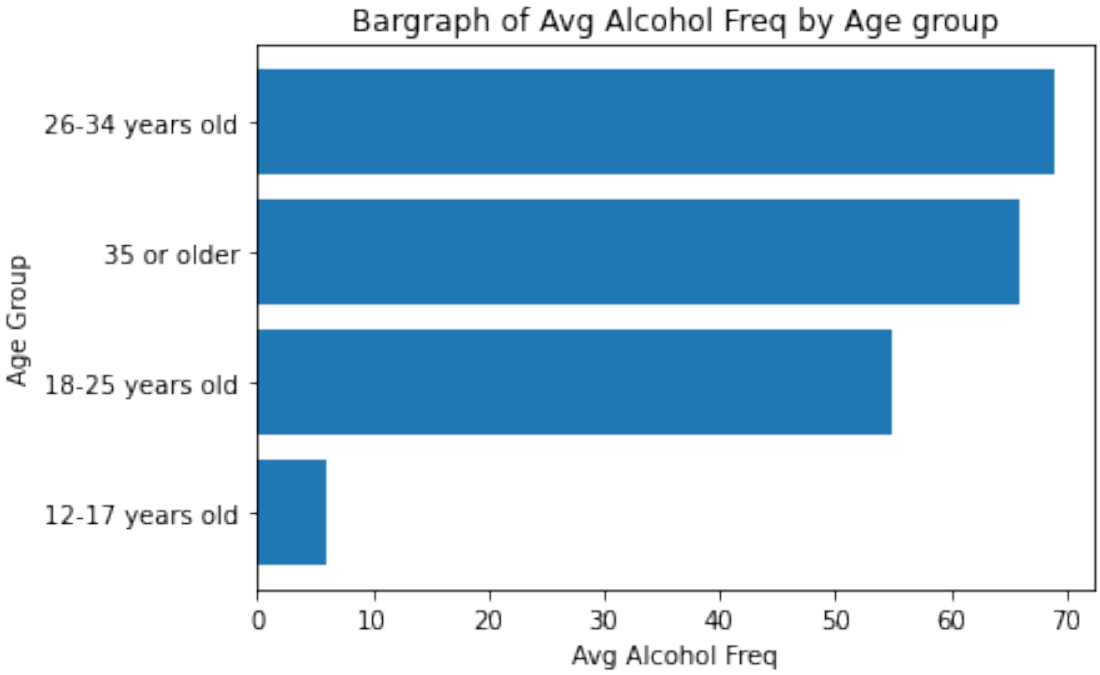



```
#importing the modules
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

#getting the unique values from CATAGE and assigning it to x
x = list(df['CATAGE'].unique())
y = []

# Taking average of y-values and rounding-off to nearest number
# Taking sum of y is not preffered, so going for mean (avg)
for i in range(len(x)):
    y.append(int(df['IRALCFY'][df['CATAGE'] == x[i]].mean().round()))

df_bar = {"Age Group": list(x), "Alcfreq": y}
dFrame = pd.DataFrame.from_dict(df_bar)
df_sorted_bar = dFrame.sort_values('Alcfreq',ascending=True)

print(df_sorted_bar.sort_values('Alcfreq',ascending=False))

## Bar Graph
plt.barh('Age Group', 'Alcfreq', data=df_sorted_bar)
plt.ylabel('Age Group')
plt.xlabel('Avg Alcohol Freq')
plt.title('Bargraph of Avg Alcohol Freq by Age group')
```

```
         Age Group  Alcfreq
1  26-34 years old       69
```

```
0      35 or older       66
3  18-25 years old       55
2  12-17 years old        6
```

```
Text(0.5, 1.0, 'Bargraph of Avg Alcohol Freq by Age group')
```

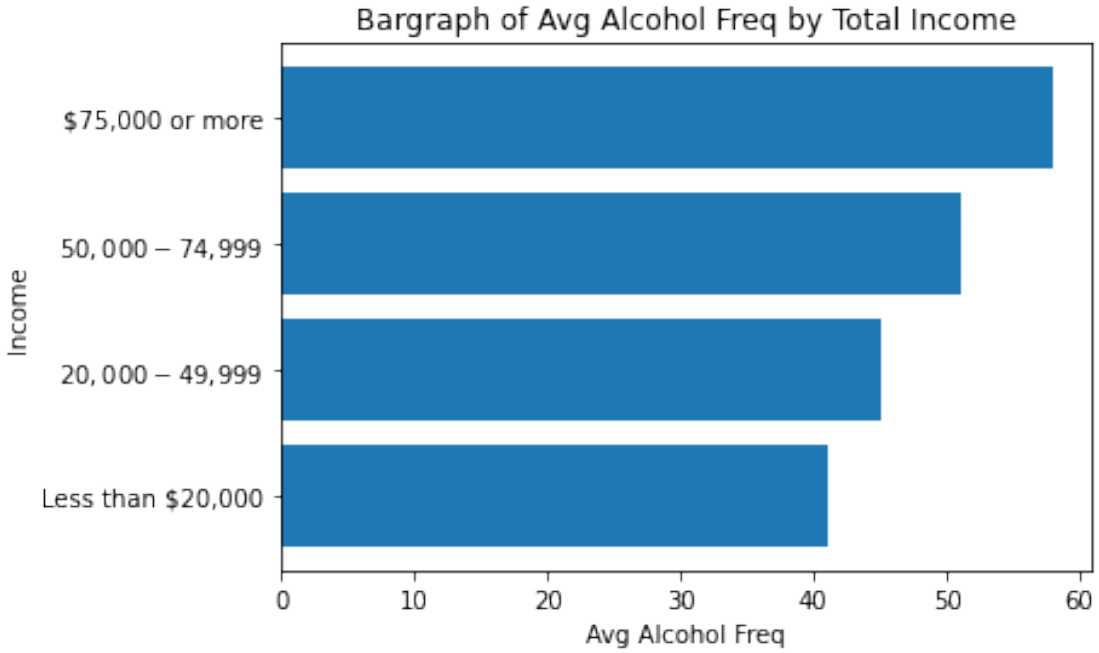



```
#importing the modules
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

#getting the unique values from INCOME and assigning it to x
x = list(df['INCOME'].unique())
y = []

# Taking average of y-values and rounding-off to nearest number
# Taking sum of y is not preffered, so going for mean (avg)
for i in range(len(x)):
    y.append(int(df['IRALCFY'][df['INCOME'] == x[i]].mean().round()))

df_bar = {"Income": list(x), "Alcfreq": y}
dFrame = pd.DataFrame.from_dict(df_bar)
df_sorted_bar = dFrame.sort_values('Alcfreq',ascending=True)

print(df_sorted_bar.sort_values('Alcfreq',ascending=False))

## Bar Graph
plt.barh('Income', 'Alcfreq', data=df_sorted_bar)
plt.ylabel('Income')
```

```
plt.xlabel('Avg Alcohol Freq')
plt.title('Bargraph of Avg Alcohol Freq by Total Income')
```

```
                Income  Alcfreq
0     $75,000 or more       58
1     $50,000-$74,999       51
2     $20,000-$49,999       45
3  Less than $20,000        41
```

```
Text(0.5, 1.0, 'Bargraph of Avg Alcohol Freq by Total Income')
```

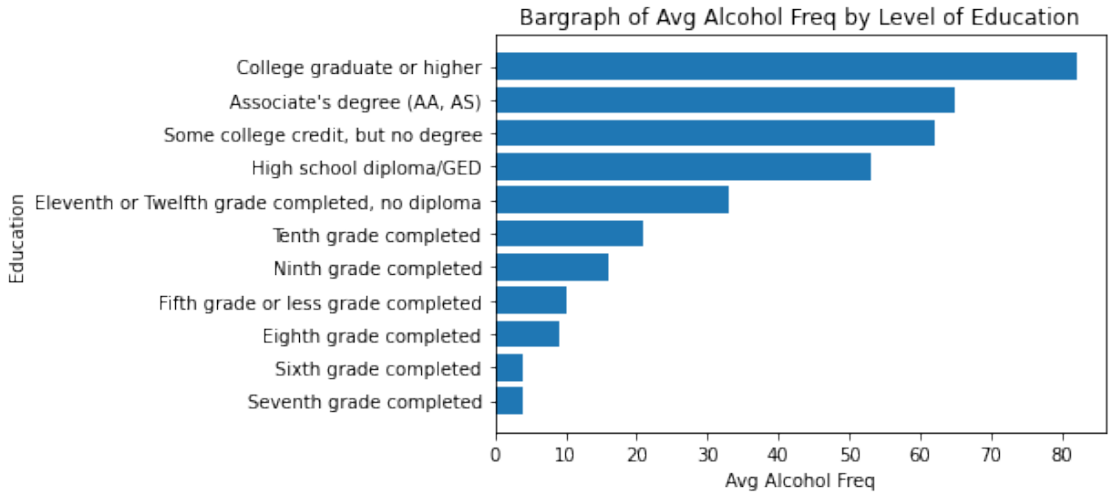



```
#importing the modules
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

#getting the unique values from IREDUHIGHST2 and assigning it to x
x = list(df['IREDUHIGHST2'].unique())
y = []

# Taking average of y-values and rounding-off to nearest number
# Taking sum of y is not preffered, so going for mean (avg)
for i in range(len(x)):
    y.append(int(df['IRALCFY'][df['IREDUHIGHST2'] ==
x[i]].mean().round()))

df_bar = {"Education": list(x), "Alcfreq": y}
dFrame = pd.DataFrame.from_dict(df_bar)
df_sorted_bar = dFrame.sort_values('Alcfreq',ascending=True)
```

```
print(df_sorted_bar.sort_values('Alcfreq',ascending=False))

## Bar Graph
plt.barh('Education', 'Alcfreq', data=df_sorted_bar)
plt.ylabel('Education')
plt.xlabel('Avg Alcohol Freq')
plt.title('Bargraph of Avg Alcohol Freq by Level of Education')
```

	Education	Alcfreq
1	College graduate or higher	82
3	Associate's degree (AA, AS)	65
0	Some college credit, but no degree	62
5	High school diploma/GED	53
8	Eleventh or Twelfth grade completed, no diploma	33
4	Tenth grade completed	21
9	Ninth grade completed	16
10	Fifth grade or less grade completed	10
2	Eighth grade completed	9
6	Seventh grade completed	4
7	Sixth grade completed	4

```
Text(0.5, 1.0, 'Bargraph of Avg Alcohol Freq by Level of Education')
```



#Testing for normality

```
from scipy import stats
x=df["ALLDGS"]
k2, p = stats.normaltest(x)
alpha = 1e-3
print("p = {:g}".format(p))
p = 8.4713e-19
if p < alpha:  # null hypothesis: x comes from a normal distribution
    print("The null hypothesis can be rejected")
else:
    print("The null hypothesis cannot be rejected")
```

```
p = 0
The null hypothesis can be rejected
```

```
from scipy import stats
x=df["IRALCFY"]
k2, p = stats.normaltest(x)
alpha = 1e-3
print("p = {:g}".format(p))
p = 8.4713e-19
if p < alpha:  # null hypothesis: x comes from a normal distribution
    print("The null hypothesis can be rejected")
else:
    print("The null hypothesis cannot be rejected")
```

```
p = 0
The null hypothesis can be rejected
```

#Testing for difference between groups

```
!pip install scikit-posthocs
```

Kruskal Wallis for Alcohol and Ethnicity

```
x1 = dv['IRALCFY'][dv['NEWRACE2']=='White'].sum()
x2 = dv['IRALCFY'][dv['NEWRACE2']=='Hispanic'].sum()
x3 = dv['IRALCFY'][dv['NEWRACE2']=='Black/African American'].sum()
x4 = dv['IRALCFY'][dv['NEWRACE2']=='Native Hawaiian/Other Pacific
Islander'].sum()
x5 = dv['IRALCFY'][dv['NEWRACE2']=='Asian'].sum()
x6 = dv['IRALCFY'][dv['NEWRACE2']=='Native American/Alaskan
Native'].sum()
x7 = dv['IRALCFY'][dv['NEWRACE2']=='More than one race'].sum()
x = [x1,x2,x3,x4,x5,x6,x7]

c1 = dv['NEWRACE2'][dv['NEWRACE2']== 'White'].count()
c2 = dv['NEWRACE2'][dv['NEWRACE2']== 'Hispanic'].count()
c3 = dv['NEWRACE2'][dv['NEWRACE2']== 'Black/African American'].count()
c4 = dv['NEWRACE2'][dv['NEWRACE2']== 'Native Hawaiian/Other Pacific
Islander'].count()
c5 = dv['NEWRACE2'][dv['NEWRACE2']== 'Asian'].count()
c6 = dv['NEWRACE2'][dv['NEWRACE2']== 'Native American/Alaskan
Native'].count()
c7 = dv['NEWRACE2'][dv['NEWRACE2']== 'More than one race'].count()
c = [c1,c2,c3,c4,c5,c6,c7]

stats. kruskal(x, c)
```

Krustal Wallis for all substances and ethnicity

```
x1 = dv['ALLDGS'][dv['NEWRACE2']=='White'].sum()
x2 = dv['ALLDGS'][dv['NEWRACE2']=='Hispanic'].sum()
x3 = dv['ALLDGS'][dv['NEWRACE2']=='Black/African American'].sum()
```

```
x4 = dv['ALLDGS'][dv['NEWRACE2']=='Native Hawaiian/Other Pacific
Islander'].sum()
x5 = dv['ALLDGS'][dv['NEWRACE2']=='Asian'].sum()
x6 = dv['ALLDGS'][dv['NEWRACE2']=='Native American/Alaskan
Native'].sum()
x7 = dv['ALLDGS'][dv['NEWRACE2']=='More than one race'].sum()
x = [x1,x2,x3,x4,x5,x6,x7]

c1 = dv['NEWRACE2'][dv['NEWRACE2']== 'White'].count()
c2 = dv['NEWRACE2'][dv['NEWRACE2']== 'Hispanic'].count()
c3 = dv['NEWRACE2'][dv['NEWRACE2']== 'Black/African American'].count()
c4 = dv['NEWRACE2'][dv['NEWRACE2']== 'Native Hawaiian/Other Pacific
Islander'].count()
c5 = dv['NEWRACE2'][dv['NEWRACE2']== 'Asian'].count()
c6 = dv['NEWRACE2'][dv['NEWRACE2']== 'Native American/Alaskan
Native'].count()
c7 = dv['NEWRACE2'][dv['NEWRACE2']== 'More than one race'].count()
c = [c1,c2,c3,c4,c5,c6,c7]

stats. kruskal(x, c)
```

KW for alcohol frequency and age

```
x1 = dv['IRALCFY'][dv['CATAGE']=='12-17 years old'].sum()
x2 = dv['IRALCFY'][dv['CATAGE']=='18-25 years old'].sum()
x3 = dv['IRALCFY'][dv['CATAGE']=='26-34 years old'].sum()
x4 = dv['IRALCFY'][dv['CATAGE']=='35 or older'].sum()
x = [x1,x2,x3,x4]

c1 = dv['CATAGE'][dv['CATAGE']== '12-17 years old'].count()
c2 = dv['CATAGE'][dv['CATAGE']== '18-25 years old'].count()
c3 = dv['CATAGE'][dv['CATAGE']== '26-34 years old'].count()
c4 = dv['CATAGE'][dv['CATAGE']== '35 or older'].count()
c = [c1,c2,c3,c4]

stats. kruskal(x, c)
```

KW for all substances and age groups

```
x1 = dv['ALLDGS'][dv['CATAGE']=='12-17 years old'].sum()
x2 = dv['ALLDGS'][dv['CATAGE']=='18-25 years old'].sum()
x3 = dv['ALLDGS'][dv['CATAGE']=='26-34 years old'].sum()
x4 = dv['ALLDGS'][dv['CATAGE']=='35 or older'].sum()
x = [x1,x2,x3,x4]

c1 = dv['CATAGE'][dv['CATAGE']== '12-17 years old'].count()
c2 = dv['CATAGE'][dv['CATAGE']== '18-25 years old'].count()
c3 = dv['CATAGE'][dv['CATAGE']== '26-34 years old'].count()
c4 = dv['CATAGE'][dv['CATAGE']== '35 or older'].count()
c = [c1,c2,c3,c4]

stats. kruskal(x, c)
```

KW for alcohol frequency and gender

```
x1 = dv['IRALCFY'][dv['IRSEX']=='Male'].sum()
x2 = dv['IRALCFY'][dv['IRSEX']=='Female'].sum()
x = [x1,x2]

c1 = dv['IRSEX'][dv['IRSEX']== 'Male'].count()
c2 = dv['IRSEX'][dv['IRSEX']== 'Female'].count()
c = [c1,c2]

stats. kruskal(x,c)
```

KW for all substances and gender

```
x1 = dv['ALLDGS'][dv['IRSEX']=='Male'].sum()
x2 = dv['ALLDGS'][dv['IRSEX']=='Female'].sum()
x = [x1,x2]

c1 = dv['IRSEX'][dv['IRSEX']== 'Male'].count()
c2 = dv['IRSEX'][dv['IRSEX']== 'Female'].count()
c = [c1,c2]

stats. kruskal(x,c)
```

KW For alcohol frequency and work status

```
x1 = dv['IRALCFY'][dv['IRWRKSTAT']=='Employed full time'].sum()
x2 = dv['IRALCFY'][dv['IRWRKSTAT']=='Employed part time'].sum()
x3 = dv['IRALCFY'][dv['IRWRKSTAT']=='Unemployed'].sum()
x4 = dv['IRALCFY'][dv['IRWRKSTAT']=='Other (incl. not in labor
force)'].sum()
x5 = dv['IRALCFY'][dv['IRWRKSTAT']=='12-14 year olds'].sum()
x=[x1,x2,x3,x4,x5]

c1 = dv['IRWRKSTAT'][dv['IRWRKSTAT']== 'Employed full time'].count()
c2 = dv['IRWRKSTAT'][dv['IRWRKSTAT']== 'Employed part time'].count()
c3 = dv['IRWRKSTAT'][dv['IRWRKSTAT']== 'Unemployed'].count()
c4 = dv['IRWRKSTAT'][dv['IRWRKSTAT']== 'Other (incl. not in labor
force)'].count()
c5 = dv['IRWRKSTAT'][dv['IRWRKSTAT']== '12-14 year olds'].count()
c=[c1,c2,c3,c4,c5]

stats. kruskal(x,c)
```

KW For alcohol frequency and education

```
x1 = dv['IRALCFY'][dv['IREDUHIGHST2']=='Fifth grade or less grade
completed'].sum()
x2 = dv['IRALCFY'][dv['IREDUHIGHST2']=='Sixth grade completed'].sum()
x3 = dv['IRALCFY'][dv['IREDUHIGHST2']=='Seventh grade
completed'].sum()
x4 = dv['IRALCFY'][dv['IREDUHIGHST2']=='Eighth grade completed'].sum()
x5 = dv['IRALCFY'][dv['IREDUHIGHST2']=='Ninth grade completed'].sum()
```

```
x6 = dv['IRALCFY'][dv['IREDUHIGHST2']=='Tenth grade completed'].sum()
x7 = dv['IRALCFY'][dv['IREDUHIGHST2']=='Eleventh or Twelfth grade
completed, no diploma'].sum()
x8 = dv['IRALCFY'][dv['IREDUHIGHST2']=='High school
diploma/GED'].sum()
x9 = dv['IRALCFY'][dv['IREDUHIGHST2']=='Some college credit, but no
degree'].sum()
x10 = dv['IRALCFY'][dv['IREDUHIGHST2']=="Associate's degree (AA,
AS)"].sum()
x11 = dv['IRALCFY'][dv['IREDUHIGHST2']=='College graduate or
higher'].sum()
x=[x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11]

c1 = dv['IREDUHIGHST2'][dv['IREDUHIGHST2']== 'Fifth grade or less
grade completed'].count()
c2 = dv['IREDUHIGHST2'][dv['IREDUHIGHST2']== 'Sixth grade
completed'].count()
c3 = dv['IREDUHIGHST2'][dv['IREDUHIGHST2']== 'Seventh grade
completed'].count()
c4 = dv['IREDUHIGHST2'][dv['IREDUHIGHST2']== 'Eighth grade
completed'].count()
c5 = dv['IREDUHIGHST2'][dv['IREDUHIGHST2']== 'Ninth grade
completed'].count()
c6 = dv['IREDUHIGHST2'][dv['IREDUHIGHST2']== 'Tenth grade
completed'].count()
c7 = dv['IREDUHIGHST2'][dv['IREDUHIGHST2']== 'Eleventh or Twelfth
grade completed, no diploma'].count()
c8 = dv['IREDUHIGHST2'][dv['IREDUHIGHST2']== 'High school
diploma/GED'].count()
c9 = dv['IREDUHIGHST2'][dv['IREDUHIGHST2']== 'Some college credit, but
no degree'].count()
c10 = dv['IREDUHIGHST2'][dv['IREDUHIGHST2']== "Associate's degree (AA,
AS)"].count()
c11 = dv['IREDUHIGHST2'][dv['IREDUHIGHST2']== 'College graduate or
higher'].count()
c=[c1,c2,c3,c4,c5,c6,c7,c8,c9,c10,c11]

stats. kruskal(x,c)
```

KW for all susbtances and education

```
x1 = dv['ALLDGS'][dv['IREDUHIGHST2']=='Fifth grade or less grade
completed'].sum()
x2 = dv['ALLDGS'][dv['IREDUHIGHST2']=='Sixth grade completed'].sum()
x3 = dv['ALLDGS'][dv['IREDUHIGHST2']=='Seventh grade completed'].sum()
x4 = dv['ALLDGS'][dv['IREDUHIGHST2']=='Eighth grade completed'].sum()
x5 = dv['ALLDGS'][dv['IREDUHIGHST2']=='Ninth grade completed'].sum()
x6 = dv['ALLDGS'][dv['IREDUHIGHST2']=='Tenth grade completed'].sum()
x7 = dv['ALLDGS'][dv['IREDUHIGHST2']=='Eleventh or Twelfth grade
completed, no diploma'].sum()
x8 = dv['ALLDGS'][dv['IREDUHIGHST2']=='High school diploma/GED'].sum()
```

```
x9 = dv['ALLDGS'][dv['IREDUHIGHST2']=='Some college credit, but no
degree'].sum()
x10 = dv['ALLDGS'][dv['IREDUHIGHST2']=="Associate's degree (AA,
AS)"].sum()
x11 = dv['ALLDGS'][dv['IREDUHIGHST2']=='College graduate or
higher'].sum()
x=[x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11]

c1 = dv['IREDUHIGHST2'][dv['IREDUHIGHST2']== 'Fifth grade or less
grade completed'].count()
c2 = dv['IREDUHIGHST2'][dv['IREDUHIGHST2']== 'Sixth grade
completed'].count()
c3 = dv['IREDUHIGHST2'][dv['IREDUHIGHST2']== 'Seventh grade
completed'].count()
c4 = dv['IREDUHIGHST2'][dv['IREDUHIGHST2']== 'Eighth grade
completed'].count()
c5 = dv['IREDUHIGHST2'][dv['IREDUHIGHST2']== 'Ninth grade
completed'].count()
c6 = dv['IREDUHIGHST2'][dv['IREDUHIGHST2']== 'Tenth grade
completed'].count()
c7 = dv['IREDUHIGHST2'][dv['IREDUHIGHST2']== 'Eleventh or Twelfth
grade completed, no diploma'].count()
c8 = dv['IREDUHIGHST2'][dv['IREDUHIGHST2']== 'High school
diploma/GED'].count()
c9 = dv['IREDUHIGHST2'][dv['IREDUHIGHST2']== 'Some college credit, but
no degree'].count()
c10 = dv['IREDUHIGHST2'][dv['IREDUHIGHST2']== "Associate's degree (AA,
AS)"].count()
c11 = dv['IREDUHIGHST2'][dv['IREDUHIGHST2']== 'College graduate or
higher'].count()
c=[c1,c2,c3,c4,c5,c6,c7,c8,c9,c10,c11]

stats. kruskal(x,c)
```

KW for alcohol frequency and population density

```
x1 = dv['IRALCFY'][dv['PDEN10']=='Segment in a CBSA with 1 million or
more persons'].sum()
x2 = dv['IRALCFY'][dv['PDEN10']=='Segm in a CBSA with fewer  than 1
million persons'].sum()
x3 = dv['IRALCFY'][dv['PDEN10']=='Segment not in CBSA'].sum()
x=[x1,x2,x3]

c1 = dv['PDEN10'][dv['PDEN10']== 'Segment in a CBSA with 1 million or
more persons'].count()
c2 = dv['PDEN10'][dv['PDEN10']== 'Segm in a CBSA with fewer  than 1
million persons'].count()
c3 = dv['PDEN10'][dv['PDEN10']== 'Segment not in CBSA'].count()
c=[c1,c2,c3]

stats. kruskal(x,c)
```

KW for alcohol frequency and county

```
x1 = dv['IRALCFY'][dv['COUTYP4']=='Large Metro'].sum()
x2 = dv['IRALCFY'][dv['COUTYP4']=='Small Metro'].sum()
x3 = dv['IRALCFY'][dv['COUTYP4']=='Nonmetro'].sum()
x=[x1,x2,x3]

c1 = dv['IRALCFY'][dv['IRALCFY']== 'Large Metro'].count()
c2 = dv['IRALCFY'][dv['IRALCFY']== 'Small Metro'].count()
c3 = dv['IRALCFY'][dv['IRALCFY']== 'Nonmetro'].count()
c=[c1,c2,c3]

stats. kruskal(x,c)
```

KW for alcohol frequency and health insurance

```
x1 = dv['IRALCFY'][dv['IRPRVHLT']=='Yes'].sum()
x2 = dv['IRALCFY'][dv['IRPRVHLT']=='No'].sum()
x=[x1,x2]

c1 = dv['IRPRVHLT'][dv['IRPRVHLT']== 'Yes'].count()
c2 = dv['IRPRVHLT'][dv['IRPRVHLT']== 'No'].count()
c = [c1,c2]

stats.kruskal(x,c)
```

KW for alcohol frequency and median household income

```
x1 = dv['IRALCFY'][dv['IRPINC3']=='Less than $10,000'].sum()
x2 = dv['IRALCFY'][dv['IRPINC3']=='$10,000-$19,999'].sum()
x3 = dv['IRALCFY'][dv['IRPINC3']=='$20,000-$29,999'].sum()
x4 = dv['IRALCFY'][dv['IRPINC3']=='$30,000-$39,999'].sum()
x5 = dv['IRALCFY'][dv['IRPINC3']=='$40,000-$49,999'].sum()
x6 = dv['IRALCFY'][dv['IRPINC3']=='$50,000-$74,999'].sum()
x7 = dv['IRALCFY'][dv['IRPINC3']=='$75,000 or more'].sum()
x=[x1,x2,x3,x4,x5,x6,x7]

c1 = dv['IRPINC3'][dv['IRPINC3']== 'Less than $10,000'].count()
c2 = dv['IRPINC3'][dv['IRPINC3']== '$10,000-$19,999'].count()
c3 = dv['IRPINC3'][dv['IRPINC3']== '$20,000-$29,999'].count()
c4 = dv['IRPINC3'][dv['IRPINC3']== '$30,000-$39,999'].count()
c5 = dv['IRPINC3'][dv['IRPINC3']== '$40,000-$49,999'].count()
c6 = dv['IRPINC3'][dv['IRPINC3']== '$50,000-$74,999'].count()
c7 = dv['IRPINC3'][dv['IRPINC3']== '$75,000 or more'].count()
c=[c1,c2,c3,c4,c5,c6,c7]

stats.kruskal(x,c)
```

KW for alcohol frequency and government assistance

```
x1 = dv['IRALCFY'][dv['IRFAMSOC']=='Yes'].sum()
x2 = dv['IRALCFY'][dv['IRFAMSOC']=='No'].sum()
x=[x1,x2]
```

```
c1 = dv['IRFAMSOC'][dv['IRFAMSOC']== 'Yes'].count()
c2 = dv['IRFAMSOC'][dv['IRFAMSOC']== 'No'].count()
c=[c1,c2]

stats.kruskal(x,c)
```

KW for alcohol frequency and perception of unmet need for mental health

```
x1 = dv['IRALCFY'][dv['AMHTXND2']=='Unknown/Aged 12-17'].sum()
x2 = dv['IRALCFY'][dv['AMHTXND2']=='Yes'].sum()
x3 = dv['IRALCFY'][dv['AMHTXND2']=='No'].sum()
x=[x1,x2,x3]

c1 = dv['AMHTXND2'][dv['AMHTXND2']== 'Unknown/Aged 12-17'].count()
c2 = dv['AMHTXND2'][dv['AMHTXND2']== 'No'].count()
c3 = dv['AMHTXND2'][dv['AMHTXND2']== 'Yes'].count()
c=[c1,c2,c3]

stats.kruskal(x,c)
```

KW for alcohol frequency and past depression

```
x1 = dv['IRALCFY'][dv['AMDEYR']=='Unknown/Aged 12-17'].sum()
x2 = dv['IRALCFY'][dv['AMDEYR']=='Yes'].sum()
x3 = dv['IRALCFY'][dv['AMDEYR']=='No'].sum()
x=[x1,x2,x3]

c1 = dv['AMDEYR'][dv['AMDEYR']== 'Unknown/Aged 12-17'].count()
c2 = dv['AMDEYR'][dv['AMDEYR']== 'No'].count()
c3 = dv['AMDEYR'][dv['AMDEYR']== 'Yes'].count()
c=[c1,c2,c3]

stats.kruskal(x,c)
```

KW for alcohol frequency and level of disability

```
x1 = dv['IRALCFY'][dv['ASDSHOM2']=='Aged 12-17/Unkn/Skip'].sum()
x2 = dv['IRALCFY'][dv['ASDSHOM2']=='None'].sum()
x3 = dv['IRALCFY'][dv['ASDSHOM2']=='Mild'].sum()
x4 = dv['IRALCFY'][dv['ASDSHOM2']=='Moderate'].sum()
x5 = dv['IRALCFY'][dv['ASDSHOM2']=='Severe'].sum()
x6 = dv['IRALCFY'][dv['ASDSHOM2']=='Very Severe'].sum()
x= [x1,x2,x3,x4,x5,x6]

c1 = dv['IRPINC3'][dv['ASDSHOM2']== 'Aged 12-17/Unkn/Skip'].count()
c2 = dv['IRPINC3'][dv['ASDSHOM2']== 'None'].count()
c3 = dv['IRPINC3'][dv['ASDSHOM2']== 'Mild'].count()
c4 = dv['IRPINC3'][dv['ASDSHOM2']== 'Moderate'].count()
c5 = dv['IRPINC3'][dv['ASDSHOM2']== 'Severe'].count()
c6 = dv['IRPINC3'][dv['ASDSHOM2']== 'Very Severe'].count()
c= [c1,c2,c3,c4,c5,c6]

stats.kruskal(x,c)
```

KW for alcohol frequency and level of functioning disability

```
x1 = dv['IRALCFY'][dv['ASDSWRK2']=='Aged 12-17/Unkn/Skip'].sum()
x2 = dv['IRALCFY'][dv['ASDSWRK2']=='None'].sum()
x3 = dv['IRALCFY'][dv['ASDSWRK2']=='Mild'].sum()
x4 = dv['IRALCFY'][dv['ASDSWRK2']=='Moderate'].sum()
x5 = dv['IRALCFY'][dv['ASDSWRK2']=='Severe'].sum()
x6 = dv['IRALCFY'][dv['ASDSWRK2']=='Very Severe'].sum()
x= [x1,x2,x3,x4,x5,x6]

c1 = dv['IRPINC3'][dv['ASDSWRK2']== 'Aged 12-17/Unkn/Skip'].count()
c2 = dv['IRPINC3'][dv['ASDSWRK2']== 'None'].count()
c3 = dv['IRPINC3'][dv['ASDSWRK2']== 'Mild'].count()
c4 = dv['IRPINC3'][dv['ASDSWRK2']== 'Moderate'].count()
c5 = dv['IRPINC3'][dv['ASDSWRK2']== 'Severe'].count()
c6 = dv['IRPINC3'][dv['ASDSWRK2']== 'Very Severe'].count()
c= [c1,c2,c3,c4,c5,c6]

stats.kruskal(x,c)
```

KW between alcohol frequency and Level of functioning disability- from depression- Close Relationships

```
x1 = dv['IRALCFY'][dv['ASDSREL2']=='Aged 12-17/Unkn/Skip'].sum()
x2 = dv['IRALCFY'][dv['ASDSREL2']=='None'].sum()
x3 = dv['IRALCFY'][dv['ASDSREL2']=='Mild'].sum()
x4 = dv['IRALCFY'][dv['ASDSREL2']=='Moderate'].sum()
x5 = dv['IRALCFY'][dv['ASDSREL2']=='Severe'].sum()
x6 = dv['IRALCFY'][dv['ASDSREL2']=='Very Severe'].sum()
x= [x1,x2,x3,x4,x5,x6]

c1 = dv['IRPINC3'][dv['ASDSREL2']== 'Aged 12-17/Unkn/Skip'].count()
c2 = dv['IRPINC3'][dv['ASDSREL2']== 'None'].count()
c3 = dv['IRPINC3'][dv['ASDSREL2']== 'Mild'].count()
c4 = dv['IRPINC3'][dv['ASDSREL2']== 'Moderate'].count()
c5 = dv['IRPINC3'][dv['ASDSREL2']== 'Severe'].count()
c6 = dv['IRPINC3'][dv['ASDSREL2']== 'Very Severe'].count()
c= [c1,c2,c3,c4,c5,c6]

stats.kruskal(x,c)
```

Kruskal Wallis between alcohol frequency use and Level of functioning disability- from depression- Social life

```
x1 = dv['IRALCFY'][dv['ASDSSOC2']=='Aged 12-17/Unkn/Skip'].sum()
x2 = dv['IRALCFY'][dv['ASDSSOC2']=='None'].sum()
x3 = dv['IRALCFY'][dv['ASDSSOC2']=='Mild'].sum()
x4 = dv['IRALCFY'][dv['ASDSSOC2']=='Moderate'].sum()
x5 = dv['IRALCFY'][dv['ASDSSOC2']=='Severe'].sum()
x6 = dv['IRALCFY'][dv['ASDSSOC2']=='Very Severe'].sum()
x= [x1,x2,x3,x4,x5,x6]
```

```
c1 = dv['IRPINC3'][dv['ASDSREL2']== 'Aged 12-17/Unkn/Skip'].count()
c2 = dv['IRPINC3'][dv['ASDSREL2']== 'None'].count()
c3 = dv['IRPINC3'][dv['ASDSREL2']== 'Mild'].count()
c4 = dv['IRPINC3'][dv['ASDSREL2']== 'Moderate'].count()
c5 = dv['IRPINC3'][dv['ASDSREL2']== 'Severe'].count()
c6 = dv['IRPINC3'][dv['ASDSREL2']== 'Very Severe'].count()
c= [c1,c2,c3,c4,c5,c6]

stats.kruskal(x,c)
```

#ML models

CLASSIFICATION

X data

```
#  One-hot encoding (Dummy encoding)

import matplotlib.pyplot as plt

# Dummy encoding on the categorical columns:
#
'ASDSHOM2','ASDSWRK2','ASDSREL2','ASDSSOC2','ASDSOVL2','IRSEX','IREDUH
IGHST2','CATAGE','NEWRACE2','IRWRKSTAT','IRPINC3','INCOME','PDEN10' &
'COUTYP4'
dummy1=pd.get_dummies(df['ASDSHOM2'])
dummy2=pd.get_dummies(df['ASDSWRK2'])
dummy3=pd.get_dummies(df['ASDSREL2'])
dummy4=pd.get_dummies(df['ASDSSOC2'])
dummy5=pd.get_dummies(df['ASDSOVL2'])
dummy6=pd.get_dummies(df['IRSEX'])
dummy7=pd.get_dummies(df['IREDUHIGHST2'])
dummy8=pd.get_dummies(df['CATAGE'])
dummy9=pd.get_dummies(df['NEWRACE2'])
dummy10=pd.get_dummies(df['IRWRKSTAT'])
dummy11=pd.get_dummies(df['IRPINC3'])
dummy12=pd.get_dummies(df['INCOME'])
dummy13=pd.get_dummies(df['PDEN10'])
dummy14=pd.get_dummies(df['COUTYP4'])

# Combining df, dummy1 to dummy14. We are adding cols, so axis=1
master = pd.concat([df, dummy1, dummy2, dummy3, dummy4, dummy5,
dummy6, dummy7, dummy8, dummy9, dummy10, dummy11, dummy12, dummy13,
dummy14],axis=1)

# Removing QUESTID2, IRALCFY:IRMETHAMYFQ, ALCFLAG:METHAMFLAG,
ASDSHOM2:ASDSOVL2 (dummy encoded), IRSEX (dummy enc'd)
# NEWRACE2 (dummy
enc'd),'IREDUHIGHST2','CATAGE','NEWRACE2','IRWRKSTAT','IRPINC3','INCOM
```

```
E','PDEN10', 'COUTYP4' (all dummy enc'd), ALCCAT, IRALCFY, ALLDGSFLAG,
ALLDGS from Master data
# Also removing dummy encoded variables and variables that were
decided to remove after correlation matrix
x =
master.drop(master.columns[[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,1
9,20,21,22,23,24,25,26,27,28,31,32,33,34,35,36,37,38,39]],axis=1)


# Renaming depression categories to avoid column duplication
i=5
x.columns.values[9-i] = 'Aged 12-17/Unkn/Skip_ASDSHOM2'
x.columns.values[10-i] = 'Mild_ASDSHOM2'
x.columns.values[11-i] = 'Moderate_ASDSHOM2'
x.columns.values[12-i] = 'None_ASDSHOM2'
x.columns.values[13-i] = 'Severe_ASDSHOM2'
x.columns.values[14-i] = 'Very Severe_ASDSHOM2'

x.columns.values[15-i] = 'Aged 12-17/Unkn/Skip_ASDSWRK2'
x.columns.values[16-i] = 'Mild_ASDSWRK2'
x.columns.values[17-i] = 'Moderate_ASDSWRK2'
x.columns.values[18-i] = 'None_ASDSWRK2'
x.columns.values[19-i] = 'Severe_ASDSWRK2'
x.columns.values[20-i] = 'Very Severe_ASDSWRK2'

x.columns.values[21-i] = 'Aged 12-17/Unkn/Skip_ASDSREL2'
x.columns.values[22-i] = 'Mild_ASDSREL2'
x.columns.values[23-i] = 'Moderate_ASDSREL2'
x.columns.values[24-i] = 'None_ASDSREL2'
x.columns.values[25-i] = 'Severe_ASDSREL2'
x.columns.values[26-i] = 'Very Severe_ASDSREL2'

x.columns.values[27-i] = 'Aged 12-17/Unkn/Skip_ASDSSOC2'
x.columns.values[28-i] = 'Mild_ASDSSOC2'
x.columns.values[29-i] = 'Moderate_ASDSSOC2'
x.columns.values[30-i] = 'None_ASDSSOC2'
x.columns.values[31-i] = 'Severe_ASDSSOC2'
x.columns.values[32-i] = 'Very Severe_ASDSSOC2'

x.columns.values[33-i] = 'Aged 12-17/Unkn/Skip_ASDSOVL2'
x.columns.values[34-i] = 'Mild_ASDSOVL2'
x.columns.values[35-i] = 'Moderate_ASDSOVL2'
x.columns.values[36-i] = 'None_ASDSOVL2'
x.columns.values[37-i] = 'Severe_ASDSOVL2'
x.columns.values[38-i] = 'Very Severe_ASDSOVL2'

x.columns.values[70] = '$20,000-$49,999_INCOME'
x.columns.values[71] = '$50,000-$74,999_INCOME'
x.columns.values[72] = '$75,000 or more_INCOME'
x.columns.values[73] = 'Less than $20,000_INCOME'
```

```
x.columns.values[58] = '12-14 year olds_IRWRKSTAT'
```

For ALCCAT (Consumed/Not consumed Alcohol)

##Y Data

```
# create target variable
y = df['ALCCAT']
```

##SMOTE

```
# Install imblearn for SMOTE
!pip install imblearn
```

```
# Takes around 6 minutes to run this code chunk
from imblearn.over_sampling import SMOTE
smote =SMOTE(random_state=42)
x_sm,y_sm=smote.fit_resample(x,y)

print("X shape before SMOTE",x.shape)
print("X shape after SMOTE",x_sm.shape)

print("y shape before SMOTE",y.shape)
print("X shape after SMOTE",y_sm.shape)
```

```
X shape before SMOTE (315661, 80)
X shape after SMOTE (384224, 80)
y shape before SMOTE (315661,)
X shape after SMOTE (384224,)
```

```
y_sm.value_counts()
```

```
0    192112
1    192112
Name: ALCCAT, dtype: int64
```

Train-Test split

```
# split data into train n test
from sklearn.model_selection import train_test_split

# Outputs 4 variables. First two variables will be 80% (train) & 20%
(test) of x and Last two variables are 80% (train) & 20% (test) of y
x_train,x_test,y_train,y_test =
train_test_split(x_sm,y_sm,test_size=0.2)

print(x_train.shape,y_train.shape)
print(x_test.shape,y_test.shape)
```

```
(307379, 80) (307379,)
(76845, 80) (76845,)
```

```
# Copies of x_test and y_test for ALCCAT
x_test_ca = x_test.copy()
y_test_ca = y_test.copy()
```

CLASSIFICATION BY LOGISTIC REGRESSION

```
## fit the model
from sklearn.linear_model import LogisticRegression
logca=LogisticRegression()
logca.fit(x_train,y_train)
```

```
LogisticRegression()
```

```
logca.score(x_train,y_train)
```

```
0.7536298836290052
```

```
logca.score(x_test,y_test)
```

```
0.7512915609343483
```

```
## The final predicted value. 0 -> Never drank; 1 -> Drank
y_pred = logca.predict(x_test)
```

```
## Confusion Matrix
from sklearn.metrics import confusion_matrix
# y_test is actual and y_pred is predicted
tn,fp,fn,tp=confusion_matrix(y_test,y_pred).ravel()
sensitivity = tp/(tp+fn)
print("sensitivity is ", sensitivity) # Also 'Recall'
specificity = tn/(tn+fp)
print("specificity is ", specificity) # Also 'Precision'
```

```
sensitivity is  0.8137882755580907
specificity is  0.6883164384993207
```

```
# Classification report
import sklearn
print(sklearn.metrics.classification_report(y_test,y_pred))
```

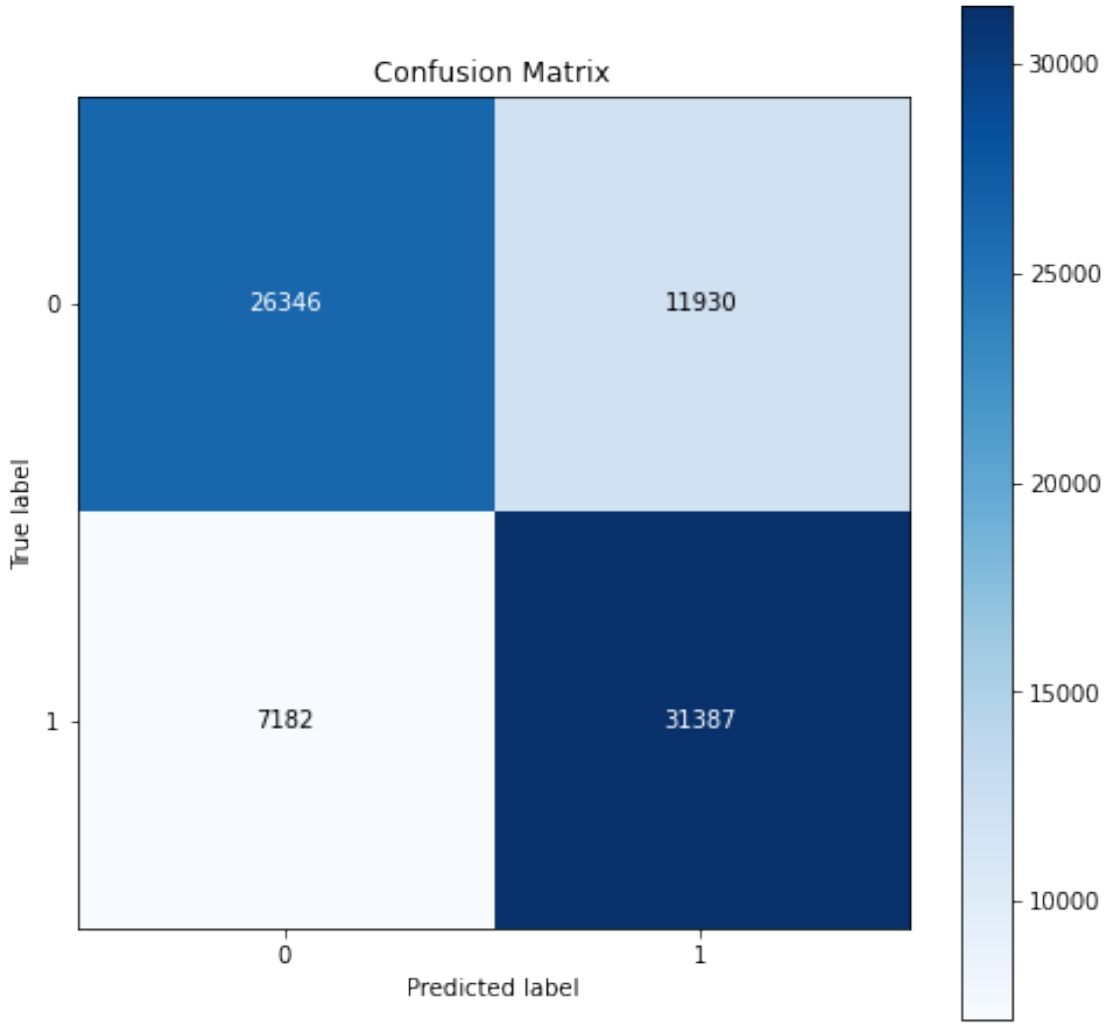
```
              precision    recall  f1-score   support

           0       0.79      0.69      0.73     38276
           1       0.72      0.81      0.77     38569

    accuracy                           0.75     76845
   macro avg       0.76      0.75      0.75     76845
weighted avg       0.76      0.75      0.75     76845
```

```
import scikitplot as skplt
```

```
skplt.metrics.plot_confusion_matrix(
```

```
    y_test,
    y_pred,
    figsize=(8,8))
```

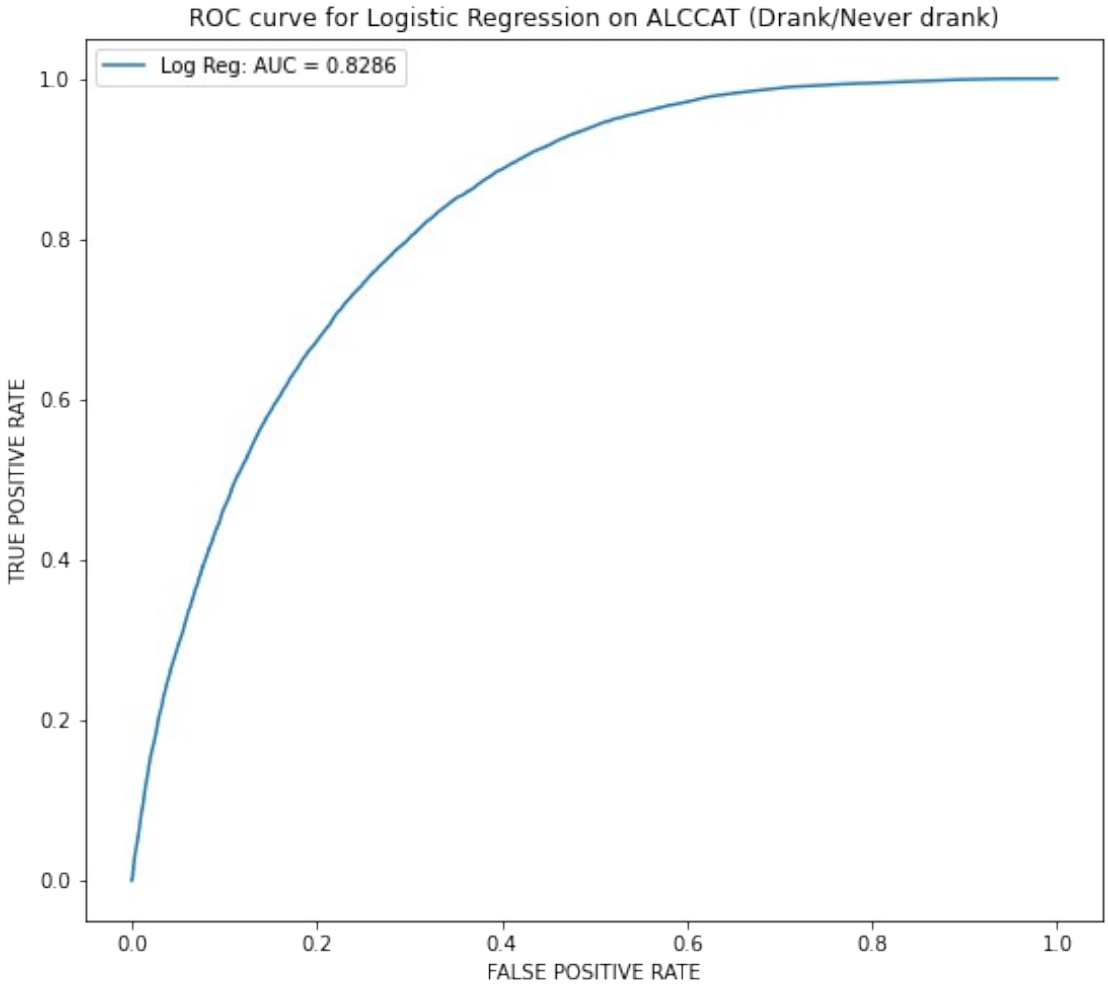
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4f10439d00>
```




```
import sklearn.metrics as metrics
# calculate the fpr and tpr for all thresholds of the classification
probs = logca.predict_proba(x_test)
preds = probs[:,1] # Extracting data under '1'th column of probs
fpr, tpr, threshold = metrics.roc_curve(y_test, preds)
roc_auc = metrics.auc(fpr, tpr)
print('Area under ROC curve:',roc_auc)
print('\n')

import matplotlib.pyplot as plt
plt.plot(fpr,tpr,label="Log Reg: AUC = "+str(round(roc_auc,4)))
plt.rcParams["figure.figsize"] = (10,10)
plt.title('ROC curve for Logistic Regression on ALCCAT (Drank/Never
```

```
drank)')
plt.xlabel('FALSE POSITIVE RATE')
plt.ylabel('TRUE POSITIVE RATE')
plt.legend(loc='upper left')
plt.show()
```

```
Area under ROC curve: 0.828621014383357
```



RANDOM FOREST CLASSIFICATION

Hyperparameter Tuning

Step 1: Constructing the Random Grid for Search

```
import numpy as np
from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.arange(2,101)]
# Number of features to consider at every split
```

```
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.arange(2,31,2)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [int(x) for x in np.arange(2,51,2)]
# Minimum number of samples required at each leaf node
min_samples_leaf = [int(x) for x in np.arange(2,51,2)]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
print(random_grid)
```

```
{'n_estimators': [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,
17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50,
51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67,
68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84,
85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100],
'max_features': ['auto', 'sqrt'], 'max_depth': [2, 4, 6, 8, 10, 12,
14, 16, 18, 20, 22, 24, 26, 28, 30, None], 'min_samples_split': [2, 4,
6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40,
42, 44, 46, 48, 50], 'min_samples_leaf': [2, 4, 6, 8, 10, 12, 14, 16,
18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48, 50],
'bootstrap': [True, False]}
```

Step 2 - Randomized Search & Model Building. Running this takes 2 hrs (approx)

```
# SKIP if you already have the hyperparameters
# Use the random grid to search for best hyperparameters
# First create the base model to tune
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available
cores
rfc_bestfit = RandomizedSearchCV(estimator = rfc, param_distributions
= random_grid, n_iter = 100, cv = 3, verbose=2, random_state=42,
n_jobs = -1)
# Fit the random search model
rfc_bestfit.fit(x_train, y_train)

rfc_bestfit.best_params_ # These are the parameters that we are
setting to our model to get the best possible accuracy
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits
```

```
# BEST PARAMETERS (for SMOTE):
# {'n_estimators': 71,
#  'min_samples_split': 22,
#  'min_samples_leaf': 4,
#  'max_features': 'sqrt',
#  'max_depth': 26,
#  'bootstrap': False}

# RUN THIS CHUNK IF YOU ALREADY KNOW THE PARAMETERS
# Model set with the optimal hyperparameters
from sklearn.ensemble import RandomForestClassifier

rf_tm_ca = RandomForestClassifier(n_estimators= 71,
 min_samples_split= 22,
 min_samples_leaf= 4,
 max_features= 'sqrt',
 max_depth= 26,
 bootstrap= False,random_state=42)

# Fit the random search model
rf_tm_ca = rf_tm_ca.fit(x_train, y_train)
```

Model Evaluation

```
# score for training data
rf_tm_ca.score(x_train,y_train)
```

```
0.7745682040737981
```

```
# score for test data
rf_tm_ca.score(x_test,y_test)
```

```
0.7464636606155247
```

```
## The final predicted value. 0 -> Never drank; 1 -> Drank
y_pred = rf_tm_ca.predict(x_test)

## Confusion Matrix
from sklearn.metrics import confusion_matrix
# y_test is actual and y_pred is predicted
tn,fp,fn,tp=confusion_matrix(y_test,y_pred).ravel()
sensitivity = tp/(tp+fn)
print("sensitivity is ", sensitivity) # Also 'Recall'
specificity = tn/(tn+fp)
print("specificity is ", specificity) # Also 'Precision'
```

```
sensitivity is  0.7920350540589592
specificity is  0.7005434214651479
```

```
# Accuracy
from sklearn import metrics
print('Accuracy of the model is:',
metrics.accuracy_score(y_test,y_pred))
```

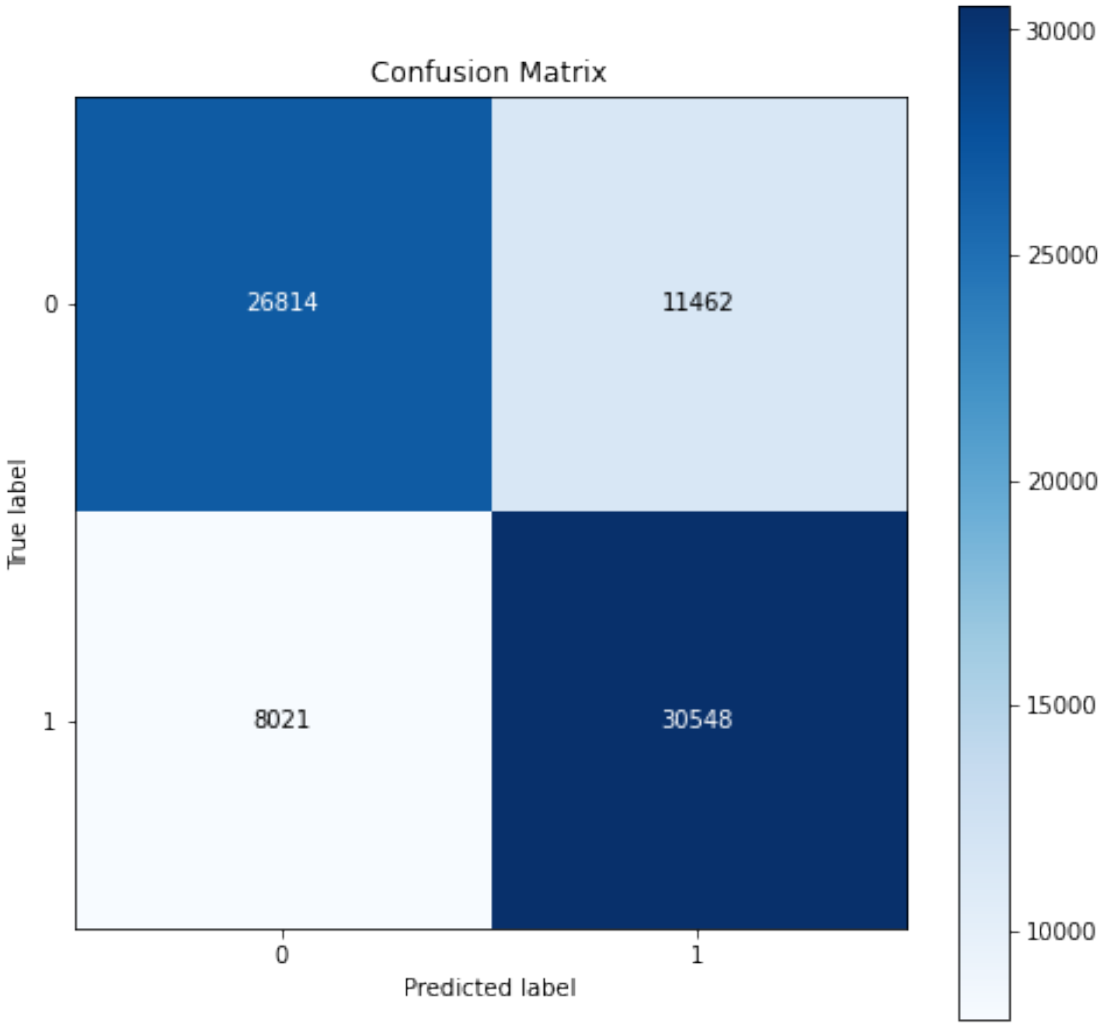
```
Accuracy of the model is: 0.7464636606155247
```

```
# Classification report
import sklearn
print(sklearn.metrics.classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.77      0.70      0.73     38276
           1       0.73      0.79      0.76     38569

    accuracy                           0.75     76845
   macro avg       0.75      0.75      0.75     76845
weighted avg       0.75      0.75      0.75     76845
```

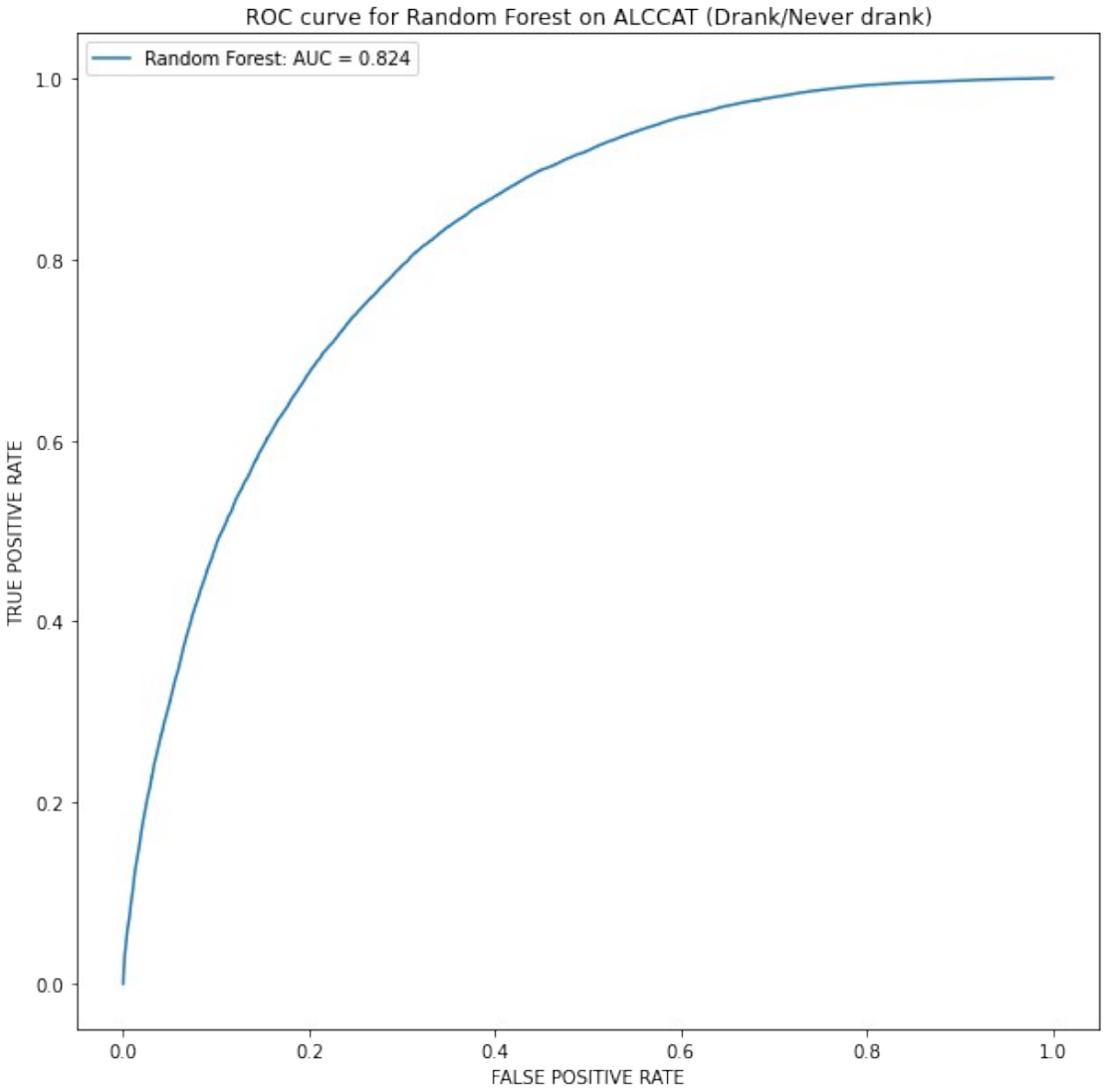
```
import scikitplot as skplt

skplt.metrics.plot_confusion_matrix(
    y_test,
    y_pred,
    figsize=(8,8))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4f102a7970>
```




```
import sklearn.metrics as metrics
# calculate the fpr and tpr for all thresholds of the classification
probs = rf_tm_ca.predict_proba(x_test)
preds = probs[:,1] # Extracting data under '1'th column of probs
fpr, tpr, threshold = metrics.roc_curve(y_test, preds)
roc_auc = metrics.auc(fpr, tpr)
print('Area under ROC curve:',roc_auc)
print('\n')

import matplotlib.pyplot as plt
plt.plot(fpr,tpr,label="Random Forest: AUC = "+str(round(roc_auc,4)))
plt.rcParams["figure.figsize"] = (10,10)
plt.title('ROC curve for Random Forest on ALCCAT (Drank/Never drank)')
plt.xlabel('FALSE POSITIVE RATE')
plt.ylabel('TRUE POSITIVE RATE')
plt.legend(loc='upper left')
plt.show()
```

```
Area under ROC curve: 0.8240041460953997
```



##CATBOOST CLASSIFIER

```
# To install scikit-optimize for skopt and catboost
!pip install scikit_optimize
!pip install catboost

# Libraries
import os
import pandas as pd
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn import metrics
```

```
import numpy as np
import pandas as pd
from time import time
import pprint
import joblib
import warnings
warnings.filterwarnings("ignore")

# Classifiers
from catboost import CatBoostClassifier

# Model selection
from sklearn.model_selection import StratifiedKFold

# Metrics
from sklearn.metrics import roc_auc_score
from sklearn.metrics import make_scorer

# Scikit optimize functions
from skopt import BayesSearchCV
from skopt.callbacks import DeadlineStopper, VerboseCallback,
DeltaXStopper
from skopt.space import Real, Categorical, Integer
from time import time

# Performance reporting for model
# Reporting util for different optimizers
def report_perf(optimizer, X, y, title, callbacks=None):
    """
    A wrapper for measuring time and performances of different
optmizers

    """
    start = time()
    if callbacks:
        optimizer.fit(X, y, callback=callbacks)
    else:
        optimizer.fit(X, y)
    d=pd.DataFrame(optimizer.cv_results_)
    best_score = optimizer.best_score_
    best_score_std = d.iloc[optimizer.best_index_].std_test_score
    best_params = optimizer.best_params_
    print((title + " took %.2f seconds,  candidates checked: %d, best
CV score: %.3f "
           +u"\u00B1"+" %.3f") % (time() - start,

len(optimizer.cv_results_['params']),
                                  best_score,
                                  best_score_std))
    print('Best parameters:')
```

```
    pprint.pprint(best_params)
    print()
    return best_params

# Using Stratified K-Fold
roc_auc = make_scorer(roc_auc_score, greater_is_better=True,
needs_threshold=True)
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=0)

# Catboost classifier
clf = CatBoostClassifier(thread_count=2,
                         loss_function='Logloss',

                         od_type = 'Iter',
                         verbose= False
                        )

# Defining your search space
search_spaces = {'iterations': Integer(10, 1000),
                 'depth': Integer(1, 8),
                 'learning_rate': Real(0.01, 1.0, 'log-uniform'),
                 'random_strength': Real(1e-9, 10, 'log-uniform'),
                 'bagging_temperature': Real(0.0, 1.0),
                 'border_count': Integer(1, 255),
                 'l2_leaf_reg': Integer(2, 30),
                 'scale_pos_weight':Real(0.01, 1.0, 'uniform')}

# Setting up BayesSearchCV
opt = BayesSearchCV(clf,
                    search_spaces,
                    scoring=roc_auc,
                    cv=skf,
                    n_iter=100,
                    n_jobs=1,  # use just 1 job with CatBoost in order
to avoid segmentation fault
                    return_train_score=False,
                    refit=True,
                    optimizer_kwargs={'base_estimator': 'GP'},
                    random_state=42)

# SKIP if you already have the hyperparameters
# Tuning for CatBoost using Baye's Search. This takes around 45 mins
to run
best_params_a = report_perf(opt,x_train, y_train,'CatBoost',
                            callbacks=[VerboseCallback(100),
                                       DeadlineStopper(60*10)])

Iteration No: 1 started. Searching for the next optimal point.
Iteration No: 1 ended. Search finished for the next optimal point.
Time taken: 154.3356
Function value obtained: -0.8314
```

```
Current minimum: -0.8314
Iteration No: 2 started. Searching for the next optimal point.
Iteration No: 2 ended. Search finished for the next optimal point.
Time taken: 257.1996
Function value obtained: -0.8101
Current minimum: -0.8314
Iteration No: 3 started. Searching for the next optimal point.
CatBoost took 449.26 seconds,  candidates checked: 2, best CV score:
0.831 ± 0.002
Best parameters:
OrderedDict([('bagging_temperature', 0.41010395885331385),
             ('border_count', 186),
             ('depth', 8),
             ('iterations', 323),
             ('l2_leaf_reg', 21),
             ('learning_rate', 0.0673344419215237),
             ('random_strength', 3.230824361824754e-06),
             ('scale_pos_weight', 0.7421091918485163)])


best_params_a # These are the best parameters

OrderedDict([('bagging_temperature', 0.41010395885331385),
             ('border_count', 186),
             ('depth', 8),
             ('iterations', 323),
             ('l2_leaf_reg', 21),
             ('learning_rate', 0.0673344419215237),
             ('random_strength', 3.230824361824754e-06),
             ('scale_pos_weight', 0.7421091918485163)])

# BEST PARAMETERS
# OrderedDict([('bagging_temperature', 0.41010395885331385),
#              ('border_count', 186),
#              ('depth', 8),
#              ('iterations', 323),
#              ('l2_leaf_reg', 21),
#              ('learning_rate', 0.0673344419215237),
#              ('random_strength', 3.230824361824754e-06),
#              ('scale_pos_weight', 0.7421091918485163)])

# DO THIS, IF YOU HAVE ALREADY RUN THE TUNING CODE
import collections
from collections import OrderedDict
best_params_a = [('bagging_temperature', 0.41010395885331385),
             ('border_count', 186),
             ('depth', 8),
             ('iterations', 323),
             ('l2_leaf_reg', 21),
             ('learning_rate', 0.0673344419215237),
             ('random_strength', 3.230824361824754e-06),
```

```
            ('scale_pos_weight', 0.7421091918485163)]
```

```
best_params_a = OrderedDict(best_params_a)

best_params_a['iterations']=1000

# Building model with optimal parameters
cat_tm_ca = CatBoostClassifier(**best_params_a,verbose=False)
cat_tm_ca = cat_tm_ca.fit(x_train,y_train)

# Prediction
y_pred = cat_tm_ca.predict(x_test)
y_pred_cat_tm_ca = y_pred.copy()

# model score for training data
cat_tm_ca.score(x_train,y_train)
```

```
0.7698639139303596
```

```
# model score for test data
cat_tm_ca.score(x_test,y_test)
```

```
0.7536859912811503
```

```
# Accuracy
from sklearn import metrics
print('Accuracy of the model is:',
metrics.accuracy_score(y_test,y_pred))
```

```
Accuracy of the model is: 0.7536859912811503
```

```
## Confusion Matrix
from sklearn.metrics import confusion_matrix
# y_test is actual and y_pred is predicted
tn,fp,fn,tp=confusion_matrix(y_test,y_pred).ravel()
sensitivity = tp/(tp+fn)
print("sensitivity is ", sensitivity) # Also 'Recall'
specificity = tn/(tn+fp)
print("specificity is ", specificity) # Also 'Precision'
```

```
sensitivity is  0.7353055562757655
specificity is  0.7722071271815236
```

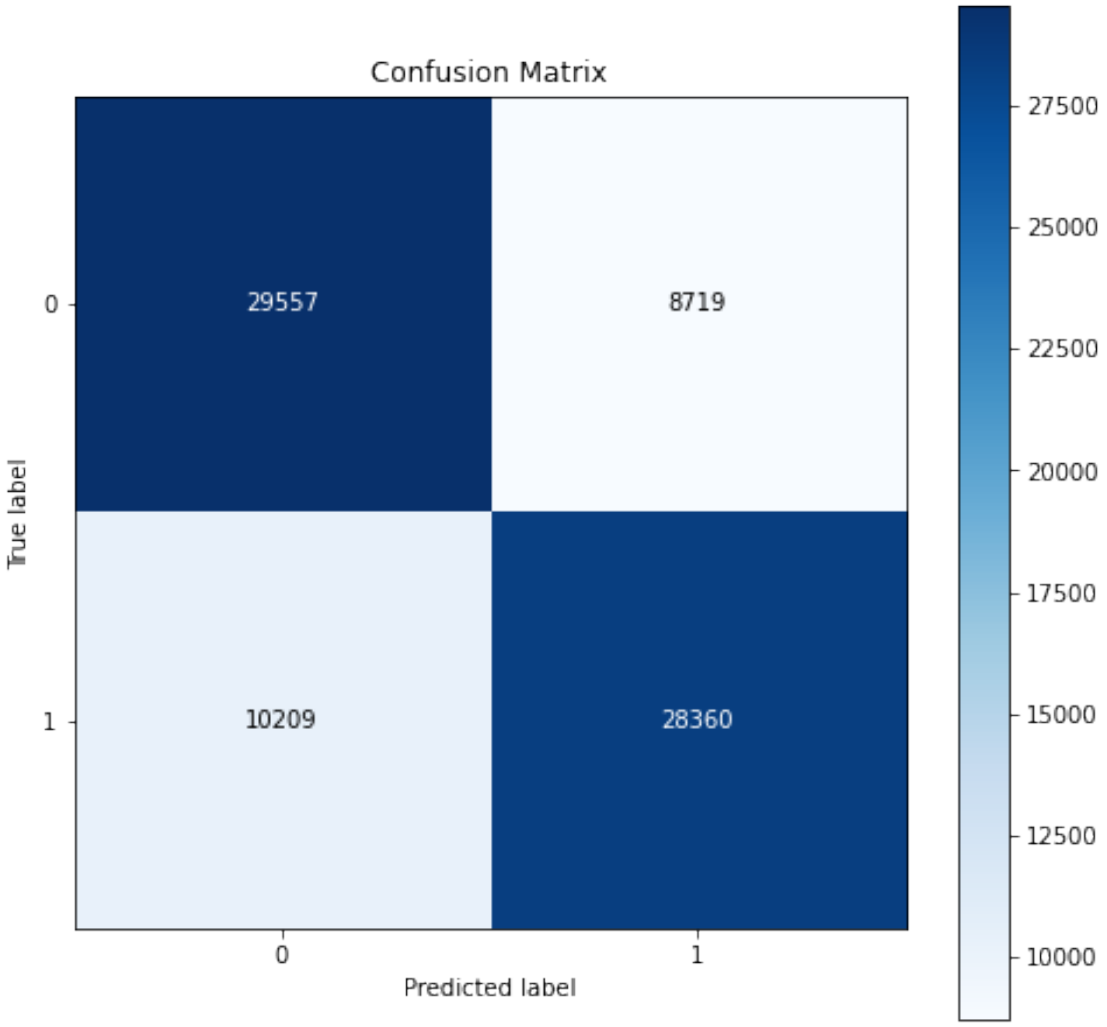
```
# Classification report
import sklearn
print(sklearn.metrics.classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.74      0.77      0.76     38276
           1       0.76      0.74      0.75     38569

    accuracy                           0.75     76845
```

```
   macro avg       0.75      0.75      0.75     76845
weighted avg       0.75      0.75      0.75     76845
```

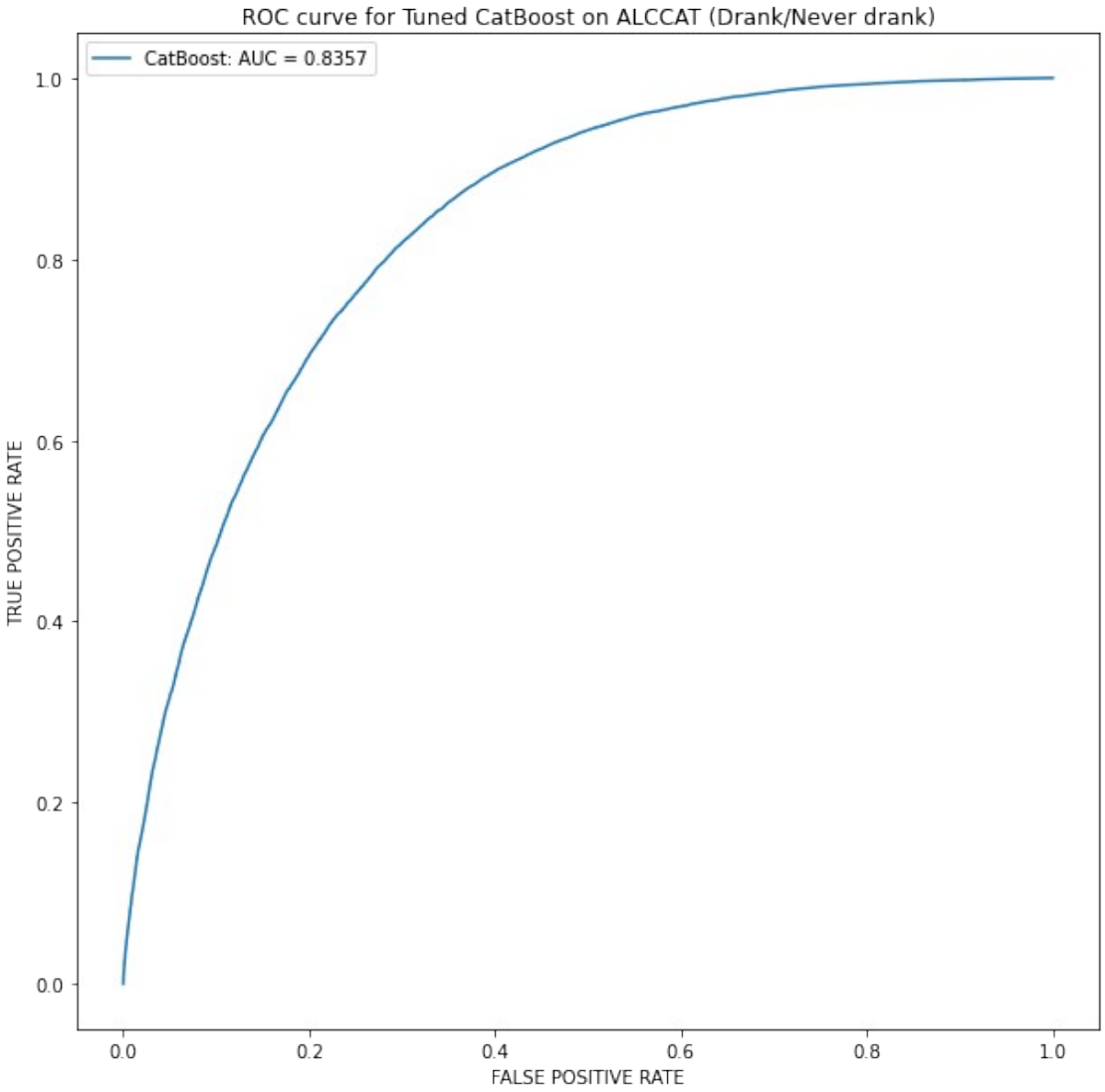
```
# CONFUSION MATRIX
!pip install scikit-plot
import scikitplot as skplt

skplt.metrics.plot_confusion_matrix(
    y_test,
    y_pred,
    figsize=(8,8))
```

```
Looking in indexes: https://pypi.org/simple, https://us-
python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: scikit-plot in
/usr/local/lib/python3.8/dist-packages (0.3.7)
Requirement already satisfied: scipy>=0.9 in
/usr/local/lib/python3.8/dist-packages (from scikit-plot) (1.7.3)
Requirement already satisfied: joblib>=0.10 in
/usr/local/lib/python3.8/dist-packages (from scikit-plot) (1.2.0)
Requirement already satisfied: matplotlib>=1.4.0 in
/usr/local/lib/python3.8/dist-packages (from scikit-plot) (3.2.2)
Requirement already satisfied: scikit-learn>=0.18 in
/usr/local/lib/python3.8/dist-packages (from scikit-plot) (1.0.2)
Requirement already satisfied: numpy>=1.11 in
/usr/local/lib/python3.8/dist-packages (from matplotlib>=1.4.0-
>scikit-plot) (1.21.6)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.8/dist-packages (from matplotlib>=1.4.0-
>scikit-plot) (1.4.4)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!
=2.1.6,>=2.0.1 in /usr/local/lib/python3.8/dist-packages (from
matplotlib>=1.4.0->scikit-plot) (3.0.9)
Requirement already satisfied: cycler>=0.10 in
/usr/local/lib/python3.8/dist-packages (from matplotlib>=1.4.0-
>scikit-plot) (0.11.0)
Requirement already satisfied: python-dateutil>=2.1 in
/usr/local/lib/python3.8/dist-packages (from matplotlib>=1.4.0-
>scikit-plot) (2.8.2)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.8/dist-packages (from python-dateutil>=2.1-
>matplotlib>=1.4.0->scikit-plot) (1.15.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.8/dist-packages (from scikit-learn>=0.18-
>scikit-plot) (3.1.0)

<matplotlib.axes._subplots.AxesSubplot at 0x7f4f1092f310>
```




```
import sklearn.metrics as metrics
# calculate the fpr and tpr for all thresholds of the classification
probs = cat_tm_ca.predict_proba(x_test)
preds = probs[:,1] # Extracting data under '1'th column of probs
fpr, tpr, threshold = metrics.roc_curve(y_test, preds)
roc_auc = metrics.auc(fpr, tpr)
print('Area under ROC curve:',roc_auc)
print('\n')

import matplotlib.pyplot as plt
plt.plot(fpr,tpr,label="CatBoost: AUC = "+str(round(roc_auc,4)))
plt.rcParams["figure.figsize"] = (9,8)
plt.title('ROC curve for Tuned CatBoost on ALCCAT (Drank/Never
drank)')
plt.xlabel('FALSE POSITIVE RATE')
plt.ylabel('TRUE POSITIVE RATE')
plt.legend(loc='upper left')
plt.show()
```

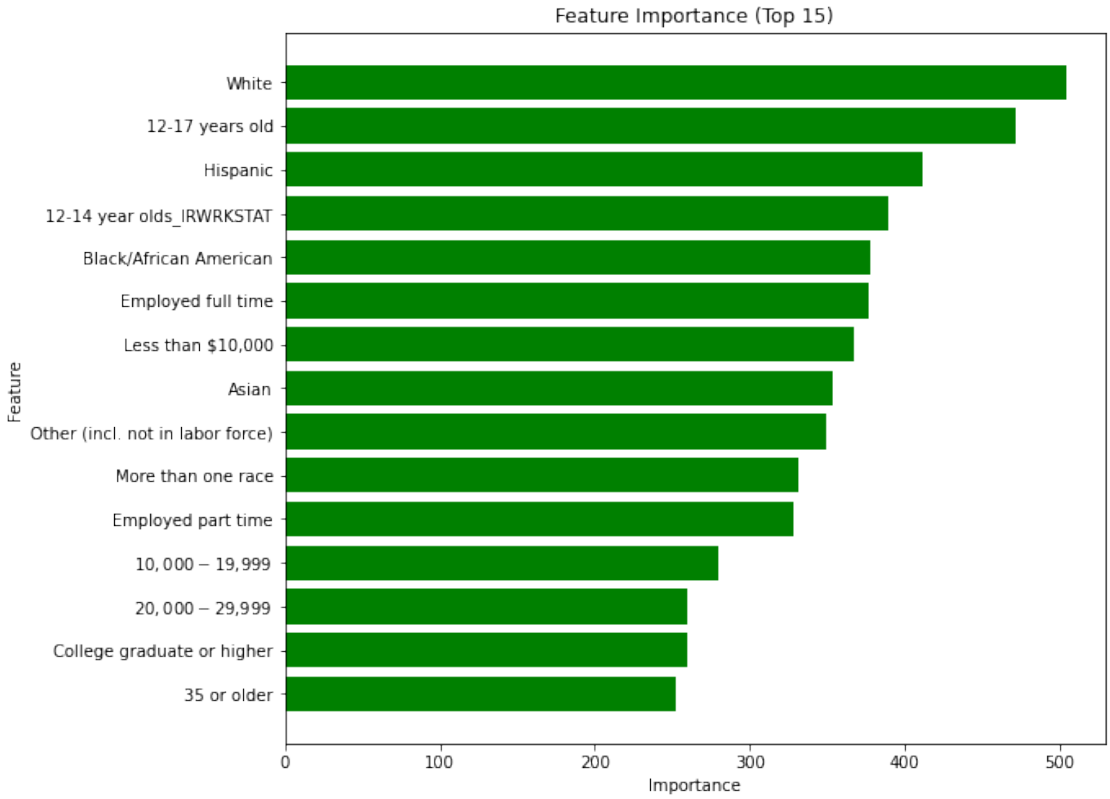
```
Area under ROC curve: 0.8357242329660786
```




```
# Feature importance in prediction
import sklearn.metrics as metrics
bcfit = cat_tm_ca
imp_feature = bcfit.feature_importances_
columns = x.columns
clf_co = pd.Series(imp_feature,columns)

## top features
# Feature importance graph
# Top 15 most imp features
dfimp =
pd.DataFrame(clf_co.sort_values(ascending=False)).reset_index()
dfimp.columns.values[0] = 'Feature'
```

```
dfimp.columns.values[1] = 'Importance'
dfimp['Importance'] = round(dfimp['Importance']*100,2)
dplt1 = dfimp.head(15)
dplt2 = dplt1.copy()
dplt2.sort_values(by='Importance',ascending=True,inplace=True)

plt.barh(dplt2['Feature'],dplt2['Importance'],color='green')
plt.title('Feature Importance (Top 15)')
plt.ylabel('Feature')
plt.xlabel('Importance')
print(dplt1)
```

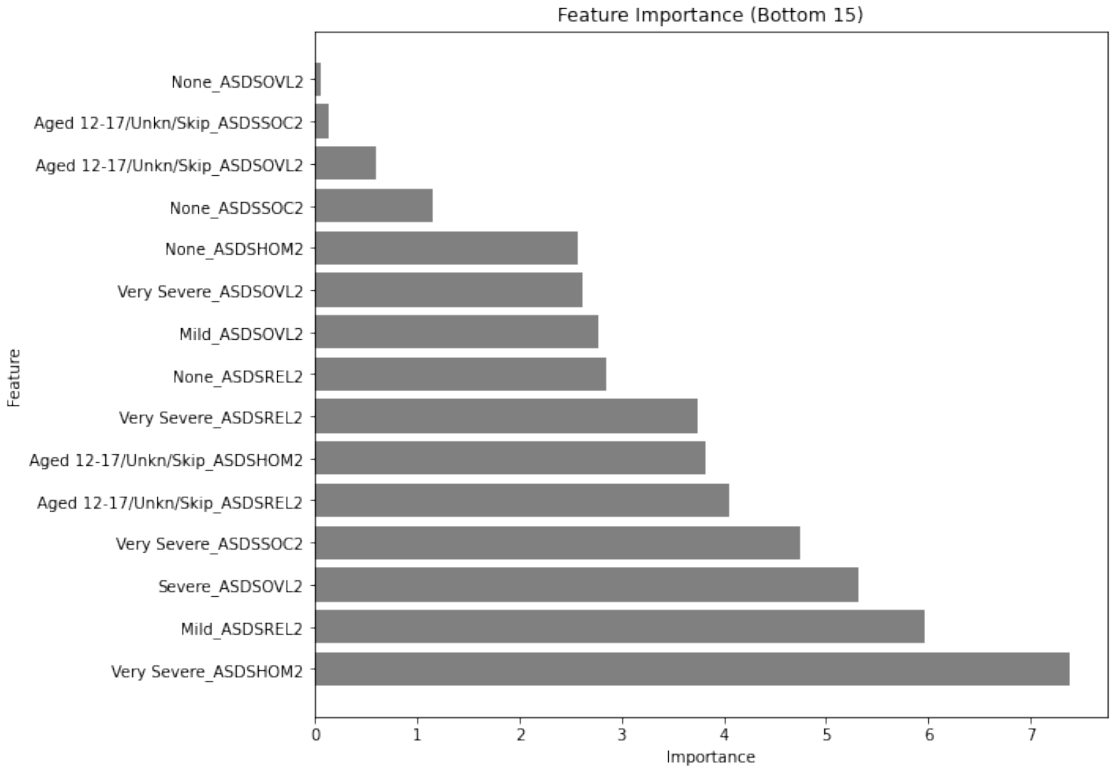
```
                             Feature  Importance
0                              White      504.09
1                      12-17 years old      471.65
2                             Hispanic      411.78
3           12-14 year olds_IRWRKSTAT      389.68
4              Black/African American      378.01
5                  Employed full time      377.03
6                   Less than $10,000      367.51
7                               Asian      353.44
8   Other (incl. not in labor force)      349.62
9                  More than one race      331.05
10                 Employed part time      328.24
11                    $10,000-$19,999      279.65
12                    $20,000-$29,999      260.10
13         College graduate or higher      259.55
14                         35 or older      252.71
```




```
## bottom features
# Feature importance graph
# Bottom 15 (15 least imp features)
dfimp =
pd.DataFrame(clf_co.sort_values(ascending=False)).reset_index()
dfimp.columns.values[0] = 'Feature'
dfimp.columns.values[1] = 'Importance'
dfimp['Importance'] = round(dfimp['Importance']*100,2)
dplt1 = dfimp.tail(15)
dplt2 = dplt1.copy()
dplt2.sort_values(by='Importance',ascending=False,inplace=True)

plt.barh(dplt2['Feature'],dplt2['Importance'],color='gray')
plt.title('Feature Importance (Bottom 15)')
plt.ylabel('Feature')
plt.xlabel('Importance')
print(dplt2.sort_values(by='Importance',ascending=True))
```

```
                          Feature  Importance
79                  None_ASDSOVL2        0.06
78  Aged 12-17/Unkn/Skip_ASDSSOC2        0.14
77  Aged 12-17/Unkn/Skip_ASDSOVL2        0.59
76                  None_ASDSSOC2        1.15
75                  None_ASDSHOM2        2.57
74           Very Severe_ASDSOVL2        2.62
73                  Mild_ASDSOVL2        2.78
```

```
72                  None_ASDSREL2        2.85
71           Very Severe_ASDSREL2        3.74
70  Aged 12-17/Unkn/Skip_ASDSHOM2        3.82
69  Aged 12-17/Unkn/Skip_ASDSREL2        4.05
68           Very Severe_ASDSSOC2        4.74
67                Severe_ASDSOVL2        5.32
66                  Mild_ASDSREL2        5.97
65           Very Severe_ASDSHOM2        7.38
```



For ALLDGSCAT (Consumed/Not consumed Alldrugs)

Y data

```
# create target variable
y = df['ALLDGSCAT']
```

SMOTE

```
from imblearn.over_sampling import SMOTE
smote =SMOTE(random_state=42)
x_sm,y_sm=smote.fit_resample(x,y)

print("X shape before SMOTE",x.shape)
print("X shape after SMOTE",x_sm.shape)

print("y shape before SMOTE",y.shape)
print("X shape after SMOTE",y_sm.shape)
```

```
X shape before SMOTE (315661, 80)
X shape after SMOTE (499482, 80)
y shape before SMOTE (315661,)
X shape after SMOTE (499482,)
```

Train-test split

```
# split data into train n test
from sklearn.model_selection import train_test_split

# Outputs 4 variables. First two variables will be 80% (train) & 20%
(test) of x and Last two variables are 80% (train) & 20% (test) of y
x_train,x_test,y_train,y_test =
train_test_split(x_sm,y_sm,test_size=0.2)

print(x_train.shape,y_train.shape)
print(x_test.shape,y_test.shape)
```

```
(399585, 80) (399585,)
(99897, 80) (99897,)
```

```
# creating copies of x_test & y_test for ALLDGSCAT
x_test_cd = x_test.copy()
y_test_cd = y_test.copy()
```

RANDOM FOREST CLASSIFICATION

Hyperparameter Tuning

Step 1: Random Grid for Search space

```
import numpy as np
from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.arange(2,101)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.arange(2,31,2)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [int(x) for x in np.arange(2,51,2)]
# Minimum number of samples required at each leaf node
min_samples_leaf = [int(x) for x in np.arange(2,51,2)]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
```

```
               'bootstrap': bootstrap}
print(random_grid)
```

Step 2 - Randomized Search & Model Building

```
# SKIP if you already have the hyperparameters
# Use the random grid to search for best hyperparameters
# First create the base model to tune
from sklearn.ensemble import RandomForestClassifier
rfca = RandomForestClassifier()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available
cores
rfca_bestfit = RandomizedSearchCV(estimator = rfca,
param_distributions = random_grid, n_iter = 100, cv = 3, verbose=2,
random_state=42, n_jobs = -1)
# Fit the random search model
rfca_bestfit.fit(x_train, y_train)

rfca_bestfit.best_params_ # These are the parameters that we are
setting to our model to get the best possible accuracy

# BEST PARAMETERS (for SMOTE):
# {'n_estimators': 71,
#  'min_samples_split': 22,
#  'min_samples_leaf': 4,
#  'max_features': 'sqrt',
#  'max_depth': 26,
#  'bootstrap': False}

# Run this if you already have the best parameters
# Model Building based on best params (Don't have to run this if u
have run the above code)
from sklearn.ensemble import RandomForestClassifier
rf_tm_cd = RandomForestClassifier(n_estimators= 71,
 min_samples_split= 22,
 min_samples_leaf= 4,
 max_features= 'sqrt',
 max_depth= 26,
 bootstrap= False, random_state=42)


# Fit the random search model
rf_tm_cd.fit(x_train, y_train)
```

```
RandomForestClassifier(bootstrap=False, max_depth=26,
max_features='sqrt',
                       min_samples_leaf=4, min_samples_split=22,
                       n_estimators=71, random_state=42)
```

Model Evaluation

```
# score for training data
rf_tm_cd.score(x_train,y_train)
```

```
0.7469674787592127
```

```
# score for test data
rf_tm_cd.score(x_test,y_test)
```

```
0.7233550557073786
```

```
## The final predicted value. 0 -> Never used Alcohol; 1 -> Ever used
Alcohol
y_pred = rf_tm_cd.predict(x_test)
```

```
## Confusion Matrix
from sklearn.metrics import confusion_matrix
# y_test is actual and y_pred is predicted
tn,fp,fn,tp=confusion_matrix(y_test,y_pred).ravel()
sensitivity = tp/(tp+fn)
print("sensitivity is ", sensitivity)
specificity = tn/(tn+fp)
print("specificity is ", specificity)
```

```
sensitivity is  0.7566746602717825
specificity is  0.6899131516136149
```

```
# Accuracy
from sklearn import metrics
print('Accuracy of the model is:',
metrics.accuracy_score(y_test,y_pred))
```

```
Accuracy of the model is: 0.7233550557073786
```

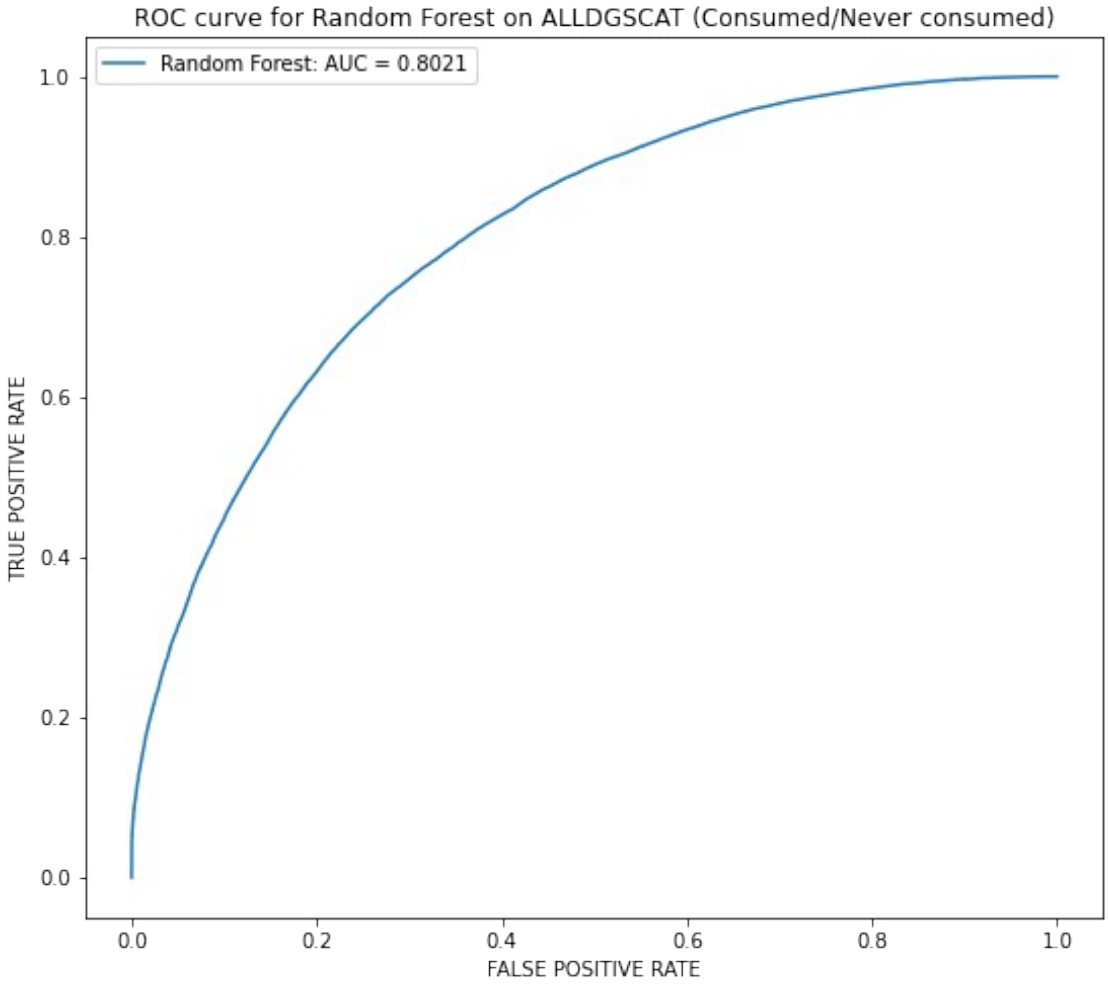
```
# Classification report
import sklearn
print(sklearn.metrics.classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.74      0.69      0.71     49857
           1       0.71      0.76      0.73     50040

    accuracy                           0.72     99897
   macro avg       0.72      0.72      0.72     99897
weighted avg       0.72      0.72      0.72     99897
```

```
import sklearn.metrics as metrics
# calculate the fpr and tpr for all thresholds of the classification
probs = rf_tm_cd.predict_proba(x_test)
preds = probs[:,1] # Extracting data under '1'th column of probs
fpr, tpr, threshold = metrics.roc_curve(y_test, preds)
```

```
roc_auc = metrics.auc(fpr, tpr)
print('Area under ROC curve:',roc_auc)
print('\n')

import matplotlib.pyplot as plt
plt.plot(fpr,tpr,label="Random Forest: AUC = "+str(round(roc_auc,4)))
plt.rcParams["figure.figsize"] = (9,8)
plt.title('ROC curve for Random Forest on ALLDGSCAT (Consumed/Never
consumed)')
plt.xlabel('FALSE POSITIVE RATE')
plt.ylabel('TRUE POSITIVE RATE')
plt.legend(loc='upper left')
plt.show()
```

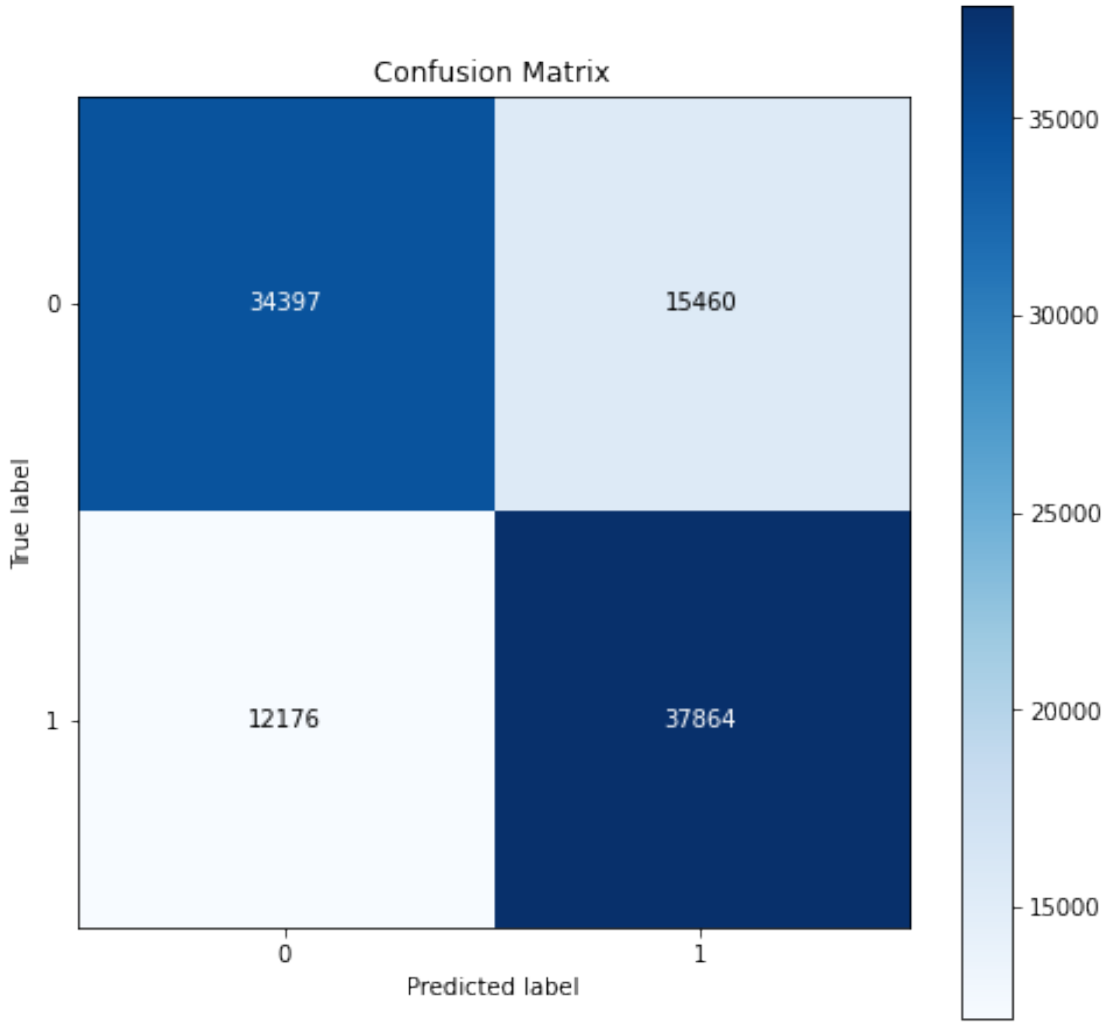
```
Area under ROC curve: 0.8020628970879096
```




```
# CONFUSION MATRIX
import scikitplot as skplt
```

```
skplt.metrics.plot_confusion_matrix(
    y_test,
    y_pred,
    figsize=(8,8))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4f10d11790>
```



CLASSIFICATION BY LOGISTIC REGRESSION

```
## fit the model
from sklearn.linear_model import LogisticRegression
logcd=LogisticRegression()
logcd.fit(x_train,y_train)
```

```
LogisticRegression()
```

```
# model score for training data
logcd.score(x_train,y_train)
```

```
0.7255702791646333
```

```
# model score for test data
logcd.score(x_test,y_test)
```

```
0.7255473137331452
```

```
## The final predicted value. 0 -> Not consumed Drugs; 1 -> Consumed Drugs
y_pred = logcd.predict(x_test)
```

```
## Confusion Matrix
from sklearn.metrics import confusion_matrix
# y_test is actual and y_pred is predicted
tn,fp,fn,tp=confusion_matrix(y_test,y_pred).ravel()
sensitivity = tp/(tp+fn)
print("sensitivity is ", sensitivity) # Also 'Recall'
specificity = tn/(tn+fp)
print("specificity is ", specificity) # Also 'Precision'
```

```
sensitivity is  0.6862909672262191
specificity is  0.7649477505666206
```

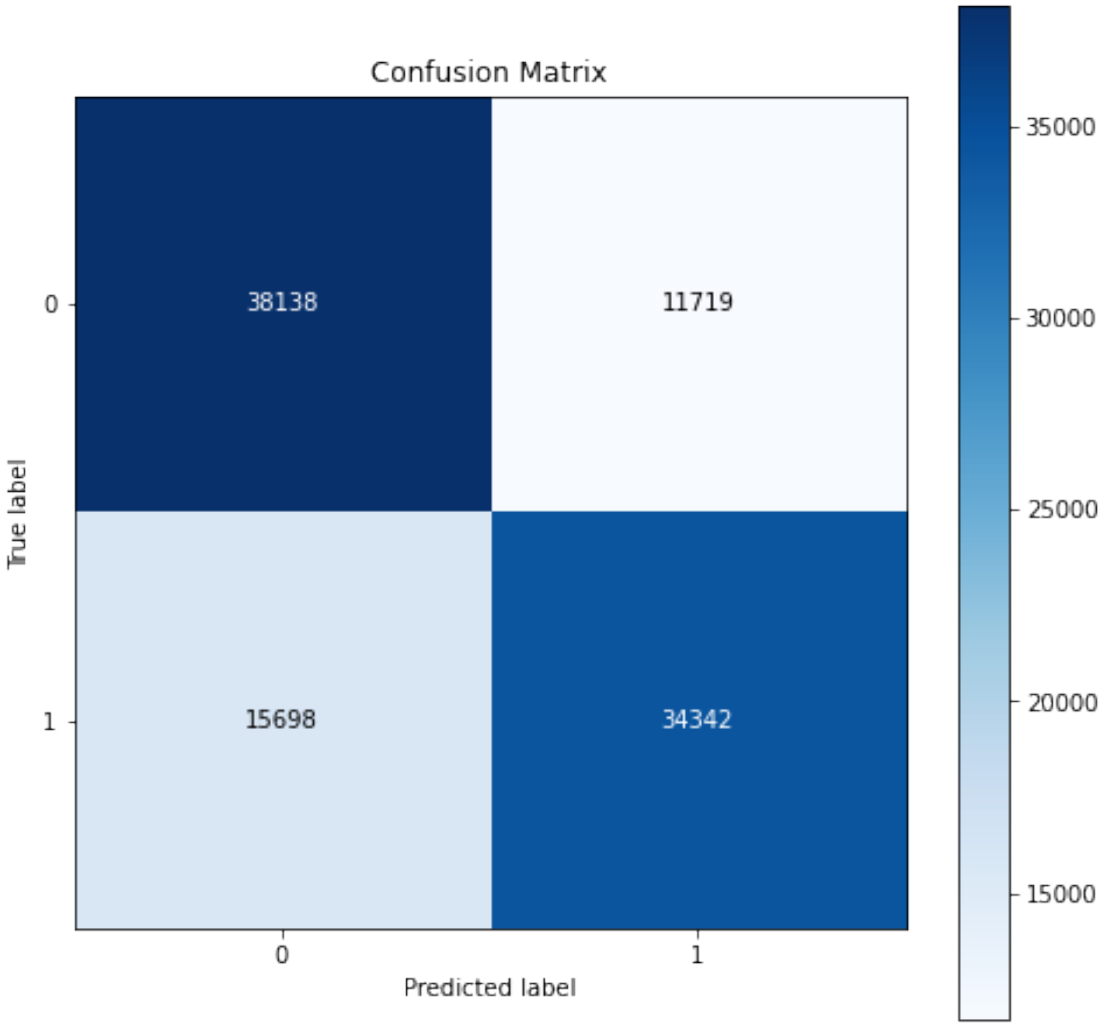
```
# Classification report
import sklearn
print(sklearn.metrics.classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.71      0.76      0.74     49857
           1       0.75      0.69      0.71     50040

    accuracy                           0.73     99897
   macro avg       0.73      0.73      0.73     99897
weighted avg       0.73      0.73      0.73     99897
```

```
!pip install scikit-plot
```

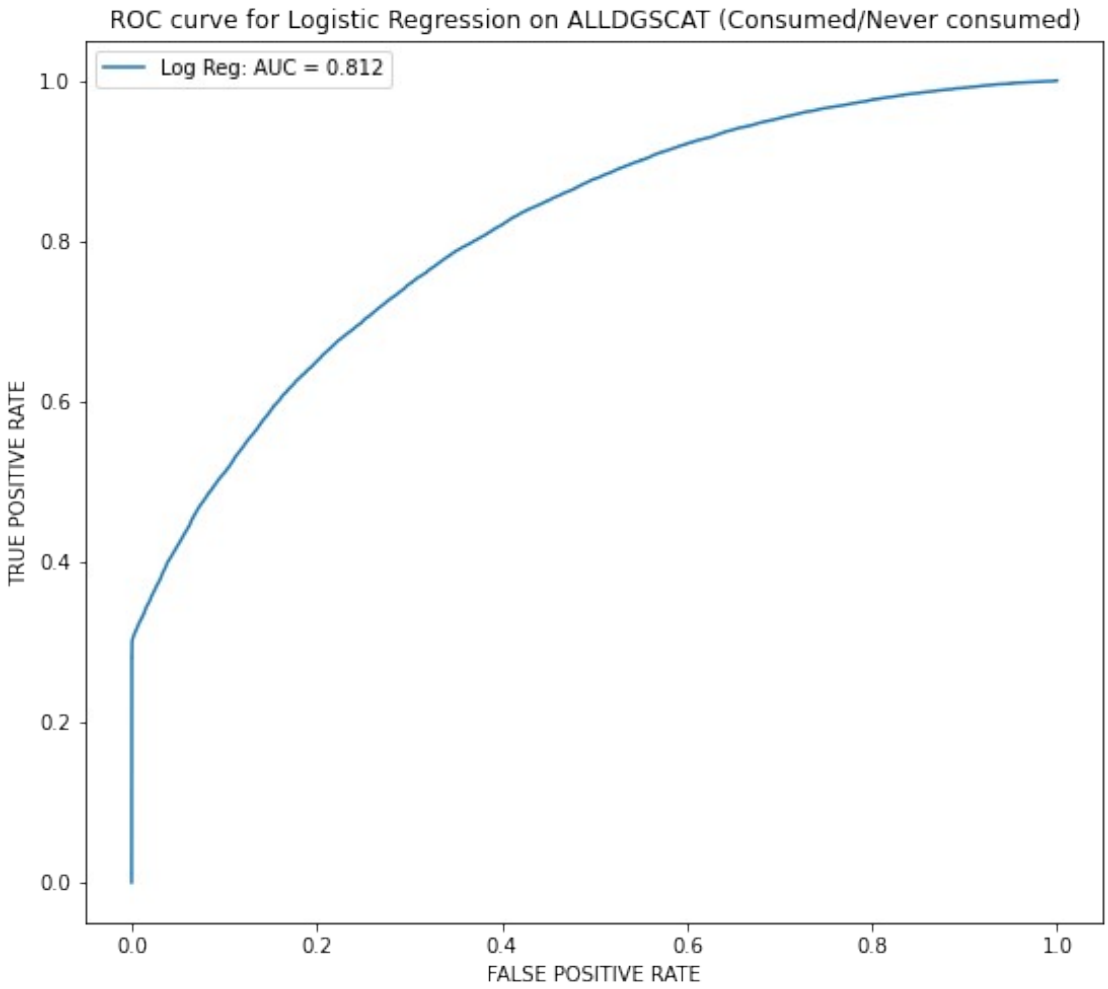
```
# CONFUSION MATRIX
import scikitplot as skplt

skplt.metrics.plot_confusion_matrix(
    y_test,
    y_pred,
    figsize=(8,8))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4f1096bfa0>
```




```
import sklearn.metrics as metrics
# calculate the fpr and tpr for all thresholds of the classification
probs = logcd.predict_proba(x_test)
preds = probs[:,1] # Extracting data under '1'th column of probs
fpr, tpr, threshold = metrics.roc_curve(y_test, preds)
roc_auc = metrics.auc(fpr, tpr)
print('Area under ROC curve:',roc_auc)
print('\n')

import matplotlib.pyplot as plt
plt.plot(fpr,tpr,label="Log Reg: AUC = "+str(round(roc_auc,4)))
plt.rcParams["figure.figsize"] = (9,8)
plt.title('ROC curve for Logistic Regression on ALLDGSCAT
(Consumed/Never consumed)')
plt.xlabel('FALSE POSITIVE RATE')
plt.ylabel('TRUE POSITIVE RATE')
plt.legend(loc='upper left')
plt.show()
```

```
Area under ROC curve: 0.8119740050469203
```



##CATBOOST CLASSIFIER

```
# Run this to install skopt and catboost
!pip install scikit_optimize
!pip install catboost

# Libraries
import os
import pandas as pd
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn import metrics
import numpy as np
import pandas as pd
from time import time
```

```
import pprint
import joblib
import warnings
warnings.filterwarnings("ignore")

# Classifiers
from catboost import CatBoostClassifier

# Model selection
from sklearn.model_selection import StratifiedKFold

# Metrics
from sklearn.metrics import roc_auc_score
from sklearn.metrics import make_scorer

# Scikit optimize functions
from skopt import BayesSearchCV
from skopt.callbacks import DeadlineStopper, VerboseCallback,
DeltaXStopper
from skopt.space import Real, Categorical, Integer
from time import time

# Performance reporting for model
# Reporting util for different optimizers
def report_perf(optimizer, X, y, title, callbacks=None):
    """
    A wrapper for measuring time and performances of different
optmizers

    optimizer = a sklearn or a skopt optimizer
    X = the training set
    y = our target
    title = a string label for the experiment
    """
    start = time()
    if callbacks:
        optimizer.fit(X, y, callback=callbacks)
    else:
        optimizer.fit(X, y)
    d=pd.DataFrame(optimizer.cv_results_)
    best_score = optimizer.best_score_
    best_score_std = d.iloc[optimizer.best_index_].std_test_score
    best_params = optimizer.best_params_
    print((title + " took %.2f seconds,  candidates checked: %d, best
CV score: %.3f "
           +u"\u00B1"+" %.3f") % (time() - start,

len(optimizer.cv_results_['params']),
                                  best_score,
                                  best_score_std))
```

```
    print('Best parameters:')
    pprint.pprint(best_params)
    print()
    return best_params

# Using Stratified K Fold
roc_auc = make_scorer(roc_auc_score, greater_is_better=True,
needs_threshold=True)
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=0)

# CatBoost classifier model
clf = CatBoostClassifier(thread_count=2,
                         loss_function='Logloss',

                         od_type = 'Iter',
                         verbose= False
                        )

# Defining your search space
search_spaces = {'iterations': Integer(10, 1000),
                 'depth': Integer(1, 8),
                 'learning_rate': Real(0.01, 1.0, 'log-uniform'),
                 'random_strength': Real(1e-9, 10, 'log-uniform'),
                 'bagging_temperature': Real(0.0, 1.0),
                 'border_count': Integer(1, 255),
                 'l2_leaf_reg': Integer(2, 30),
                 'scale_pos_weight':Real(0.01, 1.0, 'uniform')}

# Setting up BayesSearchCV
# use just 1 job with CatBoost in order to avoid segmentation fault
opt = BayesSearchCV(clf,
                    search_spaces,
                    scoring=roc_auc,
                    cv=skf,
                    n_iter=100,
                    n_jobs=1,
                    return_train_score=False,
                    refit=True,
                    optimizer_kwargs={'base_estimator': 'GP'},
                    random_state=42)

# Running this takes around 45 mins
# Tuning for CatBoost using Baye's Search
best_params = report_perf(opt, x_train, y_train,'CatBoost',
                          callbacks=[VerboseCallback(100),
                                     DeadlineStopper(60*10)])

best_params # These are the best parameters

OrderedDict([('bagging_temperature', 0.41010395885331385),
             ('border_count', 186),
             ('depth', 8),
```

```
             ('iterations', 1000),
             ('l2_leaf_reg', 21),
             ('learning_rate', 0.0673344419215237),
             ('random_strength', 3.230824361824754e-06),
             ('scale_pos_weight', 0.7421091918485163)])

# BEST PARAMETERS
# OrderedDict([('bagging_temperature', 0.41010395885331385),
#              ('border_count', 186),
#              ('depth', 8),
#              ('iterations', 1000),
#              ('l2_leaf_reg', 21),
#              ('learning_rate', 0.0673344419215237),
#              ('random_strength', 3.230824361824754e-06),
#              ('scale_pos_weight', 0.7421091918485163)])

# Do this, if you have already run the tuning code
import collections
from collections import OrderedDict
best_params = [('bagging_temperature', 0.41010395885331385),
             ('border_count', 186),
             ('depth', 8),
             ('iterations', 1000),
             ('l2_leaf_reg', 21),
             ('learning_rate', 0.0673344419215237),
             ('random_strength', 3.230824361824754e-06),
             ('scale_pos_weight', 0.7421091918485163)]

best_params = OrderedDict(best_params)

best_params['iterations']=1000

# Building the model with opt parameters
cat_tm_cd = CatBoostClassifier(**best_params,verbose=False)
cat_tm_cd.fit(x_train,y_train)

<catboost.core.CatBoostClassifier at 0x7f842a395580>

# model score for training data
cat_tm_cd.score(x_train,y_train)

0.747135152720948

# model score for test data
cat_tm_cd.score(x_test,y_test)

0.7387909546833238

## The final predicted value. 0 -> Not consumed Drugs; 1 -> Consumed
Drugs
y_pred = cat_tm_cd.predict(x_test)
```

```
## Confusion Matrix
from sklearn.metrics import confusion_matrix
# y_test is actual and y_pred is predicted
tn,fp,fn,tp=confusion_matrix(y_test,y_pred).ravel()
sensitivity = tp/(tp+fn)
print("sensitivity is ", sensitivity) # Also 'Recall'
specificity = tn/(tn+fp)
print("specificity is ", specificity) # Also 'Precision'
```

```
sensitivity is  0.636031258161072
specificity is  0.8419529909413783
```

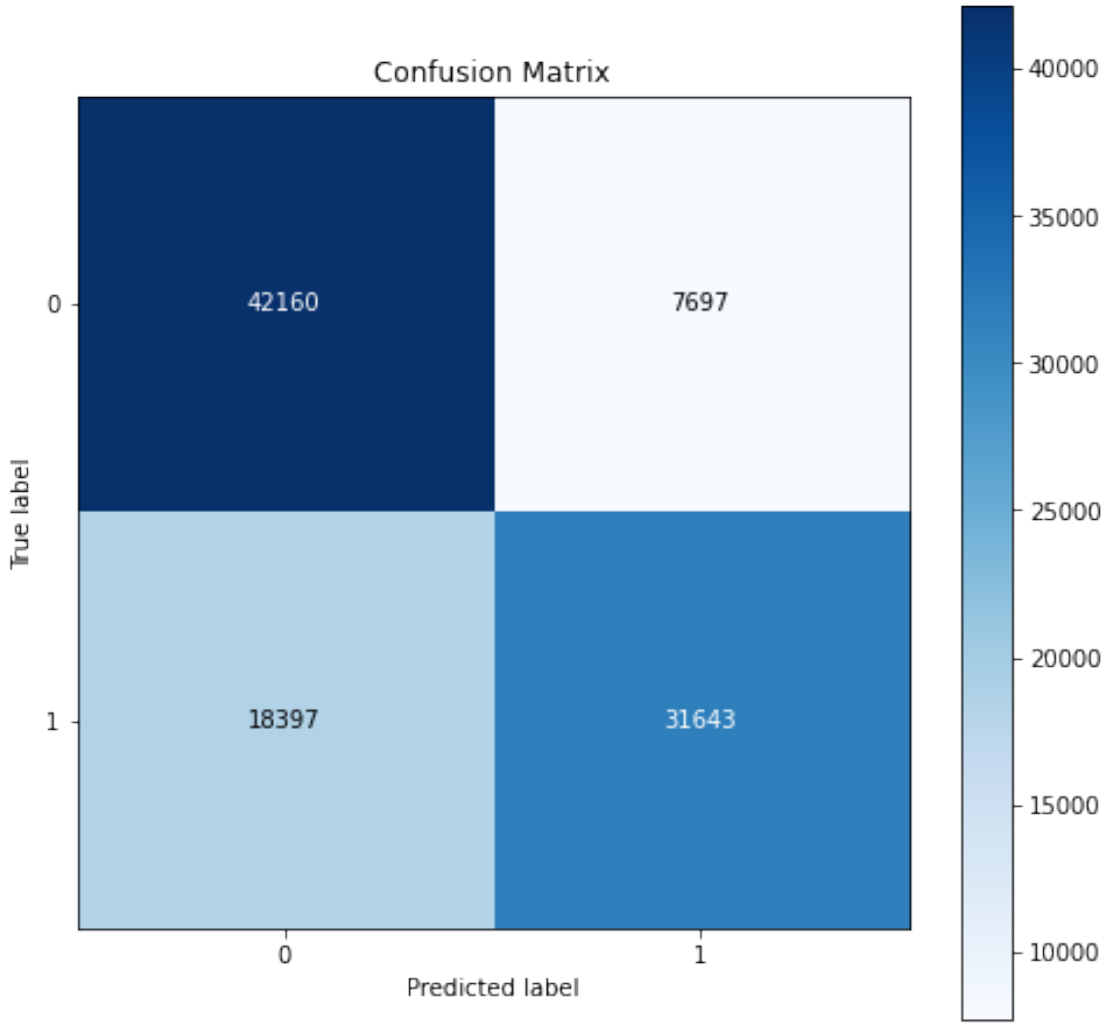
```
# Classification report
import sklearn
print(sklearn.metrics.classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.70      0.85      0.76     49857
           1       0.80      0.63      0.71     50040

    accuracy                           0.74     99897
   macro avg       0.75      0.74      0.74     99897
weighted avg       0.75      0.74      0.74     99897
```

```
# CONFUSION MATRIX
!pip install scikit-plot
import scikitplot as skplt

skplt.metrics.plot_confusion_matrix(
    y_test,
    y_pred,
    figsize=(8,8))
```

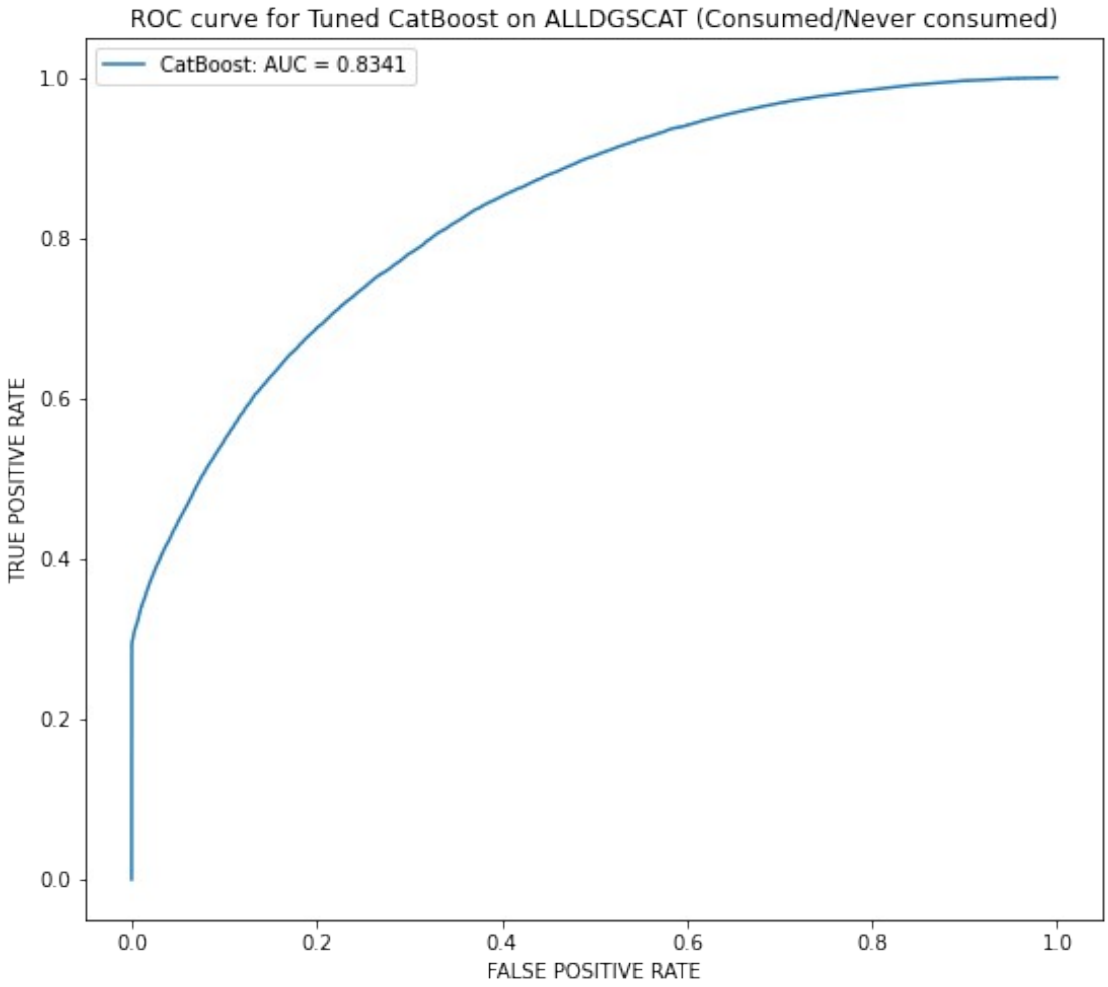
```
Looking in indexes: https://pypi.org/simple, https://us-
python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: scikit-plot in
/usr/local/lib/python3.8/dist-packages (0.3.7)
Requirement already satisfied: matplotlib>=1.4.0 in
/usr/local/lib/python3.8/dist-packages (from scikit-plot) (3.2.2)
Requirement already satisfied: joblib>=0.10 in
/usr/local/lib/python3.8/dist-packages (from scikit-plot) (1.2.0)
Requirement already satisfied: scipy>=0.9 in
/usr/local/lib/python3.8/dist-packages (from scikit-plot) (1.7.3)
Requirement already satisfied: scikit-learn>=0.18 in
/usr/local/lib/python3.8/dist-packages (from scikit-plot) (1.0.2)
Requirement already satisfied: cycler>=0.10 in
/usr/local/lib/python3.8/dist-packages (from matplotlib>=1.4.0-
>scikit-plot) (0.11.0)
Requirement already satisfied: python-dateutil>=2.1 in
```

```
/usr/local/lib/python3.8/dist-packages (from matplotlib>=1.4.0-
>scikit-plot) (2.8.2)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.8/dist-packages (from matplotlib>=1.4.0-
>scikit-plot) (1.4.4)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!
=2.1.6,>=2.0.1 in /usr/local/lib/python3.8/dist-packages (from
matplotlib>=1.4.0->scikit-plot) (3.0.9)
Requirement already satisfied: numpy>=1.11 in
/usr/local/lib/python3.8/dist-packages (from matplotlib>=1.4.0-
>scikit-plot) (1.21.6)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.8/dist-packages (from python-dateutil>=2.1-
>matplotlib>=1.4.0->scikit-plot) (1.15.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.8/dist-packages (from scikit-learn>=0.18-
>scikit-plot) (3.1.0)

<matplotlib.axes._subplots.AxesSubplot at 0x7f4f0f8af6a0>
```



```
# ROC Curve
import sklearn.metrics as metrics
# calculate the fpr and tpr for all thresholds of the classification
probs = cat_tm_cd.predict_proba(x_test)
preds = probs[:,1] # Extracting data under '1'th column of probs
fpr, tpr, threshold = metrics.roc_curve(y_test, preds)
roc_auc = metrics.auc(fpr, tpr)
print('Area under ROC curve:',roc_auc)
print('\n')

import matplotlib.pyplot as plt
plt.plot(fpr,tpr,label="CatBoost: AUC = "+str(round(roc_auc,4)))
plt.rcParams["figure.figsize"] = (9,8)
plt.title('ROC curve for Tuned CatBoost on ALLDGSCAT (Consumed/Never
consumed)')
plt.xlabel('FALSE POSITIVE RATE')
plt.ylabel('TRUE POSITIVE RATE')
plt.legend(loc='upper left')
plt.show()
```

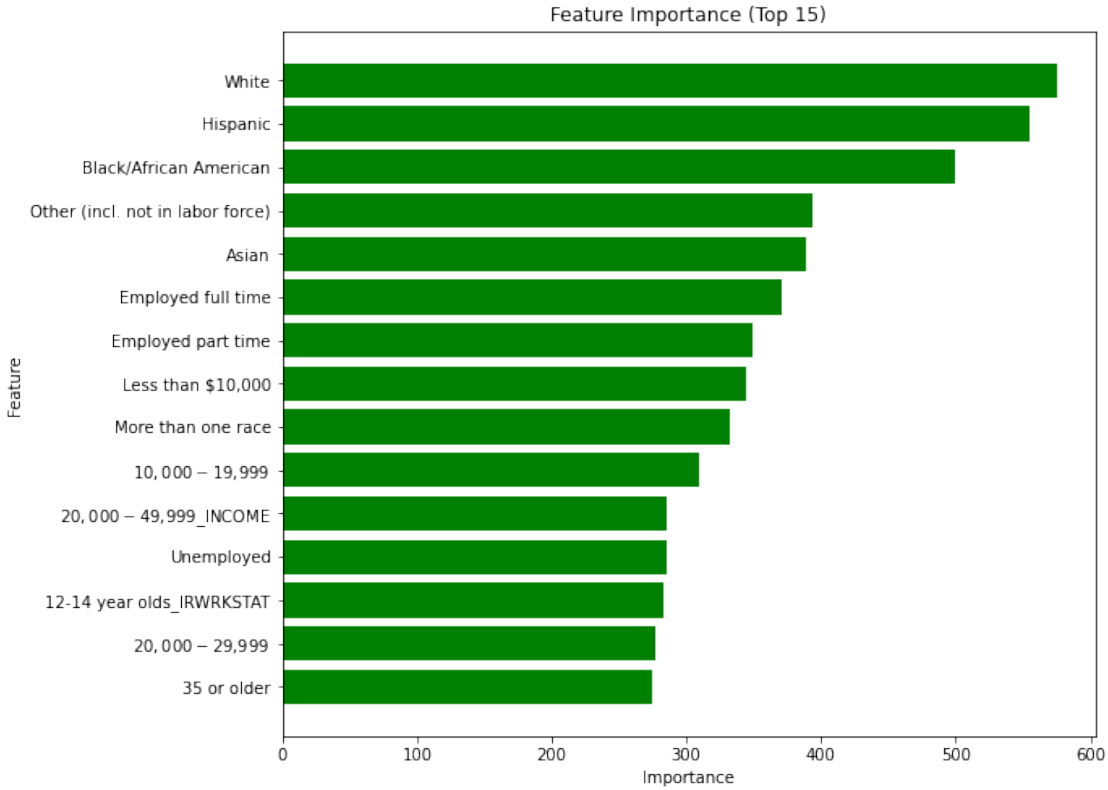
```
Area under ROC curve: 0.8340573961594108
```




```
# Feature importance in prediction
import sklearn.metrics as metrics
bcfit = cat_tm_cd
imp_feature = bcfit.feature_importances_
columns = x.columns
clf_co = pd.Series(imp_feature,columns)

## top features
# Feature importance graph
# Top 15 most imp features
dfimp =
pd.DataFrame(clf_co.sort_values(ascending=False)).reset_index()
dfimp.columns.values[0] = 'Feature'
dfimp.columns.values[1] = 'Importance'
dfimp['Importance'] = round(dfimp['Importance']*100,2)
dplt1 = dfimp.head(15)
dplt2 = dplt1.copy()
dplt2.sort_values(by='Importance',ascending=True,inplace=True)

plt.barh(dplt2['Feature'],dplt2['Importance'],color='green')
```

```
plt.title('Feature Importance (Top 15)')
plt.ylabel('Feature')
plt.xlabel('Importance')
print(dplt1)
```

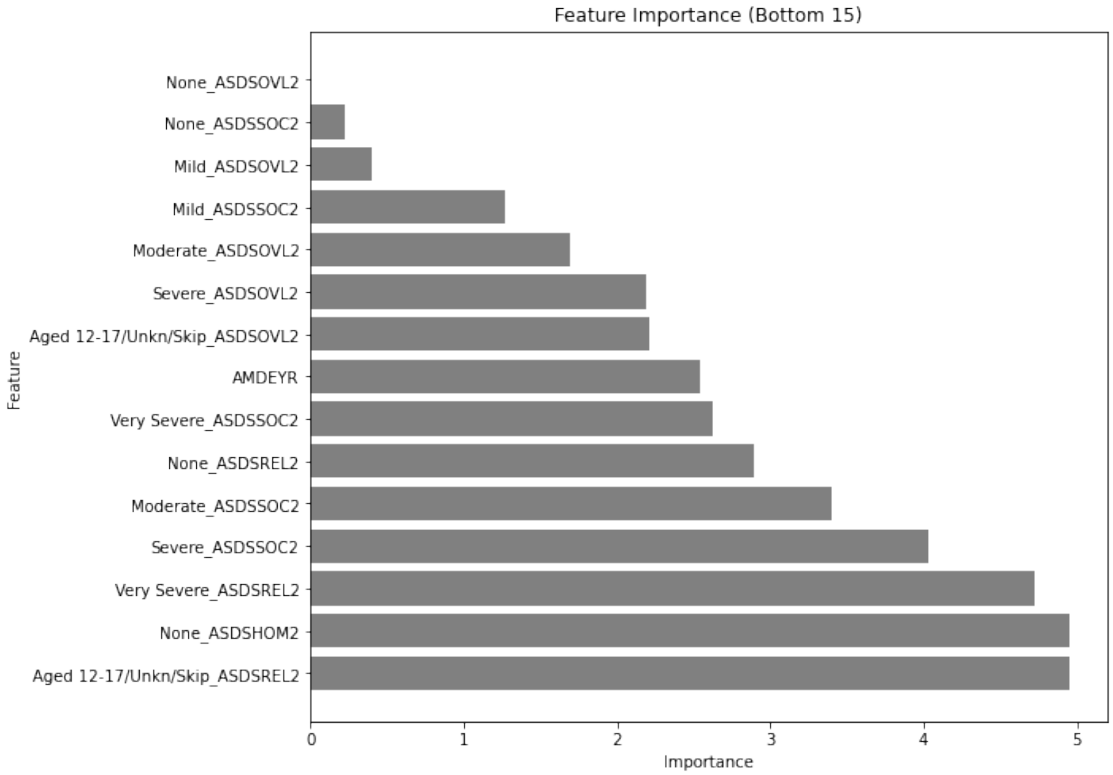
```
                              Feature  Importance
0                               White      575.23
1                            Hispanic      555.20
2               Black/African American      499.71
3   Other (incl. not in labor force)      393.92
4                               Asian      388.54
5                  Employed full time      371.30
6                  Employed part time      349.45
7                   Less than $10,000      344.11
8                  More than one race      332.72
9                     $10,000-$19,999      309.99
10             $20,000-$49,999_INCOME      285.44
11                         Unemployed      284.95
12          12-14 year olds_IRWRKSTAT      283.57
13                    $20,000-$29,999      277.44
14                        35 or older      274.83
```




```
## bottom features
# Feature importance graph
# Bottom 15 (15 least imp features)
dfimp =
```

```
pd.DataFrame(clf_co.sort_values(ascending=False)).reset_index()
dfimp.columns.values[0] = 'Feature'
dfimp.columns.values[1] = 'Importance'
dfimp['Importance'] = round(dfimp['Importance']*100,2)
dplt1 = dfimp.tail(15)
dplt2 = dplt1.copy()
dplt2.sort_values(by='Importance',ascending=False,inplace=True)

plt.barh(dplt2['Feature'],dplt2['Importance'],color='gray')
plt.title('Feature Importance (Bottom 15)')
plt.ylabel('Feature')
plt.xlabel('Importance')
print(dplt2.sort_values(by='Importance',ascending=True))
```

	Feature	Importance
79	None_ASDSOVL2	0.00
78	None_ASDSSOC2	0.22
77	Mild_ASDSOVL2	0.40
76	Mild_ASDSSOC2	1.27
75	Moderate_ASDSOVL2	1.69
74	Severe_ASDSOVL2	2.19
73	Aged 12-17/Unkn/Skip_ASDSOVL2	2.21
72	AMDEYR	2.54
71	Very Severe_ASDSSOC2	2.62
70	None_ASDSREL2	2.89
69	Moderate_ASDSSOC2	3.40
68	Severe_ASDSSOC2	4.03
67	Very Severe_ASDSREL2	4.73
65	Aged 12-17/Unkn/Skip_ASDSREL2	4.95
66	None_ASDSHOM2	4.95

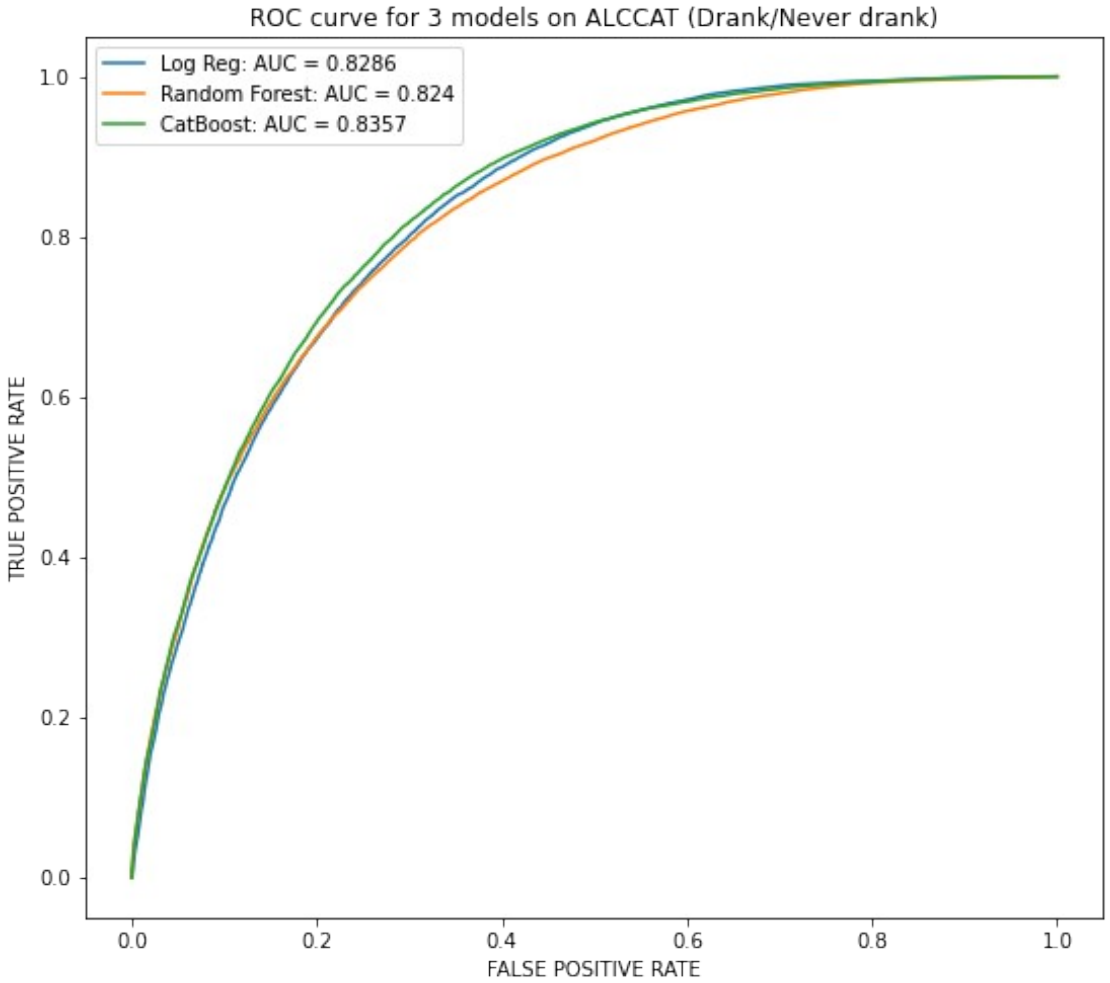


MODEL SELECTION

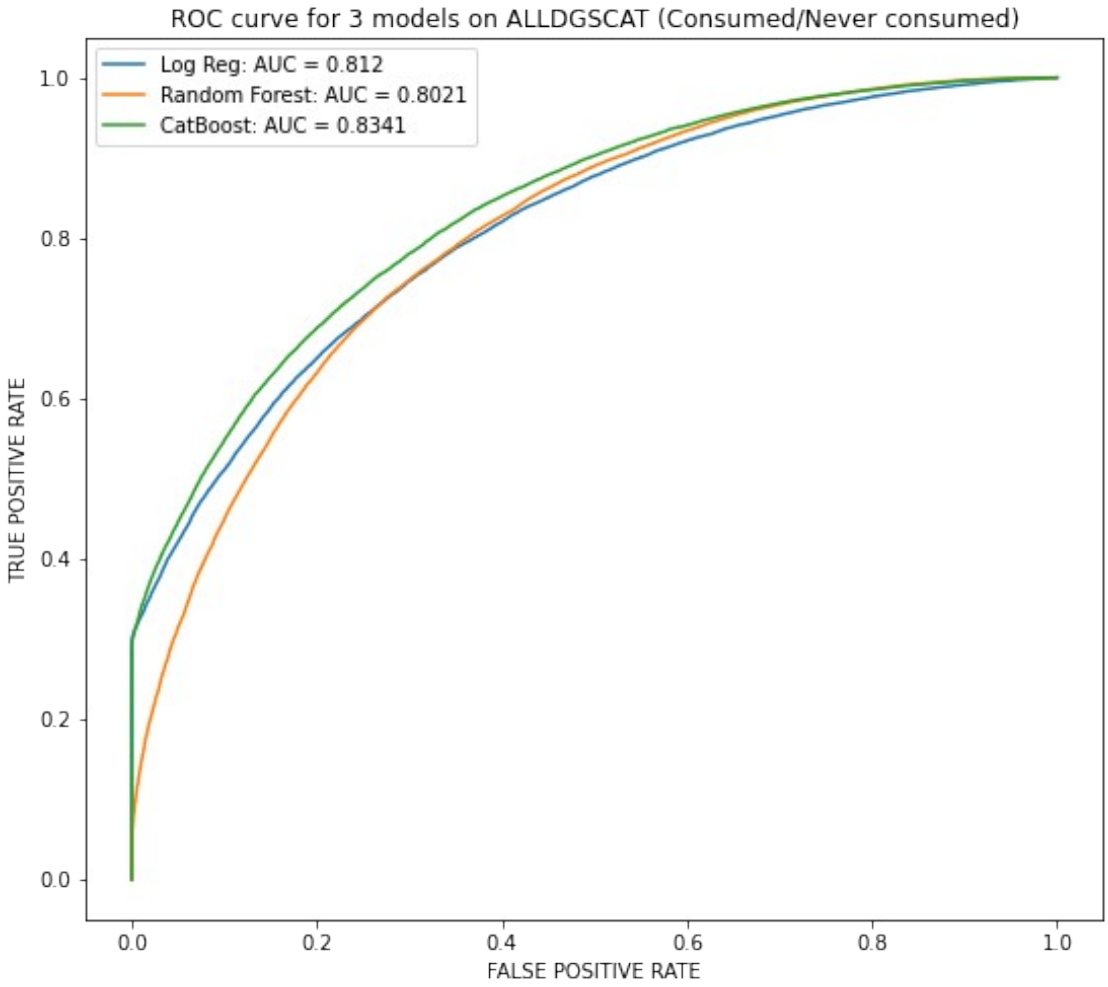
In case of ALCCAT (Consumed/Not consumed Alcohol)

```
### roc-curve for 3models ca

import sklearn.metrics as metrics

# For Log Reg
probs = logca.predict_proba(x_test_ca)
preds = probs[:,1] # Extracting data under '1'th column of probs
fpr, tpr, threshold = metrics.roc_curve(y_test_ca, preds)
roc_auc = metrics.auc(fpr, tpr)

import matplotlib.pyplot as plt
plt.plot(fpr,tpr,label="Log Reg: AUC = "+str(round(roc_auc,4)))

# RF
probs = rf_tm_ca.predict_proba(x_test_ca)
preds = probs[:,1] # Extracting data under '1'th column of probs
fpr, tpr, threshold = metrics.roc_curve(y_test_ca, preds)
roc_auc = metrics.auc(fpr, tpr)

import matplotlib.pyplot as plt
plt.plot(fpr,tpr,label="Random Forest: AUC = "+str(round(roc_auc,4)))
```

```
# CB
probs = cat_tm_ca.predict_proba(x_test_ca)
preds = probs[:,1] # Extracting data under '1'th column of probs
fpr, tpr, threshold = metrics.roc_curve(y_test_ca, preds)
roc_auc = metrics.auc(fpr, tpr)


import matplotlib.pyplot as plt
plt.plot(fpr,tpr,label="CatBoost: AUC = "+str(round(roc_auc,4)))


plt.rcParams["figure.figsize"] = (9,8)
plt.title('ROC curve for 3 models on ALCCAT (Drank/Never drank)')
plt.xlabel('FALSE POSITIVE RATE')
plt.ylabel('TRUE POSITIVE RATE')
plt.legend(loc='upper left')
plt.show()
```



```
### roc-curve for 3models cd

import sklearn.metrics as metrics

# For Log Reg
probs = logcd.predict_proba(x_test_cd)
preds = probs[:,1] # Extracting data under '1'th column of probs
fpr, tpr, threshold = metrics.roc_curve(y_test_cd, preds)
roc_auc = metrics.auc(fpr, tpr)

import matplotlib.pyplot as plt
plt.plot(fpr,tpr,label="Log Reg: AUC = "+str(round(roc_auc,4)))

# RF
probs = rf_tm_cd.predict_proba(x_test_cd)
preds = probs[:,1] # Extracting data under '1'th column of probs
fpr, tpr, threshold = metrics.roc_curve(y_test_cd, preds)
roc_auc = metrics.auc(fpr, tpr)

import matplotlib.pyplot as plt
plt.plot(fpr,tpr,label="Random Forest: AUC = "+str(round(roc_auc,4)))

# CB
probs = cat_tm_cd.predict_proba(x_test_cd)
preds = probs[:,1] # Extracting data under '1'th column of probs
fpr, tpr, threshold = metrics.roc_curve(y_test_cd, preds)
roc_auc = metrics.auc(fpr, tpr)


import matplotlib.pyplot as plt
plt.plot(fpr,tpr,label="CatBoost: AUC = "+str(round(roc_auc,4)))


plt.rcParams["figure.figsize"] = (9,8)
plt.title('ROC curve for 3 models on ALLDGSCAT (Consumed/Never
consumed)')
plt.xlabel('FALSE POSITIVE RATE')
plt.ylabel('TRUE POSITIVE RATE')
plt.legend(loc='upper left')
plt.show()
```



Therefore, from the ROC curves above, we can conclude that the CatBoost Classifiers are the most suitable for both ALCCAT and ALLDGSCAT

#**REGRESSION**

For IRALCFY (Alcohol consumption frequency per year)

##**X data** for IRALCFY (Alcohol consumption frequency per year)

```
dk = df.copy()
```

Since we are dealing with a Regression problem, we are removing rows having ALCCAT (actually IRALCFY) as 0

```
dk = dk[dk['ALCCAT'] == 1]
```

Now data has been edited to remove rows having ALCCAT = 0

```
#  One-hot encoding (Dummy encoding)

import matplotlib.pyplot as plt

# Dummy encoding on the categorical columns:
#
'ASDSHOM2','ASDSWRK2','ASDSREL2','ASDSSOC2','ASDSOVL2','IRSEX','IREDUH
IGHST2','CATAGE','NEWRACE2','IRWRKSTAT','IRPINC3','INCOME','PDEN10' &
'COUTYP4'
dummy1=pd.get_dummies(dk['ASDSHOM2'])
dummy2=pd.get_dummies(dk['ASDSWRK2'])
dummy3=pd.get_dummies(dk['ASDSREL2'])
dummy4=pd.get_dummies(dk['ASDSSOC2'])
dummy5=pd.get_dummies(dk['ASDSOVL2'])
dummy6=pd.get_dummies(dk['IRSEX'])
dummy7=pd.get_dummies(dk['IREDUHIGHST2'])
dummy8=pd.get_dummies(dk['CATAGE'])
dummy9=pd.get_dummies(dk['NEWRACE2'])
dummy10=pd.get_dummies(dk['IRWRKSTAT'])
dummy11=pd.get_dummies(dk['IRPINC3'])
dummy12=pd.get_dummies(dk['INCOME'])
dummy13=pd.get_dummies(dk['PDEN10'])
dummy14=pd.get_dummies(dk['COUTYP4'])

# Combining dk, dummy1 to dummy14. We are adding cols, so axis=1
master = pd.concat([dk, dummy1, dummy2, dummy3, dummy4, dummy5,
dummy6, dummy7, dummy8, dummy9, dummy10, dummy11, dummy12, dummy13,
dummy14],axis=1)

# Removing QUESTID2, IRALCFY:IRMETHAMYFQ, ALCFLAG:METHAMFLAG,
ASDSHOM2:ASDSOVL2 (dummy encoded), IRSEX (dummy enc'd)
# NEWRACE2 (dummy enc'd), IRWRKSTAT (dummy enc'd), PDEN10, ALLDGSFLAG,
ALLDGS from Master data
# Also removing dummy encoded variables and variables that were
decided to remove after correlation matrix
x =
master.drop(master.columns[[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,1
9,20,21,22,23,24,25,26,27,28,31,32,33,34,35,36,37,38,39]],axis=1)

# Renaming depression categories to avoid column duplication
i=5
x.columns.values[9-i] = 'Aged 12-17/Unkn/Skip_ASDSHOM2'
x.columns.values[10-i] = 'Mild_ASDSHOM2'
x.columns.values[11-i] = 'Moderate_ASDSHOM2'
x.columns.values[12-i] = 'None_ASDSHOM2'
x.columns.values[13-i] = 'Severe_ASDSHOM2'
x.columns.values[14-i] = 'Very Severe_ASDSHOM2'

x.columns.values[15-i] = 'Aged 12-17/Unkn/Skip_ASDSWRK2'
x.columns.values[16-i] = 'Mild_ASDSWRK2'
```

```
x.columns.values[17-i] = 'Moderate_ASDSWRK2'
x.columns.values[18-i] = 'None_ASDSWRK2'
x.columns.values[19-i] = 'Severe_ASDSWRK2'
x.columns.values[20-i] = 'Very Severe_ASDSWRK2'

x.columns.values[21-i] = 'Aged 12-17/Unkn/Skip_ASDSREL2'
x.columns.values[22-i] = 'Mild_ASDSREL2'
x.columns.values[23-i] = 'Moderate_ASDSREL2'
x.columns.values[24-i] = 'None_ASDSREL2'
x.columns.values[25-i] = 'Severe_ASDSREL2'
x.columns.values[26-i] = 'Very Severe_ASDSREL2'

x.columns.values[27-i] = 'Aged 12-17/Unkn/Skip_ASDSSOC2'
x.columns.values[28-i] = 'Mild_ASDSSOC2'
x.columns.values[29-i] = 'Moderate_ASDSSOC2'
x.columns.values[30-i] = 'None_ASDSSOC2'
x.columns.values[31-i] = 'Severe_ASDSSOC2'
x.columns.values[32-i] = 'Very Severe_ASDSSOC2'

x.columns.values[33-i] = 'Aged 12-17/Unkn/Skip_ASDSOVL2'
x.columns.values[34-i] = 'Mild_ASDSOVL2'
x.columns.values[35-i] = 'Moderate_ASDSOVL2'
x.columns.values[36-i] = 'None_ASDSOVL2'
x.columns.values[37-i] = 'Severe_ASDSOVL2'
x.columns.values[38-i] = 'Very Severe_ASDSOVL2'

x.columns.values[70] = '$20,000-$49,999_INCOME'
x.columns.values[71] = '$50,000-$74,999_INCOME'
x.columns.values[72] = '$75,000 or more_INCOME'
x.columns.values[73] = 'Less than $20,000_INCOME'

x.columns.values[58] = '12-14 year olds_IRWRKSTAT'
```

Y data for IRALCFY

```
# create target variable
y = dk['IRALCFY']
```

Train-Test split

```
# split data into train n test
from sklearn.model_selection import train_test_split

# Outputs 4 variables. First two variables will be 80% (train) & 20%
(test) of x and Last two variables are 80% (train) & 20% (test) of y
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2)

print(x_train.shape,y_train.shape)
print(x_test.shape,y_test.shape)
```

```
(153689, 80) (153689,)
(38423, 80) (38423,)
```

```
###################################
```

```
# WE ARE JUST EXPLORING THE OPTION OF USING BOX-COX TRANSFORM. WE ARE
NOT IMPLEMENTING IT IN OUR MODEL
# Box-Cox Transformation (To reduce skewness in the Y variable and
make it more similar to Normal distribution)
# Packages
# DO NOT RUN BOX-COX FOR RANDOM FOREST REGRESSION/CLASSIFICATION (or
any other tree-based algorithm)
!pip install feature-engine
import pandas as pd
from feature_engine import transformation as vt # pip install feature-
engine
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from scipy import stats
```

```
# Checking skewness of Y
da = pd.DataFrame(y_train)
da.IRALCFY.skew()
```

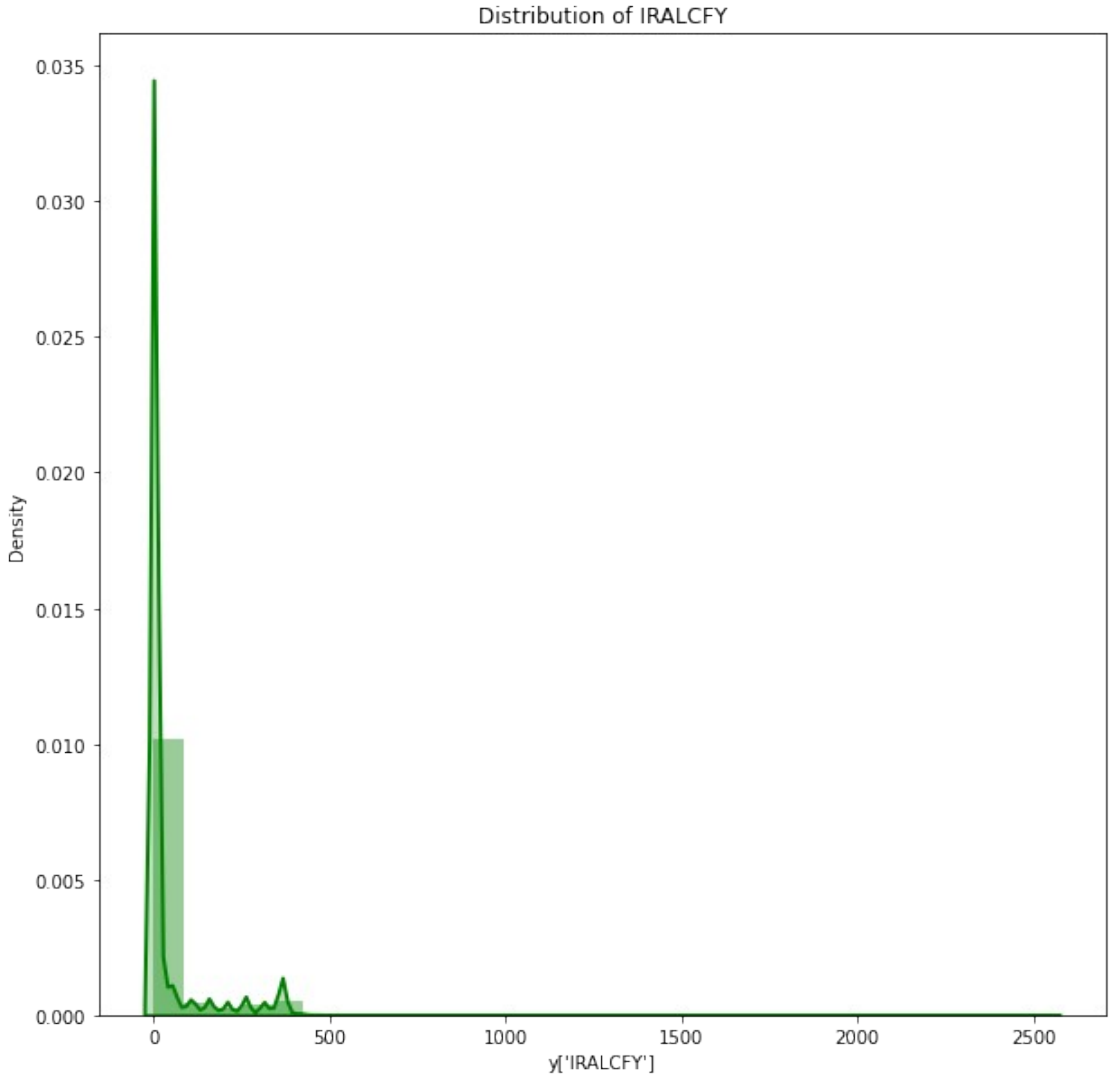
```
1.392884417441438
```

```
# Checking skewness of Y
da = pd.DataFrame(y)
da.IRALCFY.skew()
```

```
1.3947337096300492
```

```
# creating axes to draw plots
import seaborn as sns
fig, ax = plt.subplots(1, 1, figsize=(10, 10))

# plotting the original data(non-normal) and
# fitted data (normal)
sns.distplot(y, hist = True, kde = True,
             kde_kws = {'shade': True, 'linewidth': 2},
             label = "Non-Normal", color ="green", ax = ax,bins=30)
ax.title.set_text('Distribution of IRALCFY')
ax.set_xlabel("y['IRALCFY']")
```

```
Text(0.5, 0, "y['IRALCFY']")
```




```
# Applying Boxcox transform on y_train
tf = vt.BoxCoxTransformer(variables = ['IRALCFY'])
y_train_tf = tf.fit_transform(pd.DataFrame(y_train))
y_train_tf.IRALCFY.skew() # As you can see, skewness is close to zero
(Normal dist)
```

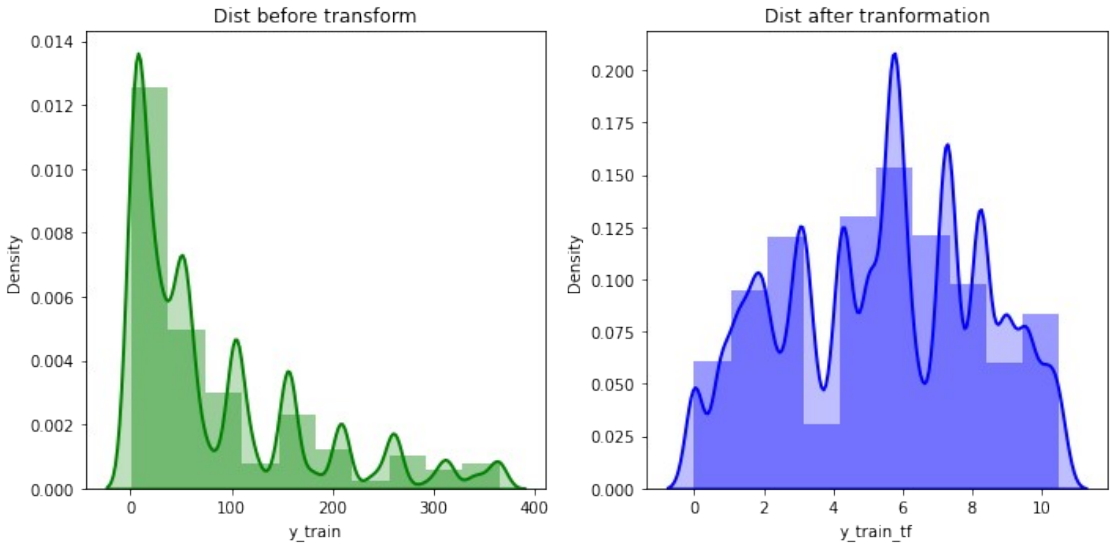
```
-0.09458868550427103
```

```
stats.boxcox((pd.DataFrame(y_train)).IRALCFY)[1] # LAMBDA value for
y_train
```

```
0.17973217438699515
```

```
# creating axes to draw plots
import seaborn as sns
fig, ax = plt.subplots(1, 2, figsize=(10, 5))

# plotting the original data(non-normal) and
```

```
# fitted data (normal)
sns.distplot(y_train, hist = True, kde = True,
             kde_kws = {'shade': True, 'linewidth': 2},
             label = "Non-Normal", color ="green", ax = ax[0],bins=10)
ax[0].title.set_text('Dist before transform')
ax[0].set_xlabel('y_train')

sns.distplot(y_train_tf, hist = True, kde = True,
             kde_kws = {'shade': True, 'linewidth': 2},
             label = "Normal", color ="blue", ax = ax[1],bins=10)
plt.xlabel('y_train_tf')
plt.title('Dist after tranformation')

fig.tight_layout()
```



But we are not applying Box-Cox transform on the target variable due to dissmilar scale. This is just checking for feasibility

LINEAR REGRESSION

```
# Linear Regression Modeling
# Fitting to the model
from sklearn import linear_model
lm = linear_model.LinearRegression()
model = lm.fit(x_train,y_train) # Passing the training data for
Predictor & Target variables
prediction = lm.predict(x_test) # The predicted y_test from test data
of predicted variable

# model scores for train and test
import sklearn as sklearn
print(model.score(x_train,y_train))
print(model.score(x_test,y_test))
```

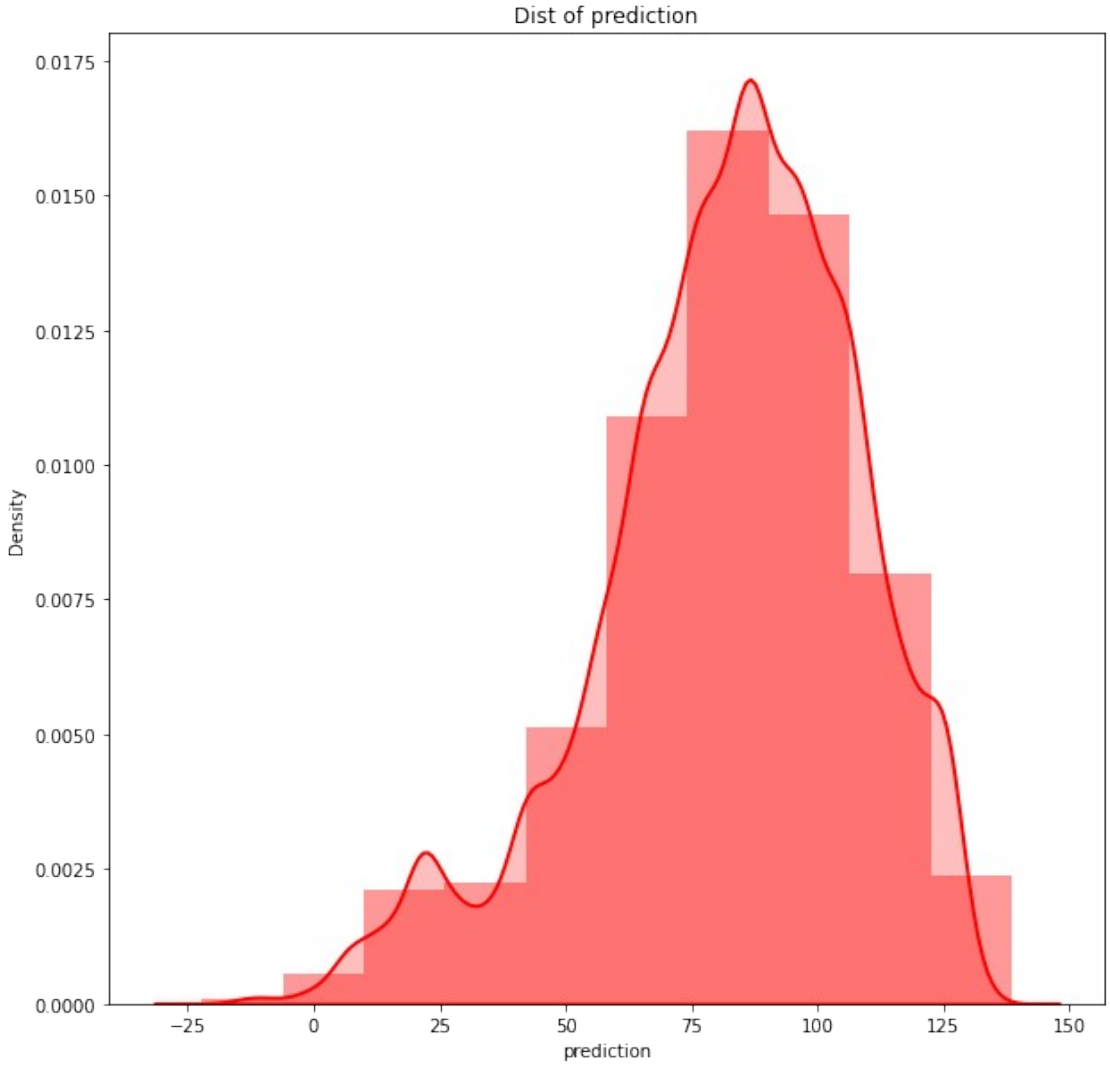
```
0.08226745078017306
0.0773372152438615
```

```
# print the coefficient
# Linear Reg eqn y = B0 + B1.x1 + B2.x2 +...+ BN.xN
print(model.intercept_) # intercept_ is used to find B0 (intercept or
c)
print(model.coef_) # other coeffcients (B1, B2...BN)
```

```
83991284538450.58
[ 7.09607835e+00  7.66369933e+00 -1.64819461e+00  3.55468537e+00
 -4.23213688e+13 -4.23213688e+13 -4.23213688e+13 -4.23213688e+13
 -4.23213688e+13 -4.23213688e+13 -2.48092953e+14 -2.48092953e+14
 -2.48092953e+14 -2.48092953e+14 -2.48092953e+14 -2.48092953e+14
 -9.95483457e+13 -9.95483457e+13 -9.95483457e+13 -9.95483457e+13
 -9.95483457e+13 -9.95483457e+13  2.89376308e+14  2.89376308e+14
  2.89376308e+14  2.89376308e+14  2.89376308e+14  2.89376308e+14
 -4.90465655e+14 -4.90465655e+14 -4.90465655e+14 -4.90465655e+14
 -4.90465655e+14 -4.90465655e+14  1.86519327e+14  1.86519327e+14
  3.63529189e+13  3.63529189e+13  3.63529189e+13  3.63529189e+13
  3.63529189e+13  3.63529189e+13  3.63529189e+13  3.63529189e+13
  3.63529189e+13  3.63529189e+13  3.63529189e+13  2.82162053e+14
  2.82162053e+14  2.82162053e+14  2.82162053e+14  4.37455185e+13
  4.37455185e+13  4.37455185e+13  4.37455185e+13  4.37455185e+13
  4.37455185e+13  4.37455185e+13 -3.33649969e+13 -3.33649969e+13
 -3.33649969e+13 -3.33649969e+13 -3.33649969e+13 -1.09500882e+14
 -1.09500882e+14 -1.09500882e+14 -1.09500882e+14 -1.09500882e+14
 -1.09500882e+14 -1.09500882e+14 -2.90717646e+14 -2.90717646e+14
 -2.90717646e+14 -2.90717646e+14 -1.85843183e+14 -1.85843183e+14
 -1.85843183e+14  5.77707621e+14  5.77707621e+14  5.77707621e+14]
```

```
# Checking the distribution of prediction
import seaborn as sns
fig, ax = plt.subplots(1,1,figsize=(10, 10))

# plotting the original data(non-normal) and
# fitted data (normal)
sns.distplot(prediction, hist = True, kde = True,
             kde_kws = {'shade': True, 'linewidth': 2},
             label = "Non-Normal", color ="red", ax = ax,bins=10)
ax.title.set_text('Dist of prediction')
ax.set_xlabel('prediction')
```

```
Text(0.5, 0, 'prediction')
```




```
print(max(prediction))
print(min(prediction))
```

```
138.515625
-22.234375
```

```
# Evaluation metrics

import sklearn.metrics as metrics
mae = metrics.mean_absolute_error(y_test, prediction)
mse = metrics.mean_squared_error(y_test, prediction)
rmse = np.sqrt(mse) #mse**(0.5)
r2 = metrics.r2_score(y_test,prediction)
mape = metrics.mean_absolute_percentage_error(y_test, prediction)

print("Results of sklearn.metrics:")
print("MAE:",mae)
print("MSE:", mse)
```

```
print("RMSE:", rmse)
print("R-Squared:", r2)
print("MAPE:",mape)
```

```
Results of sklearn.metrics:
MAE: 67.3325347936132
MSE: 7585.800442891922
RMSE: 87.09650074998376
R-Squared: 0.0773372152438615
MAPE: 6.050438707733569
```

RANDOM FOREST REGRESSION

Hyperparameter Tuning

Step 1

```
import numpy as np
from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.arange(2,101)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.arange(2,31,2)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [int(x) for x in np.arange(2,51,2)]
# Minimum number of samples required at each leaf node
min_samples_leaf = [int(x) for x in np.arange(2,51,2)]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
print(random_grid)
```

```
{'n_estimators': [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,
17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50,
51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67,
68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84,
85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100],
'max_features': ['auto', 'sqrt'], 'max_depth': [2, 4, 6, 8, 10, 12,
14, 16, 18, 20, 22, 24, 26, 28, 30, None], 'min_samples_split': [2, 4,
6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40,
42, 44, 46, 48, 50], 'min_samples_leaf': [2, 4, 6, 8, 10, 12, 14, 16,
```

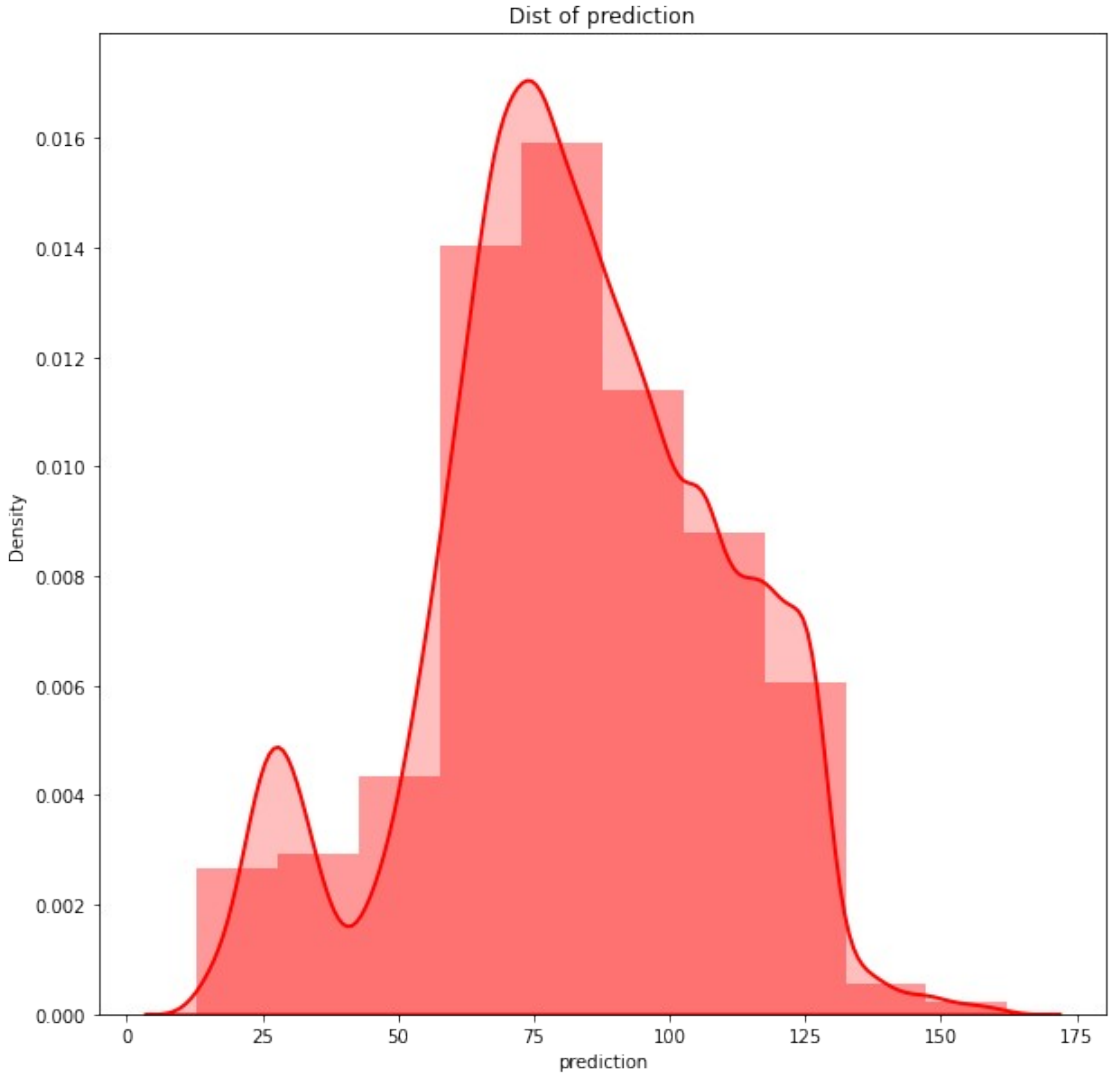
```
18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48, 50],
'bootstrap': [True, False]}
```

Step 2 - Randomized Search & Model Building

```
# SKIP if you already have the hyperparameters
# Use the random grid to search for best hyperparameters
# First create the base model to tune
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available
cores
rf_bestfit = RandomizedSearchCV(estimator = rf, param_distributions =
random_grid, n_iter = 100, cv = 3, verbose=2, random_state=42, n_jobs
= -1)
# Fit the random search model
rf_bestfit.fit(x_train, y_train)

rf_bestfit.best_params_ # These are the parameters that we are setting
to our model to get the best possible accuracy

# PARAMETERS
# {'n_estimators': 43,
#  'min_samples_split': 48,
#  'min_samples_leaf': 18,
#  'max_features': 'sqrt',
#  'max_depth': 28,
#  'bootstrap': True}

# Run this, to avoid RandomizedSearch since we already have the
parameters
from sklearn.ensemble import RandomForestRegressor
rf_bestfit = RandomForestRegressor(n_estimators= 43,
 min_samples_split= 48,
 min_samples_leaf= 18,
 max_features= 'sqrt',
 max_depth= 28,
 bootstrap= True,random_state=42)
rf_bestfit = rf_bestfit.fit(x_train,y_train)
prediction = rf_bestfit.predict(x_test)

# Checking the distribution of prediction
import seaborn as sns
fig, ax = plt.subplots(1,1,figsize=(10, 10))

# plotting the original data(non-normal) and
# fitted data (normal)
sns.distplot(prediction, hist = True, kde = True,
             kde_kws = {'shade': True, 'linewidth': 2},
             label = "Non-Normal", color ="red", ax = ax,bins=10)
```

```
ax.title.set_text('Dist of prediction')
ax.set_xlabel('prediction')
```

```
Text(0.5, 0, 'prediction')
```




```
print(min(np.array(prediction)))
print(max(np.array(prediction)))
```

```
12.825101783311226
162.28439650370748
```

Model Evaluation

```
# score on Training data
print(rf_bestfit.score(x_train,y_train))
```

```
0.11502089672821669
```

```
# score on Test data
print(rf_bestfit.score(x_test,y_test))
```

```
0.09366934372894598
```

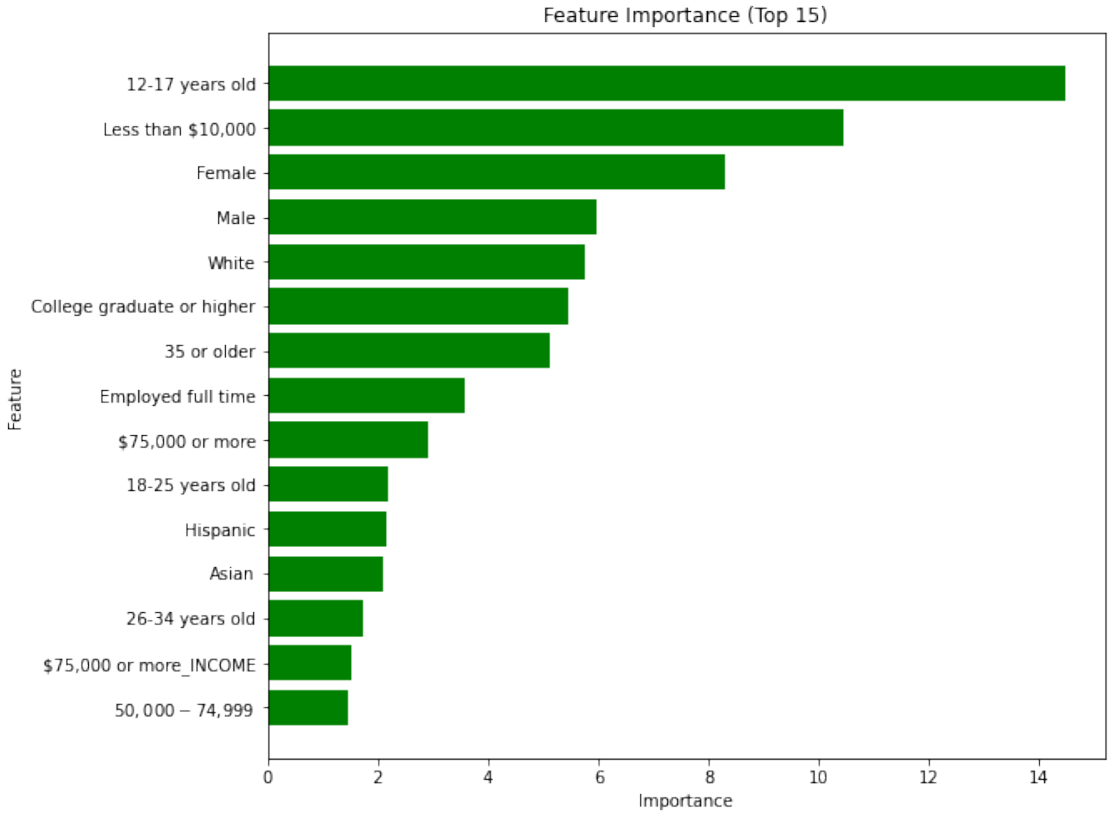
```
# Prediction
prediction = rf_bestfit.predict(x_test)

# Evaluation metrics

import sklearn.metrics as metrics
mae = metrics.mean_absolute_error(y_test, prediction)
mse = metrics.mean_squared_error(y_test, prediction)
rmse = np.sqrt(mse) #mse**(0.5)
r2 = metrics.r2_score(y_test,prediction)
mape = metrics.mean_absolute_percentage_error(y_test, prediction)

print("Results of sklearn.metrics:")
print("MAE:",mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R-Squared:", r2)
print("MAPE:",mape)
```

```
Results of sklearn.metrics:
MAE: 66.65218863624317
MSE: 7501.849392723051
RMSE: 86.61321719416183
R-Squared: 0.09366934372894598
MAPE: 5.851723056694891
```

```
# Feature importance in prediction
import sklearn.metrics as metrics
bcfit = rf_bestfit
imp_feature = bcfit.feature_importances_
columns = x.columns
clf_co = pd.Series(imp_feature,columns)

## top features
# Feature importance graph
# Top 15 most imp features
dfimp =
pd.DataFrame(clf_co.sort_values(ascending=False)).reset_index()
dfimp.columns.values[0] = 'Feature'
dfimp.columns.values[1] = 'Importance'
dfimp['Importance'] = round(dfimp['Importance']*100,2)
dplt1 = dfimp.head(15)
dplt2 = dplt1.copy()
dplt2.sort_values(by='Importance',ascending=True,inplace=True)

plt.barh(dplt2['Feature'],dplt2['Importance'],color='green')
```

```
plt.title('Feature Importance (Top 15)')
plt.ylabel('Feature')
plt.xlabel('Importance')
print(dplt1)
```

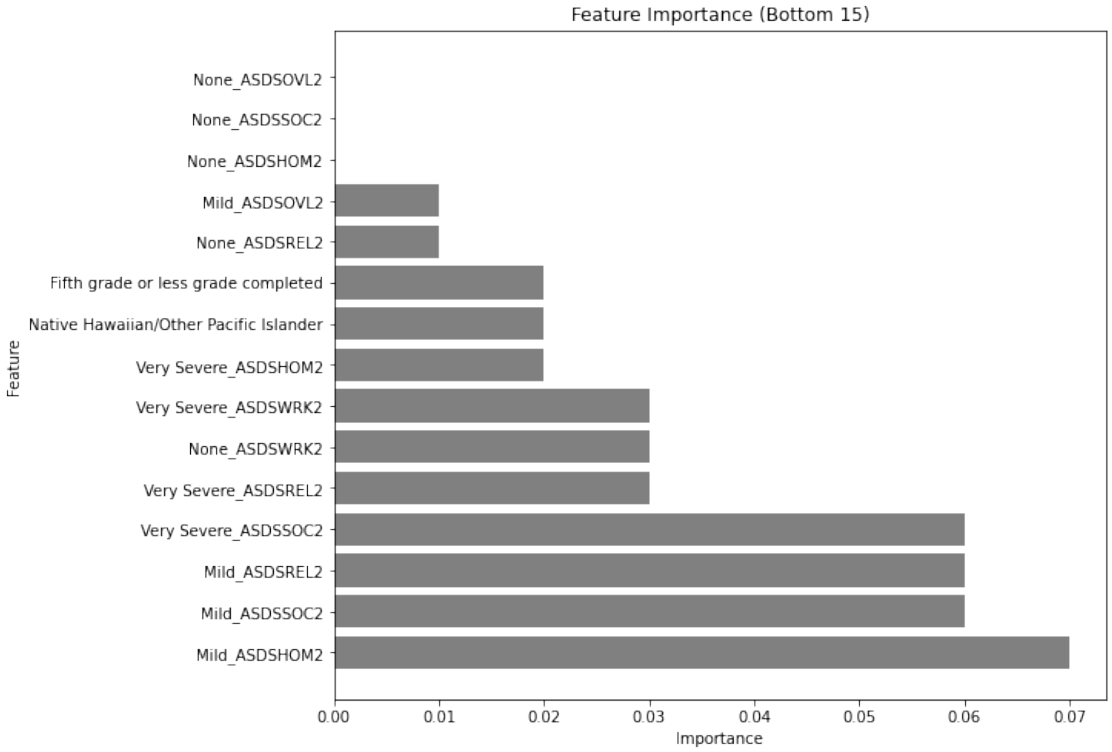
	Feature	Importance
0	12-17 years old	14.47
1	Less than $10,000	10.44
2	Female	8.29
3	Male	5.98
4	White	5.76
5	College graduate or higher	5.44
6	35 or older	5.11
7	Employed full time	3.57
8	$75,000 or more	2.91
9	18-25 years old	2.18
10	Hispanic	2.17
11	Asian	2.10
12	26-34 years old	1.74
13	$75,000 or more_INCOME	1.53
14	$50,000-$74,999	1.45




```
## bottom features
# Feature importance graph
# Bottom 15 (15 least imp features)
dfimp =
```

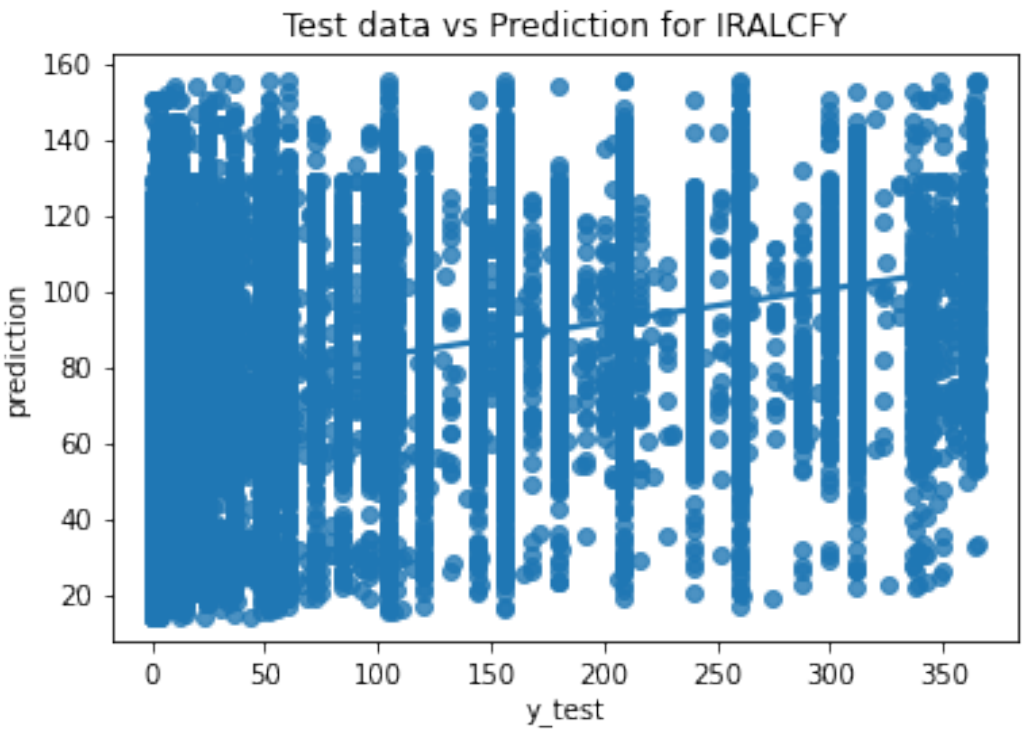
```
pd.DataFrame(clf_co.sort_values(ascending=False)).reset_index()
dfimp.columns.values[0] = 'Feature'
dfimp.columns.values[1] = 'Importance'
dfimp['Importance'] = round(dfimp['Importance']*100,2)
dplt1 = dfimp.tail(15)
dplt2 = dplt1.copy()
dplt2.sort_values(by='Importance',ascending=False,inplace=True)

plt.barh(dplt2['Feature'],dplt2['Importance'],color='gray')
plt.title('Feature Importance (Bottom 15)')
plt.ylabel('Feature')
plt.xlabel('Importance')
print(dplt2.sort_values(by='Importance',ascending=True))
```

	Feature	Importance
77	None_ASDSHOM2	0.00
78	None_ASDSSOC2	0.00
79	None_ASDSOVL2	0.00
75	None_ASDSREL2	0.01
76	Mild_ASDSOVL2	0.01
72	Very Severe_ASDSHOM2	0.02
73	Native Hawaiian/Other Pacific Islander	0.02
74	Fifth grade or less grade completed	0.02
69	Very Severe_ASDSREL2	0.03
70	None_ASDSWRK2	0.03
71	Very Severe_ASDSWRK2	0.03
66	Mild_ASDSSOC2	0.06
67	Mild_ASDSREL2	0.06
68	Very Severe_ASDSSOC2	0.06
65	Mild_ASDSHOM2	0.07




```
# Plotting the chart
import seaborn as sns
sns.regplot(y_test,prediction).set(xlabel = 'y_test', ylabel =
'prediction', title = 'Test data vs Prediction for IRALCFY')
plt.figure(figsize=(20,20))
```

```
<Figure size 1440x1440 with 0 Axes>
```




```
<Figure size 1440x1440 with 0 Axes>
```

CATBOOST REGRESSOR

```
# Selecting only ALCCAT = 'Ever used' (removing rows with 0-value for
IRALCFY)
dvn = dv[dv['ALCCAT'] == 'Ever used']

X =
dvn.drop(dvn.columns[[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,35,36,3
7,38,39]],axis=1)
y = dvn['IRALCFY']

# Declaring Categorical features
cat_features = list(range(0, X.shape[1]))
print(cat_features)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17]
```

```
# Train test split
from sklearn.model_selection import train_test_split

X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2,
random_state=0)

!pip install catboost

# catboost implementation
from catboost import CatBoostRegressor
```

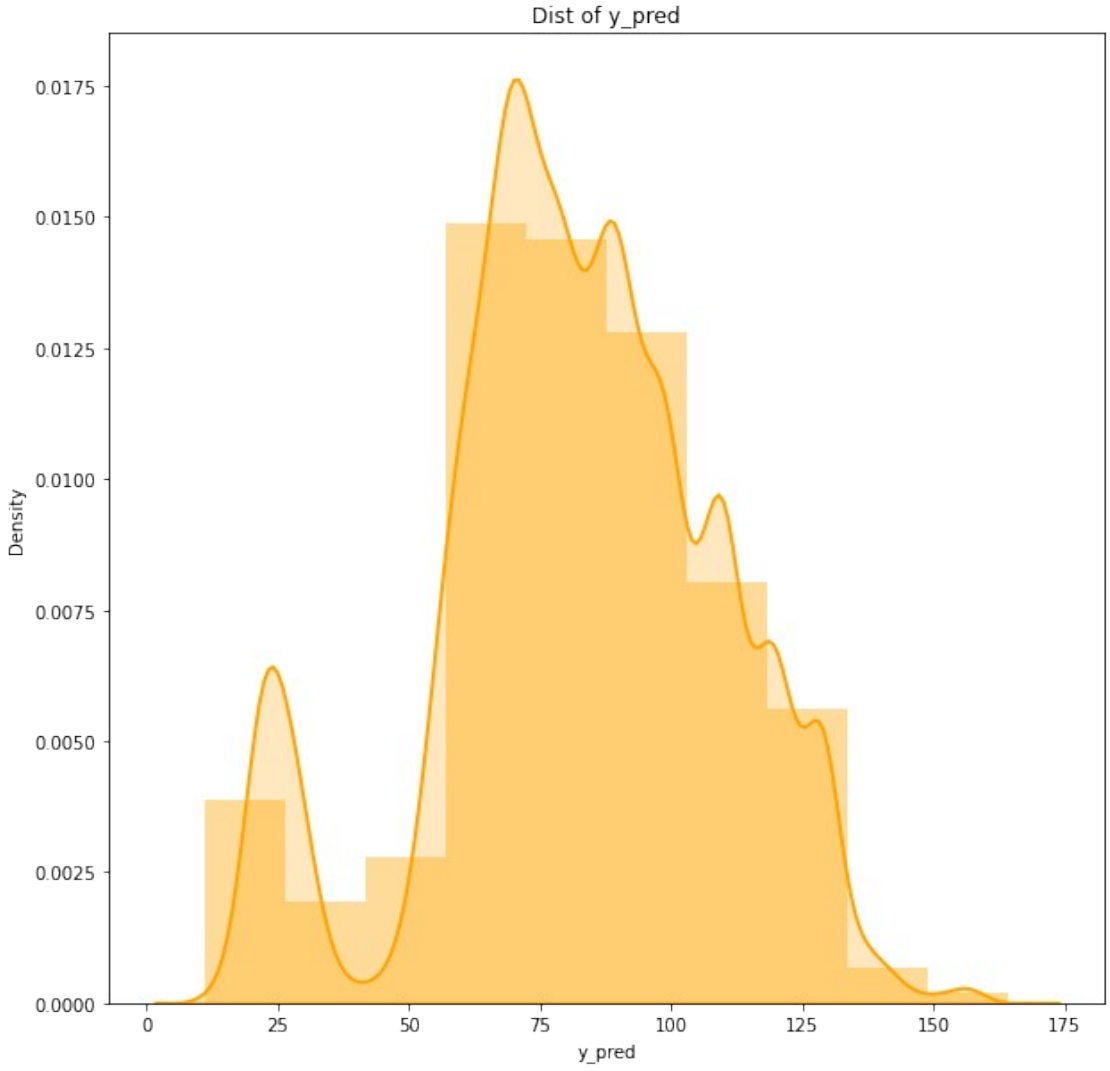
```
clf = CatBoostRegressor(
    iterations=5,
    learning_rate=0.1,
    #loss_function='CrossEntropy'
)


clf.fit(X_train, y_train,
        cat_features=cat_features,
        eval_set=(X_val, y_val),
        verbose=False
)

print('CatBoost model is fitted: ' + str(clf.is_fitted()))
print('CatBoost model parameters:')
print(clf.get_params())
```

```
CatBoost model is fitted: True
CatBoost model parameters:
{'iterations': 5, 'learning_rate': 0.1, 'loss_function': 'RMSE'}
```

```
# Stdout of the training
from catboost import CatBoostRegressor
clf = CatBoostRegressor(
    iterations=10,
#     verbose=5,
)

clf.fit(
    X_train, y_train,
    cat_features=cat_features,
    eval_set=(X_val, y_val),
)
```

```
Learning rate set to 0.5
0:    learn: 88.7743962     test: 88.6560109 best: 88.6560109 (0)
      total: 167ms    remaining: 1.5s
1:    learn: 87.9530420     test: 87.8639898 best: 87.8639898 (1)
      total: 293ms    remaining: 1.17s
2:    learn: 87.3975179     test: 87.3454853 best: 87.3454853 (2)
      total: 422ms    remaining: 985ms
3:    learn: 87.1109220     test: 87.0725560 best: 87.0725560 (3)
      total: 562ms    remaining: 844ms
4:    learn: 86.9632323     test: 86.9541484 best: 86.9541484 (4)
      total: 695ms    remaining: 695ms
5:    learn: 86.9252421     test: 86.9243924 best: 86.9243924 (5)
      total: 806ms    remaining: 537ms
6:    learn: 86.8476882     test: 86.8616067 best: 86.8616067 (6)
      total: 923ms    remaining: 395ms
```

```
7:    learn: 86.8005121     test: 86.8154670 best: 86.8154670 (7)
      total: 1.04s    remaining: 260ms
8:    learn: 86.7357402     test: 86.7608915 best: 86.7608915 (8)
      total: 1.16s    remaining: 129ms
9:    learn: 86.6889754     test: 86.7444333 best: 86.7444333 (9)
      total: 1.28s    remaining: 0us

bestTest = 86.74443326
bestIteration = 9


<catboost.core.CatBoostRegressor at 0x7f4f1165bfd0>

print(clf.predict(X_val)) # Predictions

[ 24.09447171  62.35340449  79.56607192 ... 118.2643631  116.95335757
  93.12273618]

y_pred = clf.predict(X_val)

print(min(np.array(y_pred)))
print(max(np.array(y_pred)))

11.202455774962758
164.1356278045056

# Checking the distribution of y_pred
import seaborn as sns
fig, ax = plt.subplots(1,1,figsize=(10, 10))

# plotting the original data(non-normal) and
# fitted data (normal)
sns.distplot(y_pred, hist = True, kde = True,
             kde_kws = {'shade': True, 'linewidth': 2},
             label = "Non-Normal", color ="orange", ax = ax,bins=10)
ax.title.set_text('Dist of y_pred')
ax.set_xlabel('y_pred')

Text(0.5, 0, 'y_pred')
```




```
# score on Training data
print(clf.score(X_train,y_train))
```

```
0.09134986907066O9
```

```
# score on Test data
print(clf.score(X_val,y_val))
```

```
0.08847903238038313
```

```
# Calculating error

import sklearn.metrics as metrics
mae = metrics.mean_absolute_error(y_val, y_pred)
mse = metrics.mean_squared_error(y_val, y_pred)
rmse = np.sqrt(mse) #mse**(0.5)
r2 = metrics.r2_score(y_val,y_pred)
mape = metrics.mean_absolute_percentage_error(y_val, y_pred)
```

```
print("Results of sklearn.metrics:")
print("MAE:",mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R-Squared:", r2)
print("MAPE:",mape)
```

```
Results of sklearn.metrics:
MAE: 67.0744796879123
MSE: 7524.596701757585
RMSE: 86.74443326091644
R-Squared: 0.08847903238038313
MAPE: 5.963419228648019
```

For ALLDGS (Alldrugs consumption frequency per year)

##**X data** for ALLDGS (Alldrugs consumption frequency per year)

```
dl = df.copy()
```

```
dl = dl[dl['ALLDGSCAT'] == 1] # Deleting rows with 0s in ALLDGSCAT
```

```
#  One-hot encoding (Dummy encoding)
```

```
import matplotlib.pyplot as plt
```

```
# Dummy encoding on the categorical columns:
#
'ASDSHOM2','ASDSWRK2','ASDSREL2','ASDSSOC2','ASDSOVL2','IRSEX','IREDUH
IGHST2','CATAGE','NEWRACE2','IRWRKSTAT','IRPINC3','INCOME','PDEN10' &
'COUTYP4'
dummy1=pd.get_dummies(dl['ASDSHOM2'])
dummy2=pd.get_dummies(dl['ASDSWRK2'])
dummy3=pd.get_dummies(dl['ASDSREL2'])
dummy4=pd.get_dummies(dl['ASDSSOC2'])
dummy5=pd.get_dummies(dl['ASDSOVL2'])
dummy6=pd.get_dummies(dl['IRSEX'])
dummy7=pd.get_dummies(dl['IREDUHIGHST2'])
dummy8=pd.get_dummies(dl['CATAGE'])
dummy9=pd.get_dummies(dl['NEWRACE2'])
dummy10=pd.get_dummies(dl['IRWRKSTAT'])
dummy11=pd.get_dummies(dl['IRPINC3'])
dummy12=pd.get_dummies(dl['INCOME'])
dummy13=pd.get_dummies(dl['PDEN10'])
dummy14=pd.get_dummies(dl['COUTYP4'])
```

```
# Combining dk, dummy1, dummy2, dummy3, dummy4, dummy5, dummy6, dummy7
& dummy8. We are adding cols, so axis=1
```

```
master = pd.concat([dl, dummy1, dummy2, dummy3, dummy4, dummy5,
dummy6, dummy7, dummy8, dummy9, dummy10, dummy11, dummy12, dummy13,
dummy14],axis=1)

# Removing QUESTID2, IRALCFY:IRMETHAMYFQ, ALCFLAG:METHAMFLAG,
ASDSHOM2:ASDSOVL2 (dummy encoded), IRSEX (dummy enc'd)
# NEWRACE2 (dummy enc'd), IRWRKSTAT (dummy enc'd), PDEN10, ALLDGSFLAG,
ALLDGS from Master data
# Also removing dummy encoded variables and variables that were
decided to remove after correlation matrix
x =
master.drop(master.columns[[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,1
9,20,21,22,23,24,25,26,27,28,31,32,33,34,35,36,37,38,39]],axis=1)

# Renaming depression categories to avoid column duplication
i=5
x.columns.values[9-i] = 'Aged 12-17/Unkn/Skip_ASDSHOM2'
x.columns.values[10-i] = 'Mild_ASDSHOM2'
x.columns.values[11-i] = 'Moderate_ASDSHOM2'
x.columns.values[12-i] = 'None_ASDSHOM2'
x.columns.values[13-i] = 'Severe_ASDSHOM2'
x.columns.values[14-i] = 'Very Severe_ASDSHOM2'

x.columns.values[15-i] = 'Aged 12-17/Unkn/Skip_ASDSWRK2'
x.columns.values[16-i] = 'Mild_ASDSWRK2'
x.columns.values[17-i] = 'Moderate_ASDSWRK2'
x.columns.values[18-i] = 'None_ASDSWRK2'
x.columns.values[19-i] = 'Severe_ASDSWRK2'
x.columns.values[20-i] = 'Very Severe_ASDSWRK2'

x.columns.values[21-i] = 'Aged 12-17/Unkn/Skip_ASDSREL2'
x.columns.values[22-i] = 'Mild_ASDSREL2'
x.columns.values[23-i] = 'Moderate_ASDSREL2'
x.columns.values[24-i] = 'None_ASDSREL2'
x.columns.values[25-i] = 'Severe_ASDSREL2'
x.columns.values[26-i] = 'Very Severe_ASDSREL2'

x.columns.values[27-i] = 'Aged 12-17/Unkn/Skip_ASDSSOC2'
x.columns.values[28-i] = 'Mild_ASDSSOC2'
x.columns.values[29-i] = 'Moderate_ASDSSOC2'
x.columns.values[30-i] = 'None_ASDSSOC2'
x.columns.values[31-i] = 'Severe_ASDSSOC2'
x.columns.values[32-i] = 'Very Severe_ASDSSOC2'

x.columns.values[33-i] = 'Aged 12-17/Unkn/Skip_ASDSOVL2'
x.columns.values[34-i] = 'Mild_ASDSOVL2'
x.columns.values[35-i] = 'Moderate_ASDSOVL2'
x.columns.values[36-i] = 'None_ASDSOVL2'
x.columns.values[37-i] = 'Severe_ASDSOVL2'
x.columns.values[38-i] = 'Very Severe_ASDSOVL2'
```

```
x.columns.values[70] = '$20,000-$49,999_INCOME'
x.columns.values[71] = '$50,000-$74,999_INCOME'
x.columns.values[72] = '$75,000 or more_INCOME'
x.columns.values[73] = 'Less than $20,000_INCOME'

x.columns.values[58] = '12-14 year olds_IRWRKSTAT'
```

Y data for ALLDGS

```
# create target variable
y = dl['ALLDGS']

# split data into train n test
from sklearn.model_selection import train_test_split

# Outputs 4 variables. First two variables will be 80% (train) & 20%
(test) of x and Last two variables are 80% (train) & 20% (test) of y
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2)

print(x_train.shape,y_train.shape)
print(x_test.shape,y_test.shape)
```

```
(52736, 80) (52736,)
(13184, 80) (13184,)
```

```
# Checking skewness of Y
da = pd.DataFrame(y)
da.ALLDGS.skew()
```

```
1.3265830536421945
```

```
# Checking skewness of Y
da = pd.DataFrame(y_train)
da.ALLDGS.skew()
```

```
1.3528419249715458
```

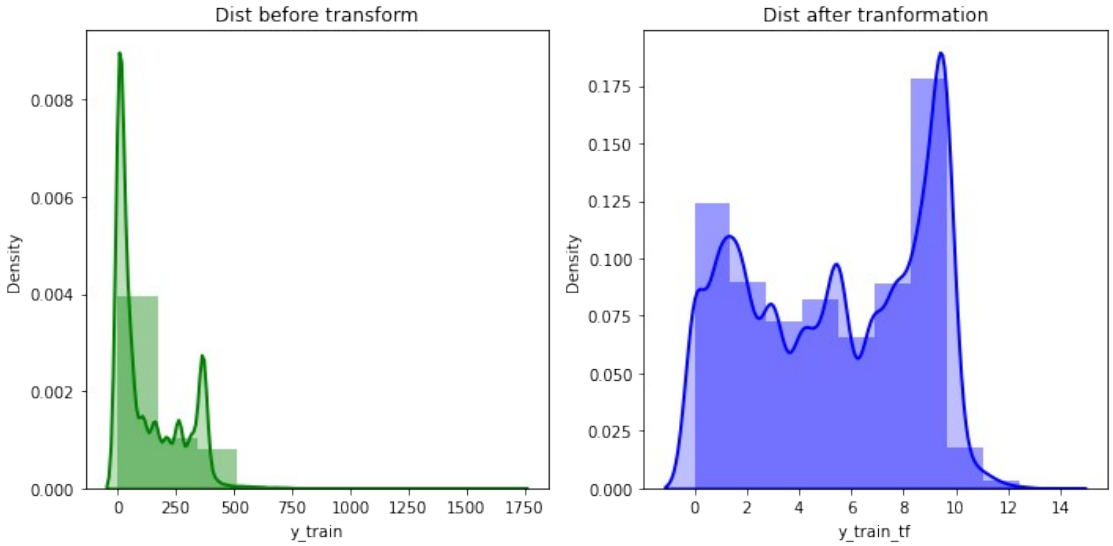
```
##########################################

# Applying Boxcox transform on y_train
tf = vt.BoxCoxTransformer(variables = ['ALLDGS'])
y_train_tf = tf.fit_transform(pd.DataFrame(y_train))
y_train_tf.ALLDGS.skew() # As you can see, skewness is close to zero
(Normal dist)
```

```
-0.14128313673533535
```

```
stats.boxcox((pd.DataFrame(y_train)).ALLDGS)[1] # LAMBDA value for
y_train
```

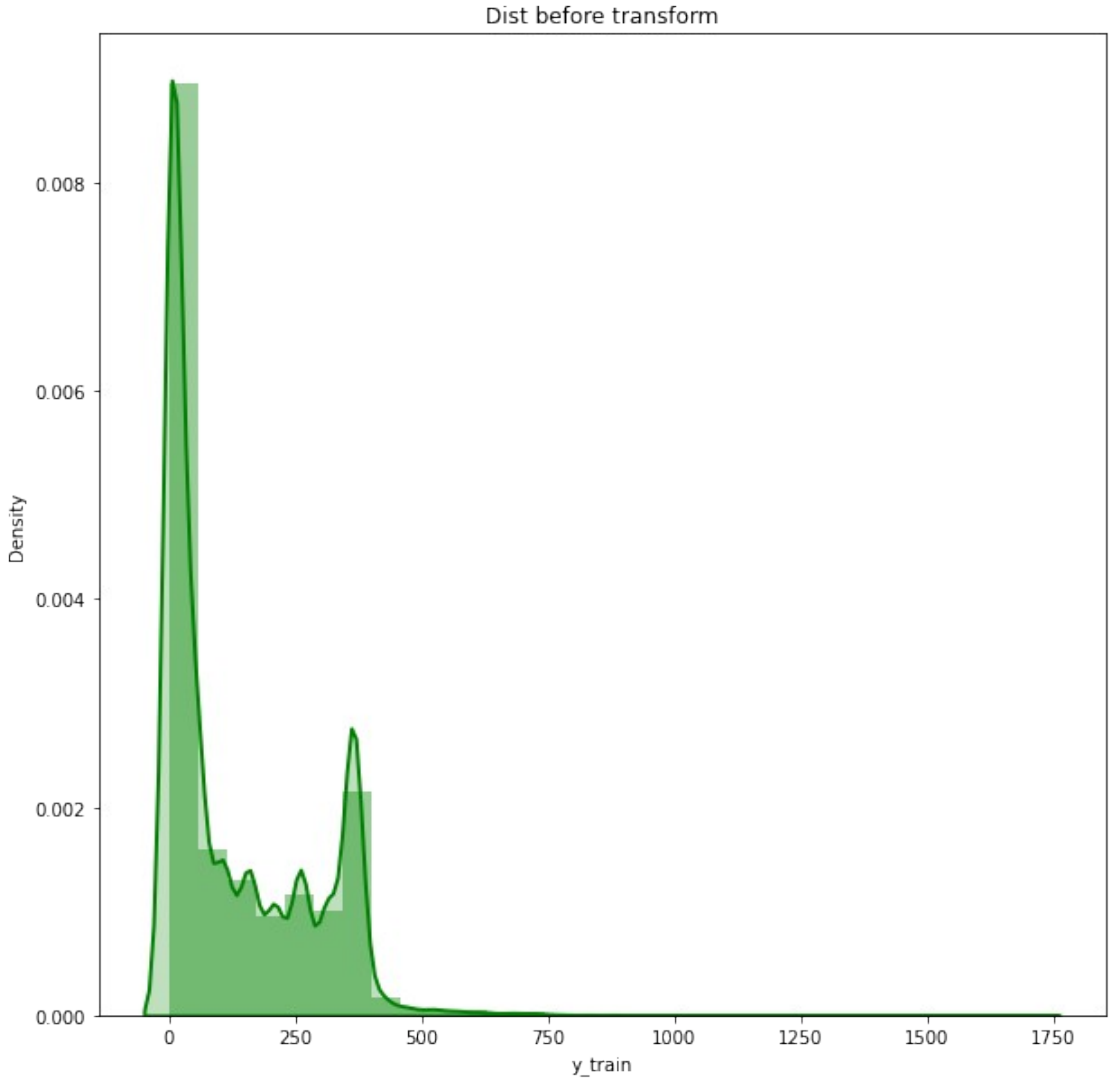
```
0.15289231046275192
```

```
# creating axes to draw plots
import seaborn as sns
fig, ax = plt.subplots(1, 2, figsize=(10, 5))

# plotting the original data(non-normal) and
# fitted data (normal)
sns.distplot(y_train, hist = True, kde = True,
             kde_kws = {'shade': True, 'linewidth': 2},
             label = "Non-Normal", color ="green", ax = ax[0],bins=10)
ax[0].title.set_text('Dist before transform')
ax[0].set_xlabel('y_train')

sns.distplot(y_train_tf, hist = True, kde = True,
             kde_kws = {'shade': True, 'linewidth': 2},
             label = "Normal", color ="blue", ax = ax[1],bins=10)
plt.xlabel('y_train_tf')
plt.title('Dist after tranformation')

fig.tight_layout()
```




```
# creating axes to draw plots
import seaborn as sns
fig, ax = plt.subplots(1, 1, figsize=(10, 10))

# plotting the original data(non-normal) and
# fitted data (normal)
sns.distplot(y_train, hist = True, kde = True,
             kde_kws = {'shade': True, 'linewidth': 2},
             label = "Non-Normal", color ="green", ax = ax,bins=30)
ax.title.set_text('Dist before transform')
ax.set_xlabel('y_train')
```

```
Text(0.5, 0, 'y_train')
```



Not using Box-Cox

LINEAR REGRESSION

```
# Linear Regression Modeling
# Fitting to the model
from sklearn import linear_model
lm = linear_model.LinearRegression()
model = lm.fit(x_train,y_train) # Passing the training data for
Predictor & Target variables
prediction = lm.predict(x_test) # The predicted y_test from test data
of predicted variable

# model scores for train and test
import sklearn as sklearn
print(model.score(x_train,y_train))
print(model.score(x_test,y_test))
```

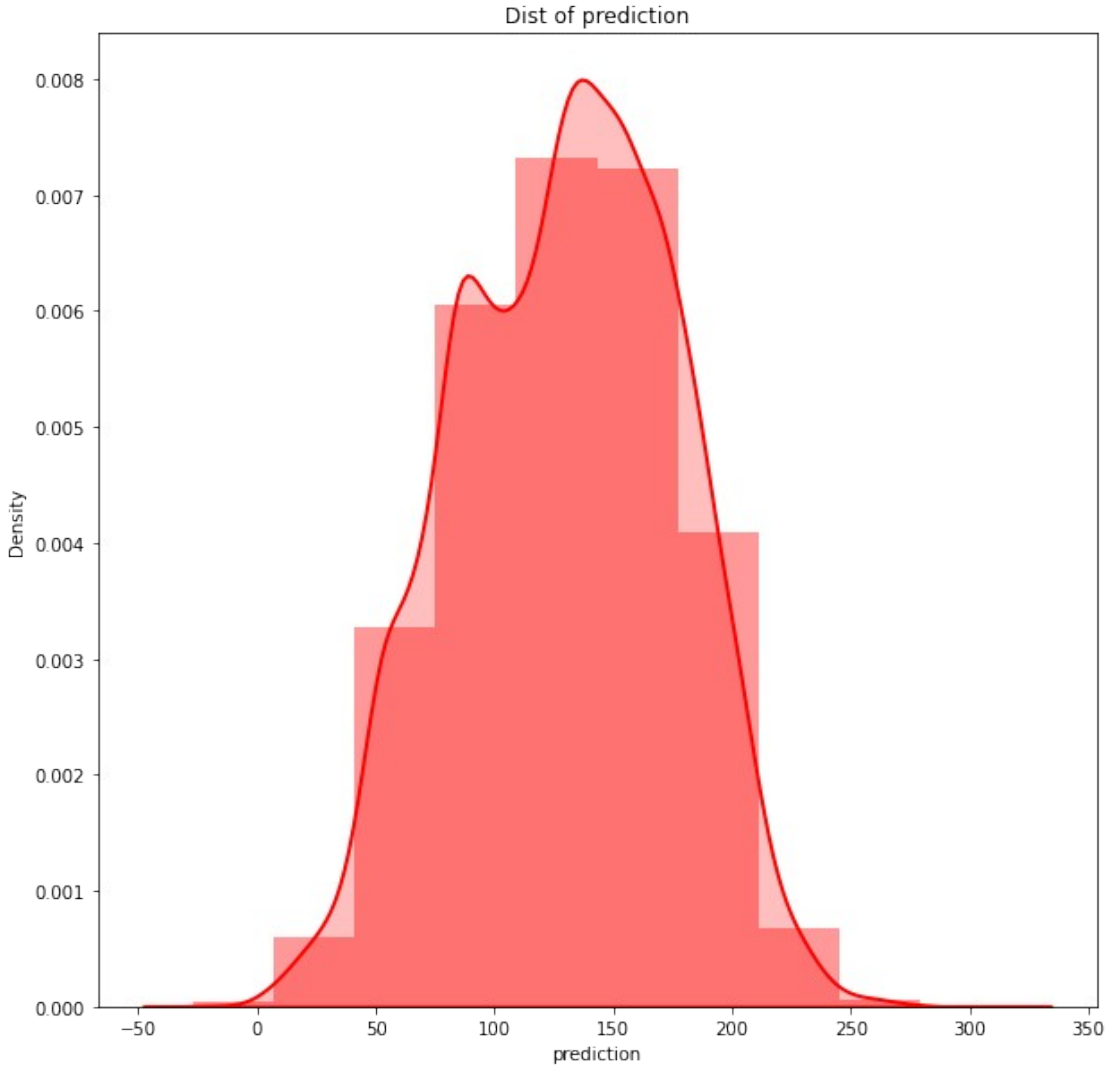
```
0.09436835695801271
0.08648754625047828
```

```
# print the coefficient
# Linear Reg eqn y = B0 + B1.x1 + B2.x2 +...+ BN.xN
print(model.intercept_) # intercept_ is used to find B0 (intercept or
c)
print(model.coef_) # other coeffcients (B1, B2...BN)
```

```
-864209604472485.9
[ 1.48021543e+01  6.95576797e+01 -2.13752329e+01  4.21708344e+00
 -8.74559232e+14 -8.74559232e+14 -8.74559232e+14 -8.74559232e+14
 -8.74559232e+14 -8.74559232e+14  2.00070968e+14  2.00070968e+14
  2.00070968e+14  2.00070968e+14  2.00070968e+14  2.00070968e+14
 -8.38725623e+14 -8.38725623e+14 -8.38725623e+14 -8.38725623e+14
 -8.38725623e+14 -8.38725623e+14 -4.98439144e+14 -4.98439144e+14
 -4.98439144e+14 -4.98439144e+14 -4.98439144e+14 -4.98439144e+14
  3.82032808e+13  3.82032808e+13  3.82032808e+13  3.82032808e+13
  3.82032808e+13  3.82032808e+13  7.20687797e+13  7.20687797e+13
  1.23322195e+14  1.23322195e+14  1.23322195e+14  1.23322195e+14
  1.23322195e+14  1.23322195e+14  1.23322195e+14  1.23322195e+14
  1.23322195e+14  1.23322195e+14  1.23322195e+14 -8.25935991e+14
 -8.25935991e+14 -8.25935991e+14 -8.25935991e+14  9.50373893e+14
  9.50373893e+14  9.50373893e+14  9.50373893e+14  9.50373893e+14
  9.50373893e+14  9.50373893e+14 -4.19542660e+14 -4.19542660e+14
 -4.19542660e+14 -4.19542660e+14 -4.19542660e+14 -1.69010055e+15
 -1.69010055e+15 -1.69010055e+15 -1.69010055e+15 -1.69010055e+15
 -1.69010055e+15 -1.69010055e+15  1.63925586e+15  1.63925586e+15
  1.63925586e+15  1.63925586e+15  1.50757367e+15  1.50757367e+15
  1.50757367e+15  1.48064416e+15  1.48064416e+15  1.48064416e+15]
```

```
# Checking the distribution of prediction
import seaborn as sns
fig, ax = plt.subplots(1,1,figsize=(10, 10))

# plotting the original data(non-normal) and
# fitted data (normal)
sns.distplot(prediction, hist = True, kde = True,
             kde_kws = {'shade': True, 'linewidth': 2},
             label = "Non-Normal", color ="red", ax = ax,bins=10)
ax.title.set_text('Dist of prediction')
ax.set_xlabel('prediction')
```

```
Text(0.5, 0, 'prediction')
```




```
print(max(prediction))
print(min(prediction))
```

```
313.375
-27.25
```

```
# Evaluation metrics

import sklearn.metrics as metrics
mae = metrics.mean_absolute_error(y_test, prediction)
mse = metrics.mean_squared_error(y_test, prediction)
rmse = np.sqrt(mse) #mse**(0.5)
r2 = metrics.r2_score(y_test,prediction)
mape = metrics.mean_absolute_percentage_error(y_test, prediction)

print("Results of sklearn.metrics:")
print("MAE:",mae)
print("MSE:", mse)
```

```
print("RMSE:", rmse)
print("R-Squared:", r2)
print("MAPE:",mape)
```

```
Results of sklearn.metrics:
MAE: 116.80163455703884
MSE: 20683.889513330552
RMSE: 143.818946990063
R-Squared: 0.08648754625047828
MAPE: 15.828576233565963
```

RANDOM FOREST REGRESSOR

```
# Hyperparameter Tuning

# Step 1

# Search space
import numpy as np
from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.arange(2,101)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.arange(2,31,2)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [int(x) for x in np.arange(2,51,2)]
# Minimum number of samples required at each leaf node
min_samples_leaf = [int(x) for x in np.arange(2,51,2)]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
pprint(random_grid)

# Step 2 - Randomized Search & Model Building

# # SKIP if you already have the hyperparameters
# Use the random grid to search for best hyperparameters
# First create the base model to tune
from sklearn.ensemble import RandomForestRegressor
rfa = RandomForestRegressor()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available
cores
```

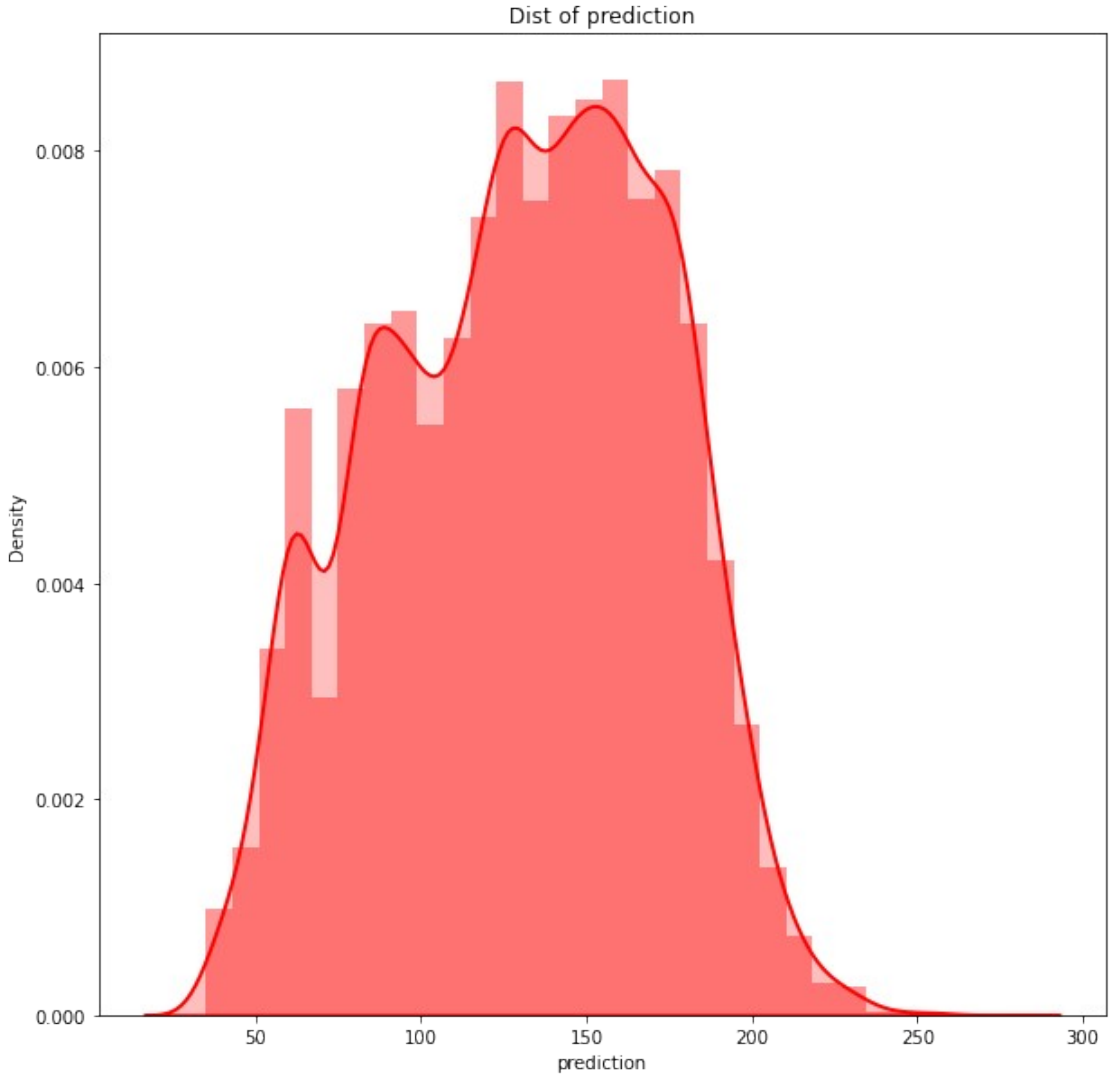
```
rfa_bestfit = RandomizedSearchCV(estimator = rfa, param_distributions
= random_grid, n_iter = 100, cv = 3, verbose=2, random_state=42,
n_jobs = -1)
# Fit the random search model
rfa_bestfit.fit(x_train, y_train)

rfa_bestfit.best_params_ # These are the parameters that we are
setting to our model to get the best possible accuracy
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits
```

```
{'n_estimators': 52,
 'min_samples_split': 26,
 'min_samples_leaf': 10,
 'max_features': 'sqrt',
 'max_depth': 14,
 'bootstrap': False}
```

```
# PARAMETERS
# {'n_estimators': 52,
#  'min_samples_split': 26,
#  'min_samples_leaf': 10,
#  'max_features': 'sqrt',
#  'max_depth': 14,
#  'bootstrap': False}

# Do this to avoid Random Search
# model built with the optimal parameters
from sklearn.ensemble import RandomForestRegressor
rfa_bestfit = RandomForestRegressor(n_estimators= 52,
 min_samples_split= 26,
 min_samples_leaf= 10,
 max_features= 'sqrt',
 max_depth= 14,
 bootstrap= False, random_state=42)
rfa_bestfit = rfa_bestfit.fit(x_train,y_train)
prediction = rfa_bestfit.predict(x_test) # Prediction

# Checking the distribution of prediction
import seaborn as sns
fig, ax = plt.subplots(1,1,figsize=(10, 10))

# plotting the original data(non-normal) and
# fitted data (normal)
sns.distplot(prediction, hist = True, kde = True,
             kde_kws = {'shade': True, 'linewidth': 2},
             label = "Non-Normal", color ="red", ax = ax,bins=30)
ax.title.set_text('Dist of prediction')
ax.set_xlabel('prediction')
```

```
Text(0.5, 0, 'prediction')
```




```
print(min(np.array(prediction)))
print(max(np.array(prediction)))
```

```
34.98647937554737
274.2373846419686
```

```
# Model Evaluation

# score on Training data
print(rfa_bestfit.score(x_train,y_train))
```

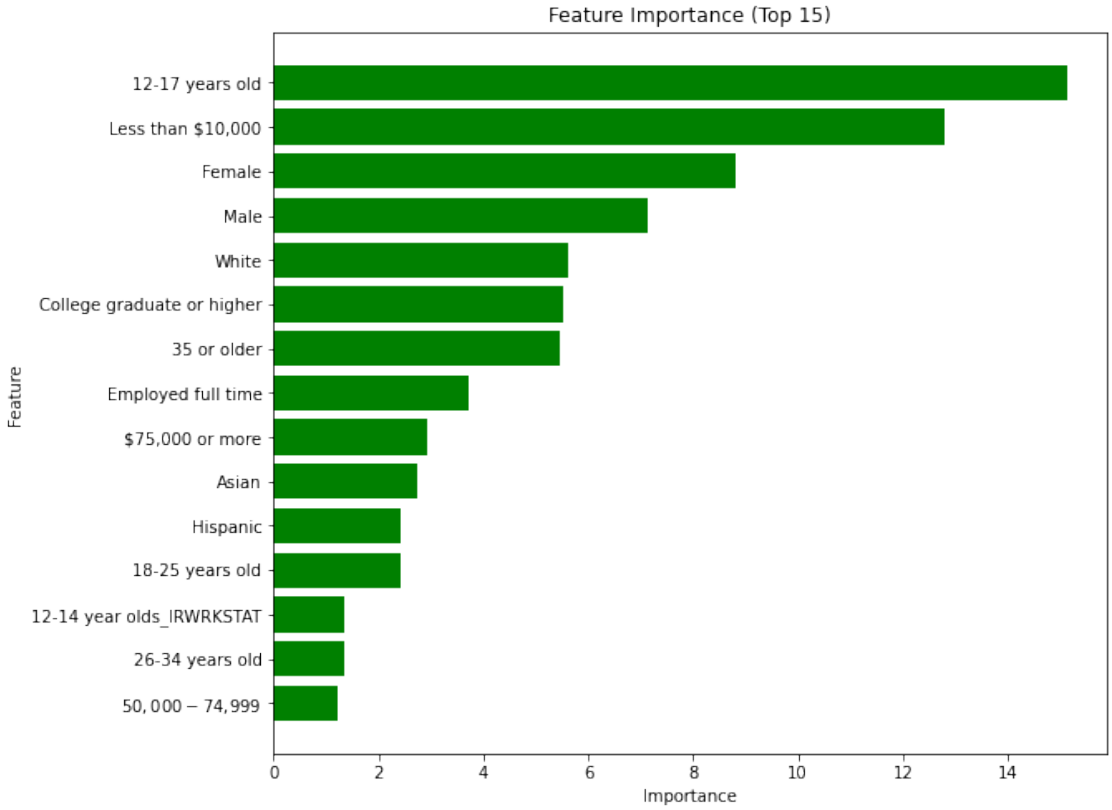
```
0.14587565491390486
```

```
# score on Test data
print(rfa_bestfit.score(x_test,y_test))
```

```
0.0887552559157
```

```
# Prediction
prediction = rf_bestfit.predict(x_test) # The predicted y_test from
test data of predicted variable

# Evaluation metrics

import sklearn.metrics as metrics
mae = metrics.mean_absolute_error(y_test, prediction)
mse = metrics.mean_squared_error(y_test, prediction)
rmse = np.sqrt(mse) #mse**(0.5)
r2 = metrics.r2_score(y_test,prediction)
mape = metrics.mean_absolute_percentage_error(y_test, prediction)

print("Results of sklearn.metrics:")
print("MAE:",mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R-Squared:", r2)
print("MAPE:",mape)

Results of sklearn.metrics:
MAE: 117.2241321299863
MSE: 20632.543682224215
RMSE: 143.64032749274912
R-Squared: 0.0887552559157
MAPE: 16.076025282912205

# Feature importance in prediction
import sklearn.metrics as metrics
bcfit = rfa_bestfit
imp_feature = bcfit.feature_importances_
columns = x.columns
clf_co = pd.Series(imp_feature,columns)

## top features
# Feature importance graph
# Top 15 most imp features
dfimp =
pd.DataFrame(clf_co.sort_values(ascending=False)).reset_index()
dfimp.columns.values[0] = 'Feature'
dfimp.columns.values[1] = 'Importance'
dfimp['Importance'] = round(dfimp['Importance']*100,2)
dplt1 = dfimp.head(15)
dplt2 = dplt1.copy()
dplt2.sort_values(by='Importance',ascending=True,inplace=True)

plt.barh(dplt2['Feature'],dplt2['Importance'],color='green')
plt.title('Feature Importance (Top 15)')
plt.ylabel('Feature')
plt.xlabel('Importance')
print(dplt1)
```

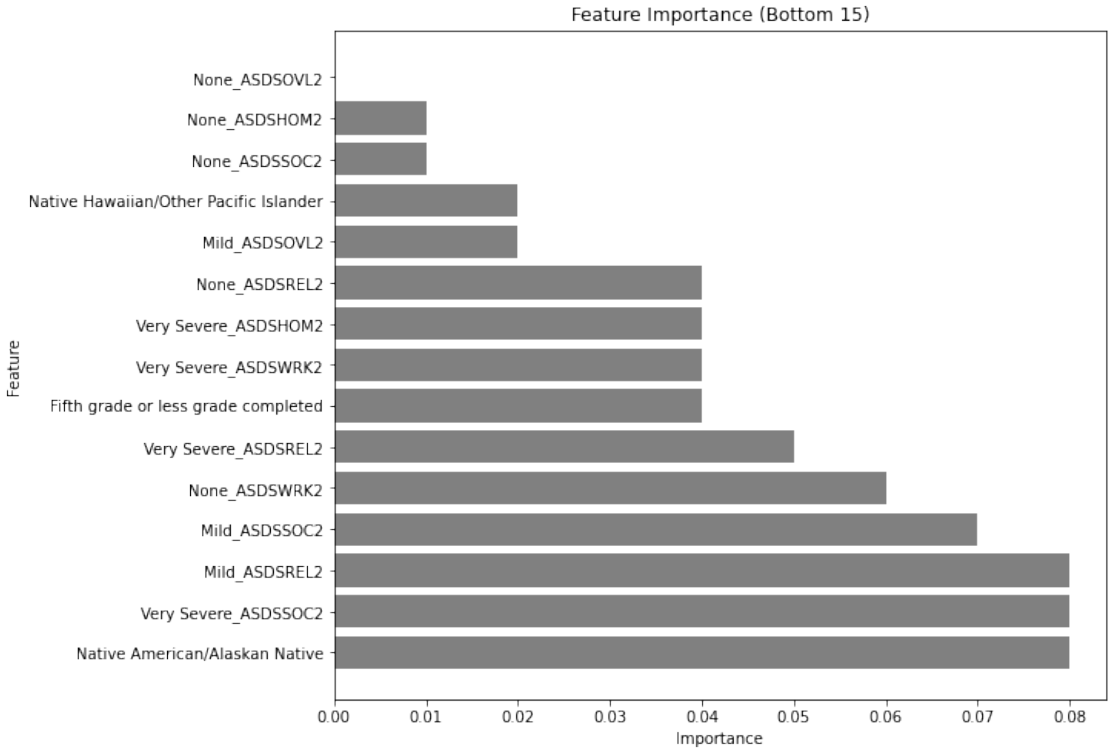
	Feature	Importance
0	12-17 years old	15.13
1	Less than $10,000	12.81
2	Female	8.82
3	Male	7.13
4	White	5.61
5	College graduate or higher	5.53
6	35 or older	5.46
7	Employed full time	3.73
8	$75,000 or more	2.94
9	Asian	2.74
10	18-25 years old	2.43
11	Hispanic	2.43
12	12-14 year olds_IRWRKSTAT	1.36
13	26-34 years old	1.34
14	$50,000-$74,999	1.21




```
## bottom features
# Feature importance graph
# Bottom 15 (15 least imp features)
dfimp =
pd.DataFrame(clf_co.sort_values(ascending=False)).reset_index()
dfimp.columns.values[0] = 'Feature'
dfimp.columns.values[1] = 'Importance'
dfimp['Importance'] = round(dfimp['Importance']*100,2)
dplt1 = dfimp.tail(15)
```

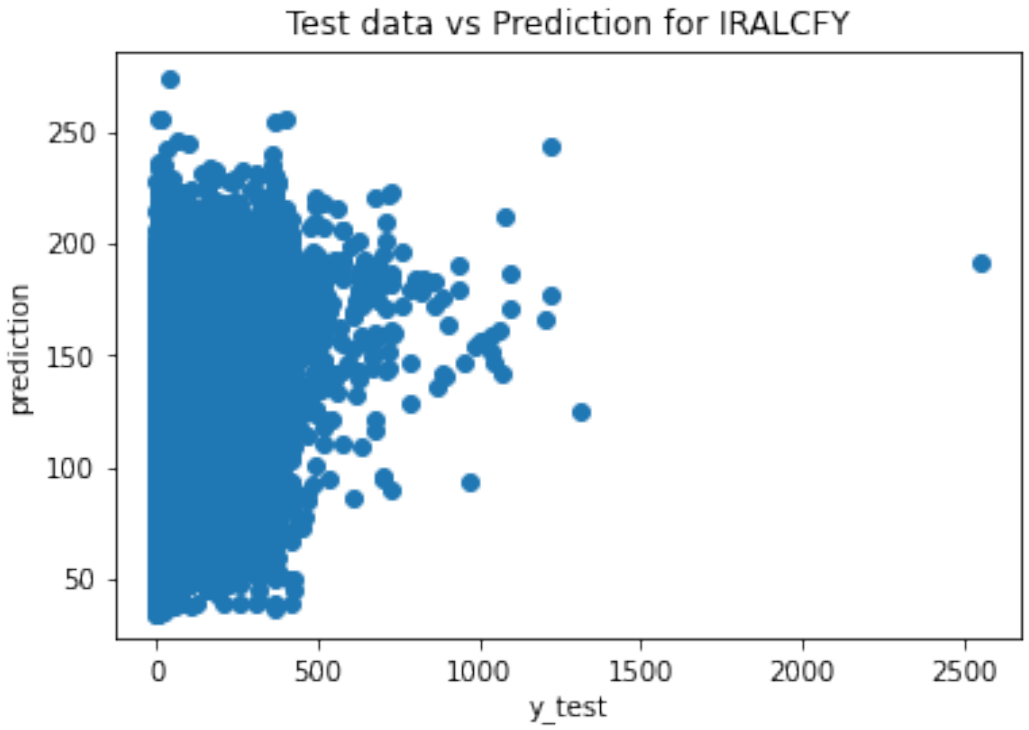
```
dplt2 = dplt1.copy()
dplt2.sort_values(by='Importance',ascending=False,inplace=True)

plt.barh(dplt2['Feature'],dplt2['Importance'],color='gray')
plt.title('Feature Importance (Bottom 15)')
plt.ylabel('Feature')
plt.xlabel('Importance')
print(dplt2.sort_values(by='Importance',ascending=True))
```

	Feature	Importance
79	None_ASDSOVL2	0.00
77	None_ASDSSOC2	0.01
78	None_ASDSHOM2	0.01
75	Mild_ASDSOVL2	0.02
76	Native Hawaiian/Other Pacific Islander	0.02
71	Fifth grade or less grade completed	0.04
72	Very Severe_ASDSWRK2	0.04
73	Very Severe_ASDSHOM2	0.04
74	None_ASDSREL2	0.04
70	Very Severe_ASDSREL2	0.05
69	None_ASDSWRK2	0.06
68	Mild_ASDSSOC2	0.07
65	Native American/Alaskan Native	0.08
66	Very Severe_ASDSSOC2	0.08
67	Mild_ASDSREL2	0.08



```
# Plotting the chart
import seaborn as sns
plt.scatter(y_test,prediction)
# .set(xlabel = 'y_test', ylabel = 'prediction', title = 'Test data vs
Prediction for IRALCFY')
plt.xlabel('y_test')
plt.ylabel('prediction')
plt.title('Test data vs Prediction for IRALCFY')
```

```
Text(0.5, 1.0, 'Test data vs Prediction for IRALCFY')
```



##CATBOOST REGRESSOR

```
# Removing rows with ALLDGSCAT = 'Never used'
dvn = dv[dv['ALLDGSCAT'] == 'Ever used']

X =
dvn.drop(dvn.columns[[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,35,36,3
7,38,39]],axis=1)
y = dvn['ALLDGS']

# Declaring Categorical features
cat_features = list(range(0, X.shape[1]))
print(cat_features)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17]
```

```
# Train test split
from sklearn.model_selection import train_test_split
```

```
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2,
random_state=0)

!pip install catboost

# catboost implementation
from catboost import CatBoostRegressor

clf = CatBoostRegressor(
    iterations=5,
    learning_rate=0.1,
    #loss_function='CrossEntropy'
)


clf.fit(X_train, y_train,
        cat_features=cat_features,
        eval_set=(X_val, y_val),
        verbose=False
)

print('CatBoost model is fitted: ' + str(clf.is_fitted()))
print('CatBoost model parameters:')
print(clf.get_params())
```

```
CatBoost model is fitted: True
CatBoost model parameters:
{'iterations': 5, 'learning_rate': 0.1, 'loss_function': 'RMSE'}
```

```
# Stdout of the training
from catboost import CatBoostRegressor
clf = CatBoostRegressor(
    iterations=10,
#     verbose=5,
)

clf.fit(
    X_train, y_train,
    cat_features=cat_features,
    eval_set=(X_val, y_val),
)
```

```
Learning rate set to 0.5
0:    learn: 145.5364664    test: 148.6019577    best: 148.6019577 (0)
      total: 113ms    remaining: 1.02s
1:    learn: 143.8200433    test: 146.8313184    best: 146.8313184 (1)
      total: 236ms    remaining: 943ms
2:    learn: 143.3409063    test: 146.3004773    best: 146.3004773 (2)
      total: 328ms    remaining: 766ms
3:    learn: 143.1069057    test: 146.0582593    best: 146.0582593 (3)
      total: 375ms    remaining: 563ms
```

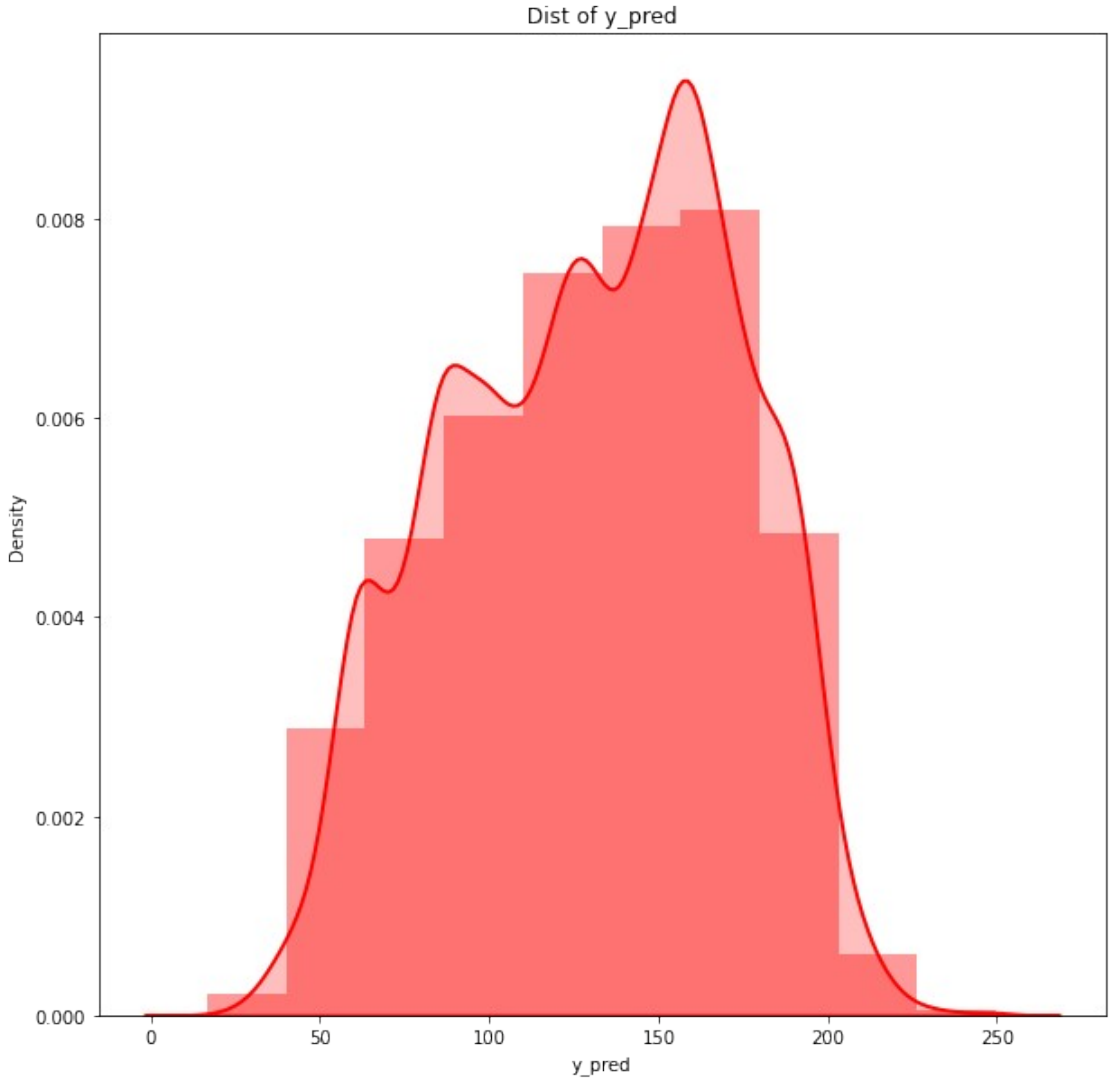
```
4:    learn: 142.7957502    test: 145.6702745    best: 145.6702745 (4)
      total: 471ms    remaining: 471ms
5:    learn: 142.6750708    test: 145.4848649    best: 145.4848649 (5)
      total: 554ms    remaining: 369ms
6:    learn: 142.5517910    test: 145.3794689    best: 145.3794689 (6)
      total: 671ms    remaining: 288ms
7:    learn: 142.3937752    test: 145.1993597    best: 145.1993597 (7)
      total: 789ms    remaining: 197ms
8:    learn: 142.2264626    test: 145.0177022    best: 145.0177022 (8)
      total: 853ms    remaining: 94.8ms
9:    learn: 142.0290909    test: 144.8759360    best: 144.8759360 (9)
      total: 940ms    remaining: 0us

bestTest = 144.875936
bestIteration = 9


<catboost.core.CatBoostRegressor at 0x7f3fcd1c0790>
```

```
print(clf.predict(X_val)) # Predictions
```

```
[157.39548786 184.53397091 168.42036269 ... 126.81700489 154.26425932
 176.29968626]
```

```
y_pred = clf.predict(X_val)

# Checking the distribution of prediction
import seaborn as sns
fig, ax = plt.subplots(1,1,figsize=(10, 10))

# plotting the original data(non-normal) and
# fitted data (normal)
sns.distplot(y_pred, hist = True, kde = True,
             kde_kws = {'shade': True, 'linewidth': 2},
             label = "Non-Normal", color ="red", ax = ax,bins=10)
ax.title.set_text('Dist of y_pred')
ax.set_xlabel('y_pred')
```

```
Text(0.5, 0, 'y_pred')
```




```
print(min(np.array(y_pred)))
print(max(np.array(y_pred)))
```

```
16.645953295045828
249.87113788248803
```

```
# score on Training data
print(clf.score(X_train,y_train))
```

```
0.09153999456372541
```

```
# score on Test data
print(clf.score(X_val,y_val))
```

```
0.09357765755734104
```

```
# Evaluation Metrics

import sklearn.metrics as metrics
```

```
mae = metrics.mean_absolute_error(y_val, y_pred)
mse = metrics.mean_squared_error(y_val, y_pred)
rmse = np.sqrt(mse) #mse**(0.5)
r2 = metrics.r2_score(y_val,y_pred)
mape = metrics.mean_absolute_percentage_error(y_val, y_pred)

print("Results of sklearn.metrics:")
print("MAE:",mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R-Squared:", r2)
print("MAPE:",mape)
```

```
Results of sklearn.metrics:
MAE: 118.36996281804136
MSE: 20989.036830604855
RMSE: 144.87593599561265
R-Squared: 0.09357765755734104
MAPE: 16.57605053667785
```

#**MODEL SELECTION**

For IRALCFY:

Linear Regression:

Results of sklearn.metrics:

- MAE: 67.3325347936132
- MSE: 7585.800442891922
- RMSE: 87.09650074998376
- R-Squared: 0.0773372152438615
- MAPE: 6.050438707733569

Random Forest:

Results of sklearn.metrics:

- MAE: 66.65218863624317
- MSE: 7501.849392723051
- RMSE: 86.61321719416183
- R-Squared: 0.09366934372894598
- MAPE: 5.851723056694891

CatBoost:

Results of sklearn.metrics:

- MAE: 67.0744796879123
- MSE: 7524.596701757585

- RMSE: 86.74443326091644
- R-Squared: 0.08847903238038313
- MAPE: 5.963419228648019

We choose Random Forest Regressor for Alcohol frequency as the best model, since it has the lowest RMSE and MAPE values

For ALLDGS:

Linear Regression:

Results of sklearn.metrics:

- MAE: 116.80163455703884
- MSE: 20683.889513330552
- RMSE: 143.818946990063
- R-Squared: 0.08648754625047828
- MAPE: 15.828576233565963

Random Forest:

Results of sklearn.metrics:

- MAE: 117.2241321299863
- MSE: 20632.543682224215
- RMSE: 143.64032749274912
- R-Squared: 0.0887552559157
- MAPE: 16.076025282912205

CatBoost:

Results of sklearn.metrics:

- MAE: 118.36996281804136
- MSE: 20989.036830604855
- RMSE: 144.87593599561265
- R-Squared: 0.09357765755734104
- MAPE: 16.57605053667785

We choose Random Forest Regressor for Alldrugs frequency as the best model, since it has the lowest RMSE value

PREDICTION

```
# NOTE: RUN THIS CODE ONLY AFTER RUNNING THE MODELS: cat_tm_ca,
cat_tm_cd, rf_bestfit & rfa_bestfit

# sample input data1
data1 = [0, 0,  1,   0,   1,   0,   0,   0,   0,   0,
```

```
          1,     0,   0,   0,   0,   0,   1,   0,   0,   0,
          0,     0,   1,   0,   0,   0,   0,   0,   1,   0,
          0,     0,   0,   0,   0,   1,   0,   0,   0,   0,
          0,     0,   0,   0,   0,   1,   0,   0,   0,   0,
          1,     0,   0,   0,   0,   0,   0,   1,   0,   1,
          0,     0,   0,   0,   0,   0,   1,   0,   0,   0,
          0,     1,   0,   0,   1,   0,   0,   0,   0,   1]

list1 = data1

# Code to predict
y_pred_alc = cat_tm_ca.predict([list1])
y_pred_drug = cat_tm_cd.predict([list1])
if y_pred_alc == 0:
  print('Does not consume Alcohol')
else:
  print('Consumed Alcohol')

if y_pred_drug == 0:
  print('Does not consume Drugs')
else:
  print('Consumed Drugs')

if y_pred_alc == 1:
  print('Avg Alcohol consumption per year:',
round(rf_bestfit.predict([list1])[0]))

if y_pred_drug == 1:
  print('Avg Alldrugs consumption per year:',
round(rfa_bestfit.predict([list1])[0]))
```

```
Consumed Alcohol
Does not consume Drugs
Avg Alcohol consumption per year: 108
```

```
# sample input data2
data2 = [0,     0,   1,   0,   1,   0,   0,   0,   0,   0,   1,
     0,   0,   0,   0,   0,   1,   0,   0,
         0,     0,   0,   1,   0,   0,   0,   0,   0,   1,   0,
     0,   0,   0,   0,   0,   1,   0,   0,
         1,     0,   0,   0,   0,   0,   0,   0,   0,   1,   0,
     0,   0,   0,   0,   0,   1,   0,   0,
         0,     0,   0,   0,   1,   0,   0,   0,   0,   0,   0,
     0,   1,   1,   0,   0,   0,   1,   0,
         0,     0,   0,   1]

list2 = data2

# Code to predict
y_pred_alc = cat_tm_ca.predict([list2])
```

```
y_pred_drug = cat_tm_cd.predict([list2])
if y_pred_alc == 0:
  print('Does not consume Alcohol')
else:
  print('Consumed Alcohol')

if y_pred_drug == 0:
  print('Does not consume Drugs')
else:
  print('Consumed Drugs')

if y_pred_alc == 1:
  print('\n')
  print('Avg Alcohol consumption per year:',
round(rf_bestfit.predict([list1])[0]))

if y_pred_drug == 1:
  print('\n')
  print('Avg Alldrugs consumption per year:',
round(rfa_bestfit.predict([list1])[0]))
```

```
Does not consume Alcohol
Does not consume Drugs
```

```
# sample input data3
data3 = [0, 0, 1, 0, 1, 0, 0, 0, 0, 0,
         1, 0, 0, 0, 0, 0, 1, 0, 0, 0,
         0, 0, 1, 0, 0, 0, 0, 0, 1, 0,
         0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
         0, 1, 0, 0, 0, 0, 0, 0, 1, 0,
         0, 0, 0, 0, 0, 0, 0, 1, 0, 1,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
         0, 0, 1, 0, 0, 1, 0, 1, 0, 0]

list1 = data3

# Code to predict
y_pred_alc = cat_tm_ca.predict([list1])
y_pred_drug = cat_tm_cd.predict([list1])
if y_pred_alc == 0:
  print('Does not consume Alcohol')
else:
  print('Consumed Alcohol')

if y_pred_drug == 0:
  print('Does not consume Drugs')
else:
  print('Consumed Drugs')

if y_pred_alc == 1:
```

```
  print('Avg Alcohol consumption per year:',
round(rf_bestfit.predict([list1])[0]))

if y_pred_drug == 1:
  print('Avg Alldrugs consumption per year:',
round(rfa_bestfit.predict([list1])[0]))
```

```
Consumed Alcohol
Consumed Drugs
Avg Alcohol consumption per year: 68
Avg Alldrugs consumption per year: 142
```