# Cocaine consumption - classification models

Jarosław Leski, Robert Kowalczyk

The goal of this work is to apply various ML algorithms to build a model explaining whether a particular person consumed cocaine in the last month based on the training sample and generate predictions for all observations from the test sample.

#### 1. Importing libraries

import pandas as pd   
import seaborn as sns  
import numpy as np   
import matplotlib.pyplot as plt   
from datetime import datetime as dt   
import warnings  
from sklearn.preprocessing import OrdinalEncoder  
from sklearn.model\_selection import KFold  
from sklearn.metrics import balanced\_accuracy\_score, confusion\_matrix, f1\_score, precision\_score, roc\_auc\_score, recall\_score, accuracy\_score  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.model\_selection import GridSearchCV  
from sklearn.pipeline import Pipeline  
from sklearn.preprocessing import RobustScaler  
from sklearn.linear\_model import LogisticRegression  
from sklearn.model\_selection import train\_test\_split  
from collections import Counter  
from imblearn.over\_sampling import SMOTE  
from sklearn import model\_selection  
from sklearn.linear\_model import LogisticRegression  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn.naive\_bayes import GaussianNB  
from sklearn.svm import SVC  
import xgboost as xgb  
import lightgbm as   
import optuna  
from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis  
from sklearn.discriminant\_analysis import QuadraticDiscriminantAnalysis  
  
%matplotlib inline  
warnings.filterwarnings("ignore")  
pd.set\_option("display.max\_columns",100)

## 2. Loading data, EDA and preprocessing

The dataset contains following columns:

* **id** – unique observation identifier
* **age** – age group of the person with the following levels: 18-24, 25-34, 35-44, 45-54, 55-64, 65+
* **gender** – gender of the person with the following levels: female, male
* **education** – education level of the person with the following levels: Left school before 16 years, Left school at 16 years, Left school at 17 years, Left school at 18 years, Some college or university, no certificate or degree, Professional certificate/ diploma, University degree, Masters degree, Doctorate degree
* **country** – country of current residence of the person with the following levels: Australia, Canada, New Zealand, Ireland, UK, USA, Other
* **ethnicity** – ethnicity of the person with the following levels: Asian, Black, Mixed-Black/Asian, Mixed-White/Asian, Mixed-White/Black, White, Other
* **personality\_neuroticism** – assessment of neuroticism of the person based on psychological tests (0-100)
* **personality\_extraversion** – assessment of extraversion of the person based on psychological tests (0-100)
* **personality\_openness** – assessment of openness to experience of the person based on psychological tests (0-100)
* **personality\_agreeableness** – assessment of agreeableness of the person based on psychological tests (0-100)
* **personality\_conscientiousness** – assessment of conscientiousness of the person based on psychological tests (0-100)
* **personality\_impulsiveness** – assessment of impulsiveness of the person based on psychological tests (0-100)
* **personality\_sensation** – assessment of sensation of the person based on psychological tests (0-100)
* **consumption\_alcohol** – declared consumption of alcohol with the following levels: never used, used over a decade ago, used in last decade, used in last year, used in last month, used in last week, used in last day
* **consumption\_amphetamines** – declared consumption of amphetamines with the following levels: never used, used over a decade ago, used in last decade, used in last year, used in last month, used in last week, used in last day
* **consumption\_caffeine** – declared consumption of caffeine with the following levels: never used, used over a decade ago, used in last decade, used in last year, used in last month, used in last week, used in last day
* **consumption\_cannabis** – declared consumption of cannabis with the following levels: never used, used over a decade ago, used in last decade, used in last year, used in last month, used in last week, used in last day
* **consumption\_chocolate** – declared consumption of chocolate with the following levels: never used, used over a decade ago, used in last decade, used in last year, used in last month, used in last week, used in last day
* **consumption\_mushrooms** – declared consumption of magic mushrooms with the following levels: never used, used over a decade ago, used in last decade, used in last year, used in last month, used in last week, used in last day
* **consumption\_nicotine** – declared consumption of nicotine with the following levels: never used, used over a decade ago, used in last decade, used in last year, used in last month, used in last week, used in last day
* **consumption\_cocaine\_last\_month** – declared consumption of cocaine in the last month with the following levels: No, Yes (outcome variable, only in the training sample)

Let's load the dataset and see if there are any missing data.

drugs = pd.read\_csv('drugs\_train.csv')  
drugs.info()  
drugs.describe()

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 1500 entries, 0 to 1499  
Data columns (total 21 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 id 1500 non-null object   
 1 age 1500 non-null object   
 2 gender 1500 non-null object   
 3 education 1500 non-null object   
 4 country 1500 non-null object   
 5 ethnicity 1500 non-null object   
 6 personality\_neuroticism 1500 non-null float64  
 7 personality\_extraversion 1500 non-null float64  
 8 personality\_openness 1500 non-null float64  
 9 personality\_agreeableness 1500 non-null float64  
 10 personality\_conscientiousness 1500 non-null float64  
 11 personality\_impulsiveness 1500 non-null float64  
 12 personality\_sensation 1500 non-null float64  
 13 consumption\_alcohol 1500 non-null object   
 14 consumption\_amphetamines 1500 non-null object   
 15 consumption\_caffeine 1500 non-null object   
 16 consumption\_cannabis 1500 non-null object   
 17 consumption\_chocolate 1500 non-null object   
 18 consumption\_mushrooms 1500 non-null object   
 19 consumption\_nicotine 1500 non-null object   
 20 consumption\_cocaine\_last\_month 1500 non-null object   
dtypes: float64(7), object(14)  
memory usage: 246.2+ KB

personality\_neuroticism personality\_extraversion \  
count 1500.000000 1500.000000   
mean 51.507267 50.053667   
std 14.958815 15.200954   
min 0.000000 0.000000   
25% 41.300000 39.400000   
50% 52.000000 50.100000   
75% 60.800000 59.700000   
max 100.000000 100.000000   
  
 personality\_openness personality\_agreeableness \  
count 1500.000000 1500.000000   
mean 53.108667 49.966000   
std 16.049880 14.340301   
min 0.000000 0.000000   
25% 41.400000 41.200000   
50% 52.700000 49.800000   
75% 64.700000 58.500000   
max 100.000000 100.000000   
  
 personality\_conscientiousness personality\_impulsiveness \  
count 1500.000000 1500.000000   
mean 49.998800 46.972200   
std 14.571548 17.361892   
min 0.000000 0.000000   
25% 40.600000 33.800000   
50% 49.900000 42.800000   
75% 58.400000 56.500000   
max 100.000000 100.000000   
  
 personality\_sensation   
count 1500.000000   
mean 52.316333   
std 23.686221   
min 0.000000   
25% 38.800000   
50% 54.000000   
75% 71.100000   
max 100.000000

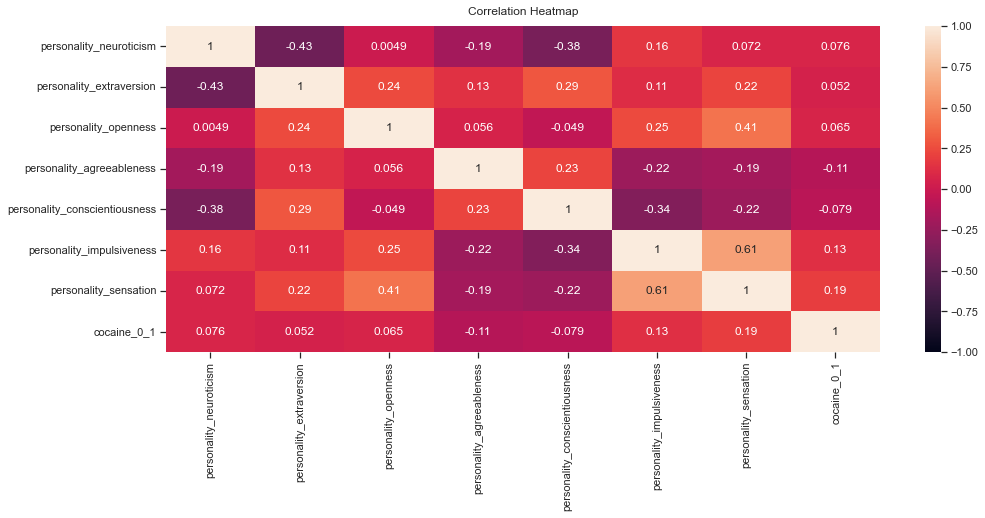
As you can see the dataset is complete with no missing data. What is more, there are no outliers at first glance. Let's visualize the data to take a closer look at the relationships and potential data quality issues.

### Exploanatory Data Analysis

Let's start by creating a correlation matrix. None of the variables are highly correlated with each other (assumed threshold >|0.7|). In particular, it is good information that the predicted variable has only very weak correlation coefficients.

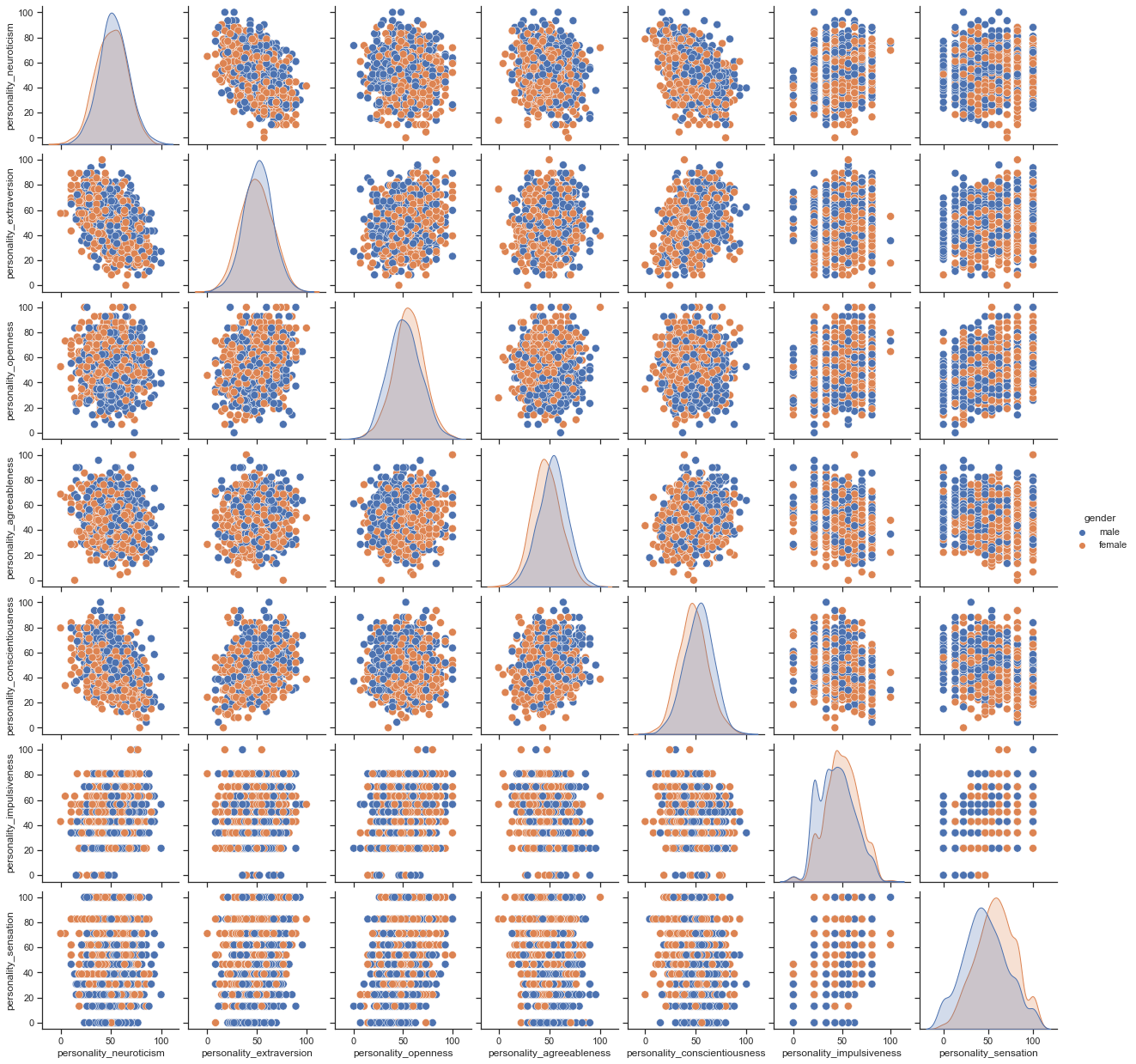
plt.figure(figsize=(16, 6))  
drugs['cocaine\_0\_1'] = np.where(drugs['consumption\_cocaine\_last\_month'] == 'Yes', 1,0)  
heatmap = sns.heatmap(drugs.corr(), vmin=-1, vmax=1, annot=True)  
heatmap.set\_title('Correlation Heatmap', fontdict={'fontsize':12}, pad=11)

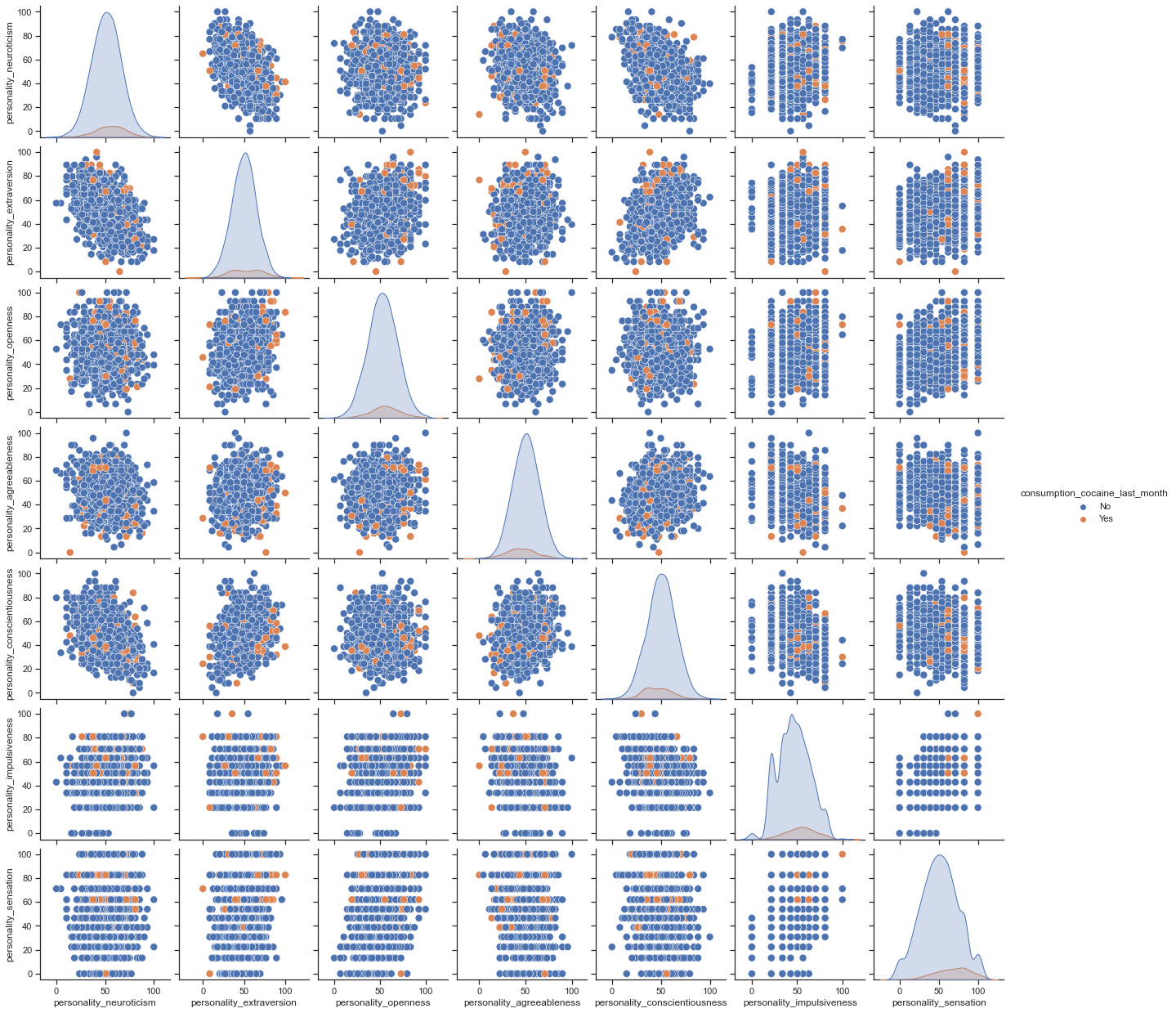
Text(0.5, 1.0, 'Correlation Heatmap')



Let's look at the distributions of the variables by gender. Based on graphical analysis only - there are no significant differences in personality traits between men and women. The distributions of personality\_impulsiveness and personality\_sensation are shifted to the right for females. Analogous conclusions can be drawn from the distributions by cocaine users in the last month.

sns.pairplot(drugs,kind="scatter", hue="gender", plot\_kws=dict(s=80, edgecolor="white", linewidth=.5))  
plt.show()  
sns.pairplot(drugs,kind="scatter", hue="consumption\_cocaine\_last\_month", plot\_kws=dict(s=80, edgecolor="white", linewidth=.5))  
plt.show()



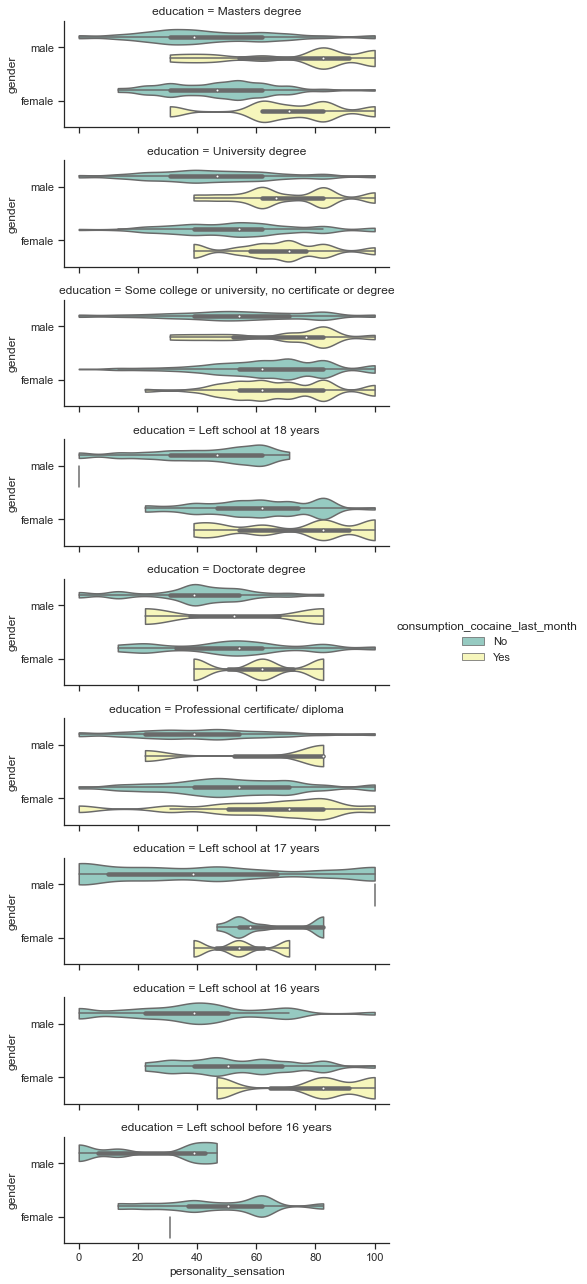


The following are the spin charts of the personality\_sensation variable by gender, education, and cocaine users. Conclusions that can be drawn from the charts:

* there is not a single person who left education under the age of 16 who has consumed cocaine in the last month
* there is not a single person who has completed education under the age of 17, is male and has consumed cocaine in the last month
* median personality\_sensation is higher for those who have consumed cocaine in the last month for almost every level of education

sns.catplot(x="personality\_sensation", y="gender",  
 hue="consumption\_cocaine\_last\_month", row="education",  
 data=drugs,  
 orient="h", height=2, aspect=3, palette="Set3",  
 kind="violin", dodge=True, cut=0, bw=.2)

<seaborn.axisgrid.FacetGrid at 0x23b10a1afa0>



Let's take a closer look at char columns distributions. Since there are few people who left education before the age of 18 (i.e., under 16,17,18) we should think about categorizing them into one group. In the case of the variable ethinicity, because of the small number of white people we will classify them into the group other. Similarly in the case of Canadians. We will replace the remaining variables with ordinal or categorical variables.

char\_cols = drugs.select\_dtypes(include=['object'])  
char\_cols = char\_cols.drop('id', axis = 1)  
for col in char\_cols:  
 print(col, ':', char\_cols[col].value\_counts())

age : 18-24 528  
25-34 375  
35-44 278  
45-54 233  
55-64 72  
65+ 14  
Name: age, dtype: int64  
gender : female 751  
male 749  
Name: gender, dtype: int64  
education : Some college or university, no certificate or degree 405  
University degree 376  
Masters degree 229  
Professional certificate/ diploma 221  
Left school at 18 years 85  
Left school at 16 years 72  
Doctorate degree 66  
Left school at 17 years 26  
Left school before 16 years 20  
Name: education, dtype: int64  
country : USA 811  
Australia 460  
New Zealand 94  
UK 73  
Other 44  
Ireland 13  
Canada 5  
Name: country, dtype: int64  
ethnicity : Mixed-Black/Asian 1372  
Mixed-White/Black 47  
Asian 25  
Black 22  
Other 16  
Mixed-White/Asian 15  
White 3  
Name: ethnicity, dtype: int64  
consumption\_alcohol : used in last week 601  
used in last day 399  
used in last month 235  
used in last year 158  
used in last decade 54  
used over a decade ago 27  
never used 26  
Name: consumption\_alcohol, dtype: int64  
consumption\_amphetamines : never used 784  
used in last decade 187  
used over a decade ago 182  
used in last year 156  
used in last day 80  
used in last month 57  
used in last week 54  
Name: consumption\_amphetamines, dtype: int64  
consumption\_caffeine : used in last day 1092  
used in last week 224  
used in last month 87  
used in last year 49  
never used 20  
used in last decade 18  
used over a decade ago 10  
Name: consumption\_caffeine, dtype: int64  
consumption\_cannabis : used in last day 373  
never used 324  
used in last decade 207  
used in last year 171  
used over a decade ago 168  
used in last week 151  
used in last month 106  
Name: consumption\_cannabis, dtype: int64  
consumption\_chocolate : used in last day 652  
used in last week 528  
used in last month 234  
used in last year 49  
never used 26  
used in last decade 8  
used over a decade ago 3  
Name: consumption\_chocolate, dtype: int64  
consumption\_mushrooms : never used 772  
used in last year 218  
used in last decade 208  
used over a decade ago 161  
used in last month 101  
used in last week 36  
used in last day 4  
Name: consumption\_mushrooms, dtype: int64  
consumption\_nicotine : used in last day 497  
never used 337  
used in last decade 165  
used over a decade ago 149  
used in last year 149  
used in last week 119  
used in last month 84  
Name: consumption\_nicotine, dtype: int64  
consumption\_cocaine\_last\_month : No 1373  
Yes 127  
Name: consumption\_cocaine\_last\_month, dtype: int64

### Data Preprocessing

Below we have defined functions to preprocess the data, based on the conclusions we have drawn from EDA. These will be as follows:

* creating dummie variables based on the columns: age, ethnicity, country, gender, consumption\_cocaine\_last\_month
* create ordinal variables based on the columns describing consumption of drugs (of course without the predicted variable)
* scaling continuous variables using RobustScaler()
* creating a function that allows easy validation of the created models

def getting\_dummies(drugs):  
 drugs['tmp'] = range(0,drugs.shape[0])  
 t1 = pd.get\_dummies(drugs['age'])  
 t1['tmp'] = range(0,drugs.shape[0])  
 drugs = pd.merge(drugs, t1, on = ['tmp'])  
  
 t1 = pd.get\_dummies(drugs['country'])  
 t1['tmp'] = range(0,drugs.shape[0])  
 drugs = pd.merge(drugs, t1, on = ['tmp'])  
 try:  
 drugs['Other\_country'] = drugs['Other'] + drugs['Ireland'] + drugs['Canada']  
  
 t1 = pd.get\_dummies(drugs['ethnicity'])  
 t1['tmp'] = range(0,drugs.shape[0])  
 drugs = pd.merge(drugs, t1, on = ['tmp'])  
 drugs['Other\_y'] = drugs['Other\_y'] + drugs['White']  
  
 drugs = drugs.drop(columns = ['tmp', 'age', 'country', 'ethnicity','Ireland', 'Canada', 'Other\_x' , 'White'])  
 except:  
 drugs['Other\_country'] = drugs['Other'] + drugs['Ireland']   
  
 t1 = pd.get\_dummies(drugs['ethnicity'])  
 t1['tmp'] = range(0,drugs.shape[0])  
 drugs = pd.merge(drugs, t1, on = ['tmp'])  
  
 drugs = drugs.drop(columns = ['tmp', 'age', 'country', 'ethnicity','Ireland', 'Other\_x' ])  
  
 return (drugs)  
  
def getting\_ordinals(drugs):  
   
 lst = ['consumption\_alcohol', 'consumption\_amphetamines', 'consumption\_caffeine',  
 'consumption\_cannabis', 'consumption\_chocolate',  
 'consumption\_mushrooms', 'consumption\_nicotine']  
 consDict = {'never used':0, 'used over a decade ago':1,  
 'used in last decade':2,'used in last year':3, 'used in last month':4, 'used in last week' : 5, 'used in last day' : 6}  
  
 eduDict = {'Left school before 16 years' : 0, 'Left school at 16 years' : 0, 'Left school at 17 years':0, 'Left school at 18 years' : 0,   
 'Some college or university, no certificate or degree':1, 'Professional certificate/ diploma':2 ,'University degree':3, 'Masters degree':4,   
 'Doctorate degree' : 5}  
  
  
 for cols in lst:  
 drugs[cols]=drugs[cols].map(consDict)  
   
 drugs['education'] = drugs['education'].map(eduDict)  
 return(drugs)  
  
def gender\_dummy(drugs):  
 drugs['female'] = np.where(drugs['gender'] == 'female', 1, 0)   
 drugs = drugs.drop(columns=['gender', 'id'])  
 return (drugs)  
  
def expl\_var\_dummy(drugs):  
 drugs['consumption\_cocaine\_last\_month'] = np.where(drugs['consumption\_cocaine\_last\_month'] == 'Yes', 1, 0)   
 return (drugs)  
  
def scalling(drugs):  
 scaled\_features = drugs.copy()  
  
 col\_names = ['personality\_neuroticism', 'personality\_extraversion',  
 'personality\_openness', 'personality\_agreeableness',  
 'personality\_conscientiousness', 'personality\_impulsiveness',  
 'personality\_sensation']  
 features = scaled\_features[col\_names]  
  
 scaler = RobustScaler().fit(features.values)  
 features = scaler.transform(features.values)  
  
 scaled\_features[col\_names] = features  
  
 return(scaled\_features)  
   
# def our\_metrics(y\_test, preds):  
# print(f'Balanced Accuracy:', balanced\_accuracy\_score(y\_test, preds),   
# '\nconfusion:\n', confusion\_matrix(y\_test, preds),  
# '\nprecision:', precision\_score(y\_test, preds) ,  
# '\naccuracy:', accuracy\_score(y\_test, preds),   
# '\nrecall:', recall\_score(y\_test, preds),  
# '\nauroc:', roc\_auc\_score(y\_test, preds) )  
def our\_metrics(y\_test, preds):  
 print(f'Balanced Accuracy:', balanced\_accuracy\_score(y\_test, preds),   
 '\nconfusion:\n', confusion\_matrix(y\_test, preds),  
 )

##### Function call and train\_test splitting

drugs = pd.read\_csv('drugs\_train.csv')  
  
drugs = getting\_dummies(drugs)  
drugs = getting\_ordinals(drugs)  
drugs = gender\_dummy(drugs)  
drugs = expl\_var\_dummy(drugs)  
drugs = scalling(drugs)  
x\_train, x\_test, y\_train, y\_test = train\_test\_split(drugs.loc[:,drugs.columns!='consumption\_cocaine\_last\_month'], drugs.consumption\_cocaine\_last\_month, test\_size=0.25)  
  
print(drugs.columns )  
drugs.head()

Index(['education', 'personality\_neuroticism', 'personality\_extraversion',  
 'personality\_openness', 'personality\_agreeableness',  
 'personality\_conscientiousness', 'personality\_impulsiveness',  
 'personality\_sensation', 'consumption\_alcohol',  
 'consumption\_amphetamines', 'consumption\_caffeine',  
 'consumption\_cannabis', 'consumption\_chocolate',  
 'consumption\_mushrooms', 'consumption\_nicotine',  
 'consumption\_cocaine\_last\_month', '18-24', '25-34', '35-44', '45-54',  
 '55-64', '65+', 'Australia', 'New Zealand', 'UK', 'USA',  
 'Other\_country', 'Asian', 'Black', 'Mixed-Black/Asian',  
 'Mixed-White/Asian', 'Mixed-White/Black', 'Other\_y', 'female'],  
 dtype='object')

education personality\_neuroticism personality\_extraversion \  
0 4 0.287179 0.354680   
1 3 -0.215385 0.832512   
2 3 0.287179 -0.334975   
3 4 1.015385 -0.931034   
4 1 0.210256 0.600985   
  
 personality\_openness personality\_agreeableness \  
0 -0.111588 -0.115607   
1 -0.300429 -0.115607   
2 0.111588 -0.242775   
3 -0.390558 0.375723   
4 0.751073 0.942197   
  
 personality\_conscientiousness personality\_impulsiveness \  
0 0.213483 0.000000   
1 0.342697 -0.396476   
2 0.000000 0.889868   
3 -1.016854 0.889868   
4 -0.421348 0.334802   
  
 personality\_sensation consumption\_alcohol consumption\_amphetamines \  
0 -0.978328 5 1   
1 -0.718266 5 0   
2 0.247678 4 0   
3 0.529412 6 0   
4 0.247678 5 0   
  
 consumption\_caffeine consumption\_cannabis consumption\_chocolate \  
0 6 5 6   
1 5 0 6   
2 6 5 5   
3 6 2 6   
4 4 4 6   
  
 consumption\_mushrooms consumption\_nicotine \  
0 0 5   
1 0 0   
2 3 4   
3 0 2   
4 3 4   
  
 consumption\_cocaine\_last\_month 18-24 25-34 35-44 45-54 55-64 65+ \  
0 0 0 0 0 1 0 0   
1 0 0 1 0 0 0 0   
2 0 1 0 0 0 0 0   
3 0 0 1 0 0 0 0   
4 0 1 0 0 0 0 0   
  
 Australia New Zealand UK USA Other\_country Asian Black \  
0 0 0 0 1 0 0 0   
1 0 0 0 1 0 0 0   
2 0 0 0 1 0 0 0   
3 0 0 0 1 0 0 0   
4 1 0 0 0 0 0 0   
  
 Mixed-Black/Asian Mixed-White/Asian Mixed-White/Black Other\_y female   
0 1 0 0 0 0   
1 1 0 0 0 0   
2 1 0 0 0 1   
3 1 0 0 0 1   
4 1 0 0 0 0

counter = Counter(y\_test)  
counter

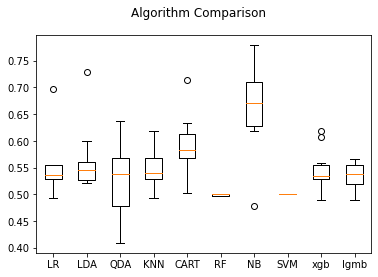
Counter({0: 340, 1: 35})

### Modeling

As we can observe, our dataset is highly inbalanced. At this point we supposed that using a over-sampling methodes will be crucial. To have a benchmark, let's see how machine learning algorithms perform on the cleaned dataset before further steps. We will use a modified script (source: [*https://machinelearningmastery.com/compare-machine-learning-algorithms-python-scikit-learn/*](https://machinelearningmastery.com/compare-machine-learning-algorithms-python-scikit-learn/)) comparing the performance of the algorithms. Our goal is to achieve the **highest** possible value of **balanced accuracy score** metric.

X, Y =drugs.loc[:,drugs.columns!='consumption\_cocaine\_last\_month'], drugs.consumption\_cocaine\_last\_month  
def comparison(X,Y):  
 models = []  
 models.append(('LR', LogisticRegression()))  
 models.append(('LDA', LinearDiscriminantAnalysis()))  
 models.append(('QDA', QuadraticDiscriminantAnalysis()))  
 models.append(('KNN', KNeighborsClassifier()))  
 models.append(('CART', DecisionTreeClassifier()))  
 models.append(('RF', RandomForestClassifier()))  
 models.append(('NB', GaussianNB()))  
 models.append(('SVM', SVC()))  
 models.append(('xgb', xgb.XGBClassifier()))  
 models.append(('lgmb', lgb.LGBMClassifier()))  
 # evaluate each model in turn  
 results = []  
 names = []  
 scoring = 'balanced\_accuracy'  
 for name, model in models:  
 kfold = model\_selection.KFold(n\_splits=10, shuffle = True, random\_state = 997)  
 cv\_results = model\_selection.cross\_val\_score(model, X, Y, cv=kfold, scoring=scoring)  
 results.append(cv\_results)  
 names.append(name)  
 msg = "%s: %f (%f)" % (name, cv\_results.mean(), cv\_results.std())  
 print(msg)  
 # boxplot algorithm comparison  
 fig = plt.figure()  
 fig.suptitle('Algorithm Comparison')  
 ax = fig.add\_subplot(111)  
 plt.boxplot(results)  
 ax.set\_xticklabels(names)  
 plt.show()  
  
comparison(X,Y)

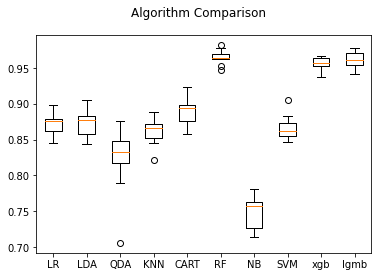
LR: 0.550559 (0.052165)  
LDA: 0.563899 (0.059334)  
QDA: 0.528075 (0.068290)  
KNN: 0.552088 (0.037535)  
CART: 0.590109 (0.055328)  
RF: 0.498545 (0.001783)  
NB: 0.664404 (0.079889)  
SVM: 0.500000 (0.000000)  
xgb: 0.547188 (0.036827)  
lgmb: 0.533665 (0.027027)



As you can see, the algorithms performed very average. Achieving average results of about 0.55 balanced accuracy score (hereinafter BAC). It is certainly not a satisfactory result, because due to the methodology of calculating the BAC, predicting only 0 or 1 will achieve a value of 0.5. We expect that the problem is probably an unbalanced set. For this purpose, we will apply the SMOTE over-sampling technique.

oversample = SMOTE()  
X\_ov, Y\_ov = oversample.fit\_resample(X, Y)  
counter\_a = Counter(Y\_ov)  
print(counter\_a)  
comparison(X\_ov,Y\_ov)

Counter({0: 1373, 1: 1373})  
LR: 0.870938 (0.015210)  
LDA: 0.873203 (0.019211)  
QDA: 0.822261 (0.044712)  
KNN: 0.861576 (0.017919)  
CART: 0.889231 (0.019372)  
RF: 0.964935 (0.009762)  
NB: 0.748429 (0.021895)  
SVM: 0.866949 (0.016841)  
xgb: 0.956238 (0.009473)  
lgmb: 0.962295 (0.010760)



This time the algorithms did very well. The LightGMB algorithm performed best on the training set with additional synthetic observations. However, we should not get overly euphoric, because we have to keep in mind that our original dataset is just unbalanced and it does not make much sense to calculate BAC on the balanced set. In the next steps, before applying SMOTE, we will do a data split extracting 15% of the observations that will serve as an unbalanced test set.

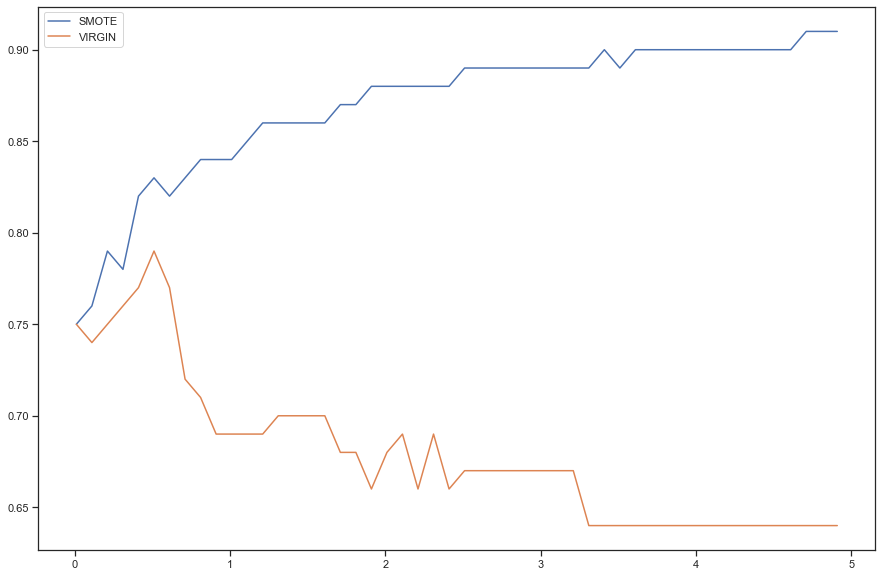
x\_train, x\_test, y\_train, y\_test = train\_test\_split(drugs.loc[:,drugs.columns!='consumption\_cocaine\_last\_month'], drugs.consumption\_cocaine\_last\_month, test\_size=0.15, stratify= drugs.consumption\_cocaine\_last\_month)  
  
oversample = SMOTE()  
X, y = oversample.fit\_resample(x\_train, y\_train)  
counter\_a = Counter(y)  
print('SMOTE y distribution:', counter\_a)  
  
x\_trainSM, x\_testSM, y\_trainSM, y\_testSM = train\_test\_split(X,y, test\_size=0.05)  
counter = Counter(y\_test)  
print('Virgin y distribution:' , counter)

SMOTE y distribution: Counter({0: 1167, 1: 1167})  
Virgin y distribution: Counter({0: 206, 1: 19})

#### SVM

#SVM   
  
c\_param = np.arange(0.01, 5, 0.1) # [0.01, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.5, 0.6, 0.8, 1, 1.5, 2, 2.5]  
sm = []  
vg = []  
for c in c\_param:  
   
 print()  
 kf = KFold(n\_splits=5)  
 ba\_smote = []  
 ba\_virg = []  
   
 clf = SVC(C=c, kernel='rbf', max\_iter=-1, probability=True, tol=0.1,random\_state= 13, verbose=False)  
  
 for train, test in kf.split(drugs.index.values):  
   
 clf.fit(x\_trainSM, y\_trainSM)  
   
 #Over-sampled DS   
 prob = clf.predict(x\_testSM)  
 ba\_smote.append(balanced\_accuracy\_score(y\_testSM, prob))  
  
 #Virgin DS  
 prob = clf.predict(x\_test)  
 ba\_virg.append(balanced\_accuracy\_score(y\_test, prob))  
 sm.append(np.round(np.mean(ba\_smote), 2))  
 vg.append(np.round(np.mean(ba\_virg), 2))  
   
 print('C value is: ', c,'Average bas on SMOTE is: ', np.round(np.mean(ba\_smote), 4), 'Average bas on VIRGIN is:', np.round(np.mean(ba\_virg), 4))  
   
  
best = dict(zip(vg, c\_param))  
vg\_m = max(list(best.keys()))  
  
clf = SVC(C=best[vg\_m], kernel='rbf', max\_iter=-1, probability=True, tol=0.1, random\_state= 13, verbose=False)  
clf.fit(x\_trainSM, y\_trainSM)  
prob = clf.predict(x\_test)  
our\_metrics(y\_test, prob)  
  
  
my\_dict = dict(x=c\_param,y=sm,z=vg)  
data = pd.DataFrame(my\_dict)  
  
plt.figure(figsize=(15,10))  
sns.lineplot(c\_param, sm, label = 'SMOTE', data = data,   
 markers=True, dashes=True)  
sns.lineplot(c\_param, vg, label = 'VIRGIN', data = data,   
 markers=True, dashes=True)  
plt.show()

C value is: 0.01 Average bas on SMOTE is: 0.7521 Average bas on VIRGIN is: 0.7508  
  
C value is: 0.11 Average bas on SMOTE is: 0.7647 Average bas on VIRGIN is: 0.7418  
  
C value is: 0.21000000000000002 Average bas on SMOTE is: 0.7914 Average bas on VIRGIN is: 0.754  
  
C value is: 0.31000000000000005 Average bas on SMOTE is: 0.7836 Average bas on VIRGIN is: 0.7564  
  
C value is: 0.41000000000000003 Average bas on SMOTE is: 0.8181 Average bas on VIRGIN is: 0.7685  
  
C value is: 0.51 Average bas on SMOTE is: 0.8275 Average bas on VIRGIN is: 0.7855  
  
C value is: 0.6100000000000001 Average bas on SMOTE is: 0.8181 Average bas on VIRGIN is: 0.7689  
  
C value is: 0.7100000000000001 Average bas on SMOTE is: 0.8259 Average bas on VIRGIN is: 0.7163  
  
C value is: 0.81 Average bas on SMOTE is: 0.8448 Average bas on VIRGIN is: 0.7138  
  
C value is: 0.91 Average bas on SMOTE is: 0.8448 Average bas on VIRGIN is: 0.69  
  
C value is: 1.01 Average bas on SMOTE is: 0.8448 Average bas on VIRGIN is: 0.6948  
  
C value is: 1.11 Average bas on SMOTE is: 0.8526 Average bas on VIRGIN is: 0.6948  
  
C value is: 1.2100000000000002 Average bas on SMOTE is: 0.862 Average bas on VIRGIN is: 0.6948  
  
C value is: 1.31 Average bas on SMOTE is: 0.862 Average bas on VIRGIN is: 0.6972  
  
C value is: 1.4100000000000001 Average bas on SMOTE is: 0.862 Average bas on VIRGIN is: 0.6997  
  
C value is: 1.51 Average bas on SMOTE is: 0.862 Average bas on VIRGIN is: 0.6972  
  
C value is: 1.61 Average bas on SMOTE is: 0.862 Average bas on VIRGIN is: 0.6997  
  
C value is: 1.7100000000000002 Average bas on SMOTE is: 0.8698 Average bas on VIRGIN is: 0.6806  
  
C value is: 1.81 Average bas on SMOTE is: 0.8698 Average bas on VIRGIN is: 0.6806  
  
C value is: 1.9100000000000001 Average bas on SMOTE is: 0.8777 Average bas on VIRGIN is: 0.6592  
  
C value is: 2.01 Average bas on SMOTE is: 0.8777 Average bas on VIRGIN is: 0.6831  
  
C value is: 2.11 Average bas on SMOTE is: 0.8777 Average bas on VIRGIN is: 0.6855  
  
C value is: 2.21 Average bas on SMOTE is: 0.8777 Average bas on VIRGIN is: 0.6616  
  
C value is: 2.31 Average bas on SMOTE is: 0.8777 Average bas on VIRGIN is: 0.6903  
  
C value is: 2.41 Average bas on SMOTE is: 0.8777 Average bas on VIRGIN is: 0.6616  
  
C value is: 2.51 Average bas on SMOTE is: 0.8871 Average bas on VIRGIN is: 0.6665  
  
C value is: 2.61 Average bas on SMOTE is: 0.8871 Average bas on VIRGIN is: 0.6689  
  
C value is: 2.71 Average bas on SMOTE is: 0.8871 Average bas on VIRGIN is: 0.6713  
  
C value is: 2.81 Average bas on SMOTE is: 0.8871 Average bas on VIRGIN is: 0.6713  
  
C value is: 2.91 Average bas on SMOTE is: 0.8871 Average bas on VIRGIN is: 0.6689  
  
C value is: 3.01 Average bas on SMOTE is: 0.8949 Average bas on VIRGIN is: 0.6689  
  
C value is: 3.11 Average bas on SMOTE is: 0.8871 Average bas on VIRGIN is: 0.6713  
  
C value is: 3.21 Average bas on SMOTE is: 0.8949 Average bas on VIRGIN is: 0.6713  
  
C value is: 3.31 Average bas on SMOTE is: 0.8949 Average bas on VIRGIN is: 0.6426  
  
C value is: 3.41 Average bas on SMOTE is: 0.9043 Average bas on VIRGIN is: 0.645  
  
C value is: 3.51 Average bas on SMOTE is: 0.8949 Average bas on VIRGIN is: 0.645  
  
C value is: 3.61 Average bas on SMOTE is: 0.9043 Average bas on VIRGIN is: 0.645  
  
C value is: 3.71 Average bas on SMOTE is: 0.9043 Average bas on VIRGIN is: 0.645  
  
C value is: 3.81 Average bas on SMOTE is: 0.9043 Average bas on VIRGIN is: 0.645  
  
C value is: 3.91 Average bas on SMOTE is: 0.9043 Average bas on VIRGIN is: 0.6426  
  
C value is: 4.01 Average bas on SMOTE is: 0.9043 Average bas on VIRGIN is: 0.645  
  
C value is: 4.11 Average bas on SMOTE is: 0.9043 Average bas on VIRGIN is: 0.645  
  
C value is: 4.21 Average bas on SMOTE is: 0.9043 Average bas on VIRGIN is: 0.645  
  
C value is: 4.31 Average bas on SMOTE is: 0.9043 Average bas on VIRGIN is: 0.645  
  
C value is: 4.41 Average bas on SMOTE is: 0.9043 Average bas on VIRGIN is: 0.645  
  
C value is: 4.51 Average bas on SMOTE is: 0.9043 Average bas on VIRGIN is: 0.645  
  
C value is: 4.61 Average bas on SMOTE is: 0.9043 Average bas on VIRGIN is: 0.645  
  
C value is: 4.71 Average bas on SMOTE is: 0.9138 Average bas on VIRGIN is: 0.645  
  
C value is: 4.8100000000000005 Average bas on SMOTE is: 0.9138 Average bas on VIRGIN is: 0.645  
  
C value is: 4.91 Average bas on SMOTE is: 0.9138 Average bas on VIRGIN is: 0.645  
Balanced Accuracy: 0.7855135411343894   
confusion:  
 [[161 45]  
 [ 4 15]]



### KNN

from sklearn.neighbors import KNeighborsClassifier  
sm = []  
vg = []  
ran = range(1,40,1)  
for n in ran:  
 knn = KNeighborsClassifier(n\_neighbors=n)  
 knn.fit(x\_trainSM, y\_trainSM)  
 knn\_predSM = knn.predict(x\_testSM)  
 knn\_predVG = knn.predict(x\_test)  
 vg.append(np.round(balanced\_accuracy\_score(y\_test, knn\_predVG), 2))  
 print('N\_neigbours is: ', n, 'Average BAS on SMOTE is: ', balanced\_accuracy\_score(y\_testSM, knn\_predSM), 'Average BAS on VIRGIN is:', balanced\_accuracy\_score(y\_test, knn\_predVG))  
  
best = dict(zip(vg, ran))  
vg\_m = max(list(best.keys()))  
  
knn = KNeighborsClassifier(n\_neighbors=best[vg\_m])  
knn.fit(x\_trainSM, y\_trainSM)  
knn\_predVG = knn.predict(x\_test)  
  
our\_metrics(y\_test, knn\_predVG)  
print('best n\_neigh:', best[vg\_m], 'max bas:', vg\_m)

N\_neigbours is: 1 Average BAS on SMOTE is: 0.9099999999999999 Average BAS on VIRGIN is: 0.6138221768012263  
N\_neigbours is: 2 Average BAS on SMOTE is: 0.9099999999999999 Average BAS on VIRGIN is: 0.6142054164537558  
N\_neigbours is: 3 Average BAS on SMOTE is: 0.89 Average BAS on VIRGIN is: 0.6543178334184977  
N\_neigbours is: 4 Average BAS on SMOTE is: 0.9 Average BAS on VIRGIN is: 0.6498467041389883  
N\_neigbours is: 5 Average BAS on SMOTE is: 0.85 Average BAS on VIRGIN is: 0.64665304036791  
N\_neigbours is: 6 Average BAS on SMOTE is: 0.87 Average BAS on VIRGIN is: 0.6397547266223812  
N\_neigbours is: 7 Average BAS on SMOTE is: 0.8300000000000001 Average BAS on VIRGIN is: 0.6632600919775167  
N\_neigbours is: 8 Average BAS on SMOTE is: 0.87 Average BAS on VIRGIN is: 0.682677567705672  
N\_neigbours is: 9 Average BAS on SMOTE is: 0.85 Average BAS on VIRGIN is: 0.7061829330608074  
N\_neigbours is: 10 Average BAS on SMOTE is: 0.88 Average BAS on VIRGIN is: 0.6920030659172203  
N\_neigbours is: 11 Average BAS on SMOTE is: 0.85 Average BAS on VIRGIN is: 0.6725855901890649  
N\_neigbours is: 12 Average BAS on SMOTE is: 0.85 Average BAS on VIRGIN is: 0.6847215125191619  
N\_neigbours is: 13 Average BAS on SMOTE is: 0.8300000000000001 Average BAS on VIRGIN is: 0.6964741951967297  
N\_neigbours is: 14 Average BAS on SMOTE is: 0.8525373134328358 Average BAS on VIRGIN is: 0.6847215125191619  
N\_neigbours is: 15 Average BAS on SMOTE is: 0.8525373134328358 Average BAS on VIRGIN is: 0.7252171691364333  
N\_neigbours is: 16 Average BAS on SMOTE is: 0.8525373134328358 Average BAS on VIRGIN is: 0.7134644864588656  
N\_neigbours is: 17 Average BAS on SMOTE is: 0.8225373134328358 Average BAS on VIRGIN is: 0.7203628002043945  
N\_neigbours is: 18 Average BAS on SMOTE is: 0.8425373134328358 Average BAS on VIRGIN is: 0.7300715380684721  
N\_neigbours is: 19 Average BAS on SMOTE is: 0.8325373134328358 Average BAS on VIRGIN is: 0.717935615738375  
N\_neigbours is: 20 Average BAS on SMOTE is: 0.8425373134328358 Average BAS on VIRGIN is: 0.7300715380684721  
N\_neigbours is: 21 Average BAS on SMOTE is: 0.8425373134328358 Average BAS on VIRGIN is: 0.7203628002043945  
N\_neigbours is: 22 Average BAS on SMOTE is: 0.8525373134328358 Average BAS on VIRGIN is: 0.6940470107307103  
N\_neigbours is: 23 Average BAS on SMOTE is: 0.8325373134328358 Average BAS on VIRGIN is: 0.7130812468063362  
N\_neigbours is: 24 Average BAS on SMOTE is: 0.8325373134328358 Average BAS on VIRGIN is: 0.6964741951967297  
N\_neigbours is: 25 Average BAS on SMOTE is: 0.8225373134328358 Average BAS on VIRGIN is: 0.7321154828819623  
N\_neigbours is: 26 Average BAS on SMOTE is: 0.8225373134328358 Average BAS on VIRGIN is: 0.7393970362800204  
N\_neigbours is: 27 Average BAS on SMOTE is: 0.8225373134328358 Average BAS on VIRGIN is: 0.7272611139499234  
N\_neigbours is: 28 Average BAS on SMOTE is: 0.8225373134328358 Average BAS on VIRGIN is: 0.7369698518140011  
N\_neigbours is: 29 Average BAS on SMOTE is: 0.8125373134328358 Average BAS on VIRGIN is: 0.7248339294839039  
N\_neigbours is: 30 Average BAS on SMOTE is: 0.8225373134328358 Average BAS on VIRGIN is: 0.7009453244762391  
N\_neigbours is: 31 Average BAS on SMOTE is: 0.8125373134328358 Average BAS on VIRGIN is: 0.7248339294839039  
N\_neigbours is: 32 Average BAS on SMOTE is: 0.8225373134328358 Average BAS on VIRGIN is: 0.7296882984159427  
N\_neigbours is: 33 Average BAS on SMOTE is: 0.8 Average BAS on VIRGIN is: 0.7462953500255494  
N\_neigbours is: 34 Average BAS on SMOTE is: 0.8300000000000001 Average BAS on VIRGIN is: 0.7248339294839039  
N\_neigbours is: 35 Average BAS on SMOTE is: 0.81 Average BAS on VIRGIN is: 0.7151251916198262  
N\_neigbours is: 36 Average BAS on SMOTE is: 0.8200000000000001 Average BAS on VIRGIN is: 0.7175523760858458  
N\_neigbours is: 37 Average BAS on SMOTE is: 0.8 Average BAS on VIRGIN is: 0.7175523760858458  
N\_neigbours is: 38 Average BAS on SMOTE is: 0.81 Average BAS on VIRGIN is: 0.7199795605518651  
N\_neigbours is: 39 Average BAS on SMOTE is: 0.8 Average BAS on VIRGIN is: 0.7414409810935105  
Balanced Accuracy: 0.7462953500255494   
confusion:  
 [[134 72]  
 [ 3 16]]  
best n\_neigh: 33 max bas: 0.75

### LDA / QDA

ldasvd = LinearDiscriminantAnalysis(solver="svd", store\_covariance=True)  
ldalsqr = LinearDiscriminantAnalysis(solver="lsqr", store\_covariance=True)  
ldaeigen = LinearDiscriminantAnalysis(solver="eigen", store\_covariance=True)  
qda = QuadraticDiscriminantAnalysis(store\_covariance=True)  
  
estsvd = ldasvd.fit(x\_trainSM, y\_trainSM)  
estlsqr = ldalsqr.fit(x\_trainSM, y\_trainSM)  
esteigen = ldaeigen.fit(x\_trainSM, y\_trainSM)  
estqda = qda.fit(x\_trainSM, y\_trainSM)   
  
predslvd = estsvd.predict(x\_test)  
predslsqr = estlsqr.predict(x\_test)  
predseigen = esteigen.predict(x\_test)  
predsqda = estqda.predict(x\_test)  
  
print('LDA solver = svd:')  
our\_metrics(y\_test,predslvd)  
print('LDA solver = lsqr:')  
our\_metrics(y\_test,predslsqr)  
print('LDA solver = eigen:')  
our\_metrics(y\_test,predseigen)  
print('QDA:')  
our\_metrics(y\_test,predsqda)

LDA solver = svd:  
Balanced Accuracy: 0.7385028104241186   
confusion:  
 [[185 21]  
 [ 8 11]]  
LDA solver = lsqr:  
Balanced Accuracy: 0.7385028104241186   
confusion:  
 [[185 21]  
 [ 8 11]]  
LDA solver = eigen:  
Balanced Accuracy: 0.7385028104241186   
confusion:  
 [[185 21]  
 [ 8 11]]  
QDA:  
Balanced Accuracy: 0.5   
confusion:  
 [[206 0]  
 [ 19 0]]

The SVM and KNN algorithms performed quite well. On the test set (keeping the original distributions of the predicted variable), SVM achieved a balanced accuracy score of **0.786**. Noting the confusion matrix, we see that the model predicts a lot of false ones. Since our goal is **BAC maximization** and we know the disparity of the distribution of the predictor variable, we can accept this - it is **much better** for us to have **more false truths than false negatives**. The key aspect was to train the model on an appropriate kernel and an appropriate value of the parameter c, which is responsible for penalizing the algorithm for errors. In the graph, we see that initially the BAC increases as the parameter c increases, reaching a maximum at c = 0.51, then gradually decreases. This happens only on unbalanced data. On data passed through SMOTE(), BAC increases logarithmically as the parameter increases approaching 1. The KNN algorithm did just as well, achieving a balanced\_accuracy score of **0.7462**. It predicted even more false ones than SVM and this was more than the value that increased the BAS measure. LDA algorithms did not do badly, regardless of the implemented parameter solver, however, achieved less satisfactory results than SVM or KNN. The QDA algorithm completely failed the task. It did not predict any 1, which as mentioned above in the script, leads to a value of BAC measure = 0.5.

### LightGBM

Since the LightGBM algorithm performed the best of the classifiers on the data after Over-Sampling, we attempted to boost the hyperparameters to get even better results than the SVM algorithm. We used the optuna package for this purpose. Below is a printout for n\_trials = 15. We also tested on n\_trails = 1000 and unfortunately the maximum BAC value we achieved on the unbalanced test set was around 0.67. The conclusion from this exercise is that LightGBM performs very well on the balanced set and unfortunately very average on the unbalanced set. The algorithm predicted far too few ones even despite the interference with the threshold level.

def objective(trial):  
   
 param = {  
 "objective": "binary",  
 "metric": "binary\_logloss",  
 "n\_estimators": trial.suggest\_int("n\_estimators", 1000, 8000),  
 "learning\_rate": trial.suggest\_float("learning\_rate", 0.01, 0.3),  
 "verbosity": -1,  
 "boosting\_type": "gbdt",  
 "lambda\_l1": trial.suggest\_float("lambda\_l1", 1e-8, 1.5, log=True),  
 "lambda\_l2": trial.suggest\_float("lambda\_l2", 1e-8, 1.5, log=True),  
 "num\_leaves": trial.suggest\_int("num\_leaves", 4000, 12000),  
 "feature\_fraction": trial.suggest\_float("feature\_fraction", 0.4, 1.0),  
 "bagging\_fraction": trial.suggest\_float("bagging\_fraction", 0.4, 1.0),  
 "bagging\_freq": trial.suggest\_int("bagging\_freq", 1, 7),  
 "min\_child\_samples": trial.suggest\_int("min\_child\_samples", 5, 140),  
 }  
  
 lst = []  
 for i in range(3):  
 dtrain = lgb.Dataset(x\_trainSM, label=y\_trainSM)  
 gbm = lgb.train(param, dtrain)  
 preds = gbm.predict(x\_test)  
 pred\_labels1 = np.where(preds>0.5,1,0)  
 x1 = balanced\_accuracy\_score(y\_test, pred\_labels1)  
 pred\_labels = x1   
 lst.append(pred\_labels)  
   
 accuracy = np.mean(lst)   
   
 return accuracy  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 study = optuna.create\_study(direction="maximize")  
 study.optimize(objective, n\_trials=15)  
   
 print("Number of finished trials: {}".format(len(study.trials)))  
  
 print("Best trial:")  
 trial = study.best\_trial  
  
 print(" Value: {}".format(trial.value))  
  
 print(" Params: ")  
 for key, value in trial.params.items():  
 print(" {}: {}".format(key, value))

[I 2022-06-03 10:37:40,944] A new study created in memory with name: no-name-af83a771-3017-4cca-a58d-2b103267b223  
[I 2022-06-03 10:38:04,962] Trial 0 finished with value: 0.5955544200306592 and parameters: {'n\_estimators': 5912, 'learning\_rate': 0.22617224642796793, 'lambda\_l1': 7.025642246187133e-07, 'lambda\_l2': 0.15527102988046176, 'num\_leaves': 8397, 'feature\_fraction': 0.7911991474249863, 'bagging\_fraction': 0.7516771569191477, 'bagging\_freq': 2, 'min\_child\_samples': 70}. Best is trial 0 with value: 0.5955544200306592.  
[I 2022-06-03 10:38:22,765] Trial 1 finished with value: 0.5955544200306592 and parameters: {'n\_estimators': 7825, 'learning\_rate': 0.16020566770422498, 'lambda\_l1': 0.405133258577505, 'lambda\_l2': 0.01155966954241298, 'num\_leaves': 11502, 'feature\_fraction': 0.8525886198033977, 'bagging\_fraction': 0.7860050074172116, 'bagging\_freq': 2, 'min\_child\_samples': 96}. Best is trial 0 with value: 0.5955544200306592.  
[I 2022-06-03 10:38:58,848] Trial 2 finished with value: 0.6218702095043434 and parameters: {'n\_estimators': 6153, 'learning\_rate': 0.28733030557012956, 'lambda\_l1': 9.606503301929183e-08, 'lambda\_l2': 7.360580183657517e-08, 'num\_leaves': 11425, 'feature\_fraction': 0.41059589722097806, 'bagging\_fraction': 0.525873930999362, 'bagging\_freq': 4, 'min\_child\_samples': 100}. Best is trial 2 with value: 0.6218702095043434.  
[I 2022-06-03 10:39:22,975] Trial 3 finished with value: 0.5955544200306592 and parameters: {'n\_estimators': 3355, 'learning\_rate': 0.29841213884892287, 'lambda\_l1': 0.00010687787028963682, 'lambda\_l2': 0.005643543846719381, 'num\_leaves': 9389, 'feature\_fraction': 0.5871373155474913, 'bagging\_fraction': 0.5894249786751671, 'bagging\_freq': 4, 'min\_child\_samples': 52}. Best is trial 2 with value: 0.6218702095043434.  
[I 2022-06-03 10:39:38,604] Trial 4 finished with value: 0.5979816044966786 and parameters: {'n\_estimators': 7343, 'learning\_rate': 0.23239491189246317, 'lambda\_l1': 3.688335812212771e-07, 'lambda\_l2': 2.0333812057765197e-08, 'num\_leaves': 4752, 'feature\_fraction': 0.4392377319647347, 'bagging\_fraction': 0.6168722130596996, 'bagging\_freq': 2, 'min\_child\_samples': 118}. Best is trial 2 with value: 0.6218702095043434.  
[I 2022-06-03 10:40:19,161] Trial 5 finished with value: 0.6242973939703628 and parameters: {'n\_estimators': 5484, 'learning\_rate': 0.23401834324016133, 'lambda\_l1': 1.257988800181301e-07, 'lambda\_l2': 0.012870216106228015, 'num\_leaves': 11745, 'feature\_fraction': 0.4010085846066598, 'bagging\_fraction': 0.5948114027062876, 'bagging\_freq': 7, 'min\_child\_samples': 37}. Best is trial 5 with value: 0.6242973939703628.  
[I 2022-06-03 10:40:22,636] Trial 6 finished with value: 0.5453500255493102 and parameters: {'n\_estimators': 3244, 'learning\_rate': 0.13558113120187223, 'lambda\_l1': 0.008753185439941458, 'lambda\_l2': 0.00011184275931870781, 'num\_leaves': 4160, 'feature\_fraction': 0.9843267796303482, 'bagging\_fraction': 0.9564914188295864, 'bagging\_freq': 6, 'min\_child\_samples': 80}. Best is trial 5 with value: 0.6242973939703628.  
[I 2022-06-03 10:40:42,917] Trial 7 finished with value: 0.5955544200306592 and parameters: {'n\_estimators': 7085, 'learning\_rate': 0.2462817777207214, 'lambda\_l1': 9.530165762188304e-08, 'lambda\_l2': 0.16050695027788953, 'num\_leaves': 4745, 'feature\_fraction': 0.750499126345262, 'bagging\_fraction': 0.8767332292797719, 'bagging\_freq': 7, 'min\_child\_samples': 116}. Best is trial 5 with value: 0.6242973939703628.  
[I 2022-06-03 10:41:10,206] Trial 8 finished with value: 0.5979816044966786 and parameters: {'n\_estimators': 7078, 'learning\_rate': 0.09968019692725294, 'lambda\_l1': 4.1722051891736084e-06, 'lambda\_l2': 0.004588494838892006, 'num\_leaves': 6101, 'feature\_fraction': 0.6674852338089806, 'bagging\_fraction': 0.603658175266402, 'bagging\_freq': 5, 'min\_child\_samples': 94}. Best is trial 5 with value: 0.6242973939703628.  
[I 2022-06-03 10:41:19,152] Trial 9 finished with value: 0.6242973939703628 and parameters: {'n\_estimators': 4354, 'learning\_rate': 0.2517347577779429, 'lambda\_l1': 8.641847435224883e-07, 'lambda\_l2': 1.906444547961692e-08, 'num\_leaves': 4429, 'feature\_fraction': 0.48976626127875994, 'bagging\_fraction': 0.8057174670770221, 'bagging\_freq': 1, 'min\_child\_samples': 35}. Best is trial 5 with value: 0.6242973939703628.  
[I 2022-06-03 10:41:32,835] Trial 10 finished with value: 0.5477772100153295 and parameters: {'n\_estimators': 1275, 'learning\_rate': 0.036207672580414615, 'lambda\_l1': 8.340510817040341e-05, 'lambda\_l2': 9.784786776215269e-06, 'num\_leaves': 10036, 'feature\_fraction': 0.5474524635615632, 'bagging\_fraction': 0.40196926731834426, 'bagging\_freq': 7, 'min\_child\_samples': 6}. Best is trial 5 with value: 0.6242973939703628.  
[I 2022-06-03 10:41:48,564] Trial 11 finished with value: 0.5477772100153295 and parameters: {'n\_estimators': 4688, 'learning\_rate': 0.196382136299499, 'lambda\_l1': 1.9286961237755584e-08, 'lambda\_l2': 1.9038259066090794e-06, 'num\_leaves': 6784, 'feature\_fraction': 0.49913715723817603, 'bagging\_fraction': 0.8299965116703919, 'bagging\_freq': 1, 'min\_child\_samples': 29}. Best is trial 5 with value: 0.6242973939703628.  
[I 2022-06-03 10:42:07,693] Trial 12 finished with value: 0.6242973939703628 and parameters: {'n\_estimators': 4468, 'learning\_rate': 0.19169641068161833, 'lambda\_l1': 2.272334812996656e-06, 'lambda\_l2': 0.0002640197713466866, 'num\_leaves': 6998, 'feature\_fraction': 0.5863509877100236, 'bagging\_fraction': 0.6972784595072375, 'bagging\_freq': 5, 'min\_child\_samples': 42}. Best is trial 5 with value: 0.6242973939703628.  
[I 2022-06-03 10:42:40,189] Trial 13 finished with value: 0.5716658150229944 and parameters: {'n\_estimators': 4697, 'learning\_rate': 0.26626556153028064, 'lambda\_l1': 1.4986542288817657e-05, 'lambda\_l2': 1.1730636159307277, 'num\_leaves': 9686, 'feature\_fraction': 0.4067499312095804, 'bagging\_fraction': 0.6904352340746399, 'bagging\_freq': 3, 'min\_child\_samples': 19}. Best is trial 5 with value: 0.6242973939703628.  
[I 2022-06-03 10:42:58,332] Trial 14 finished with value: 0.5979816044966786 and parameters: {'n\_estimators': 3364, 'learning\_rate': 0.19270155759016586, 'lambda\_l1': 0.0015057700809077616, 'lambda\_l2': 6.044946969894971e-07, 'num\_leaves': 8421, 'feature\_fraction': 0.49469324578285145, 'bagging\_fraction': 0.4696722075392503, 'bagging\_freq': 1, 'min\_child\_samples': 55}. Best is trial 5 with value: 0.6242973939703628.

Number of finished trials: 15  
Best trial:  
 Value: 0.6242973939703628  
 Params:   
 n\_estimators: 5484  
 learning\_rate: 0.23401834324016133  
 lambda\_l1: 1.257988800181301e-07  
 lambda\_l2: 0.012870216106228015  
 num\_leaves: 11745  
 feature\_fraction: 0.4010085846066598  
 bagging\_fraction: 0.5948114027062876  
 bagging\_freq: 7  
 min\_child\_samples: 37

study.best\_trial.params

{'n\_estimators': 6584,  
 'learning\_rate': 0.2585180342596222,  
 'lambda\_l1': 0.24869615891449395,  
 'lambda\_l2': 1.5185272830960025e-07,  
 'num\_leaves': 5328,  
 'feature\_fraction': 0.5516054799322556,  
 'bagging\_fraction': 0.9202901878168293,  
 'bagging\_freq': 7,  
 'min\_child\_samples': 127}

param = {  
 "objective": "binary",  
 "metric": "binary\_logloss",  
 "n\_estimators":study.best\_trial.params['n\_estimators'],  
 "learning\_rate": study.best\_trial.params['learning\_rate'],  
 "verbosity": -1,  
 "boosting\_type": "gbdt",  
 "lambda\_l1": study.best\_trial.params['lambda\_l1'],  
 "lambda\_l2": study.best\_trial.params['lambda\_l2'] ,  
 "num\_leaves": study.best\_trial.params['num\_leaves'],  
 "feature\_fraction": study.best\_trial.params['feature\_fraction'],  
 "bagging\_fraction": study.best\_trial.params['bagging\_fraction'],  
 "bagging\_freq": study.best\_trial.params['bagging\_freq'],  
 "min\_child\_samples": study.best\_trial.params['min\_child\_samples'],  
 }  
  
lst = []  
for i in range(5):  
 train\_x, valid\_x, train\_y, valid\_y = train\_test\_split(drugs.loc[:,drugs.columns!='consumption\_cocaine\_last\_month'], drugs.consumption\_cocaine\_last\_month, test\_size=0.15, random\_state = i+1\*11, stratify = drugs.consumption\_cocaine\_last\_month )  
 dtrain = lgb.Dataset(train\_x, label=train\_y)  
  
 gbm = lgb.train(param, dtrain)  
 preds = gbm.predict(valid\_x)  
 pred\_labels = np.where(preds>0.2,1,0)  
 lst.append(balanced\_accuracy\_score(valid\_y, pred\_labels))  
  
print(np.mean(lst))  
our\_metrics(valid\_y, pred\_labels)

0.583495145631068  
Balanced Accuracy: 0.6145886561062851   
confusion:  
 [[199 7]  
 [ 14 5]]

### Test set loading, transformation and prediction

drugs\_test = pd.read\_csv('drugs\_test.csv')  
drugs\_test = getting\_dummies(drugs\_test)  
drugs\_test = getting\_ordinals(drugs\_test)  
drugs\_test = gender\_dummy(drugs\_test)  
drugs\_test = scalling(drugs\_test)  
  
print(drugs.columns, drugs\_test.columns )  
drugs\_test.head()  
print(drugs.shape, drugs\_test.shape)

Index(['education', 'personality\_neuroticism', 'personality\_extraversion',  
 'personality\_openness', 'personality\_agreeableness',  
 'personality\_conscientiousness', 'personality\_impulsiveness',  
 'personality\_sensation', 'consumption\_alcohol',  
 'consumption\_amphetamines', 'consumption\_caffeine',  
 'consumption\_cannabis', 'consumption\_chocolate',  
 'consumption\_mushrooms', 'consumption\_nicotine',  
 'consumption\_cocaine\_last\_month', '18-24', '25-34', '35-44', '45-54',  
 '55-64', '65+', 'Australia', 'New Zealand', 'UK', 'USA',  
 'Other\_country', 'Asian', 'Black', 'Mixed-Black/Asian',  
 'Mixed-White/Asian', 'Mixed-White/Black', 'Other\_y', 'White', 'female'],  
 dtype='object') Index(['education', 'personality\_neuroticism', 'personality\_extraversion',  
 'personality\_openness', 'personality\_agreeableness',  
 'personality\_conscientiousness', 'personality\_impulsiveness',  
 'personality\_sensation', 'consumption\_alcohol',  
 'consumption\_amphetamines', 'consumption\_caffeine',  
 'consumption\_cannabis', 'consumption\_chocolate',  
 'consumption\_mushrooms', 'consumption\_nicotine', '18-24', '25-34',  
 '35-44', '45-54', '55-64', '65+', 'Australia', 'New Zealand', 'UK',  
 'USA', 'Other\_country', 'Asian', 'Black', 'Mixed-Black/Asian',  
 'Mixed-White/Asian', 'Mixed-White/Black', 'Other\_y', 'female'],  
 dtype='object')  
(1500, 35) (385, 33)

preds\_test = clf.predict\_proba(drugs\_test)  
print(np.sum(preds\_test))  
drugs\_test.shape  
  
final\_predictions = pd.DataFrame(preds\_test)  
final\_predictions.to\_csv("classification\_predictions.csv", sep = ',', header=None, index = None)  
final\_predictions

109

0  
0 1  
1 0  
2 0  
3 0  
4 1  
.. ..  
380 0  
381 0  
382 0  
383 0  
384 0  
  
[385 rows x 1 columns]