Final_Project

December 2, 2019

[1]:
```python
# Load libraries
import matplotlib
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
import matplotlib as mpl
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RandomizedSearchCV
from sklearn.feature_selection import SelectFromModel
from sklearn.neighbors import KNeighborsClassifier
import seaborn as sns
from sklearn import metrics
from scipy import stats
from scipy.stats import pearsonr
from scipy.stats import spearmanr
import math
from sklearn.model_selection import train_test_split
from scipy.cluster.hierarchy import linkage, fcluster
from sklearn.cluster import KMeans, DBSCAN
```

[2]:
```python
# Get training and testing data
train_data = pd.read_csv('train_data.csv', delimiter='\t', index_col=0)
test_data  = pd.read_csv('test_data.csv',  delimiter='\t', index_col=0)
# Drop ID, chocolate, the ficticious drug Semer, and legal substances
train_data.drop(columns=['ID', 'Choc', 'Semer', 'Alcohol', 'Nicotine', 'Caff',␣
 ↪'Legalh'], inplace=True)
test_data.drop (columns=['ID', 'Choc', 'Semer', 'Alcohol', 'Nicotine', 'Caff',␣
 ↪'Legalh'], inplace=True)

# Convert categories to integers
for column in train_data.loc[:, 'Amphet':]:
    train_data[column] = train_data[column].astype('category').cat.codes
    train_data[column] = train_data[column].astype('int32')
```

```
for column in test_data.loc[:, 'Amphet':]:
    test_data[column]  = test_data[column].astype('category').cat.codes
    test_data[column]  = test_data[column].astype('int32')

train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1319 entries, 1287 to 684
Data columns (total 25 columns):
Age          1319 non-null float64
Gender       1319 non-null float64
Education    1319 non-null float64
Country      1319 non-null float64
Ethnicity    1319 non-null float64
Nscore       1319 non-null float64
Escore       1319 non-null float64
Oscore       1319 non-null float64
Ascore       1319 non-null float64
Cscore       1319 non-null float64
Impulsive    1319 non-null float64
SS           1319 non-null float64
Amphet       1319 non-null int32
Amyl         1319 non-null int32
Benzos       1319 non-null int32
Cannabis     1319 non-null int32
Coke         1319 non-null int32
Crack        1319 non-null int32
Ecstacy      1319 non-null int32
Heroin       1319 non-null int32
Ketamine     1319 non-null int32
LSD          1319 non-null int32
Meth         1319 non-null int32
Mushrooms    1319 non-null int32
VSA          1319 non-null int32
dtypes: float64(12), int32(13)
memory usage: 200.9 KB
```

[3]:
```
def is_drug_user(row):
    row = row['Amphet':]
    num_zeros = (row == 0).astype(bool).sum()
    if num_zeros == row.size:
        return False
    return True

# Add 'Drug User' column
train_data['Drug User'] = train_data.apply(is_drug_user, axis=1)
test_data['Drug User']  = test_data.apply (is_drug_user, axis=1)
```

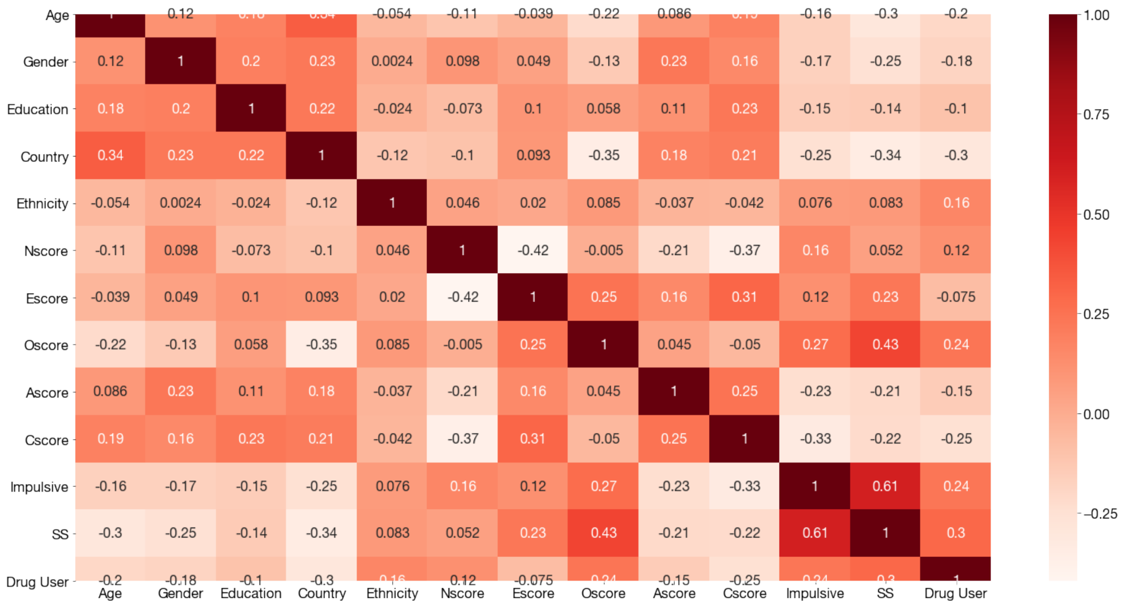
```
train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1319 entries, 1287 to 684
Data columns (total 26 columns):
Age          1319 non-null float64
Gender       1319 non-null float64
Education    1319 non-null float64
Country      1319 non-null float64
Ethnicity    1319 non-null float64
Nscore       1319 non-null float64
Escore       1319 non-null float64
Oscore       1319 non-null float64
Ascore       1319 non-null float64
Cscore       1319 non-null float64
Impulsive    1319 non-null float64
SS           1319 non-null float64
Amphet       1319 non-null int32
Amyl         1319 non-null int32
Benzos       1319 non-null int32
Cannabis     1319 non-null int32
Coke         1319 non-null int32
Crack        1319 non-null int32
Ecstacy      1319 non-null int32
Heroin       1319 non-null int32
Ketamine     1319 non-null int32
LSD          1319 non-null int32
Meth         1319 non-null int32
Mushrooms    1319 non-null int32
VSA          1319 non-null int32
Drug User    1319 non-null bool
dtypes: bool(1), float64(12), int32(13)
memory usage: 202.2 KB
```

[4]:
```
# Examine input variables
matplotlib.rc('font', family='Helvetica Neue', size=16)
fig  = plt.gcf()
fig.set_size_inches(20, 10)
corr = train_data.drop(columns=['Amphet',
                                'Amyl',
                                'Benzos',
                                'Cannabis',
                                'Coke',
                                'Crack',
                                'Ecstacy',
                                'Heroin',
                                'Ketamine',
```

```
                                 'LSD',
                                 'Meth',
                                 'Mushrooms',
                                 'VSA']).corr()
sns.heatmap(corr, annot=True, cmap=plt.cm.Reds)
plt.tight_layout()
plt.savefig('figures/heatmap.png', dpi=300)

# Correlation with output variable
corr_target = abs(corr['Drug User'])

# Select relevant variables
variables   = corr_target[corr_target > corr_target.loc['Age':'SS'].median()]
variables   = variables.loc['Age':'SS'].index.values
var_indices = [train_data.columns.get_loc(variable) for variable in variables]
print('Relevant variables (corr > %.3f):'%corr_target.loc['Age':'SS'].median(),␣
 ↪variables)
```

```
Relevant variables (corr > 0.192): ['Age' 'Country' 'Oscore' 'Cscore'
'Impulsive' 'SS']
```



[5]:
```
drop_cols = ['Amphet', 'Amyl', 'Benzos', 'Cannabis',
             'Coke', 'Crack', 'Ecstacy', 'Heroin',
             'Ketamine', 'LSD', 'Meth', 'Mushrooms',
             'VSA', 'Drug User']
x_train = train_data.drop(columns=drop_cols)
x_test  = test_data.drop (columns=drop_cols)
```

```
y_train = train_data['Drug User']
y_test  = test_data ['Drug User']

# Standardize training and testing data
scaler         = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled  = scaler.fit_transform(x_test)
```

[6]:
```
# Build a decision tree classifier to classify people as drug users or non-users
classifier = DecisionTreeClassifier(criterion='entropy', random_state=0)
classifier.fit(x_train_scaled[:, var_indices], y_train)
```

[6]:
```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=None,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=0, splitter='best')
```

[7]:
```
# Show the structure of the decision tree classifier
print(classifier.tree_.__getstate__()['nodes'])
len(classifier.tree_.__getstate__()['nodes'])
```

```
[(  1,  88,  1,  0.37104077, 0.61578458, 1319, 1.319e+03)
 (  2,  33,  1, -1.03708935, 0.21804385,  603, 6.030e+02)
 (  3,   4,  3, -0.71024391, 0.09916557,  389, 3.890e+02)
 ( -1,  -1, -2, -2.        , 0.        ,  124, 1.240e+02)
 (  5,   6,  4, -0.48617859, 0.1350362 ,  265, 2.650e+02)
 ( -1,  -1, -2, -2.        , 0.        ,   66, 6.600e+01)
 (  7,  16,  0, -0.6268507 , 0.16932446,  199, 1.990e+02)
 (  8,   9,  2,  1.30229104, 0.07360348,  112, 1.120e+02)
 ( -1,  -1, -2, -2.        , 0.        ,   89, 8.900e+01)
 ( 10,  15,  2,  1.5078187 , 0.25801867,   23, 2.300e+01)
 ( 11,  12,  4,  0.75859267, 0.54356444,    8, 8.000e+00)
 ( -1,  -1, -2, -2.        , 0.        ,    6, 6.000e+00)
 ( 13,  14,  3,  0.49862912, 1.        ,    2, 2.000e+00)
 ( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
 ( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
 ( -1,  -1, -2, -2.        , 0.        ,   15, 1.500e+01)
 ( 17,  30,  3,  1.46841699, 0.2690553 ,   87, 8.700e+01)
 ( 18,  25,  5, -0.37598695, 0.22028327,   85, 8.500e+01)
 ( 19,  24,  2,  1.41152394, 0.65002242,   12, 1.200e+01)
 ( 20,  23,  4, -0.00517242, 0.43949699,   11, 1.100e+01)
 ( 21,  22,  3,  0.22490337, 0.81127812,    4, 4.000e+00)
 ( -1,  -1, -2, -2.        , 0.        ,    3, 3.000e+00)
 ( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
 ( -1,  -1, -2, -2.        , 0.        ,    7, 7.000e+00)
```

```
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
( 26,  29,  2, -0.66125342, 0.10441908,   73, 7.300e+01)
( 27,  28,  5,  1.40527081, 0.81127812,    4, 4.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    3, 3.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.        , 0.        ,   69, 6.900e+01)
( 31,  32,  2,  1.70191941, 1.        ,    2, 2.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
( 34,  59,  5, -0.70359236, 0.38346413,  214, 2.140e+02)
( 35,  58,  3,  2.89427984, 0.81127812,   36, 3.600e+01)
( 36,  57,  3,  1.2207939 , 0.77551266,   35, 3.500e+01)
( 37,  38,  2, -2.10688007, 0.83664074,   30, 3.000e+01)
( -1,  -1, -2, -2.        , 0.        ,    4, 4.000e+00)
( 39,  56,  0,  1.65218514, 0.89049164,   26, 2.600e+01)
( 40,  41,  2, -1.75221407, 0.85545081,   25, 2.500e+01)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
( 42,  43,  2, -1.05371639, 0.81127812,   24, 2.400e+01)
( -1,  -1, -2, -2.        , 0.        ,    5, 5.000e+00)
( 44,  45,  5, -1.8749342 , 0.89974376,   19, 1.900e+01)
( -1,  -1, -2, -2.        , 0.        ,    4, 4.000e+00)
( 46,  47,  4, -1.10488191, 0.97095059,   15, 1.500e+01)
( -1,  -1, -2, -2.        , 0.        ,    3, 3.000e+00)
( 48,  51,  3, -1.12799984, 0.81127812,   12, 1.200e+01)
( 49,  50,  0,  0.21027907, 0.91829583,    3, 3.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    2, 2.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
( 52,  55,  2, -0.53011969, 0.50325833,    9, 9.000e+00)
( 53,  54,  0, -0.6268507 , 1.        ,    2, 2.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    7, 7.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    5, 5.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
( 60,  61,  3, -0.4606221 , 0.23919262,  178, 1.780e+02)
( -1,  -1, -2, -2.        , 0.        ,   76, 7.600e+01)
( 62,  87,  3,  0.85076225, 0.36078057,  102, 1.020e+02)
( 63,  86,  2,  1.97521383, 0.42806963,   80, 8.000e+01)
( 64,  65,  4, -0.00517242, 0.38774318,   79, 7.900e+01)
( -1,  -1, -2, -2.        , 0.        ,   31, 3.100e+01)
( 66,  85,  2,  0.62622762, 0.54356444,   48, 4.800e+01)
( 67,  84,  4,  1.16431427, 0.68403844,   33, 3.300e+01)
( 68,  73,  3,  0.06268557, 0.79504028,   25, 2.500e+01)
( 69,  70,  0,  0.21027907, 0.41381685,   12, 1.200e+01)
( -1,  -1, -2, -2.        , 0.        ,    9, 9.000e+00)
( 71,  72,  2, -0.65939173, 0.91829583,    3, 3.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
```

```
( -1,  -1, -2, -2.        , 0.        ,    2, 2.000e+00)
( 74,  81,  3,  0.50337908, 0.9612366 ,   13, 1.300e+01)
( 75,  80,  0,  0.21027907, 0.91829583,    6, 6.000e+00)
( 76,  77,  2,  0.19164676, 0.91829583,    3, 3.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
( 78,  79,  5,  0.2593739 , 1.        ,    2, 2.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    3, 3.000e+00)
( 82,  83,  5, -0.06127545, 0.59167278,    7, 7.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    6, 6.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    8, 8.000e+00)
( -1,  -1, -2, -2.        , 0.        ,   15, 1.500e+01)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.        , 0.        ,   22, 2.200e+01)
( 89, 390,  5, -0.06127545, 0.81348426,  716, 7.160e+02)
( 90, 153,  3, -0.20422167, 0.92869829,  433, 4.330e+02)
( 91, 152,  0,  2.51666594, 0.69734097,  117, 1.170e+02)
( 92,  93,  3, -1.84134251, 0.66657836,  115, 1.150e+02)
( -1,  -1, -2, -2.        , 0.        ,    7, 7.000e+00)
( 94, 125,  5, -0.70359236, 0.69128987,  108, 1.080e+02)
( 95, 102,  0, -0.6268507 , 0.81127812,   52, 5.200e+01)
( 96, 101,  4,  0.75859267, 0.97095059,   10, 1.000e+01)
( 97, 100,  2, -0.72547463, 0.81127812,    8, 8.000e+00)
( 98,  99,  4, -2.08610392, 0.91829583,    3, 3.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    2, 2.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    5, 5.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    2, 2.000e+00)
(103, 124,  4, -0.48617859, 0.65002242,   42, 4.200e+01)
(104, 121,  0,  1.65218514, 0.79732651,   29, 2.900e+01)
(105, 120,  0,  0.88757563, 0.72192809,   25, 2.500e+01)
(106, 117,  3, -0.4606221 , 0.87398105,   17, 1.700e+01)
(107, 108,  2, -1.89755994, 0.74959526,   14, 1.400e+01)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
(109, 110,  2, -1.35220063, 0.61938219,   13, 1.300e+01)
( -1,  -1, -2, -2.        , 0.        ,    7, 7.000e+00)
(111, 116,  5, -1.04381639, 0.91829583,    6, 6.000e+00)
(112, 113,  2, -1.13100407, 0.91829583,    3, 3.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
(114, 115,  3, -0.58359382, 1.        ,    2, 2.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    3, 3.000e+00)
(118, 119,  2, -1.89755994, 0.91829583,    3, 3.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    2, 2.000e+00)
```

```
( -1,  -1, -2, -2.        , 0.        ,    8, 8.000e+00)
(122, 123,  5, -1.04381639, 1.        ,    4, 4.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    2, 2.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    2, 2.000e+00)
( -1,  -1, -2, -2.        , 0.        ,   13, 1.300e+01)
(126, 127,  2, -1.7622633 , 0.54356444,   56, 5.600e+01)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
(128, 129,  4, -0.48617859, 0.49716776,   55, 5.500e+01)
( -1,  -1, -2, -2.        , 0.        ,   12, 1.200e+01)
(130, 133,  3, -1.4426015 , 0.58301942,   43, 4.300e+01)
(131, 132,  3, -1.57019365, 0.97095059,    5, 5.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    3, 3.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    2, 2.000e+00)
(134, 147,  3, -0.33532771, 0.48546076,   38, 3.800e+01)
(135, 144,  2,  0.94170904, 0.33729007,   32, 3.200e+01)
(136, 143,  5, -0.37598695, 0.2108423 ,   30, 3.000e+01)
(137, 142,  3, -0.76873922, 0.50325833,    9, 9.000e+00)
(138, 141,  0,  0.88757563, 0.81127812,    4, 4.000e+00)
(139, 140,  2, -0.39899582, 1.        ,    2, 2.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    2, 2.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    5, 5.000e+00)
( -1,  -1, -2, -2.        , 0.        ,   21, 2.100e+01)
(145, 146,  5, -0.37598695, 1.        ,    2, 2.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
(148, 151,  0,  0.21027907, 0.91829583,    6, 6.000e+00)
(149, 150,  2, -1.20443997, 0.91829583,    3, 3.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    2, 2.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    3, 3.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    2, 2.000e+00)
(154, 321,  2, -0.11631877, 0.97205045,  316, 3.160e+02)
(155, 316,  0,  1.65218514, 0.99540013,  213, 2.130e+02)
(156, 253,  3,  0.85076225, 0.99142668,  202, 2.020e+02)
(157, 252,  4,  0.3922534 , 0.95735567,  124, 1.240e+02)
(158, 203,  2, -0.79162669, 0.96995049,  118, 1.180e+02)
(159, 168,  2, -1.75221407, 0.89486923,   61, 6.100e+01)
(160, 167,  3,  0.50337908, 0.97986876,   12, 1.200e+01)
(161, 162,  0,  0.21027907, 0.86312057,    7, 7.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
(163, 164,  4, -1.10488191, 0.65002242,    6, 6.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    4, 4.000e+00)
(165, 166,  2, -2.01397288, 1.        ,    2, 2.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    5, 5.000e+00)
```

```
(169, 202,  3,  0.67402029, 0.80309098,   49, 4.900e+01)
(170, 199,  4, -0.00517242, 0.86312057,   42, 4.200e+01)
(171, 186,  5, -1.04381639, 0.82128094,   39, 3.900e+01)
(172, 185,  2, -0.91954565, 0.93666738,   17, 1.700e+01)
(173, 174,  0, -0.6268507 , 0.98522814,   14, 1.400e+01)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
(175, 176,  5, -1.8749342 , 0.9612366 ,   13, 1.300e+01)
( -1,  -1, -2, -2.        , 0.        ,    3, 3.000e+00)
(177, 180,  3,  0.06268557, 1.        ,   10, 1.000e+01)
(178, 179,  5, -1.40861261, 0.81127812,    4, 4.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    3, 3.000e+00)
(181, 184,  4, -1.10488191, 0.91829583,    6, 6.000e+00)
(182, 183,  2, -1.55224192, 0.91829583,    3, 3.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    2, 2.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    3, 3.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    3, 3.000e+00)
(187, 188,  2, -1.20147705, 0.68403844,   22, 2.200e+01)
( -1,  -1, -2, -2.        , 0.        ,    7, 7.000e+00)
(189, 190,  3, -0.06993568, 0.83664074,   15, 1.500e+01)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
(191, 194,  4, -1.10488191, 0.74959526,   14, 1.400e+01)
(192, 193,  0,  0.21027907, 0.91829583,    3, 3.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    2, 2.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
(195, 198,  3,  0.06268557, 0.43949699,   11, 1.100e+01)
(196, 197,  4, -0.48617859, 0.91829583,    3, 3.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    2, 2.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    8, 8.000e+00)
(200, 201,  3,  0.19535667, 0.91829583,    3, 3.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    2, 2.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    7, 7.000e+00)
(204, 213,  4, -1.10488191, 0.99977797,   57, 5.700e+01)
(205, 206,  0,  0.21027907, 0.93666738,   17, 1.700e+01)
( -1,  -1, -2, -2.        , 0.        ,    5, 5.000e+00)
(207, 212,  2, -0.53012955, 0.41381685,   12, 1.200e+01)
(208, 209,  3,  0.42542024, 0.72192809,    5, 5.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    3, 3.000e+00)
(210, 211,  5, -0.54245467, 1.        ,    2, 2.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    7, 7.000e+00)
(214, 237,  0,  0.21027907, 0.98370826,   40, 4.000e+01)
(215, 216,  3, -0.06993568, 0.98522814,   21, 2.100e+01)
( -1,  -1, -2, -2.        , 0.        ,    2, 2.000e+00)
```

```
(217, 222,  4, -0.48617859, 0.99800088,   19, 1.900e+01)
(218, 221,  3,  0.58981118, 0.91829583,    6, 6.000e+00)
(219, 220,  5, -1.23485631, 0.91829583,    3, 3.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    2, 2.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    3, 3.000e+00)
(223, 224,  5, -0.70359236, 0.9612366 ,   13, 1.300e+01)
( -1,  -1, -2, -2.        , 0.        ,    3, 3.000e+00)
(225, 234,  4, -0.00517242, 1.        ,   10, 1.000e+01)
(226, 233,  0, -0.6268507 , 0.91829583,    6, 6.000e+00)
(227, 228,  2, -0.66125342, 1.        ,    4, 4.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
(229, 230,  3,  0.58981118, 0.91829583,    3, 3.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
(231, 232,  2, -0.46396762, 1.        ,    2, 2.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    2, 2.000e+00)
(235, 236,  3,  0.06268557, 0.81127812,    4, 4.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    3, 3.000e+00)
(238, 251,  2, -0.39899582, 0.83147439,   19, 1.900e+01)
(239, 240,  5, -1.8749342 , 0.94028596,   14, 1.400e+01)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
(241, 242,  5, -0.87734866, 0.89049164,   13, 1.300e+01)
( -1,  -1, -2, -2.        , 0.        ,    5, 5.000e+00)
(243, 248,  2, -0.66125342, 1.        ,    8, 8.000e+00)
(244, 245,  4, -0.48617859, 0.81127812,    4, 4.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    2, 2.000e+00)
(246, 247,  3,  0.51185234, 1.        ,    2, 2.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
(249, 250,  3,  0.06268557, 0.81127812,    4, 4.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    3, 3.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    5, 5.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    6, 6.000e+00)
(254, 263,  3,  1.0380559 , 0.9923985 ,   78, 7.800e+01)
(255, 256,  0, -0.6268507 , 0.77322667,   22, 2.200e+01)
( -1,  -1, -2, -2.        , 0.        ,    2, 2.000e+00)
(257, 262,  2, -0.26371399, 0.6098403 ,   20, 2.000e+01)
(258, 261,  2, -1.35220063, 0.48546076,   19, 1.900e+01)
(259, 260,  0,  0.21027907, 1.        ,    4, 4.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    2, 2.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    2, 2.000e+00)
( -1,  -1, -2, -2.        , 0.        ,   15, 1.500e+01)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
(264, 291,  0,  0.21027907, 0.99631652,   56, 5.600e+01)
```

```
(265, 290,  4, -0.00517242, 0.91829583,   27, 2.700e+01)
(266, 289,  3,  1.92687446, 0.97602065,   22, 2.200e+01)
(267, 268,  2, -1.75221407, 0.99800088,   19, 1.900e+01)
( -1,  -1, -2, -2.        , 0.        ,    3, 3.000e+00)
(269, 288,  4, -0.48617859, 0.98869941,   16, 1.600e+01)
(270, 271,  2, -1.61691743, 0.99679163,   15, 1.500e+01)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
(272, 287,  5, -0.37598695, 1.        ,   14, 1.400e+01)
(273, 280,  5, -1.04381639, 0.99572745,   13, 1.300e+01)
(274, 279,  5, -1.8749342 , 0.91829583,    6, 6.000e+00)
(275, 276,  3,  1.29844373, 0.91829583,    3, 3.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
(277, 278,  0, -0.6268507 , 1.        ,    2, 2.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    3, 3.000e+00)
(281, 286,  3,  1.63621718, 0.86312057,    7, 7.000e+00)
(282, 283,  2, -0.53011969, 0.65002242,    6, 6.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    4, 4.000e+00)
(284, 285,  0, -0.6268507 , 1.        ,    2, 2.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    3, 3.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    5, 5.000e+00)
(292, 293,  2, -1.61691743, 0.97844933,   29, 2.900e+01)
( -1,  -1, -2, -2.        , 0.        ,    3, 3.000e+00)
(294, 315,  3,  2.18686354, 0.99572745,   26, 2.600e+01)
(295, 296,  2, -1.27491295, 1.        ,   24, 2.400e+01)
( -1,  -1, -2, -2.        , 0.        ,    2, 2.000e+00)
(297, 298,  2, -0.91954565, 0.99403021,   22, 2.200e+01)
( -1,  -1, -2, -2.        , 0.        ,    4, 4.000e+00)
(299, 314,  3,  1.72043622, 0.99107606,   18, 1.800e+01)
(300, 301,  5, -1.40861261, 0.954434  ,   16, 1.600e+01)
( -1,  -1, -2, -2.        , 0.        ,    2, 2.000e+00)
(302, 307,  2, -0.66125342, 0.98522814,   14, 1.400e+01)
(303, 306,  2, -0.79162669, 0.72192809,    5, 5.000e+00)
(304, 305,  3,  1.38266277, 1.        ,    2, 2.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    3, 3.000e+00)
(308, 313,  3,  1.54606682, 0.99107606,    9, 9.000e+00)
(309, 312,  0,  0.88757563, 0.86312057,    7, 7.000e+00)
(310, 311,  5, -1.04381639, 1.        ,    4, 4.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    2, 2.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    2, 2.000e+00)
```

```
( -1,  -1, -2, -2.        , 0.        ,    3, 3.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    2, 2.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    2, 2.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    2, 2.000e+00)
(317, 318,  3,  0.86131394, 0.84535094,   11, 1.100e+01)
( -1,  -1, -2, -2.        , 0.        ,    7, 7.000e+00)
(319, 320,  4, -1.73034325, 0.81127812,    4, 4.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    3, 3.000e+00)
(322, 361,  3,  1.0380559 , 0.8575588 ,  103, 1.030e+02)
(323, 348,  3,  0.50337908, 0.74551784,   66, 6.600e+01)
(324, 347,  0,  1.65218514, 0.877962  ,   37, 3.700e+01)
(325, 338,  4, -0.48617859, 0.85240518,   36, 3.600e+01)
(326, 337,  2,  0.77426478, 0.97095059,   20, 2.000e+01)
(327, 332,  3,  0.19535667, 0.99750255,   17, 1.700e+01)
(328, 331,  0,  0.21027907, 0.81127812,    8, 8.000e+00)
(329, 330,  5, -0.37598695, 0.91829583,    3, 3.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    2, 2.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    5, 5.000e+00)
(333, 334,  0, -0.2940312 , 0.91829583,    9, 9.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    2, 2.000e+00)
(335, 336,  5, -1.40861261, 0.59167278,    7, 7.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    6, 6.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    3, 3.000e+00)
(339, 340,  4, -0.00517242, 0.54356444,   16, 1.600e+01)
( -1,  -1, -2, -2.        , 0.        ,    7, 7.000e+00)
(341, 346,  2,  1.11812109, 0.76420451,    9, 9.000e+00)
(342, 345,  2,  0.19571585, 0.54356444,    8, 8.000e+00)
(343, 344,  0,  0.88757563, 0.91829583,    3, 3.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    2, 2.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    5, 5.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
(349, 360,  2,  0.34604435, 0.47983202,   29, 2.900e+01)
(350, 355,  2,  0.19571585, 0.67229482,   17, 1.700e+01)
(351, 354,  2,  0.04131827, 0.39124356,   13, 1.300e+01)
(352, 353,  3,  0.67402029, 0.72192809,    5, 5.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    4, 4.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    8, 8.000e+00)
(356, 357,  4, -0.48617859, 1.        ,    4, 4.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
(358, 359,  5, -0.37598695, 0.91829583,    3, 3.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    2, 2.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
```

```
( -1,  -1, -2, -2.        , 0.        ,   12, 1.200e+01)
(362, 389,  3,  1.92687446, 0.97402486,   37, 3.700e+01)
(363, 388,  4,  0.3922534 , 0.99403021,   33, 3.300e+01)
(364, 379,  0,  0.88757563, 0.99924925,   31, 3.100e+01)
(365, 378,  4, -0.48617859, 0.93666738,   17, 1.700e+01)
(366, 377,  2,  1.21441585, 0.99572745,   13, 1.300e+01)
(367, 372,  3,  1.54606682, 0.99403021,   11, 1.100e+01)
(368, 371,  2,  0.27100848, 0.65002242,    6, 6.000e+00)
(369, 370,  3,  1.38419795, 0.91829583,    3, 3.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    2, 2.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    3, 3.000e+00)
(373, 374,  4, -1.10488191, 0.72192809,    5, 5.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    3, 3.000e+00)
(375, 376,  3,  1.72043622, 1.        ,    2, 2.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    2, 2.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    4, 4.000e+00)
(380, 383,  4, -1.10488191, 0.94028596,   14, 1.400e+01)
(381, 382,  2,  0.11635413, 0.81127812,    4, 4.000e+00)
( -1,  -1, -2, -2.        , 1.        ,    2, 2.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    2, 2.000e+00)
(384, 385,  3,  1.38419795, 0.72192809,   10, 1.000e+01)
( -1,  -1, -2, -2.        , 0.        ,    5, 5.000e+00)
(386, 387,  3,  1.54606682, 0.97095059,    5, 5.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    2, 2.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    3, 3.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    2, 2.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    4, 4.000e+00)
(391, 412,  3,  0.19535667, 0.49852882,  283, 2.830e+02)
(392, 411,  2, -0.11631877, 0.18970512,  172, 1.720e+02)
(393, 410,  5,  0.61550494, 0.35001059,   76, 7.600e+01)
(394, 395,  3, -0.83304617, 0.46899559,   50, 5.000e+01)
( -1,  -1, -2, -2.        , 0.        ,   13, 1.300e+01)
(396, 403,  4,  0.3922534 , 0.57135497,   37, 3.700e+01)
(397, 398,  3, -0.06993568, 0.25801867,   23, 2.300e+01)
( -1,  -1, -2, -2.        , 0.        ,   19, 1.900e+01)
(399, 402,  0,  0.88757563, 0.81127812,    4, 4.000e+00)
(400, 401,  4, -0.84193926, 1.        ,    2, 2.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    2, 2.000e+00)
(404, 405,  3, -0.71024391, 0.86312057,   14, 1.400e+01)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
(406, 407,  5,  0.2593739 , 0.77934984,   13, 1.300e+01)
( -1,  -1, -2, -2.        , 0.        ,    7, 7.000e+00)
(408, 409,  0,  0.21027907, 1.        ,    6, 6.000e+00)
```

```
( -1,  -1, -2, -2.        , 0.        ,    3, 3.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    3, 3.000e+00)
( -1,  -1, -2, -2.        , 0.        ,   26, 2.600e+01)
( -1,  -1, -2, -2.        , 0.        ,   96, 9.600e+01)
(413, 468,  2,  0.48921742, 0.78531981,  111, 1.110e+02)
(414, 419,  2, -1.35220063, 0.89974376,   76, 7.600e+01)
(415, 418,  2, -1.61691743, 0.97095059,   10, 1.000e+01)
(416, 417,  5,  0.2593739 , 0.98522814,    7, 7.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    4, 4.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    3, 3.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    3, 3.000e+00)
(420, 441,  0,  0.21027907, 0.84535094,   66, 6.600e+01)
(421, 426,  3,  0.85076225, 0.66096234,   35, 3.500e+01)
(422, 423,  2,  0.34604435, 0.28639696,   20, 2.000e+01)
( -1,  -1, -2, -2.        , 0.        ,   17, 1.700e+01)
(424, 425,  5,  0.44842939, 0.91829583,    3, 3.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    2, 2.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
(427, 440,  5,  1.64387596, 0.91829583,   15, 1.500e+01)
(428, 433,  3,  1.72043622, 0.86312057,   14, 1.400e+01)
(429, 430,  2,  0.34604435, 0.54356444,    8, 8.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    6, 6.000e+00)
(431, 432,  0, -0.6268507 , 1.        ,    2, 2.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
(434, 435,  4, -0.00517242, 1.        ,    6, 6.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    2, 2.000e+00)
(436, 439,  0, -0.6268507 , 0.81127812,    4, 4.000e+00)
(437, 438,  4,  0.57923037, 1.        ,    2, 2.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    2, 2.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
(442, 453,  4, -0.00517242, 0.96290041,   31, 3.100e+01)
(443, 444,  5,  0.2593739 , 0.72192809,   15, 1.500e+01)
( -1,  -1, -2, -2.        , 0.        ,    6, 6.000e+00)
(445, 452,  3,  1.29844373, 0.91829583,    9, 9.000e+00)
(446, 451,  3,  0.85076225, 0.81127812,    8, 8.000e+00)
(447, 450,  2,  0.19571585, 0.97095059,    5, 5.000e+00)
(448, 449,  3,  0.67402029, 0.81127812,    4, 4.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    3, 3.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    3, 3.000e+00)
( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
(454, 455,  0,  0.88757563, 0.98869941,   16, 1.600e+01)
( -1,  -1, -2, -2.        , 0.        ,    3, 3.000e+00)
(456, 461,  2, -0.53012955, 0.99572745,   13, 1.300e+01)
```

```
 (457, 458,  4,  1.16431427, 0.72192809,    5, 5.000e+00)
 ( -1,  -1, -2, -2.        , 0.        ,    3, 3.000e+00)
 (459, 460,  3,  0.58981118, 1.        ,    2, 2.000e+00)
 ( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
 ( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
 (462, 467,  0,  1.65218514, 0.81127812,    8, 8.000e+00)
 (463, 466,  4,  0.3922534 , 0.59167278,    7, 7.000e+00)
 (464, 465,  3,  0.85076225, 0.91829583,    3, 3.000e+00)
 ( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
 ( -1,  -1, -2, -2.        , 0.        ,    2, 2.000e+00)
 ( -1,  -1, -2, -2.        , 0.        ,    4, 4.000e+00)
 ( -1,  -1, -2, -2.        , 0.        ,    1, 1.000e+00)
 (469, 474,  5,  0.61550494, 0.31599713,   35, 3.500e+01)
 (470, 471,  4,  0.3922534 , 0.50325833,   18, 1.800e+01)
 ( -1,  -1, -2, -2.        , 0.        ,   12, 1.200e+01)
 (472, 473,  2,  0.77426478, 0.91829583,    6, 6.000e+00)
 ( -1,  -1, -2, -2.        , 0.        ,    4, 4.000e+00)
 ( -1,  -1, -2, -2.        , 0.        ,    2, 2.000e+00)
 ( -1,  -1, -2, -2.        , 0.        ,   17, 1.700e+01)]
```

[7]: 475

[8]:
```
# The mean accuracy and the 95% confidence interval of 10-fold cross validation
scores = cross_val_score(classifier, x_train_scaled[:, var_indices], y_train,
                         cv=10, scoring='accuracy')
print('Accuracy: %0.3f (+/- %0.3f)'%(scores.mean(), scores.std() * 2))

# The mean F1 score and the 95% confidence interval of 10-fold cross validation
scores = cross_val_score(classifier, x_train_scaled[:, var_indices], y_train,
                         cv=10, scoring='f1')
print('F1 score: %0.3f (+/- %0.3f)'%(scores.mean(), scores.std() * 2))
```

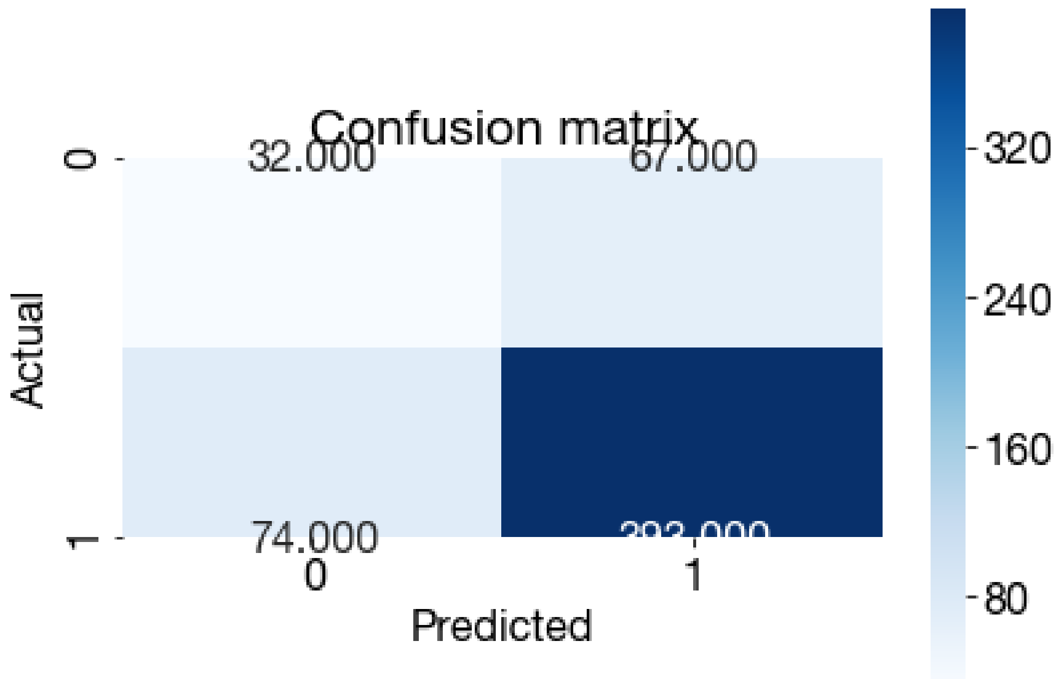
```
Accuracy: 0.790 (+/- 0.082)
F1 score: 0.874 (+/- 0.054)
```

[9]:
```
# Feature importances (aka Gini importance)
count = 0
print('Gini importance:')
for (variable, feature_importance) in sorted(zip(variables,
    classifier.feature_importances_), key=lambda x: x[1], reverse=True):
    count += 1
    print('(%d)' % count, '%0.3f,'%feature_importance, variable)
```

```
Gini importance:
(1) 0.243, Cscore
(2) 0.203, Oscore
(3) 0.173, SS
(4) 0.135, Country
```

```
(5) 0.131, Age
(6) 0.115, Impulsive
```

[10]:
```
# Predict the class labels for the test set using the decision tree classifier
y_pred = classifier.predict(x_test_scaled[:, var_indices])

# Plot the corresponding confusion matrix
conf_matrix = metrics.confusion_matrix(y_test, y_pred)
sns.heatmap(conf_matrix, annot=True, fmt='.3f', square=True, cmap=plt.cm.Blues)
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Confusion matrix')
plt.tight_layout()
plt.savefig('figures/conf_matrix_dt.png', dpi=300)
```



[11]:
```
# Compute evaluation metrics for the decision tree classifier
accuracy  = metrics.accuracy_score(y_test, y_pred)
error     = 1 - metrics.accuracy_score(y_test, y_pred)
precision = metrics.precision_score(y_test, y_pred, average=None)
recall    = metrics.recall_score(y_test, y_pred, average=None)
f1_score  = metrics.f1_score(y_test, y_pred, average=None)
print('Accuracy: ', '%0.3f'%accuracy)
print('Error:    ', '%0.3f'%error)
print('Precision:', '[%0.3f'%precision[0], '%0.3f]'%precision[1])
```

```
print('Recall:   ', '[%0.3f'%recall[0],    '%0.3f]'%recall[1])
print('F1 score: ', '[%0.3f'%f1_score[0],  '%0.3f]'%f1_score[1])
```

```
Accuracy:  0.751
Error:     0.249
Precision: [0.302 0.854]
Recall:    [0.323 0.842]
F1 score:  [0.312 0.848]
```

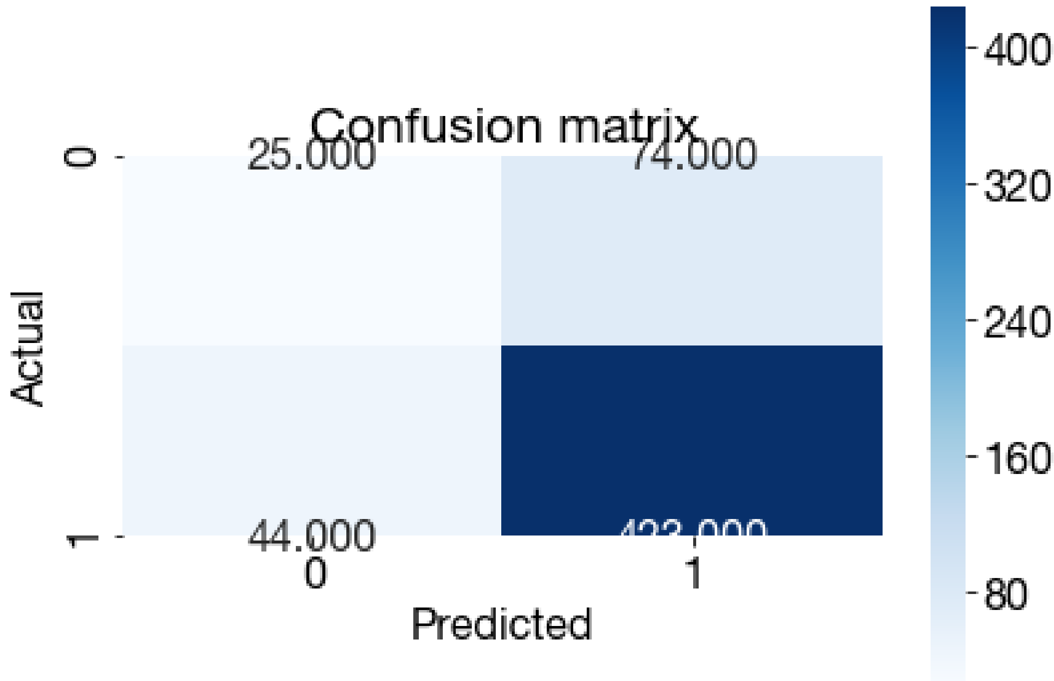
[12]:
```
# Build a k-nearest neighbors classifier to classify people as drug users or␣
↪non-users
classifier = KNeighborsClassifier(n_neighbors=3)
classifier.fit(x_train_scaled[:, var_indices], y_train)
```

[12]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=3, p=2,
                     weights='uniform')
```

[13]:
```
# The mean accuracy and the 95% confidence interval of 10-fold cross validation
scores = cross_val_score(classifier, x_train_scaled[:, var_indices], y_train,
                         cv=10, scoring='accuracy')
print('Accuracy: %0.3f (+/- %0.3f)'%(scores.mean(), scores.std() * 2))

# The mean F1 score and the 95% confidence interval of 10-fold cross validation
scores = cross_val_score(classifier, x_train_scaled[:, var_indices], y_train,
                         cv=10, scoring='f1')
print('F1 score: %0.3f (+/- %0.3f)'%(scores.mean(), scores.std() * 2))
```

```
Accuracy: 0.809 (+/- 0.060)
F1 score: 0.888 (+/- 0.037)
```

[14]:
```
# Predict the class labels for the test set using the k-nearest neighbors␣
↪classifier
y_pred = classifier.predict(x_test_scaled[:, var_indices])

# Plot the corresponding confusion matrix
conf_matrix = metrics.confusion_matrix(y_test, y_pred)
sns.heatmap(conf_matrix, annot=True, fmt='.3f', square=True, cmap=plt.cm.Blues)
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Confusion matrix')
plt.tight_layout()
plt.savefig('figures/conf_matrix_knn.png', dpi=300)
```



[15]:
```
# Compute evaluation metrics for the k-nearest neighbors classifier
accuracy  = metrics.accuracy_score(y_test, y_pred)
error     = 1 - metrics.accuracy_score(y_test, y_pred)
precision = metrics.precision_score(y_test, y_pred, average=None)
recall    = metrics.recall_score(y_test, y_pred, average=None)
f1_score  = metrics.f1_score(y_test, y_pred, average=None)
print('Accuracy: ', '%0.3f'%accuracy)
print('Error:    ', '%0.3f'%error)
print('Precision:', '[%0.3f'%precision[0], '%0.3f]'%precision[1])
print('Recall:   ', '[%0.3f'%recall[0],    '%0.3f]'%recall[1])
print('F1 score: ', '[%0.3f'%f1_score[0],  '%0.3f]'%f1_score[1])
```

```
Accuracy:  0.792
Error:     0.208
Precision: [0.362 0.851]
Recall:    [0.253 0.906]
F1 score:  [0.298 0.878]
```

[2]:
```
def gender_map(x):
    if math.isclose(x, 0.48246):
        return 'Female'
    else:
        return 'Male'
```

[3]:
```
def country_map(x):
    countries = [(-0.09765, 'Australia'), (0.24923,'Canada'),
                 (-0.46841,'New Zealand'), (0.21128,'Republic of Ireland'),
                 (0.96082,'UK'), (-0.57009,'USA'), (-0.28519,'Other')]
    for (value, country) in countries:
        if math.isclose(x, value):
            return country;
    return None
```

[4]:
```
# Function to determine illegal drug usage per person
def is_drug_user(row):
    row = row['Amphet':]
    num_zeros = (row == 0).astype(bool).sum()
    if num_zeros == row.size:
        return False
    return True
```

[5]:
```
train_data = pd.read_csv("train_data.csv", delimiter='\t', index_col=0)
test_data = pd.read_csv("test_data.csv", delimiter='\t', index_col=0)
```

[6]:
```
def create_plot(ax, data, color='#99004f'):
    usage = data.apply(pd.Series.value_counts, axis=0)
    drug = usage.iloc[:,0].name
    usages = usage.sort_index().iloc[:,0].values
    width = 0.05
    xpos = np.arange(len(usage.index),dtype='float64')
    xpos *= 0.10
    ax.bar(xpos, usages, width=width, color=color)
    rects = ax.patches
    for rect, usage in zip(rects,usages):
        x = rect.get_x() + rect.get_width()/2
        y = rect.get_height() + 0.5
        ax.text(x, y, usage, ha='center',va='bottom', fontsize=5)

    usage_texts = ['Never Used','Decade Ago','Last Decade',
                   'Last Year', 'Last Month',
                   'Last Week', 'Last Day']
    ax.set_ylim(bottom=0, top=1800)
    ax.set_xticks(xpos)
    ax.set_xticklabels(usage_texts)
    ax.tick_params(axis='y', labelsize=4)
    ax.tick_params(axis ='x', labelrotation=15, labelsize=4, width=0.7)
    ax.text(0.5, 0.9, drug, horizontalalignment='center',
            transform=ax.transAxes, fontsize=9)
```

[7]:
```
def create_usage_subplots(data):
```

```
    colors = ['#99004f', '#007acc', '#009900', '#e67300',
              '#cc0000','#0000b3', '#7a00cc', '#e6e600',
              '#2eb8b8']

    data = data.rename(columns={'Amphet':'Amphetamine', 'Amyl':'Amyl Nitrite',
            'Benzos':'Benzodiazepine','Caff':'Caffeine','Coke':'Cocaine',
            'Meth':'Methamphetamine'})


    drug_data = data.loc[:,'Alcohol':].apply(lambda series: series.astype(bool).sum(),axis=0)
    drug_data = drug_data.sort_values(ascending=False)
    print(drug_data)
    drugs = drug_data.index.values

    nrows = 5
    ncols= 4
    fig, axs = plt.subplots(nrows=nrows, ncols=ncols, sharex=False, sharey=True, figsize=(10,8))
    plt.subplots_adjust(wspace=0.02, hspace= 0.3, top = 0.95)
    color_idx = 0
    i = 0
    for row in range(nrows):
        for col in range(ncols):
            if col != 0:
                axs[row, col].tick_params(axis='y', width=0)
            if i >= len(drugs):
                axs[row, col].set_visible(False)
            else:
                create_plot(axs[row, col], data[[drugs[i]]], colors[color_idx])
            i += 1
            color_idx = (color_idx+1)%len(colors)
    fig.savefig('figures/drugs.png', dpi=300)
    plt.show()
    plt.clf()
```

[8]:
```
plot_data = pd.read_csv('drug_consumption.csv')
mpl.rcParams['axes.linewidth'] = 0.5
```

[9]:
```
# label encode categorical variables
for column in plot_data.loc[:,'Alcohol':'VSA']:
    # get label encoding for column
    plot_data[column] = plot_data[column].astype('category').cat.codes
    # convert column to numeric type
    plot_data[column] = plot_data[column].astype('int32')
```

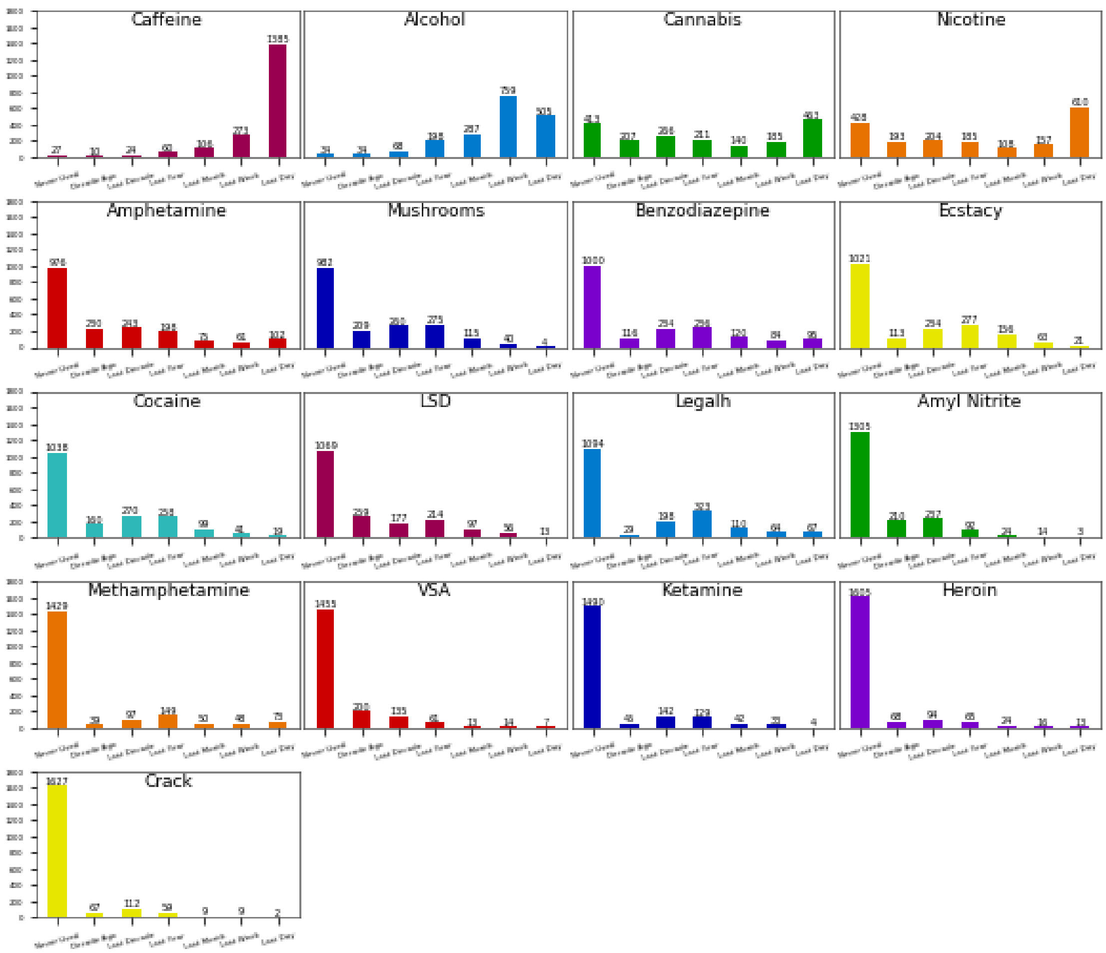
[10]:

```
# drop fake drug
del plot_data['Semer']
# drop ID variable
del plot_data['ID']
# drop chocolate
del plot_data['Choc']
```

[11]:

```
create_usage_subplots(plot_data)
```

```
Caffeine           1858
Alcohol            1851
Cannabis           1472
Nicotine           1457
Amphetamine         909
Mushrooms           903
Benzodiazepine      885
Ecstacy             864
Cocaine             847
LSD                 816
Legalh              791
Amyl Nitrite        580
Methamphetamine     456
VSA                 430
Ketamine            395
Heroin              280
Crack               258
dtype: int64
```




```
<Figure size 432x288 with 0 Axes>
```

[12]:
```python
min_score = plot_data.loc[:,'Nscore':'Cscore'].min().values.min()
plot_data.loc[:,'Nscore':'Cscore'] += abs(min_score)
```

[13]:
```python
# Create drug user columns for illegal drug usage
plot_data['Drug User'] = plot_data.drop(['Alcohol', 'Nicotine','Caff',␣
 ↪'Legalh'], axis=1).apply(is_drug_user, axis=1)
plot_data['Gender']= plot_data['Gender'].apply(gender_map)
plot_data['Country']=plot_data['Country'].apply(country_map)
```

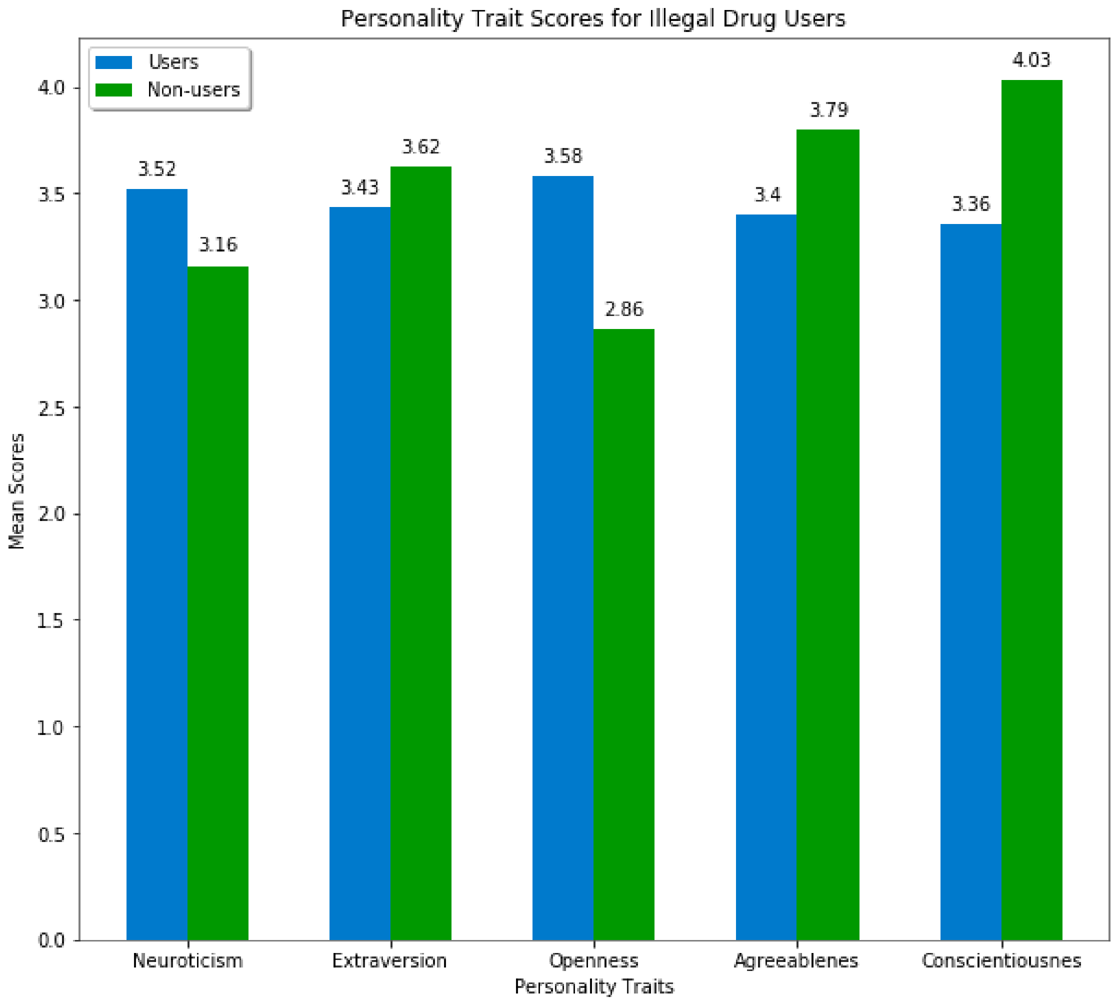
[14]:
```python
# Create dataset for drug users
users = plot_data.loc[plot_data['Drug User'] == True]
# Create dataset for non-drug users
nonusers = plot_data.loc[plot_data['Drug User'] == False]
```

[15]:
```python
# Get data for male drug users
male_users = users.loc[(users['Gender'] == 'Male')]
male_users = male_users.reset_index(drop=True)
```

[16]:
```python
# Get data for female drug users
female_users = users.loc[(users['Gender'] == 'Female')]
female_users = female_users.reset_index(drop=True)
```

[17]:
```python
def create_personality_plot(users, nonusers):
    traits = ['Neuroticism', 'Extraversion', 'Openness',
              'Agreeablenes', 'Conscientiousnes']
    users_scores = users.loc[:,'Nscore':'Cscore'].mean().values
    nonusers_scores = nonusers.loc[:,'Nscore':'Cscore'].mean().values
    width = 0.3
    users_xpos = np.arange(len(users_scores))
    nonusers_xpos = [x + width for x in users_xpos]

    plt.bar(users_xpos, users_scores, width=width, color=['#007acc'],␣
 ↪label='Users')
    plt.bar(nonusers_xpos, nonusers_scores, width=width, color=['#009900'],␣
 ↪label='Non-users')
    ax = plt.gca()
    rects = ax.patches
    for rect, score in zip(rects[:5],users_scores):
        x = rect.get_x() + rect.get_width()/2
        y = rect.get_height() + 0.05
        ax.text(x, y, np.round(score, 2), ha='center',va='bottom', fontsize=10)
    for rect, score in zip(rects[5:],nonusers_scores):
        x = rect.get_x() + rect.get_width()/2
        y = rect.get_height() + 0.05
        ax.text(x, y, np.round(score, 2), ha='center',va='bottom', fontsize=10)
    plt.title('Personality Trait Scores for Illegal Drug Users')
    plt.xlabel('Personality Traits')
    plt.ylabel('Mean Scores')
    plt.xticks(users_xpos+width/2, traits)
    plt.legend(loc='upper left',shadow=True)
    fig = plt.gcf()
    fig.set_size_inches(10,9)
    plt.savefig('figures/traits.png', dpi=300)
    plt.show()
    plt.clf()
```

[18]:
```python
create_personality_plot(users, nonusers)
```



<Figure size 432x288 with 0 Axes>

[19]:

```
traits = users.loc[:,'Nscore':'Cscore'].columns.values
t_stats = []
p_values = []
for trait in traits:
    ␣
 ↪print(trait,'**************************************************************')
    w, p_value = stats.levene( users.loc[:,trait],nonusers.loc[:,trait])
    print('levene p-value: ', p_value)
    if p_value > 0.05:
        t, p_value = stats.ttest_ind(users.loc[:,trait],nonusers.loc[:,trait],␣
 ↪equal_var = True)
    else:
```

```
        t, p_value = stats.ttest_ind(users.loc[:,trait],nonusers.loc[:,trait],␣
↪equal_var = False)

    t_stats.append(t)
    p_values.append(p_value/2.0)
    #p-value / 2.0 for one-tailed test, t > 0 for right-tailed test and t < 0␣
↪for left-tailed test.
    print('t-statistic: ', t, ' p-value: ', p_value/2.0)
```

```
Nscore **************************************************************
levene p-value:  0.007322061691877124
t-statistic:  6.343403033915174  p-value:  2.692405475634671e-10
Escore **************************************************************
levene p-value:  0.0019303812678958126
t-statistic:  -3.4179100269094023  p-value:  0.00034277409057745523
Oscore **************************************************************
levene p-value:  0.0008990864122367897
t-statistic:  13.226093187757758  p-value:  1.8034917246083255e-34
Ascore **************************************************************
levene p-value:  0.27106273015431914
t-statistic:  -6.301675344146927  p-value:  1.8286403859983095e-10
Cscore **************************************************************
levene p-value:  3.331880478169075e-05
t-statistic:  -12.753011706461825  p-value:  1.2411894819567094e-32
```

[20]:
```
illegal_drugs = plot_data.drop(['Alcohol', 'Nicotine','Caff', 'Legalh'],␣
↪axis=1).loc[:, 'Amphet':'VSA']
plot_data['Last Used'] = illegal_drugs.apply(lambda series: series.max(),␣
↪axis=1)
males = plot_data.loc[(plot_data['Gender'] == 'Male')]
males = males.reset_index(drop=True)
males.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 943 entries, 0 to 942
Data columns (total 31 columns):
Age            943 non-null float64
Gender         943 non-null object
Education      943 non-null float64
Country        943 non-null object
Ethnicity      943 non-null float64
Nscore         943 non-null float64
Escore         943 non-null float64
Oscore         943 non-null float64
Ascore         943 non-null float64
Cscore         943 non-null float64
Impulsive      943 non-null float64
```

```
SS            943 non-null float64
Alcohol       943 non-null int32
Amphet        943 non-null int32
Amyl          943 non-null int32
Benzos        943 non-null int32
Caff          943 non-null int32
Cannabis      943 non-null int32
Coke          943 non-null int32
Crack         943 non-null int32
Ecstacy       943 non-null int32
Heroin        943 non-null int32
Ketamine      943 non-null int32
Legalh        943 non-null int32
LSD           943 non-null int32
Meth          943 non-null int32
Mushrooms     943 non-null int32
Nicotine      943 non-null int32
VSA           943 non-null int32
Drug User     943 non-null bool
Last Used     943 non-null int64
dtypes: bool(1), float64(10), int32(17), int64(1), object(2)
memory usage: 159.4+ KB
```

[21]:
```
females = plot_data.loc[(plot_data['Gender'] == 'Female')]
females = females.reset_index(drop=True)
```

[22]:
```
male_freq = males['Last Used'].value_counts().sort_index().values
male_freq
```

[22]:
```
array([ 84,  60,  80,  84,  86, 146, 403])
```

[23]:
```
female_freq = females['Last Used'].value_counts().sort_index().values
female_freq
```

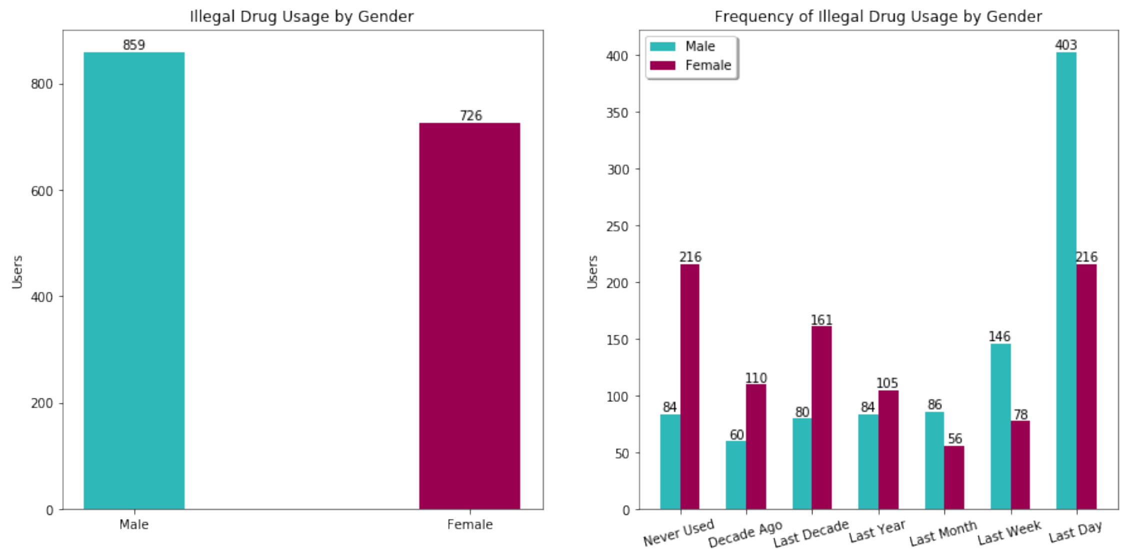
[23]:
```
array([216, 110, 161, 105,  56,  78, 216])
```

[24]:
```
def create_gender_freq_plot(ax, male_freq, female_freq):
    usage_texts = ['Never Used','Decade Ago','Last Decade',
                   'Last Year', 'Last Month',
                   'Last Week', 'Last Day']
    width = 0.3
    males_xpos = np.arange(len(male_freq))
    females_xpos = [x + width for x in males_xpos]
    ax.bar(males_xpos, male_freq, width=width, color=['#2eb8b8'], label='Male')
    ax.bar(females_xpos, female_freq, width=width, color=['#99004f'],
 ↪label='Female')
    rects = ax.patches
```

```python
    for rect, freq in zip(rects[:7], male_freq):
        x = rect.get_x() + rect.get_width()/2
        y = rect.get_height() + 0.05
        ax.text(x, y, freq, ha='center',va='bottom', fontsize=10)
    for rect, freq in zip(rects[7:], female_freq):
        x = rect.get_x() + rect.get_width()/2
        y = rect.get_height() + 0.05
        ax.text(x, y, freq, ha='center',va='bottom', fontsize=10)
    ax.tick_params(axis ='x', labelrotation=15)
    ax.legend(loc='upper left',shadow=True)
    ax.set_xticks(males_xpos+width/2)
    ax.set_xticklabels(usage_texts)
    ax.set_ylabel('Users')
    ax.set_title('Frequency of Illegal Drug Usage by Gender')
```

[25]:
```python
def create_gender_plot(ax, male_users, female_users):
    xpos = np.arange(2)
    bars = [len(male_users.index), len(female_users.index)]

    ax.bar(xpos, bars, width=0.3, color=['#2eb8b8','#99004f'], align='center')

    rects = ax.patches
    for rect, score in zip(rects,bars):
        x = rect.get_x() + rect.get_width()/2
        y = rect.get_height() + 0.05
        ax.text(x, y, np.round(score, 2), ha='center',va='bottom', fontsize=10)

    ax.set_xticks(xpos)
    ax.set_xticklabels(['Male', 'Female'])
    ax.set_title('Illegal Drug Usage by Gender')
    ax.set_ylabel('Users')
```

[26]:
```python
fig, (ax1, ax2) = plt.subplots(1, 2, sharey=False, figsize=(15,7))
create_gender_plot(ax1, male_users, female_users)
create_gender_freq_plot(ax2, male_freq, female_freq)
fig.savefig('figures/gender_freq.png', dpi=300)
plt.show()
plt.clf()
```




```
<Figure size 432x288 with 0 Axes>
```

[40]:

```
train_data = train_data.reset_index(drop=True)
test_data = test_data.reset_index(drop=True)
```

[41]:

```
train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1319 entries, 0 to 1318
Data columns (total 32 columns):
ID            1319 non-null int64
Age           1319 non-null float64
Gender        1319 non-null float64
Education     1319 non-null float64
Country       1319 non-null float64
Ethnicity     1319 non-null float64
Nscore        1319 non-null float64
Escore        1319 non-null float64
Oscore        1319 non-null float64
Ascore        1319 non-null float64
Cscore        1319 non-null float64
Impulsive     1319 non-null float64
SS            1319 non-null float64
Alcohol       1319 non-null object
Amphet        1319 non-null object
Amyl          1319 non-null object
Benzos        1319 non-null object
Caff          1319 non-null object
```

```
Cannabis     1319 non-null object
Choc         1319 non-null object
Coke         1319 non-null object
Crack        1319 non-null object
Ecstacy      1319 non-null object
Heroin       1319 non-null object
Ketamine     1319 non-null object
Legalh       1319 non-null object
LSD          1319 non-null object
Meth         1319 non-null object
Mushrooms    1319 non-null object
Nicotine     1319 non-null object
Semer        1319 non-null object
VSA          1319 non-null object
dtypes: float64(12), int64(1), object(19)
memory usage: 329.9+ KB
```

[42]:
```
# Drop ID, Chocolate, the fake drug Semer, and legal substances
train_data.drop(['ID', 'Choc', 'Semer', 'Alcohol', 'Nicotine',␣
 ↪'Caff','Legalh'], axis=1, inplace=True)
test_data.drop(['ID', 'Choc', 'Semer',  'Alcohol', 'Nicotine',␣
 ↪'Caff','Legalh'], axis=1, inplace=True)
```

[43]:
```
for column in train_data.loc[:,'Amphet':]:
    # get label encoding for column
    train_data[column] = train_data[column].astype('category').cat.codes.
 ↪astype('int32')
    test_data[column] = test_data[column].astype('category').cat.codes.
 ↪astype('int32')
```

[44]:
```
# Combine illegal drug usage into a single boolean variable
train_data['Drug User'] = train_data.apply(is_drug_user, axis=1)
test_data['Drug User'] = test_data.apply(is_drug_user, axis=1)
train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1319 entries, 0 to 1318
Data columns (total 26 columns):
Age          1319 non-null float64
Gender       1319 non-null float64
Education    1319 non-null float64
Country      1319 non-null float64
Ethnicity    1319 non-null float64
Nscore       1319 non-null float64
Escore       1319 non-null float64
Oscore       1319 non-null float64
Ascore       1319 non-null float64
Cscore       1319 non-null float64
```

```
Impulsive    1319 non-null float64
SS           1319 non-null float64
Amphet       1319 non-null int32
Amyl         1319 non-null int32
Benzos       1319 non-null int32
Cannabis     1319 non-null int32
Coke         1319 non-null int32
Crack        1319 non-null int32
Ecstacy      1319 non-null int32
Heroin       1319 non-null int32
Ketamine     1319 non-null int32
LSD          1319 non-null int32
Meth         1319 non-null int32
Mushrooms    1319 non-null int32
VSA          1319 non-null int32
Drug User    1319 non-null bool
dtypes: bool(1), float64(12), int32(13)
memory usage: 192.0 KB
```

[45]:
```
X_train = train_data.loc[:,'Age':'SS']
y_train = train_data['Drug User']
```

[46]:
```
# Feature selection
sel = SelectFromModel(RandomForestClassifier(random_state=0), threshold=0.05)
sel.fit(X_train, y_train)
selected_feat= X_train.columns[(sel.get_support())]
print(selected_feat.values)
```

```
['Age' 'Education' 'Country' 'Nscore' 'Escore' 'Oscore' 'Ascore' 'Cscore'
 'Impulsive' 'SS']
```

```
/Users/manasakandimalla/opt/anaconda3/lib/python3.7/site-
packages/sklearn/ensemble/forest.py:245: FutureWarning: The default value of
n_estimators will change from 10 in version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

[47]:
```
# Grid search for random forest (30,240 base estimators)
max_features = ['auto', 'sqrt']
max_depth = [int(x) for x in np.linspace(5, 100, num = 20)] + [None]
n_estimators = [int(x) for x in np.linspace(start = 100, stop = 1000, num = 20)]
min_samples_split = [2, 5, 10]
min_samples_leaf = [1, 2, 4]
criterion=['entropy','gini']
bootstrap = [True, False]
grid = {'max_features': max_features,
        'max_depth': max_depth,
        'n_estimators': n_estimators,
        'min_samples_split': min_samples_split,
```

```
        'min_samples_leaf': min_samples_leaf,
        'criterion': criterion,
        'bootstrap': bootstrap}
```

[48]:
```
# Base classifier used for grid search
forest = RandomForestClassifier(random_state=0)
# Use 100 iterations and 10-fold cross-validation, using 4 cores
forest_grid_search = RandomizedSearchCV(estimator = forest, param_distributions␣
 ↪= grid,
                              n_iter = 100, cv = 10, verbose=2,␣
 ↪random_state=0, n_jobs = 4)
```

[49]:
```
# Use features selected for training and testing
X_train = train_data[selected_feat.values]
y_train = train_data['Drug User']

X_test = test_data[selected_feat.values]
y_test = test_data['Drug User']
```

[50]:
```
# Fit features selected to grid search (slow process)
forest_grid_search.fit(X_train, y_train)
```

```
Fitting 10 folds for each of 100 candidates, totalling 1000 fits

[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=4)]: Done  33 tasks      | elapsed:    9.2s
[Parallel(n_jobs=4)]: Done 154 tasks      | elapsed:   39.4s
[Parallel(n_jobs=4)]: Done 357 tasks      | elapsed:  1.6min
[Parallel(n_jobs=4)]: Done 640 tasks      | elapsed:  2.9min
[Parallel(n_jobs=4)]: Done 1000 out of 1000 | elapsed:  4.2min finished
```

[50]:
```
RandomizedSearchCV(cv=10, error_score='raise-deprecating',
                   estimator=RandomForestClassifier(bootstrap=True,
                                                    class_weight=None,
                                                    criterion='gini',
                                                    max_depth=None,
                                                    max_features='auto',
                                                    max_leaf_nodes=None,
                                                    min_impurity_decrease=0.0,
                                                    min_impurity_split=None,
                                                    min_samples_leaf=1,
                                                    min_samples_split=2,
min_weight_fraction_leaf=0.0,
                                                    n_estimators='warn',
                                                    n_jobs=None,
                                                    oob_s…
                                          'max_depth': [5, 10, 15, 20, 25, 30, 35,
```

```
                                                        40, 45, 50, 55, 60, 65,
                                                        70, 75, 80, 85, 90, 95,
                                                        100, None],
                                         'max_features': ['auto', 'sqrt'],
                                         'min_samples_leaf': [1, 2, 4],
                                         'min_samples_split': [2, 5, 10],
                                         'n_estimators': [100, 147, 194, 242,
                                                          289, 336, 384, 431,
                                                          478, 526, 573, 621,
                                                          668, 715, 763, 810,
                                                          857, 905, 952, 1000]},
                   pre_dispatch='2*n_jobs', random_state=0, refit=True,
                   return_train_score=False, scoring=None, verbose=2)
```

[51]:

```
# Store the dictionary of best parameters
params = forest_grid_search.best_params_
params
```

[51]:

```
{'n_estimators': 621,
 'min_samples_split': 10,
 'min_samples_leaf': 4,
 'max_features': 'auto',
 'max_depth': 70,
 'criterion': 'entropy',
 'bootstrap': True}
```

[52]:

```
search_data = pd.DataFrame(forest_grid_search.cv_results_)
search_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 25 columns):
mean_fit_time               100 non-null float64
std_fit_time                100 non-null float64
mean_score_time             100 non-null float64
std_score_time              100 non-null float64
param_n_estimators          100 non-null object
param_min_samples_split     100 non-null object
param_min_samples_leaf      100 non-null object
param_max_features          100 non-null object
param_max_depth             95 non-null object
param_criterion             100 non-null object
param_bootstrap             100 non-null object
params                      100 non-null object
split0_test_score           100 non-null float64
split1_test_score           100 non-null float64
split2_test_score           100 non-null float64
```

```
split3_test_score          100 non-null float64
split4_test_score          100 non-null float64
split5_test_score          100 non-null float64
split6_test_score          100 non-null float64
split7_test_score          100 non-null float64
split8_test_score          100 non-null float64
split9_test_score          100 non-null float64
mean_test_score            100 non-null float64
std_test_score             100 non-null float64
rank_test_score            100 non-null int32
dtypes: float64(16), int32(1), object(8)
memory usage: 19.3+ KB
```

[53]:
```
# Convert estimators column to integer type
search_data['param_n_estimators'] = search_data['param_n_estimators'].
 ↪astype('int32')
```

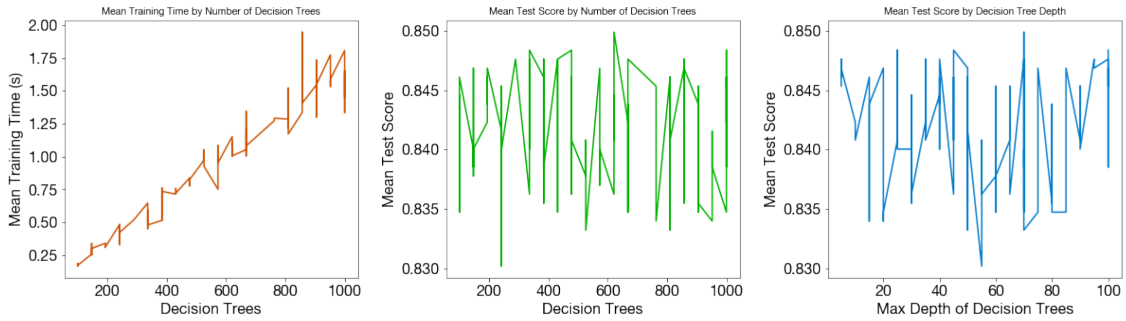
[54]:
```
estimators = search_data[['param_n_estimators', 'mean_fit_time',␣
 ↪'mean_test_score']].sort_values(
    by=['param_n_estimators'])
estimators.head
```

[54]:
```
<bound method NDFrame.head of     param_n_estimators  mean_fit_time
mean_test_score
97                  100       0.174355         0.842305
46                  100       0.168606         0.834723
39                  100       0.166035         0.844579
92                  100       0.186894         0.836240
68                  100       0.171658         0.846096
..                  …            …               …
56                 1000       1.653674         0.838514
43                 1000       1.637751         0.848370
45                 1000       1.596318         0.842305
1                  1000       1.332806         0.846096
75                 1000       1.625391         0.845337

[100 rows x 3 columns]>
```

[55]:
```
max_depths = search_data[['param_max_depth', 'mean_test_score']]
# Remove rows with max depth of None for plotting
max_depths = max_depths[max_depths['param_max_depth'].notnull()]
# Sort values by max depth
max_depths.sort_values(by=['param_max_depth'], inplace=True)
max_depths.reset_index(drop=True, inplace=True)
max_depths.head
```

[55]:
```
<bound method NDFrame.head of     param_max_depth  mean_test_score
0                 5         0.845337
1                 5         0.846096
2                 5         0.847612
3                 5         0.846096
4                 5         0.846854
..              …              …
90              100         0.847612
91              100         0.845337
92              100         0.846854
93              100         0.838514
94              100         0.848370

[95 rows x 2 columns]>
```

[56]:
```python
# Plot visuals for grid search
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, sharex=False, sharey=False)
plt.subplots_adjust(wspace=0.3)
fig.set_size_inches(20, 5)

ax1.plot(estimators['param_n_estimators'], estimators['mean_fit_time'],␣
↪color='#cc5200')
ax1.set_xlabel('Decision Trees')
ax1.set_ylabel('Mean Training Time (s)')
ax1.set_title('Mean Training Time by Number of Decision Trees',␣
↪fontdict={'fontsize':10.5})

ax2.plot(estimators['param_n_estimators'], estimators['mean_test_score'],␣
↪color='#00b300')
ax2.set_xlabel('Decision Trees')
ax2.set_ylabel('Mean Test Score')
ax2.set_title('Mean Test Score by Number of Decision␣
↪Trees',fontdict={'fontsize':10.5})

ax3.plot(max_depths['param_max_depth'], max_depths['mean_test_score'],␣
↪color='#007acc')
ax3.set_xlabel('Max Depth of Decision Trees')
ax3.set_ylabel('Mean Test Score')
ax3.set_title('Mean Test Score by Decision Tree Depth',fontdict={'fontsize':10.
↪5})
plt.savefig('gridsearch.png', dpi=300)
```



[57]:
```
# Get mean cross-validation score using best paramaters from grid search
forest = RandomForestClassifier(n_estimators=params['n_estimators'],␣
↪min_samples_split=params['min_samples_split'],
                                min_samples_leaf=params['min_samples_leaf'],␣
↪max_features=params['max_features'],
                                max_depth=params['max_depth'],␣
↪bootstrap=params['bootstrap'], random_state=0)
```

[58]:
```
scores = cross_val_score(forest, X_train, y_train, scoring='accuracy', cv=10)
```

[59]:
```
print("Mean Accuracy: %f"%(scores.mean()))
```

```
Mean Accuracy: 0.844579
```

[60]:
```
scores = cross_val_score(forest, X_train, y_train, scoring='f1', cv=10)
```

[61]:
```
print("Mean F1 Score: %f"%(scores.mean()))
```

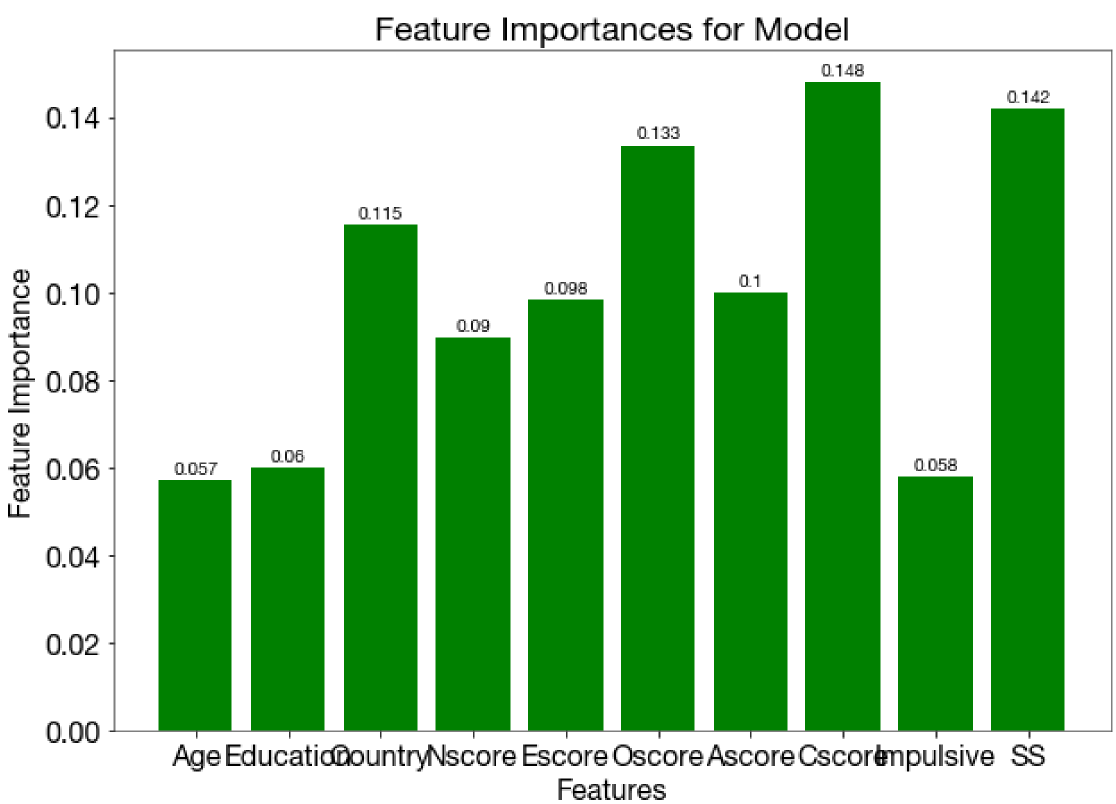
```
Mean F1 Score: 0.913897
```

[62]:
```
forest = RandomForestClassifier(n_estimators=params['n_estimators'],␣
↪min_samples_split=params['min_samples_split'],
                                min_samples_leaf=params['min_samples_leaf'],␣
↪max_features=params['max_features'],
                                max_depth=params['max_depth'],␣
↪bootstrap=params['bootstrap'], random_state=0)
forest.fit(X_train, y_train)
```

[62]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=70, max_features='auto', max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=4, min_samples_split=10,
                       min_weight_fraction_leaf=0.0, n_estimators=621,
                       n_jobs=None, oob_score=False, random_state=0, verbose=0,
                       warm_start=False)
```

[63]:
```
plt.bar(np.arange(len(X_train.columns)), forest.feature_importances_,␣
↪tick_label=X_train.columns.values, color='g')
fig = plt.gcf()
ax = plt.gca()
rects = ax.patches
for rect, importance in zip(rects, forest.feature_importances_):
    x = rect.get_x() + rect.get_width()/2
    y = rect.get_height() + 0.001
    ax.text(x, y, np.round(importance, 3), ha='center',va='bottom', fontsize=10)
fig.set_size_inches(10,7)
plt.ylabel('Feature Importance')
plt.xlabel('Features')
plt.title('Feature Importances for Model')
```

[63]:
```
Text(0.5, 1.0, 'Feature Importances for Model')
```

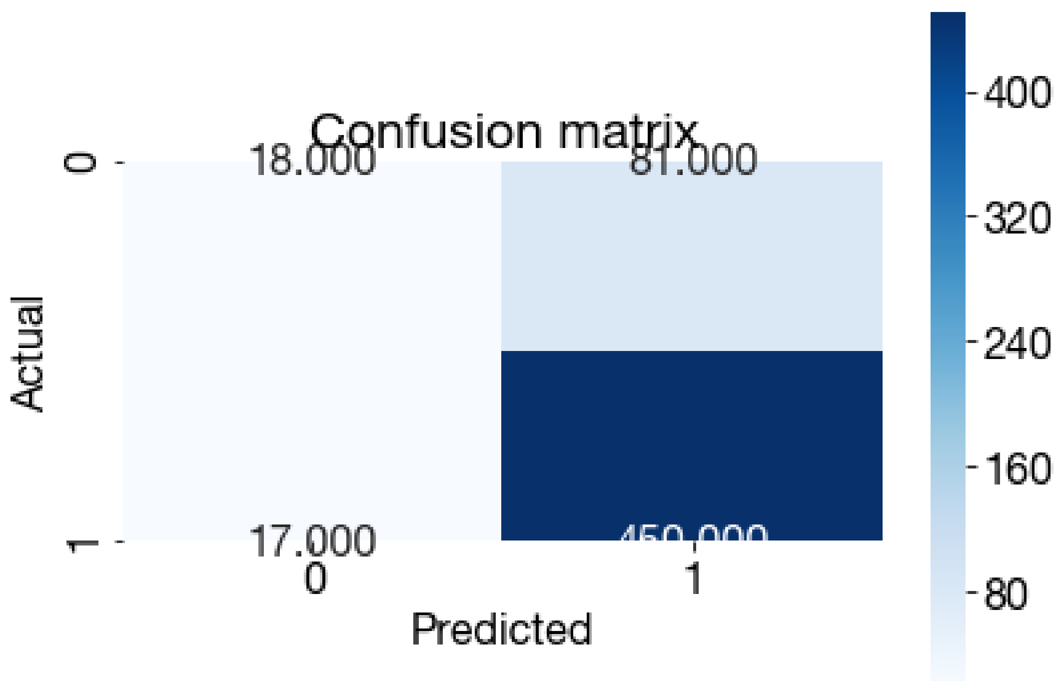


[64]:
```
# Calculate metrics
y_pred = forest.predict(X_test)
accuracy  = metrics.accuracy_score(y_test, y_pred)
error     = 1 - metrics.accuracy_score(y_test, y_pred)
precision = metrics.precision_score(y_test, y_pred, average=None)
```

```
recall    = metrics.recall_score(y_test, y_pred, average=None)
f1_score  = metrics.f1_score(y_test, y_pred, average=None)
print('Accuracy:  ', '%0.3f'%(accuracy))
print('Error:     ', '%0.3f'%(error))
print('Precision: ', precision)
print('Recall:    ', recall)
print('F1 score:  ', f1_score)
```

```
Accuracy:   0.827
Error:      0.173
Precision:  [0.51428571 0.84745763]
Recall:     [0.18181818 0.96359743]
F1 score:   [0.26865672 0.90180361]
```

[65]:
```
conf_matrix = metrics.confusion_matrix(y_test, y_pred)
sns.heatmap(conf_matrix, annot=True, fmt='.3f', square=True, cmap=plt.cm.Blues)
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Confusion matrix')
plt.tight_layout()
plt.savefig('figures/random_forest_matrix.png', dpi=300)
```



[ ]:

[66]:
```
data = pd.read_csv("drug_consumption.csv")
data.head()
```

[66]:
```
   ID      Age   Gender  Education  Country  Ethnicity   Nscore   Escore  \
0   1  0.49788  0.48246   -0.05921  0.96082    0.12600  0.31287 -0.57545
1   2 -0.07854 -0.48246    1.98437  0.96082   -0.31685 -0.67825  1.93886
2   3  0.49788 -0.48246   -0.05921  0.96082   -0.31685 -0.46725  0.80523
3   4 -0.95197  0.48246    1.16365  0.96082   -0.31685 -0.14882 -0.80615
4   5  0.49788  0.48246    1.98437  0.96082   -0.31685  0.73545 -1.63340

    Oscore   Ascore  … Ecstacy Heroin Ketamine Legalh  LSD Meth  \
0 -0.58331 -0.91699  …     CL0    CL0      CL0    CL0  CL0  CL0
1  1.43533  0.76096  …     CL4    CL0      CL2    CL0  CL2  CL3
2 -0.84732 -1.62090  …     CL0    CL0      CL0    CL0  CL0  CL0
3 -0.01928  0.59042  …     CL0    CL0      CL2    CL0  CL0  CL0
4 -0.45174 -0.30172  …     CL1    CL0      CL0    CL1  CL0  CL0

  Mushrooms Nicotine Semer  VSA
0       CL0      CL2   CL0  CL0
1       CL0      CL4   CL0  CL0
2       CL1      CL0   CL0  CL0
3       CL0      CL2   CL0  CL0
4       CL2      CL2   CL0  CL0

[5 rows x 32 columns]
```

[67]:
```
#Source: https://drugs.laws.com/list-of-illegal-drugs
#Grouping Illegal and Non-Illegal drugs
illegal_drugs = ['Amphet','Coke','Crack','Ecstacy','Heroin','LSD','Mushrooms']
Non_illegal =[i for i in data.columns[13:] if i not in illegal_drugs]
```

[68]:
```
#Stripping all 'CL' from all the drug columns
for drug in data.columns[13:]:
        data[drug] = data[drug].map(lambda x: int(str(x).lstrip('CL')))
```

[69]:
```
cluster_tab = pd.DataFrame(columns=['Type','Single Linkage','Complete␣
 ↪Linkage','K-Means','DBSCAN'])
```

[ ]:
```

```

[70]:
```
#Start with Clusters
#Cluster1: Using all the columns
#Cluster1 with Hierarchial clustering single linkage
```

[71]:
```
variables = data.columns
var_indices = [data.columns.get_loc(variable) for variable in variables]
```

[72]:
```
print(var_indices)
```

```
[0, 1, 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]
```

[73]:
```
#Standardizing the data
x = data.iloc[:,var_indices]
scaler = StandardScaler()
x_scaled = scaler.fit_transform(x)
```

[74]:
```
clustering = linkage(x_scaled,method="single",metric="euclidean")
clusters = fcluster(clustering, 2, criterion = 'maxclust')
clusters = clusters - 1
print(clusters)
```

```
[1 1 1 … 1 1 1]
```

[75]:
```
data['clusters'] = clusters
```

[76]:
```
#Silhouette coefficient
sl_sil = metrics.silhouette_score(x_scaled, data['clusters'], metric =␣
↪"euclidean")
print(sl_sil)
```

```
0.5079507583610615
```

[77]:
```
#Cluster1 with Hierarchial clustering Complete Linkage
```

[78]:
```
clustering = linkage(x_scaled,method="complete",metric="euclidean")
clusters = fcluster(clustering, 2, criterion = 'maxclust')
clusters = clusters - 1
print(clusters)
```

```
[1 1 1 … 1 1 1]
```

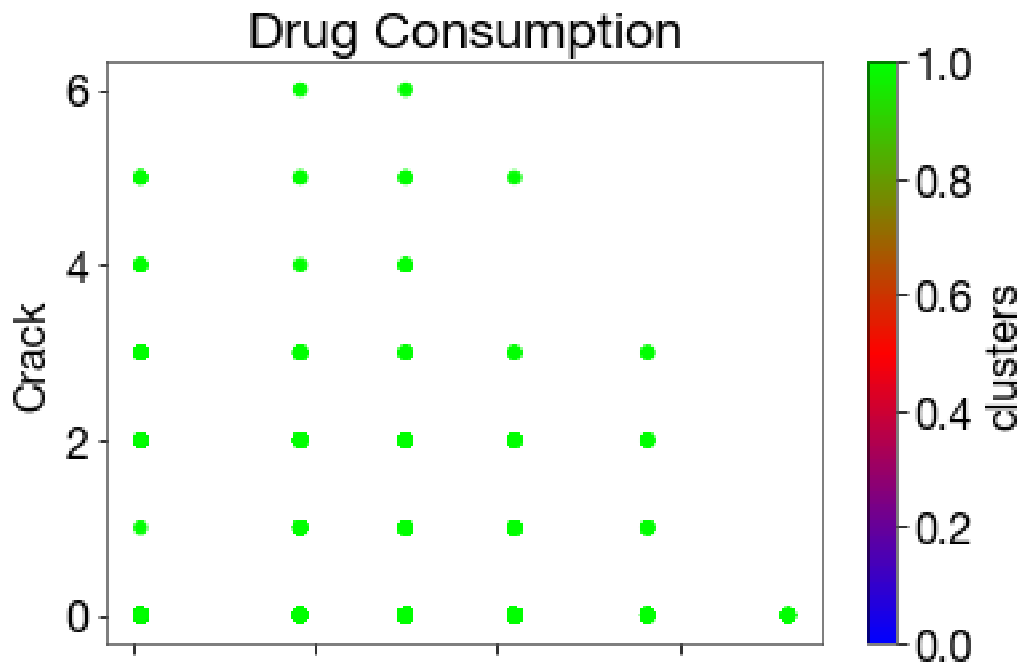
[79]:
```
data['clusters'] = clusters
```

[80]:
```
#Silhouette coefficient
cl_sil = metrics.silhouette_score(x_scaled, data['clusters'], metric =␣
↪"euclidean")
print(cl_sil)
```

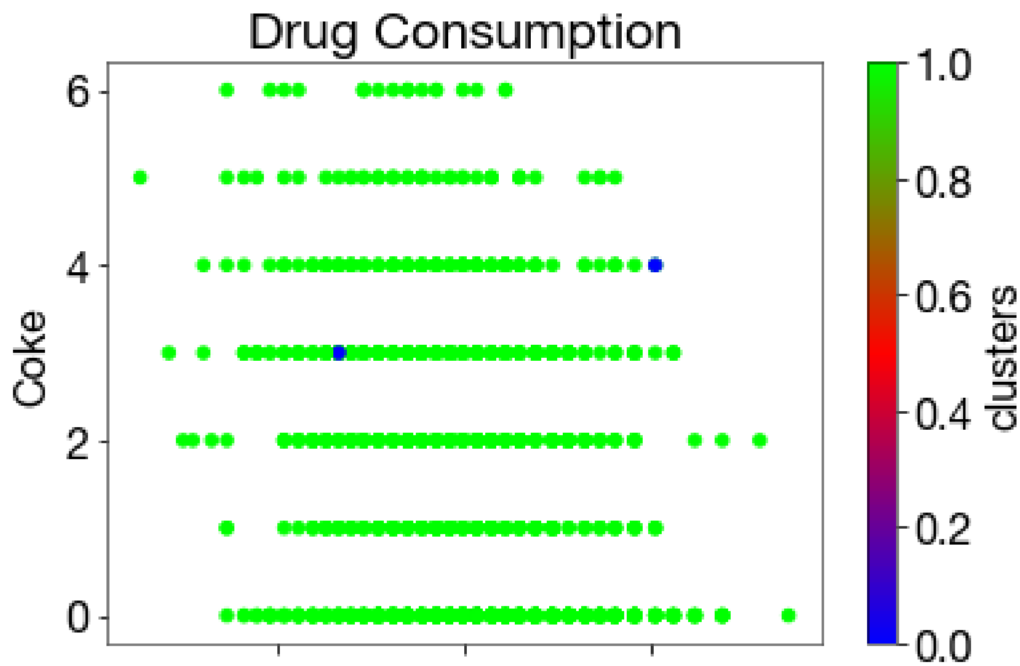
```
0.6629990546490399
```

[81]:
```
ax = data.plot(kind = 'scatter', x = 'Age', y = 'Crack', c = 'clusters',␣
↪colormap = plt.cm.brg)
ax.set(title = 'Drug Consumption', xlabel = 'Age', ylabel = 'Crack')
```

[81]: [Text(0, 0.5, 'Crack'),
 Text(0.5, 0, 'Age'),
 Text(0.5, 1.0, 'Drug Consumption')]

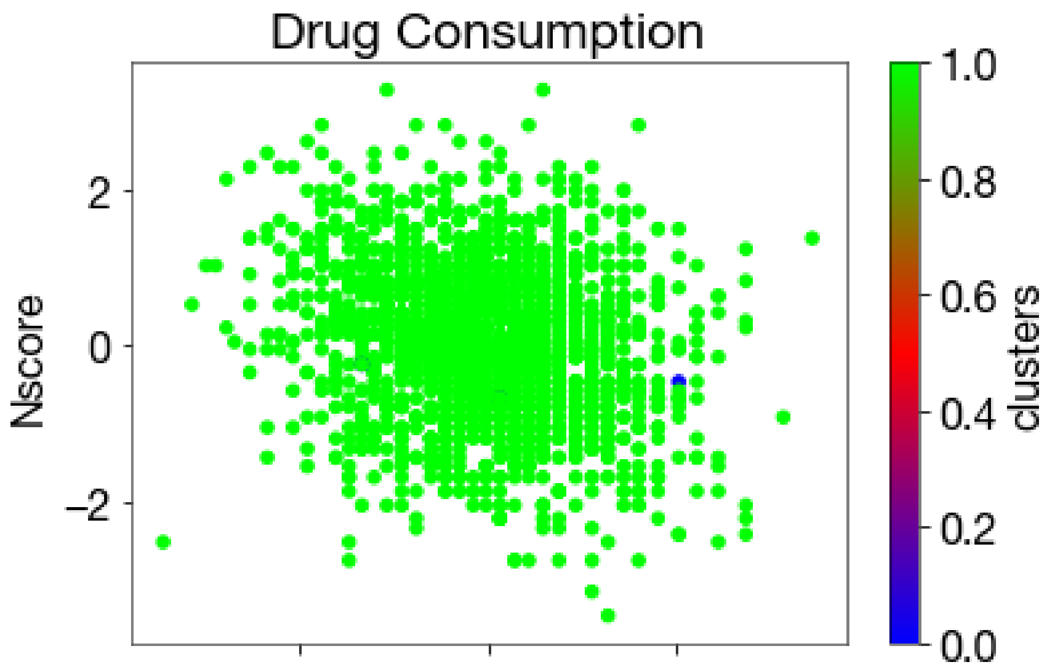


[82]:
```
ax = data.plot(kind = 'scatter', x = 'Ascore', y = 'Coke', c = 'clusters',␣
↪colormap = plt.cm.brg)
ax.set(title = 'Drug Consumption',ylabel = 'Coke', xlabel = 'Ascore')
plt.savefig('figures/cluster1-AscorevsCoke', dpi=300)
```



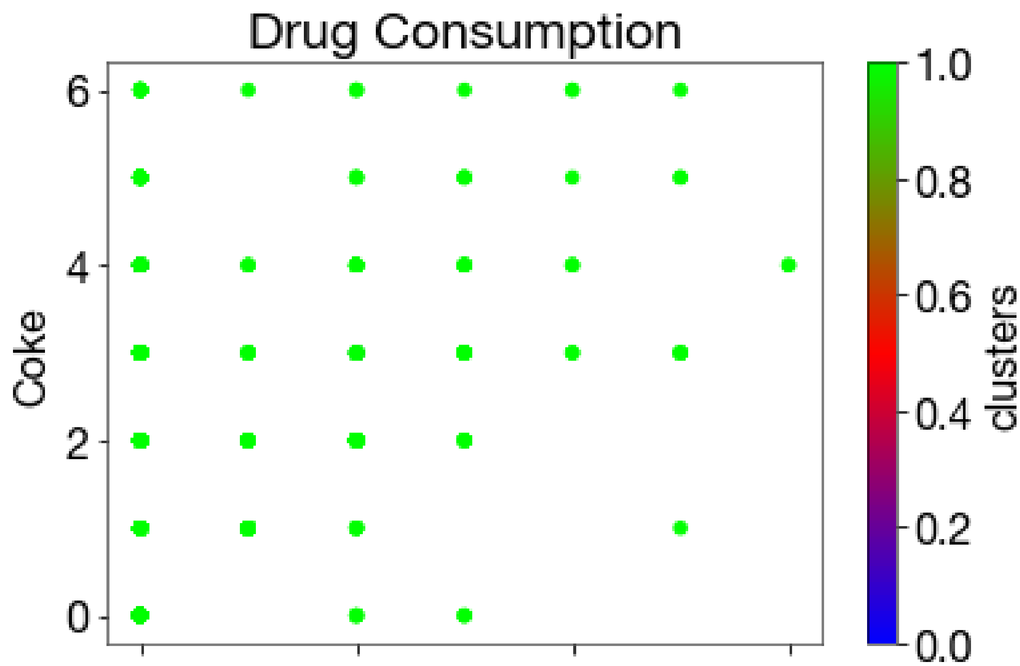
[83]:
```
ax = data.plot(kind = 'scatter', x = 'Ascore', y = 'Nscore', c = 'clusters',␣
 ↪colormap = plt.cm.brg)
ax.set(title = 'Drug Consumption', xlabel = 'Ascore', ylabel = 'Nscore')
```

[83]:
```
[Text(0, 0.5, 'Nscore'),
 Text(0.5, 0, 'Ascore'),
 Text(0.5, 1.0, 'Drug Consumption')]
```



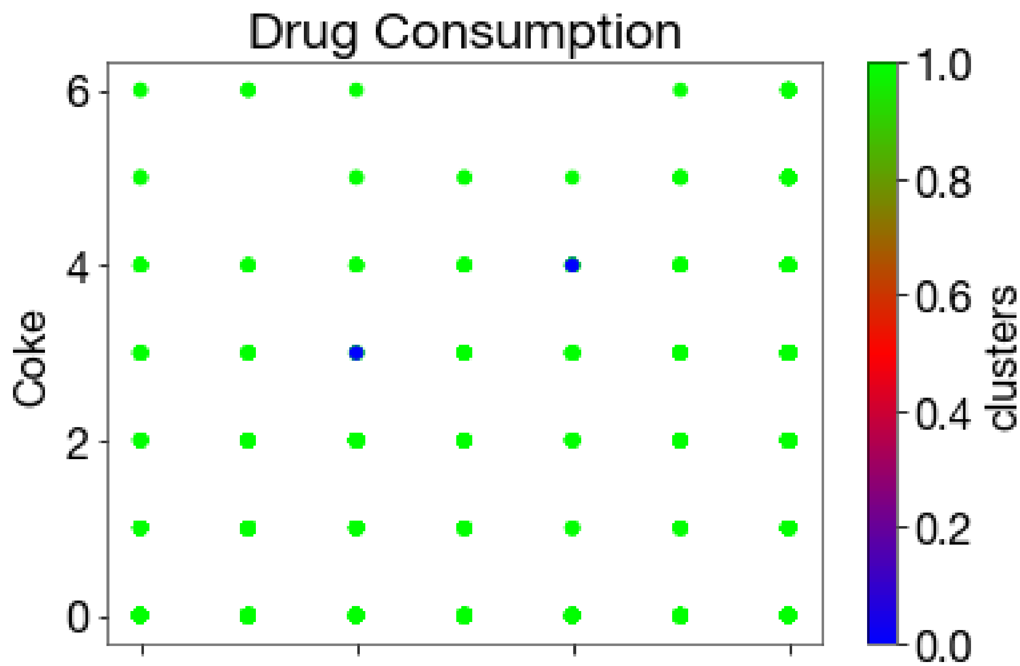

```
[84]: ax = data.plot(kind = 'scatter', x = 'Crack', y = 'Coke', c = 'clusters',␣
      ↪colormap = plt.cm.brg)
      ax.set(title = 'Drug Consumption', xlabel = 'Crack',ylabel = 'Coke')
```

```
[84]: [Text(0, 0.5, 'Coke'),
       Text(0.5, 0, 'Crack'),
       Text(0.5, 1.0, 'Drug Consumption')]
```



[85]:
```
ax = data.plot(kind = 'scatter', x = 'Nicotine', y = 'Coke', c = 'clusters',␣
↪colormap = plt.cm.brg)
ax.set(title = 'Drug Consumption', xlabel = 'Nicotine', ylabel = 'Coke')
```

[85]:
```
[Text(0, 0.5, 'Coke'),
 Text(0.5, 0, 'Nicotine'),
 Text(0.5, 1.0, 'Drug Consumption')]
```



[86]:
```
#Cluster1 with K-Means
```

[87]:
```
clustering = KMeans(n_clusters = 2, init = 'random', n_init = 1, random_state =␣
↪0).fit(x_scaled)
clusters = clustering.labels_
print(clusters)
data['clusters'] = clusters
```

```
[0 1 0 … 1 1 1]
```

[88]:
```
#Silhouette coefficient
km_sil = metrics.silhouette_score(x_scaled, data['clusters'], metric =␣
↪"euclidean")
print(km_sil)
```

```
0.18301116784616245
```

[89]:
```
#Cluster1 with DBSCAN
```

[90]:
```
clustering = DBSCAN(eps = 2, min_samples = 4, metric = "euclidean").
↪fit(x_scaled)
clusters = clustering.labels_
data['clusters'] = clusters
```

[91]:
```
#Silhouette coefficient
db_sil = metrics.silhouette_score(x_scaled, data['clusters'], metric =␣
 ↪"euclidean")
print(db_sil)
```

-0.1898724108691829

[92]:
```
# print(metrics.adjusted_rand_score(data['clusters']))
# We cannot use rand index cause we do not have anything to compare the␣
 ↪clusters against
```

[93]:
```
cluster_tab = cluster_tab.append({'Type':'All Features','Single Linkage':␣
 ↪sl_sil, 'Complete Linkage':cl_sil,'K-Means':km_sil,'DBSCAN':
 ↪db_sil},ignore_index=True)
```

[ ]:
```

```

[94]:
```
#Cluster2: Using only Non-illegal drugs
```

[95]:
```
variables = Non_illegal
var_indices = [data.columns.get_loc(variable) for variable in variables]
```

[96]:
```
#Standardizing the data
x = data.iloc[:,var_indices]
scaler = StandardScaler()
x_scaled = scaler.fit_transform(x)
```

[97]:
```
clustering = linkage(x_scaled,method="single",metric="euclidean")
clusters = fcluster(clustering, 2, criterion = 'maxclust')
clusters = clusters - 1
print(clusters)
```

[0 0 0 … 0 0 0]

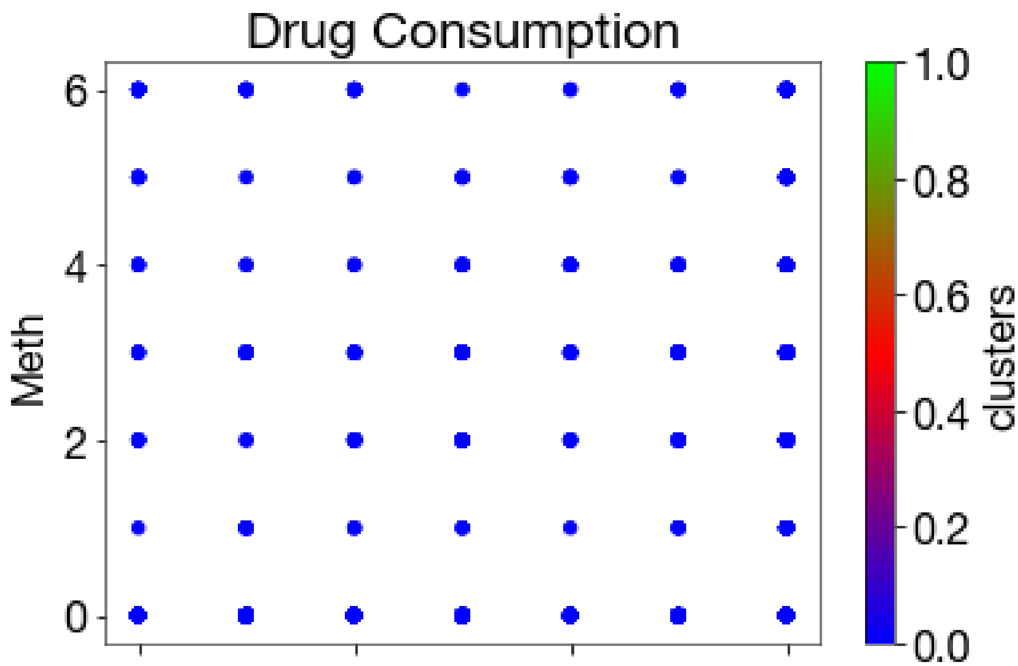
[98]:
```
data['clusters'] = clusters
```

[100]:
```
#Silhouette coefficient
sl_sil = metrics.silhouette_score(x_scaled, data['clusters'], metric =␣
 ↪"euclidean")
print(sl_sil)
```

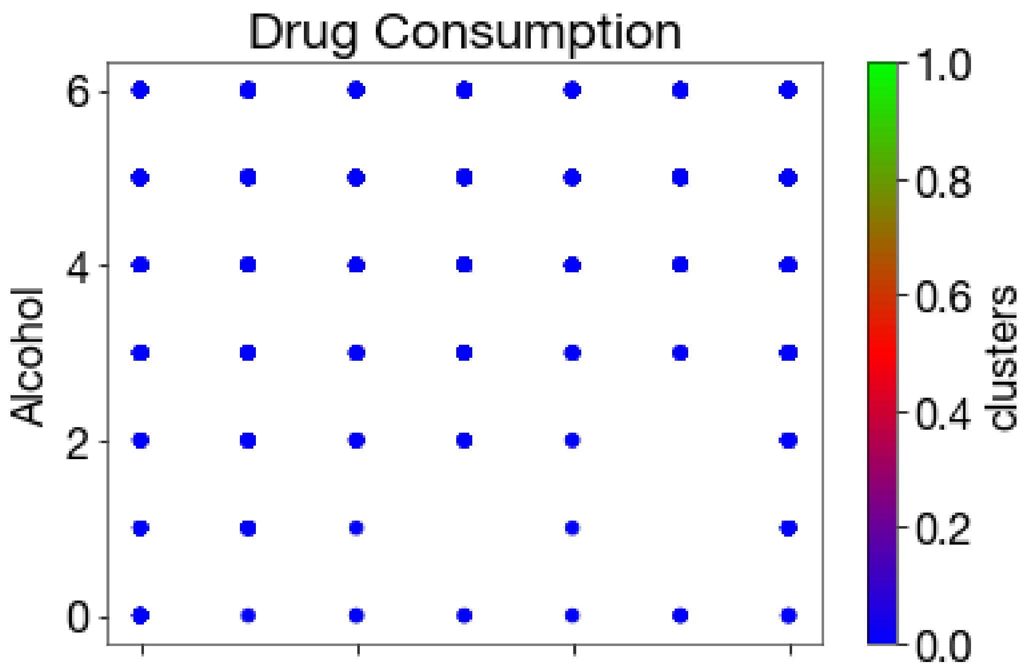
0.8233760416358061

[101]:
```
ax = data.plot(kind = 'scatter', x = 'Nicotine', y = 'Meth', c = 'clusters',␣
 ↪colormap = plt.cm.brg)
ax.set(title = 'Drug Consumption', xlabel = 'Nicotine', ylabel = 'Meth')
```

[101]: [Text(0, 0.5, 'Meth'),
 Text(0.5, 0, 'Nicotine'),
 Text(0.5, 1.0, 'Drug Consumption')]

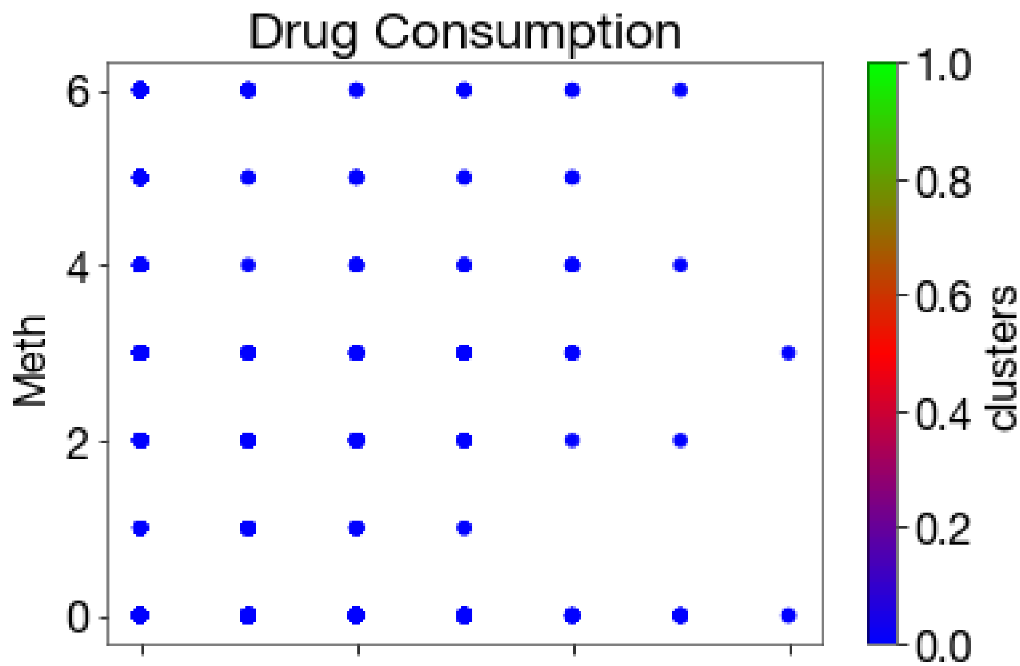



```
[102]: ax = data.plot(kind = 'scatter', x = 'Nicotine', y = 'Alcohol', c = 'clusters',␣
        ↪colormap = plt.cm.brg)
       ax.set(title = 'Drug Consumption', xlabel = 'Nicotine', ylabel = 'Alcohol')
       plt.savefig('figures/cluster2-NicotineVsAlcohol.png', dpi=300)
```




```
[103]: ax = data.plot(kind = 'scatter', x = 'Amyl', y = 'Meth', c = 'clusters',␣
        ↪colormap = plt.cm.brg)
       ax.set(title = 'Drug Consumption', xlabel = 'Amyl', ylabel = 'Meth')
```

```
[103]: [Text(0, 0.5, 'Meth'),
        Text(0.5, 0, 'Amyl'),
        Text(0.5, 1.0, 'Drug Consumption')]
```



[104]:
```
#Cluster2 with complete Linkage
```

[105]:
```
clustering = linkage(x_scaled,method="complete",metric="euclidean")
clusters = fcluster(clustering, 2, criterion = 'maxclust')
clusters = clusters - 1
print(clusters)
```

```
[1 1 1 … 1 1 1]
```

[106]:
```
data['clusters'] = clusters
```

[107]:
```
#Silhouette coefficient
cl_sil = metrics.silhouette_score(x_scaled, data['clusters'], metric =␣
 ↪"euclidean")
print(cl_sil)
```

```
0.7931811284335685
```

[108]:
```
#Cluster2 with K-Means
```

[109]:
```
clustering = KMeans(n_clusters = 2, init = 'random', n_init = 1, random_state =␣
 ↪0).fit(x_scaled)
clusters = clustering.labels_
print(clusters)
data['clusters'] = clusters
```

```
[0 1 0 … 1 1 1]
```

[110]:
```
#Silhouette coefficient
km_sil = metrics.silhouette_score(x_scaled, data['clusters'], metric =␣
↪"euclidean")
print(km_sil)
```

```
0.22503062682481748
```

[111]:
```
#Cluster2 with DBSCAN
```

[112]:
```
clustering = DBSCAN(eps = 2, min_samples = 3, metric = "euclidean").
↪fit(x_scaled)
clusters = clustering.labels_
data['clusters'] = clusters
```

[113]:
```
#Silhouette coefficient
db_sil = metrics.silhouette_score(x_scaled, data['clusters'], metric =␣
↪"euclidean")
print(db_sil)
```

```
0.09401786652049447
```

[114]:
```
cluster_tab = cluster_tab.append({'Type':'Non_Illegal','Single Linkage':␣
↪sl_sil, 'Complete Linkage':cl_sil,'K-Means':km_sil,'DBSCAN':
↪db_sil},ignore_index=True)
```

[ ]:
```

```

[115]:
```
#Cluster3: Using only Illegal drugs
```

[116]:
```
variables = illegal_drugs
var_indices = [data.columns.get_loc(variable) for variable in variables]
print(var_indices)
```

```
[14, 20, 21, 22, 23, 26, 28]
```

[117]:
```
#Standardizing the data
x = data.iloc[:,var_indices]
scaler = StandardScaler()
x_scaled = scaler.fit_transform(x)
```

[118]:
```
#Cluster3 using Hierarchial clustering Single Linkage
clustering = linkage(x_scaled,method="single",metric="euclidean")
clusters = fcluster(clustering, 2, criterion = 'maxclust')
clusters = clusters - 1
print(clusters)
```

[0 0 0 … 0 0 0]

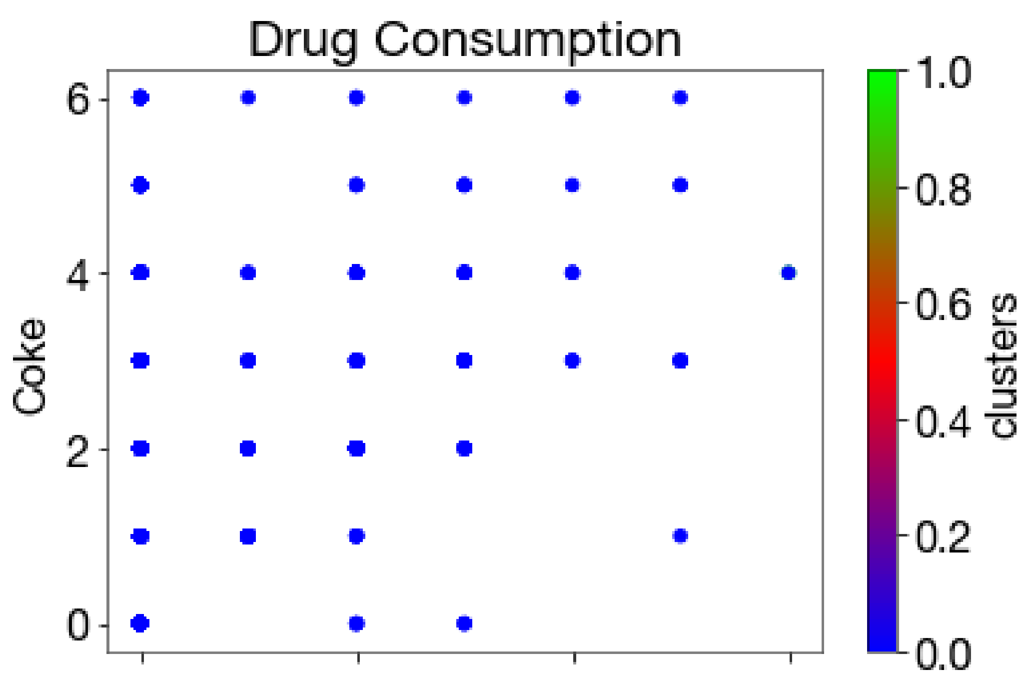
[119]:
```
data['clusters'] = clusters
```

[120]:
```
#Silhouette coefficient
sl_sil = metrics.silhouette_score(x_scaled, data['clusters'], metric =␣
↪"euclidean")
print(sl_sil)
```

0.6082079307311252

[121]:
```
ax = data.plot(kind = 'scatter', x = 'Crack', y = 'Coke', c = 'clusters',␣
↪colormap = plt.cm.brg)
ax.set(title = 'Drug Consumption', xlabel = 'Crack', ylabel = 'Coke')
```

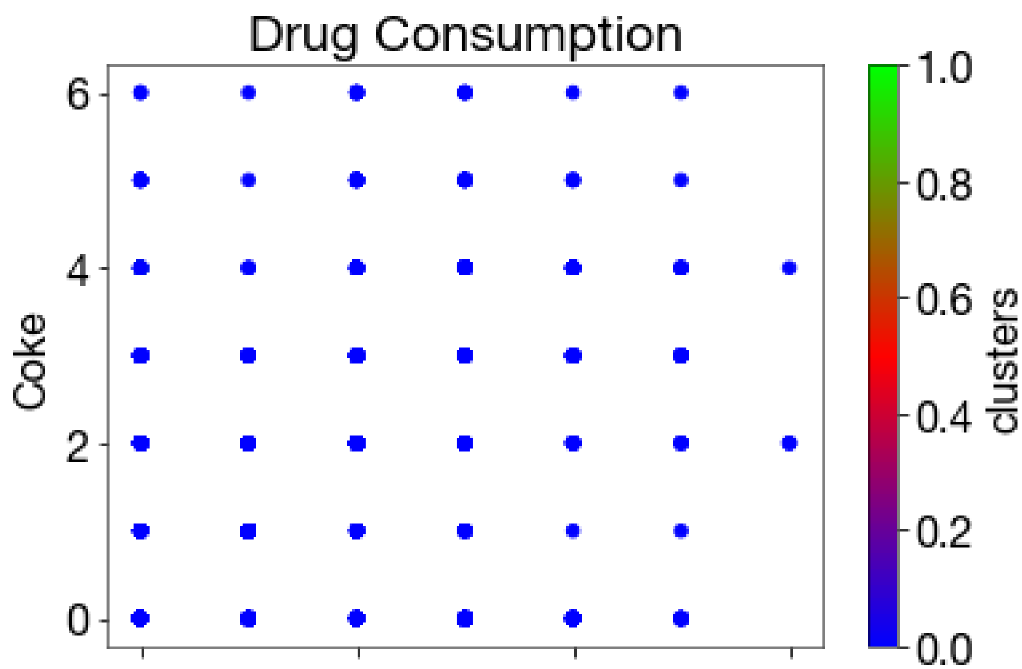
[121]:
```
[Text(0, 0.5, 'Coke'),
 Text(0.5, 0, 'Crack'),
 Text(0.5, 1.0, 'Drug Consumption')]
```



[122]:
```
ax = data.plot(kind = 'scatter', x = 'Mushrooms', y = 'Coke', c = 'clusters',␣
↪colormap = plt.cm.brg)
ax.set(title = 'Drug Consumption', xlabel = 'Mushrooms', ylabel = 'Coke')
```

[122]:
```
[Text(0, 0.5, 'Coke'),
 Text(0.5, 0, 'Mushrooms'),
```

```
Text(0.5, 1.0, 'Drug Consumption')]
```



[123]:
```
#Cluster3 using Hierarchial clustering Complete Linkage
```

[124]:
```
clustering = linkage(x_scaled,method="complete",metric="euclidean")
clusters = fcluster(clustering, 2, criterion = 'maxclust')
clusters = clusters - 1
print(clusters)
```

```
[1 1 1 … 1 1 1]
```

[125]:
```
data['clusters'] = clusters
```

[126]:
```
#Silhouette coefficient
cl_sil = metrics.silhouette_score(x_scaled, data['clusters'], metric =␣
↪"euclidean")
print(cl_sil)
```

```
0.5415889060633275
```

[127]:
```
#Cluster3 with K-Means
```

[128]:
```
clustering = KMeans(n_clusters = 2, init = 'random', n_init = 1, random_state =␣
↪0).fit(x_scaled)
clusters = clustering.labels_
```

```
print(clusters)
data['clusters'] = clusters
```

```
[0 1 0 … 1 1 1]
```

[129]:
```
#Silhouette coefficient
km_sil = metrics.silhouette_score(x_scaled, data['clusters'], metric =␣
↪"euclidean")
print(km_sil)
```

```
0.4487703444541595
```

[130]:
```
#Cluster3 with DBSCAN
```

[131]:
```
clustering = DBSCAN(eps = 2, min_samples = 3, metric = "euclidean").
↪fit(x_scaled)
clusters = clustering.labels_
data['clusters'] = clusters
```

[132]:
```
#Silhouette coefficient
db_sil = metrics.silhouette_score(x_scaled, data['clusters'], metric =␣
↪"euclidean")
print(db_sil)
```

```
0.5160039017438653
```

[133]:
```
cluster_tab = cluster_tab.append({'Type':'Illegal','Single Linkage': sl_sil,␣
↪'Complete Linkage':cl_sil,'K-Means':km_sil,'DBSCAN':
↪db_sil},ignore_index=True)
```

[ ]:
```

```

[134]:
```
#cluster4: Using only the drugs
```

[135]:
```
variables = illegal_drugs + Non_illegal
var_indices = [data.columns.get_loc(variable) for variable in variables]
print(var_indices)
```

```
[14, 20, 21, 22, 23, 26, 28, 13, 15, 16, 17, 18, 19, 24, 25, 27, 29, 30, 31]
```

[136]:
```
#Standardizing the data
x = data.iloc[:,var_indices]
scaler = StandardScaler()
x_scaled = scaler.fit_transform(x)
```

[137]:
```
clustering = linkage(x_scaled,method="single",metric="euclidean")
clusters = fcluster(clustering, 2, criterion = 'maxclust')
```

```
clusters = clusters - 1
print(clusters)
```

```
[0 0 0 … 0 0 0]
```

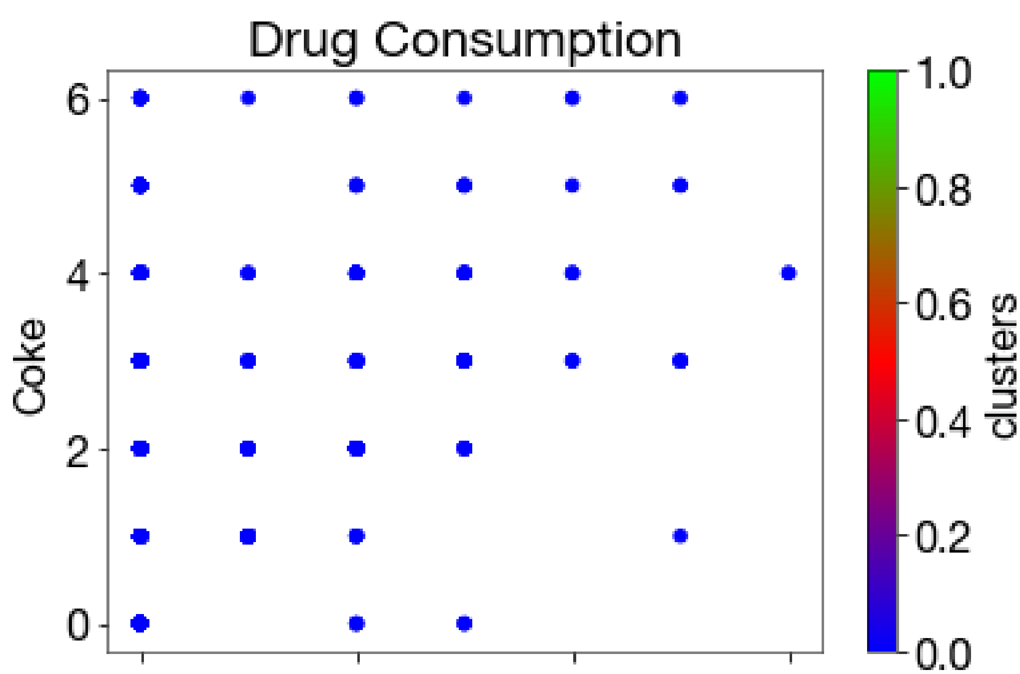
[138]:
```
data['clusters'] = clusters
```

[139]:
```
#Silhouette coefficient
sl_sil = metrics.silhouette_score(x_scaled, data['clusters'], metric =␣
↪"euclidean")
print(sl_sil)
```

```
0.7799966615627284
```

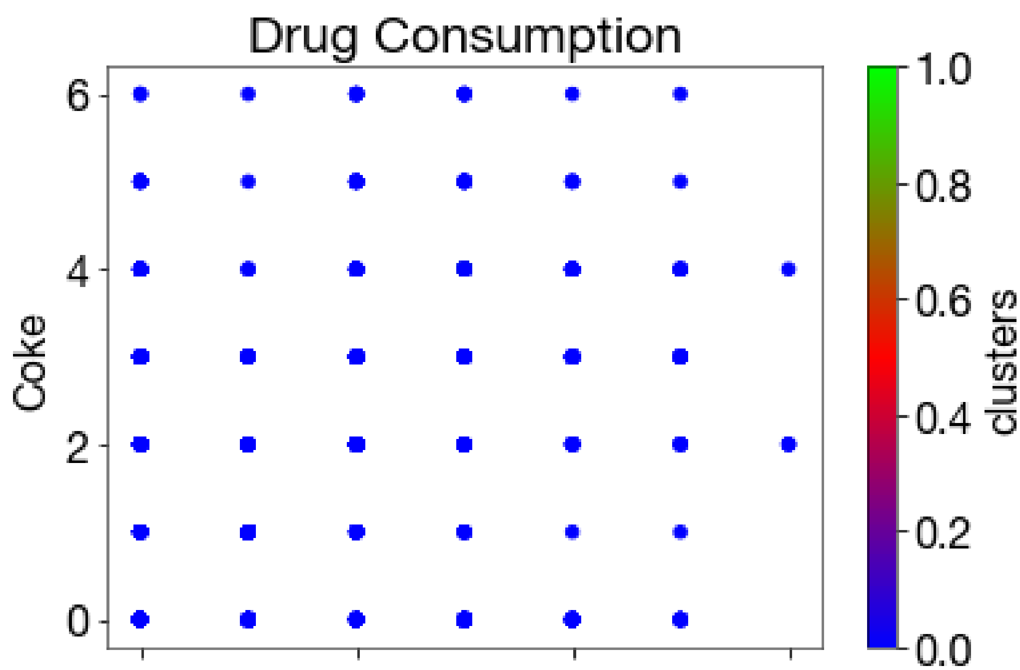
[140]:
```
ax = data.plot(kind = 'scatter', x = 'Crack', y = 'Coke', c = 'clusters',␣
↪colormap = plt.cm.brg)
ax.set(title = 'Drug Consumption', xlabel = 'Crack', ylabel = 'Coke')
```

[140]:
```
[Text(0, 0.5, 'Coke'),
 Text(0.5, 0, 'Crack'),
 Text(0.5, 1.0, 'Drug Consumption')]
```



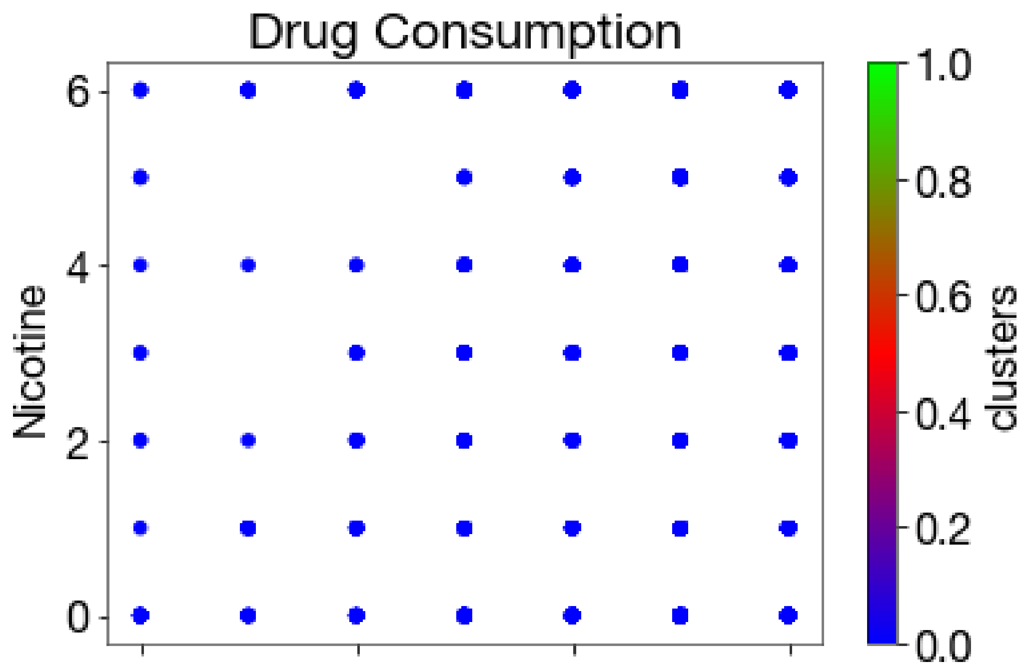
[141]:
```
ax = data.plot(kind = 'scatter', x = 'Mushrooms', y = 'Coke', c = 'clusters',␣
↪colormap = plt.cm.brg)
ax.set(title = 'Drug Consumption', xlabel = 'Mushrooms', ylabel = 'Coke')
```

[141]: [Text(0, 0.5, 'Coke'),
 Text(0.5, 0, 'Mushrooms'),
 Text(0.5, 1.0, 'Drug Consumption')]



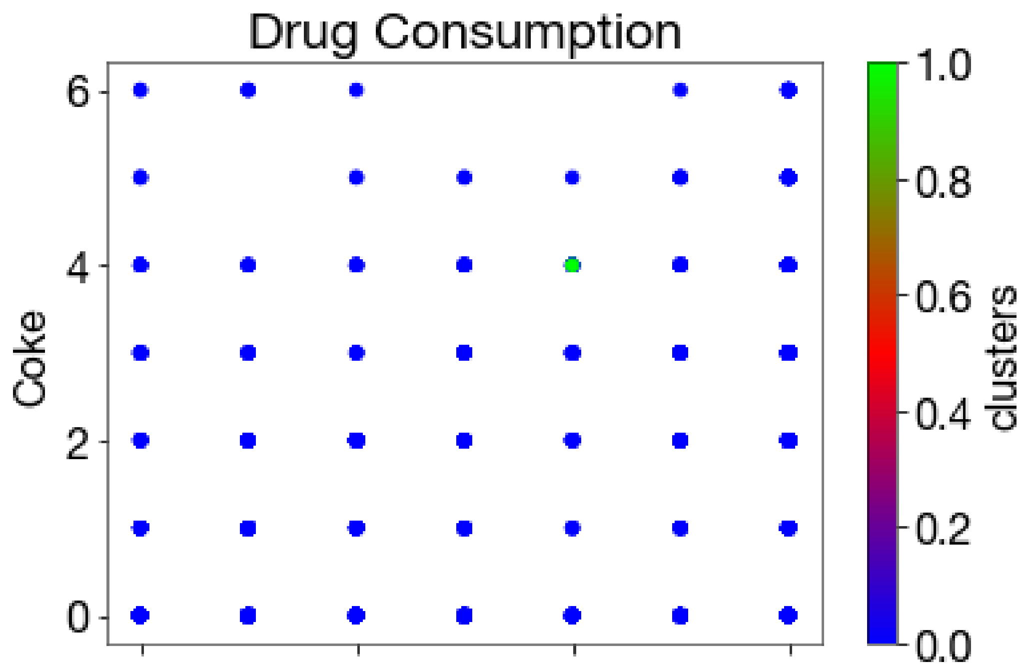

```
[142]: ax = data.plot(kind = 'scatter', x = 'Alcohol', y = 'Nicotine', c = 'clusters',␣
    ↪colormap = plt.cm.brg)
ax.set(title = 'Drug Consumption', xlabel = 'Alcohol', ylabel = 'Nicotine')
```

[142]: [Text(0, 0.5, 'Nicotine'),
 Text(0.5, 0, 'Alcohol'),
 Text(0.5, 1.0, 'Drug Consumption')]




```
[143]: ax = data.plot(kind = 'scatter', x = 'Nicotine', y = 'Coke', c = 'clusters',␣
        ↪colormap = plt.cm.brg)
       ax.set(title = 'Drug Consumption', xlabel = 'Nicotine', ylabel = 'Coke')
```

```
[143]: [Text(0, 0.5, 'Coke'),
        Text(0.5, 0, 'Nicotine'),
        Text(0.5, 1.0, 'Drug Consumption')]
```



[144]:
```
#Cluster4 using Hierarchial clustering Complete Linkage
```

[145]:
```
clustering = linkage(x_scaled,method="complete",metric="euclidean")
clusters = fcluster(clustering, 2, criterion = 'maxclust')
clusters = clusters - 1
print(clusters)
```

```
[1 1 1 … 1 1 1]
```

[146]:
```
data['clusters'] = clusters
```

[147]:
```
#Silhouette coefficient
cl_sil = metrics.silhouette_score(x_scaled, data['clusters'], metric =␣
 ↪"euclidean")
print(cl_sil)
```

```
0.7432570016576694
```

[148]:
```
#Cluster4 with K-Means
```

[149]:
```
clustering = KMeans(n_clusters = 2, init = 'random', n_init = 1, random_state =␣
 ↪0).fit(x_scaled)
clusters = clustering.labels_
print(clusters)
data['clusters'] = clusters
```

```
[0 1 0 … 1 1 1]
```

[150]:
```
#Silhouette coefficient
km_sil = metrics.silhouette_score(x_scaled, data['clusters'], metric =␣
↪"euclidean")
print(km_sil)
```

```
0.2684663279061382
```

[151]:
```
#Cluster4 with DBSCAN
```

[152]:
```
clustering = DBSCAN(eps = 2, min_samples = 3, metric = "euclidean").
↪fit(x_scaled)
clusters = clustering.labels_
data['clusters'] = clusters
```

[153]:
```
#Silhouette coefficient
db_sil = metrics.silhouette_score(x_scaled, data['clusters'], metric =␣
↪"euclidean")
print(db_sil)
```

```
-0.06633855889724637
```

[154]:
```
cluster_tab = cluster_tab.append({'Type':'Only Drugs','Single Linkage': sl_sil,␣
↪'Complete Linkage':cl_sil,'K-Means':km_sil,'DBSCAN':
↪db_sil},ignore_index=True)
```

[ ]:
```

```

[155]:
```
#Cluster5: Not using any drugs
```

[156]:
```
variables = data.columns[0:13]
var_indices = [data.columns.get_loc(variable) for variable in variables]
print(var_indices)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
```

[157]:
```
#Standardizing the data
x = data.iloc[:,var_indices]
scaler = StandardScaler()
x_scaled = scaler.fit_transform(x)
```

[158]:
```
#Cluster5 with Hierarchial clustering with Single Linkage
```

[159]:
```
clustering = linkage(x_scaled,method='single',metric="euclidean")
clusters = fcluster(clustering, 2, criterion = 'maxclust')
clusters = clusters - 1
```

```
print(clusters)
```

```
[0 0 0 … 0 0 0]
```

[160]:
```
#Silhouette coefficient
sl_sil = metrics.silhouette_score(x_scaled, data['clusters'], metric =␣
 ↪"euclidean")
print(sl_sil)
```

```
-0.21114670320992815
```

[161]:
```
#Cluster5 with Hierarchial Clustering with Complete Linkage
```

[162]:
```
clustering = linkage(x_scaled,method='complete',metric="euclidean")
clusters = fcluster(clustering, 2, criterion = 'maxclust')
clusters = clusters - 1
print(clusters)
```

```
[1 1 1 … 1 1 1]
```

[163]:
```
#Silhouette coefficient
cl_sil = metrics.silhouette_score(x_scaled, data['clusters'], metric =␣
 ↪"euclidean")
print(cl_sil)
```

```
-0.21114670320992815
```

[164]:
```
#Cluster 5 with K-Means
```

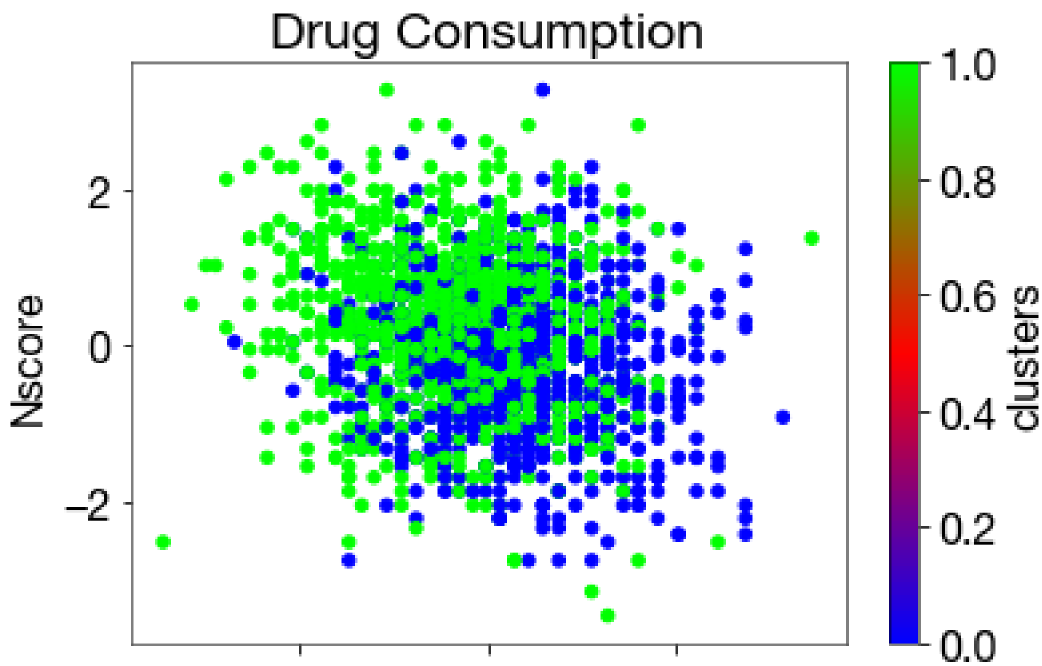
[165]:
```
clustering = KMeans(n_clusters = 2, init = 'random', n_init = 1, random_state =␣
 ↪0).fit(x_scaled)
clusters = clustering.labels_
print(clusters)
data['clusters'] = clusters
```

```
[0 0 0 … 1 1 1]
```

[166]:
```
#Silhouette coefficient
km_sil = metrics.silhouette_score(x_scaled, data['clusters'], metric =␣
 ↪"euclidean")
print(km_sil)
```

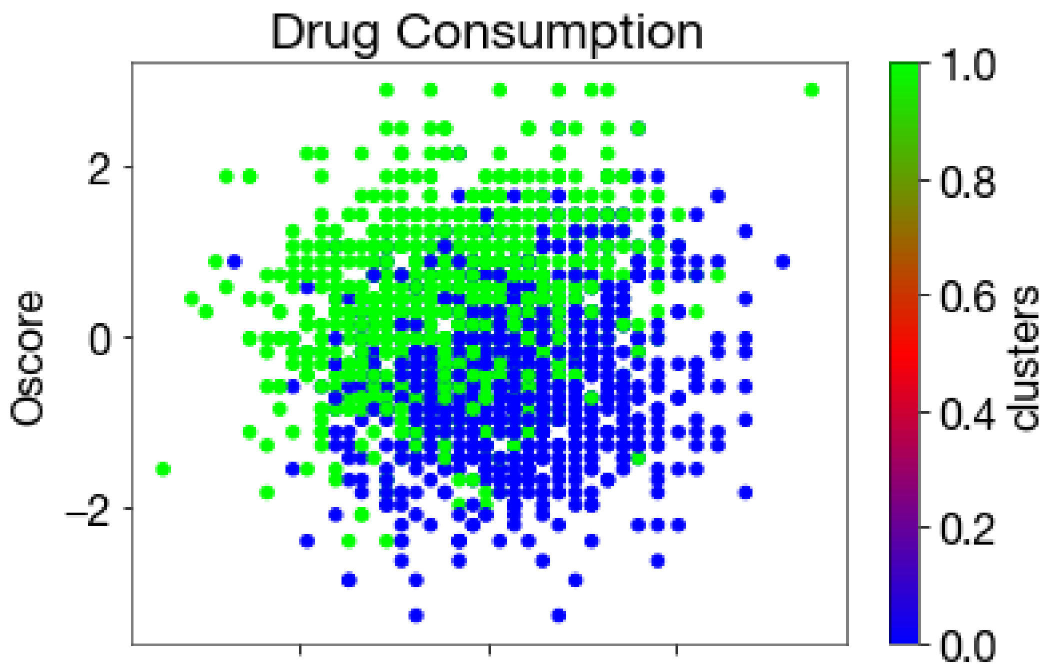
```
0.15804112591503822
```

[167]:
```
ax = data.plot(kind = 'scatter', x = 'Ascore', y = 'Nscore', c = 'clusters',␣
 ↪colormap = plt.cm.brg)
ax.set(title = 'Drug Consumption', xlabel = 'Ascore', ylabel = 'Nscore')
plt.savefig('figures/cluster5-AscoreVsNscore.png', dpi=300)
```



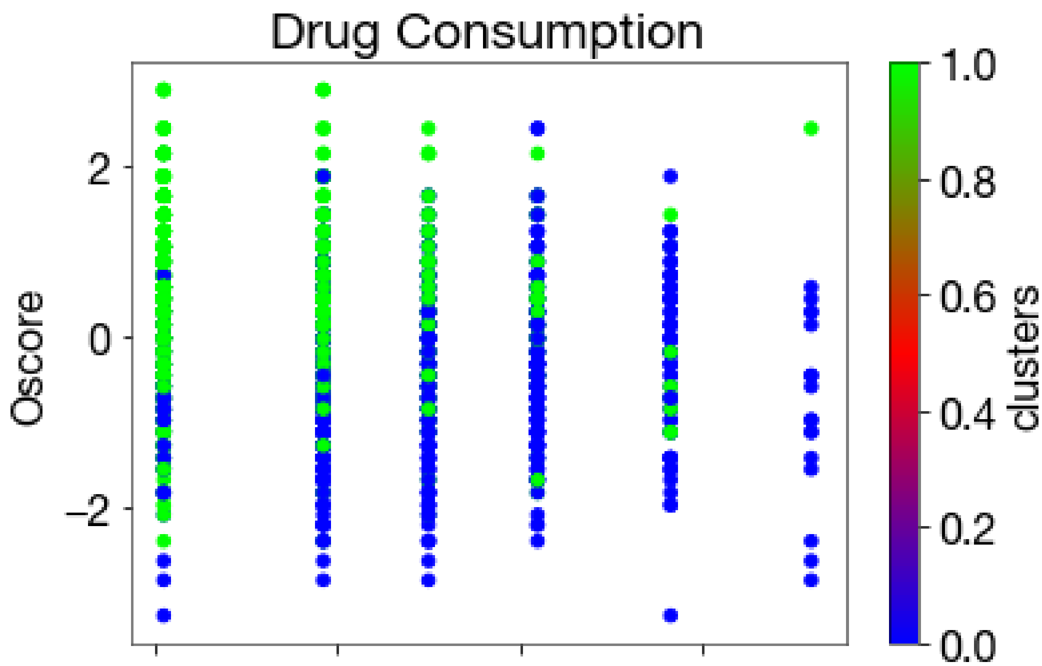
[168]:
```
ax = data.plot(kind = 'scatter', x = 'Ascore', y = 'Oscore', c = 'clusters',␣
 ↪colormap = plt.cm.brg)
ax.set(title = 'Drug Consumption', xlabel = 'Ascore', ylabel = 'Oscore')
```

[168]:
```
[Text(0, 0.5, 'Oscore'),
 Text(0.5, 0, 'Ascore'),
 Text(0.5, 1.0, 'Drug Consumption')]
```



[169]:
```
ax = data.plot(kind = 'scatter', x = 'Age', y = 'Oscore', c = 'clusters',␣
 ↪colormap = plt.cm.brg)
ax.set(title = 'Drug Consumption', xlabel = 'Age', ylabel = 'Oscore')
```

[169]:
```
[Text(0, 0.5, 'Oscore'),
 Text(0.5, 0, 'Age'),
 Text(0.5, 1.0, 'Drug Consumption')]
```



[170]:
```
#Cluster5 with DBSCAN
```

[171]:
```
clustering = DBSCAN(eps = 2, min_samples = 3, metric = "euclidean").
 ↪fit(x_scaled)
clusters = clustering.labels_
data['clusters'] = clusters
```

[172]:
```
#Silhouette coefficient
db_sil = metrics.silhouette_score(x_scaled, data['clusters'], metric =␣
 ↪"euclidean")
print(db_sil)
```

```
-0.11510965619210076
```

[173]:
```
cluster_tab = cluster_tab.append({'Type':'Not Using Drugs','Single Linkage':␣
 ↪sl_sil, 'Complete Linkage':cl_sil,'K-Means':km_sil,'DBSCAN':
 ↪db_sil},ignore_index=True)
```

[ ]:
```

```

[174]:
```
#Cluster6: Using Personality traits
```

[175]:
```
variables = data.columns[6:13]
var_indices = [data.columns.get_loc(variable) for variable in variables]
print(var_indices)
```

```
[6, 7, 8, 9, 10, 11, 12]
```

[176]:
```
#Standardizing the data
x = data.iloc[:,var_indices]
scaler = StandardScaler()
x_scaled = scaler.fit_transform(x)
```

[177]:
```
#Cluster6 with Hierarchial clustering with Single Linkage
```

[178]:
```
clustering = linkage(x_scaled,method="single",metric="euclidean")
clusters = fcluster(clustering, 2, criterion = 'maxclust')
clusters = clusters - 1
print(clusters)
```

```
[0 0 0 … 0 0 0]
```

[179]:
```
#Silhouette coefficient
sl_sil = metrics.silhouette_score(x_scaled, data['clusters'], metric =␣
 ↪"euclidean")
print(sl_sil)
```

```
-0.32290072471567954
```

[180]:
```
#Cluster6 with Hierarchial Clustering with Complete Linkage
```

[181]:
```
clustering = linkage(x_scaled,method='complete',metric="euclidean")
clusters = fcluster(clustering, 2, criterion = 'maxclust')
clusters = clusters - 1
print(clusters)
```

```
[1 1 0 … 0 0 1]
```

[182]:
```
#Silhouette coefficient
cl_sil = metrics.silhouette_score(x_scaled, data['clusters'], metric =␣
 ↪"euclidean")
print(cl_sil)
```

```
-0.32290072471567954
```

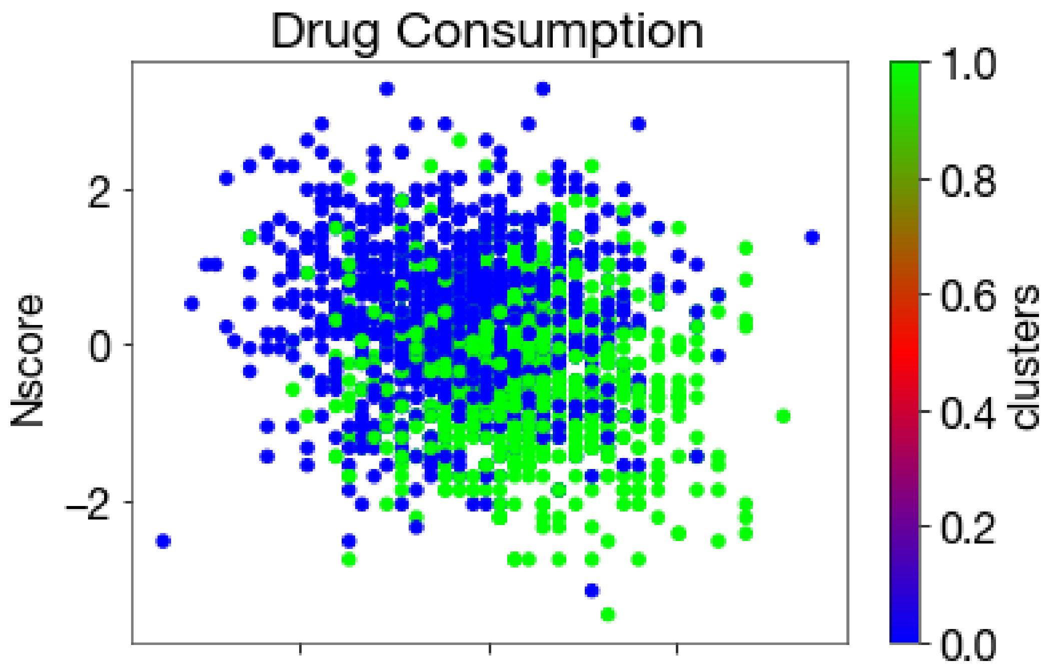
[183]:
```
#Cluster6 with K-Means
```

[184]:
```
clustering = KMeans(n_clusters = 2, init = 'random', n_init = 1, random_state =␣
 ↪0).fit(x_scaled)
clusters = clustering.labels_
print(clusters)
data['clusters'] = clusters
```

```
[1 1 1 … 0 0 0]
```

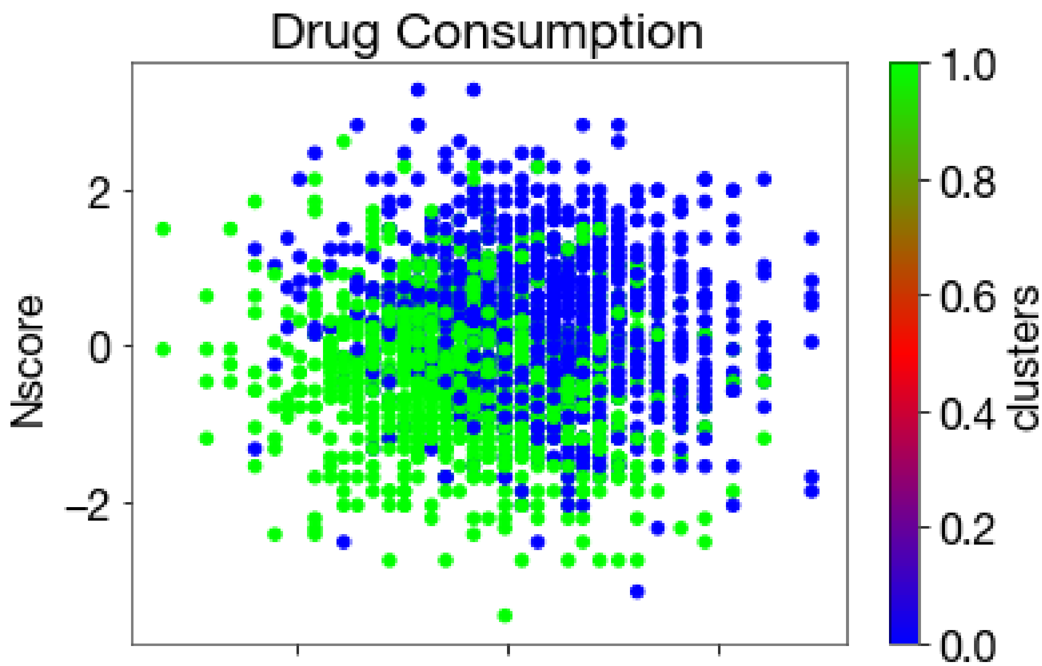
[185]:
```
ax = data.plot(kind = 'scatter', x = 'Ascore', y = 'Nscore', c = 'clusters',␣
↪colormap = plt.cm.brg)
ax.set(title = 'Drug Consumption', xlabel = 'Ascore', ylabel = 'Nscore')
```

[185]:
```
[Text(0, 0.5, 'Nscore'),
 Text(0.5, 0, 'Ascore'),
 Text(0.5, 1.0, 'Drug Consumption')]
```

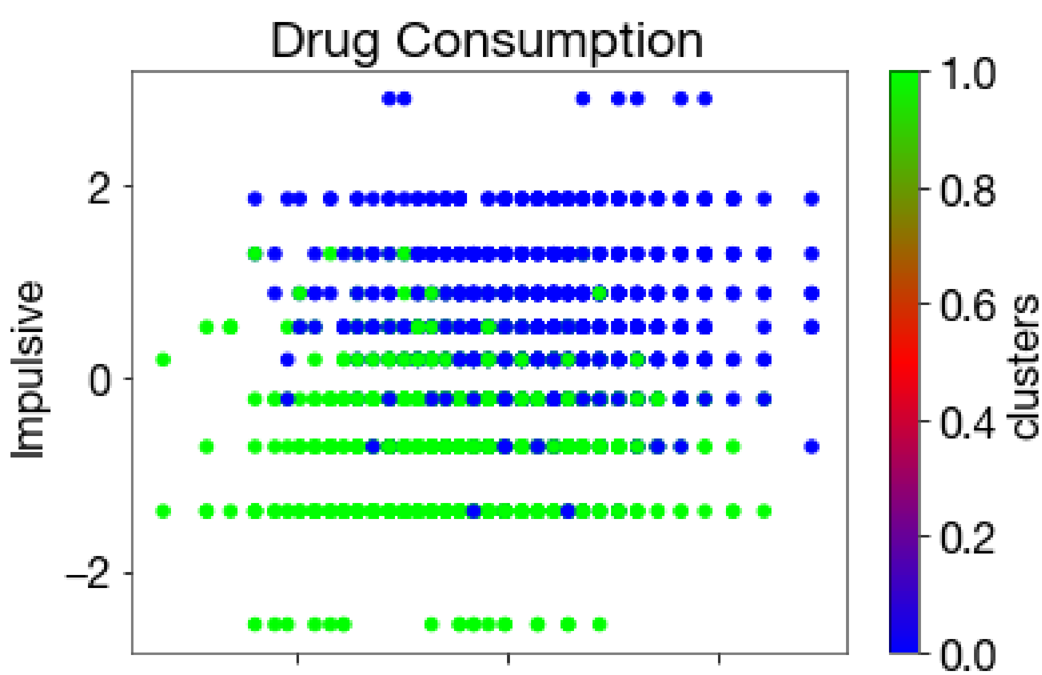


[186]:
```
ax = data.plot(kind = 'scatter', x = 'Oscore', y = 'Nscore', c = 'clusters',␣
↪colormap = plt.cm.brg)
ax.set(title = 'Drug Consumption', xlabel = 'Oscore', ylabel = 'Nscore')
```

[186]:
```
[Text(0, 0.5, 'Nscore'),
 Text(0.5, 0, 'Oscore'),
 Text(0.5, 1.0, 'Drug Consumption')]
```



[187]:
```
ax = data.plot(kind = 'scatter', x = 'Oscore', y = 'Impulsive', c = 'clusters',␣
↪colormap = plt.cm.brg)
ax.set(title = 'Drug Consumption', xlabel = 'Oscore', ylabel = 'Impulsive')
plt.savefig('figures/cluster6-OscoreVImpulsive.png', dpi=300)
```



[188]:
```
#Silhouette coefficient
km_sil = metrics.silhouette_score(x_scaled, data['clusters'], metric =␣
↪"euclidean")
print(km_sil)
```

0.1898276389135005

[189]:
```
#Cluster6 with DBSCAN
```

[190]:
```
clustering = DBSCAN(eps = 2, min_samples = 3, metric = "euclidean").
↪fit(x_scaled)
clusters = clustering.labels_
data['clusters'] = clusters
```

[191]:
```
#Silhouette coefficient
db_sil = metrics.silhouette_score(x_scaled, data['clusters'], metric =␣
↪"euclidean")
print(db_sil)
```

0.3122595933194069

[192]:
```
cluster_tab = cluster_tab.append({'Type':'Personality Traits','Single Linkage':␣
↪sl_sil, 'Complete Linkage':cl_sil,'K-Means':km_sil,'DBSCAN':
↪db_sil},ignore_index=True)
```

[ ]:
```

```

[193]:
```
#Cluster7: Using Personality traits and Non_illegal drugs
```

[194]:
```
variables = Non_illegal
var_indices = [data.columns.get_loc(variable) for variable in variables] +␣
↪[6,7,8,9,10,11,12]
print(var_indices)
```

[13, 15, 16, 17, 18, 19, 24, 25, 27, 29, 30, 31, 6, 7, 8, 9, 10, 11, 12]

[195]:
```
#Standardizing the data
x = data.iloc[:,var_indices]
scaler = StandardScaler()
x_scaled = scaler.fit_transform(x)
```

[196]:
```
#Cluster7 with Hierarchial Clustering with Single Linkage
```

[197]:
```
clustering = linkage(x_scaled,method="single",metric="euclidean")
clusters = fcluster(clustering, 2, criterion = 'maxclust')
clusters = clusters - 1
print(clusters)
```

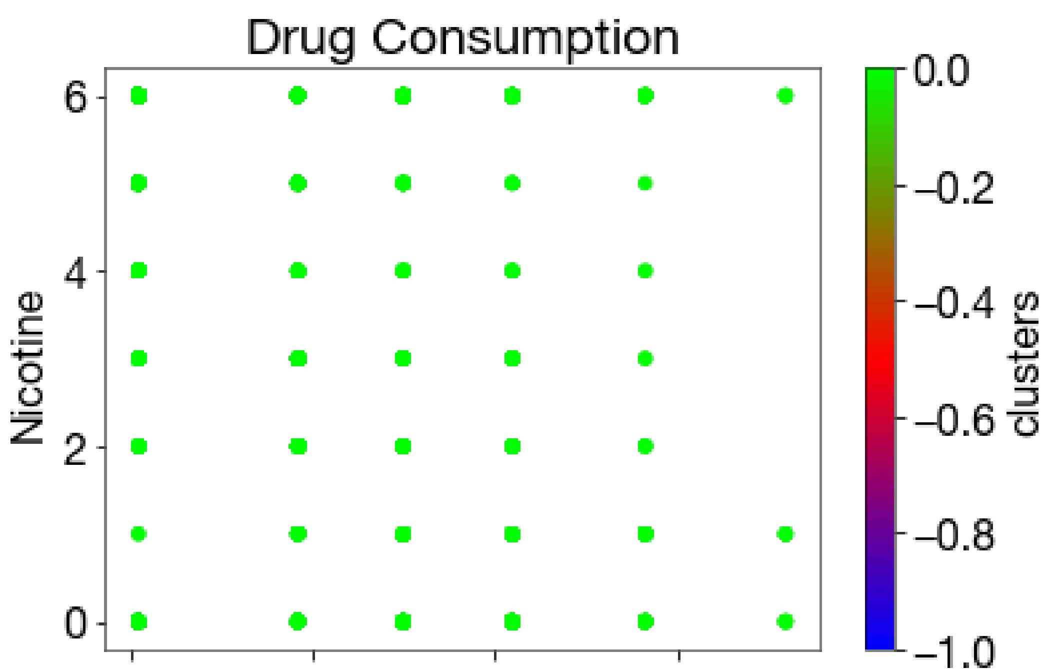
```
[0 0 0 … 0 0 0]
```

[198]:
```
#Silhouette coefficient
sl_sil = metrics.silhouette_score(x_scaled, data['clusters'], metric =␣
↪"euclidean")
print(sl_sil)
```

```
0.2200681697633267
```

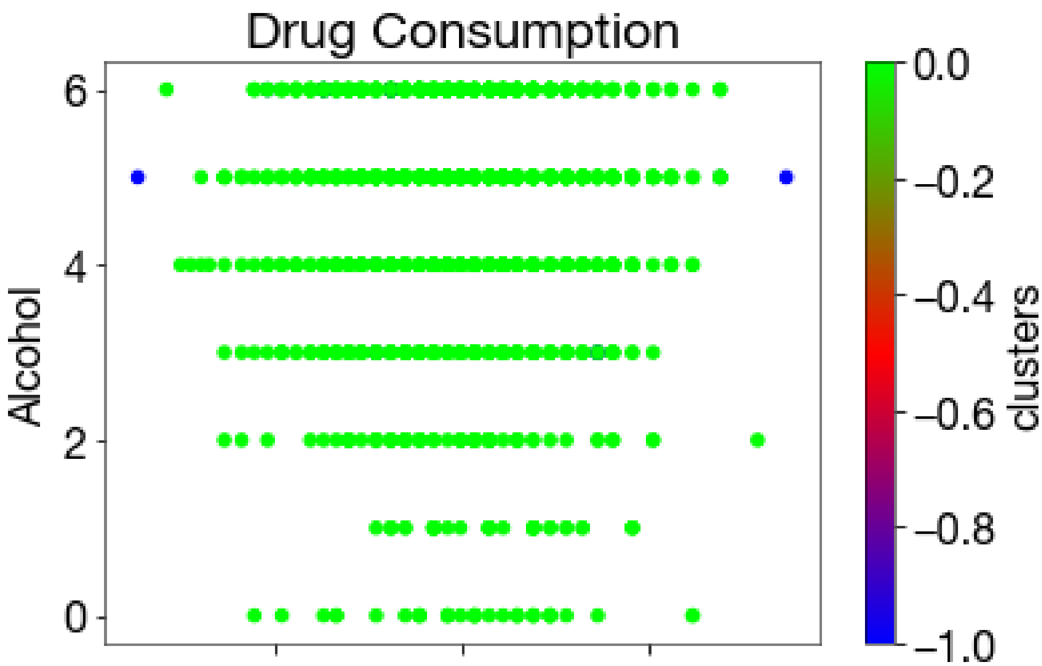
[199]:
```
ax = data.plot(kind = 'scatter', x = 'Age', y = 'Nicotine', c = 'clusters',␣
↪colormap = plt.cm.brg)
ax.set(title = 'Drug Consumption', xlabel = 'Age', ylabel = 'Nicotine')
```

[199]:
```
[Text(0, 0.5, 'Nicotine'),
 Text(0.5, 0, 'Age'),
 Text(0.5, 1.0, 'Drug Consumption')]
```



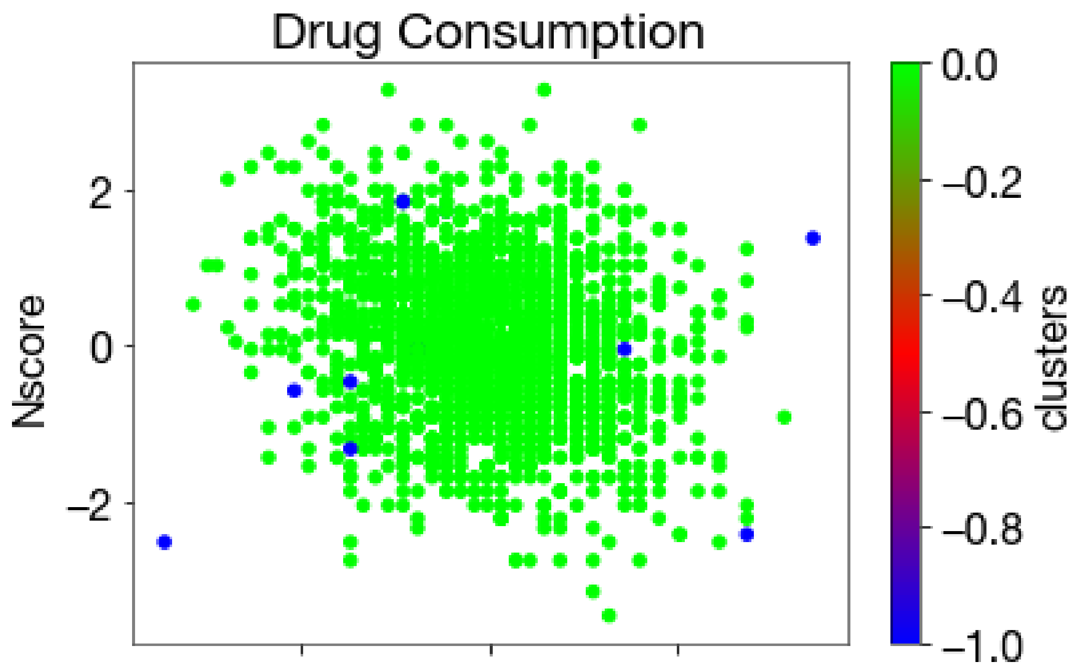
[200]:
```
ax = data.plot(kind = 'scatter', x = 'Ascore', y = 'Alcohol', c = 'clusters',␣
↪colormap = plt.cm.brg)
ax.set(title = 'Drug Consumption', xlabel = 'Ascore', ylabel = 'Alcohol')
```

[200]: [Text(0, 0.5, 'Alcohol'),
 Text(0.5, 0, 'Ascore'),
 Text(0.5, 1.0, 'Drug Consumption')]



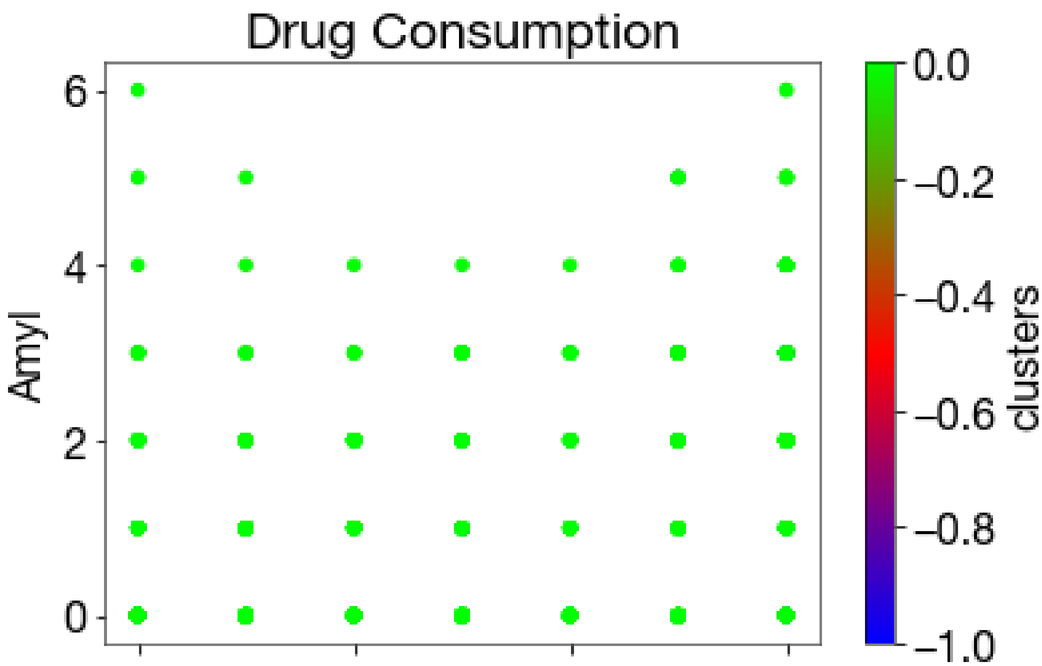

```
[201]: ax = data.plot(kind = 'scatter', x = 'Ascore', y = 'Nscore', c = 'clusters',␣
        ↪colormap = plt.cm.brg)
       ax.set(title = 'Drug Consumption', xlabel = 'Ascore', ylabel = 'Nscore')
```

[201]: [Text(0, 0.5, 'Nscore'),
 Text(0.5, 0, 'Ascore'),
 Text(0.5, 1.0, 'Drug Consumption')]



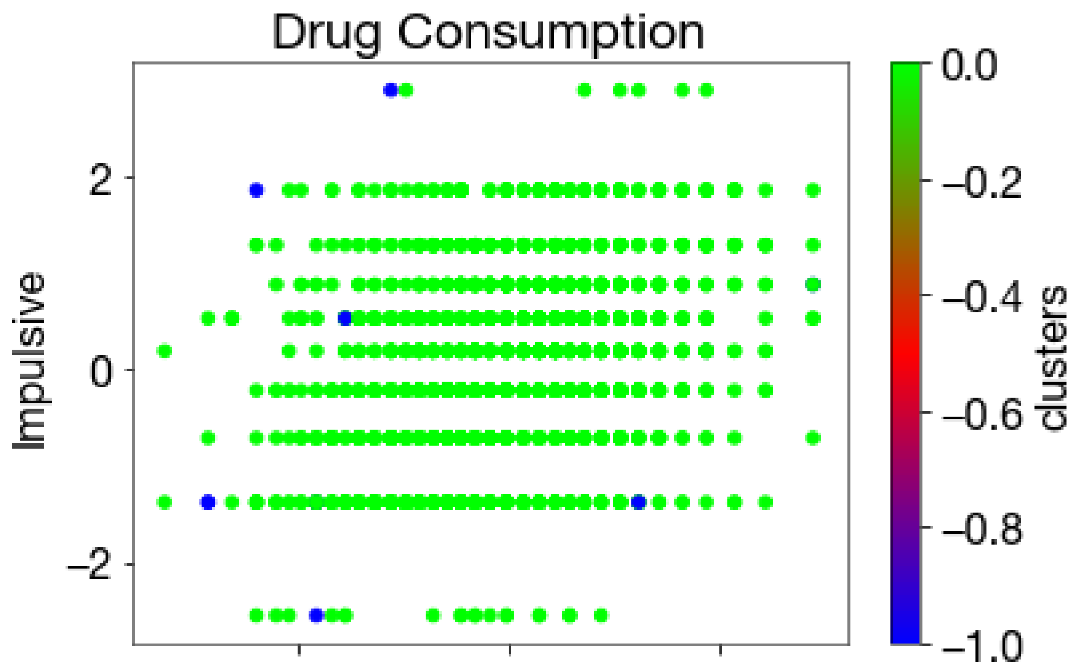
[202]:
```
ax = data.plot(kind = 'scatter', x = 'Nicotine', y = 'Amyl', c = 'clusters',␣
 ↪colormap = plt.cm.brg)
ax.set(title = 'Drug Consumption', xlabel = 'Nicotine', ylabel = 'Amyl')
```

[202]:
```
[Text(0, 0.5, 'Amyl'),
 Text(0.5, 0, 'Nicotine'),
 Text(0.5, 1.0, 'Drug Consumption')]
```




```
[203]: ax = data.plot(kind = 'scatter', x = 'Oscore', y = 'Impulsive', c = 'clusters',
        ↪colormap = plt.cm.brg)
       ax.set(title = 'Drug Consumption', xlabel = 'Oscore', ylabel = 'Impulsive')
```

```
[203]: [Text(0, 0.5, 'Impulsive'),
        Text(0.5, 0, 'Oscore'),
        Text(0.5, 1.0, 'Drug Consumption')]
```



[204]:
```
#Cluster7 with Hierarchial Clustering with Complete Linkage
```

[205]:
```
clustering = linkage(x_scaled,method='complete',metric="euclidean")
clusters = fcluster(clustering, 2, criterion = 'maxclust')
clusters = clusters - 1
print(clusters)
```

```
[1 1 1 … 1 1 1]
```

[206]:
```
#Silhouette coefficient
cl_sil = metrics.silhouette_score(x_scaled, data['clusters'], metric =␣
↪"euclidean")
print(cl_sil)
```

```
0.2200681697633267
```

[ ]:
```
#Cluster7 with K-Means
```

[ ]:
```
clustering = KMeans(n_clusters = 2, init = 'random', n_init = 1, random_state =␣
↪0).fit(x_scaled)
clusters = clustering.labels_
print(clusters)
data['clusters'] = clusters
```

[ ]:
```
#Silhouette coefficient
km_sil = metrics.silhouette_score(x_scaled, data['clusters'], metric =␣
↪"euclidean")
print(km_sil)
```

[ ]:
```
#Cluster7 with DBSCAN
```

[ ]:
```
clustering = DBSCAN(eps = 2, min_samples = 3, metric = "euclidean").
↪fit(x_scaled)
clusters = clustering.labels_
data['clusters'] = clusters
```

[ ]:
```
#Silhouette coefficient
db_sil = metrics.silhouette_score(x_scaled, data['clusters'], metric =␣
↪"euclidean")
print(db_sil)
```

[ ]:
```
cluster_tab = cluster_tab.append({'Type':'Personality Traits and Non-Illegal␣
↪drugs','Single Linkage': sl_sil, 'Complete Linkage':cl_sil,'K-Means':
↪km_sil,'DBSCAN':db_sil},ignore_index=True)
```

[ ]:
```

```

[ ]:
```
#Cluster8: Using Personality traits and Illegal drugs
```

[ ]:
```
variables = illegal_drugs
var_indices = [data.columns.get_loc(variable) for variable in variables] +␣
↪[6,7,8,9,10,11,12]
print(var_indices)
```

[ ]:
```
#Standardizing the data
x = data.iloc[:,var_indices]
scaler = StandardScaler()
x_scaled = scaler.fit_transform(x)
```

[ ]:
```
#Cluster8 with Hierarchial Clustering with Single Linkage
```

[ ]:
```
clustering = linkage(x_scaled,method="single",metric="euclidean")
clusters = fcluster(clustering, 2, criterion = 'maxclust')
clusters = clusters - 1
print(clusters)
```

[ ]:
```
#Silhouette coefficient
sl_sil = metrics.silhouette_score(x_scaled, data['clusters'], metric =␣
↪"euclidean")
print(sl_sil)
```

[ ]:
```
#Cluster8 with Hierarchial Clustering with Complete Linkage
```

[ ]:
```
clustering = linkage(x_scaled,method='complete',metric="euclidean")
clusters = fcluster(clustering, 2, criterion = 'maxclust')
clusters = clusters - 1
print(clusters)
```

[ ]:
```
#Silhouette coefficient
cl_sil = metrics.silhouette_score(x_scaled, data['clusters'], metric =␣
 ↪"euclidean")
print(cl_sil)
```

[ ]:
```
#Cluster8 with K-Means
```

[ ]:
```
clustering = KMeans(n_clusters = 2, init = 'random', n_init = 1, random_state =␣
 ↪0).fit(x_scaled)
clusters = clustering.labels_
print(clusters)
data['clusters'] = clusters
```

[ ]:
```
#Silhouette coefficient
km_sil = metrics.silhouette_score(x_scaled, data['clusters'], metric =␣
 ↪"euclidean")
print(km_sil)
```

[ ]:
```
ax = data.plot(kind = 'scatter', x = 'Ascore', y = 'Coke', c = 'clusters',␣
 ↪colormap = plt.cm.brg)
ax.set(title = 'Drug Consumption', xlabel = 'Ascore', ylabel = 'Coke')
```

[ ]:
```
ax = data.plot(kind = 'scatter', x = 'Ascore', y = 'Nscore', c = 'clusters',␣
 ↪colormap = plt.cm.brg)
ax.set(title = 'Drug Consumption', xlabel = 'Ascore', ylabel = 'Nscore')
```

[ ]:
```
ax = data.plot(kind = 'scatter', x = 'Crack', y = 'Coke', c = 'clusters',␣
 ↪colormap = plt.cm.brg)
ax.set(title = 'Drug Consumption', xlabel = 'Crack', ylabel = 'Coke')
```

[ ]:
```
ax = data.plot(kind = 'scatter', x = 'Oscore', y = 'Mushrooms', c = 'clusters',␣
 ↪colormap = plt.cm.brg)
ax.set(title = 'Drug Consumption', xlabel = 'Oscore', ylabel = 'Mushrooms')
```

[ ]:
```
#Cluster8 with DBSCAN
```

[ ]:
```
clustering = DBSCAN(eps = 2, min_samples = 3, metric = "euclidean").
 ↪fit(x_scaled)
clusters = clustering.labels_
```

```
data['clusters'] = clusters
```

[ ]:
```
#Silhouette coefficient
db_sil = metrics.silhouette_score(x_scaled, data['clusters'], metric =␣
↪"euclidean")
print(db_sil)
```

[ ]:
```
cluster_tab = cluster_tab.append({'Type':'Personality traits and Illegal␣
↪Drugs','Single Linkage': sl_sil, 'Complete Linkage':cl_sil,'K-Means':
↪km_sil,'DBSCAN':db_sil},ignore_index=True)
```

[ ]:
```
cluster_tab
```