In [1]:
```python
import pandas as pd
import sklearn as sk
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
```

In [2]:
```python
pip install -U notebook-as-pdf
```

```
Requirement already up-to-date: notebook-as-pdf in /opt/anaconda3/lib/python3.8/site-packages (0.3.1)
Requirement already satisfied, skipping upgrade: nbconvert in /opt/anaconda3/lib/python3.8/site-packages (fr
om notebook-as-pdf) (6.0.7)
Requirement already satisfied, skipping upgrade: pyppeteer in /opt/anaconda3/lib/python3.8/site-packages (fr
om notebook-as-pdf) (0.2.5)
Requirement already satisfied, skipping upgrade: PyPDF2 in /opt/anaconda3/lib/python3.8/site-packages (from
notebook-as-pdf) (1.26.0)
Requirement already satisfied, skipping upgrade: jupyter-core in /opt/anaconda3/lib/python3.8/site-packages
(from nbconvert->notebook-as-pdf) (4.6.3)
Requirement already satisfied, skipping upgrade: nbformat>=4.4 in /opt/anaconda3/lib/python3.8/site-packages
(from nbconvert->notebook-as-pdf) (5.0.7)
Requirement already satisfied, skipping upgrade: testpath in /opt/anaconda3/lib/python3.8/site-packages (fro
m nbconvert->notebook-as-pdf) (0.4.4)
Requirement already satisfied, skipping upgrade: entrypoints>=0.2.2 in /opt/anaconda3/lib/python3.8/site-pac
kages (from nbconvert->notebook-as-pdf) (0.3)
Requirement already satisfied, skipping upgrade: pygments in /opt/anaconda3/lib/python3.8/site-packages (fro
m nbconvert->notebook-as-pdf) (2.6.1)
Requirement already satisfied, skipping upgrade: pandocfilters>=1.4.1 in /opt/anaconda3/lib/python3.8/site-p
ackages (from nbconvert->notebook-as-pdf) (1.4.2)
Requirement already satisfied, skipping upgrade: defusedxml in /opt/anaconda3/lib/python3.8/site-packages (f
rom nbconvert->notebook-as-pdf) (0.6.0)
Requirement already satisfied, skipping upgrade: mistune<2,>=0.8.1 in /opt/anaconda3/lib/python3.8/site-pack
ages (from nbconvert->notebook-as-pdf) (0.8.4)
Requirement already satisfied, skipping upgrade: jinja2>=2.4 in /opt/anaconda3/lib/python3.8/site-packages (
from nbconvert->notebook-as-pdf) (2.11.2)
Requirement already satisfied, skipping upgrade: bleach in /opt/anaconda3/lib/python3.8/site-packages (from
nbconvert->notebook-as-pdf) (3.1.5)
Requirement already satisfied, skipping upgrade: traitlets>=4.2 in /opt/anaconda3/lib/python3.8/site-package
s (from nbconvert->notebook-as-pdf) (4.3.3)
Requirement already satisfied, skipping upgrade: pyee<9.0.0,>=8.1.0 in /opt/anaconda3/lib/python3.8/site-pac
kages (from pyppeteer->notebook-as-pdf) (8.1.0)
Requirement already satisfied, skipping upgrade: tqdm<5.0.0,>=4.42.1 in /opt/anaconda3/lib/python3.8/site-pa
ckages (from pyppeteer->notebook-as-pdf) (4.47.0)
Requirement already satisfied, skipping upgrade: appdirs<2.0.0,>=1.4.3 in /opt/anaconda3/lib/python3.8/site-
packages (from pyppeteer->notebook-as-pdf) (1.4.4)
Requirement already satisfied, skipping upgrade: websockets<9.0,>=8.1 in /opt/anaconda3/lib/python3.8/site-
packages (from pyppeteer->notebook-as-pdf) (8.1)
Requirement already satisfied, skipping upgrade: urllib3<2.0.0,>=1.25.8 in /opt/anaconda3/lib/python3.8/site
-packages (from pyppeteer->notebook-as-pdf) (1.25.9)
Requirement already satisfied, skipping upgrade: jsonschema!=2.5.0,>=2.4 in /opt/anaconda3/lib/python3.8/sit
e-packages (from nbformat>=4.4->nbconvert->notebook-as-pdf) (3.2.0)
Requirement already satisfied, skipping upgrade: ipython-genutils in /opt/anaconda3/lib/python3.8/site-packa
ges (from nbformat>=4.4->nbconvert->notebook-as-pdf) (0.2.0)
Requirement already satisfied, skipping upgrade: MarkupSafe>=0.23 in /opt/anaconda3/lib/python3.8/site-packa
ges (from jinja2>=2.4->nbconvert->notebook-as-pdf) (1.1.1)
Requirement already satisfied, skipping upgrade: webencodings in /opt/anaconda3/lib/python3.8/site-packages
(from bleach->nbconvert->notebook-as-pdf) (0.5.1)
Requirement already satisfied, skipping upgrade: packaging in /opt/anaconda3/lib/python3.8/site-packages (fr
om bleach->nbconvert->notebook-as-pdf) (20.4)
Requirement already satisfied, skipping upgrade: six>=1.9.0 in /opt/anaconda3/lib/python3.8/site-packages (f
rom bleach->nbconvert->notebook-as-pdf) (1.15.0)
Requirement already satisfied, skipping upgrade: decorator in /opt/anaconda3/lib/python3.8/site-packages (fr
om traitlets>=4.2->nbconvert->notebook-as-pdf) (4.4.2)
Requirement already satisfied, skipping upgrade: attrs>=17.4.0 in /opt/anaconda3/lib/python3.8/site-packages
(from jsonschema!=2.5.0,>=2.4->nbformat>=4.4->nbconvert->notebook-as-pdf) (19.3.0)
Requirement already satisfied, skipping upgrade: setuptools in /opt/anaconda3/lib/python3.8/site-packages (f
rom jsonschema!=2.5.0,>=2.4->nbformat>=4.4->nbconvert->notebook-as-pdf) (49.2.0.post20200714)
Requirement already satisfied, skipping upgrade: pyrsistent>=0.14.0 in /opt/anaconda3/lib/python3.8/site-pac
kages (from jsonschema!=2.5.0,>=2.4->nbformat>=4.4->nbconvert->notebook-as-pdf) (0.16.0)
Requirement already satisfied, skipping upgrade: pyparsing>=2.0.2 in /opt/anaconda3/lib/python3.8/site-packa
ges (from packaging->bleach->nbconvert->notebook-as-pdf) (2.4.7)
Note: you may need to restart the kernel to use updated packages.
```

In [3]:
```python
dataset = pd.read_csv('drug_consumption.csv')
data= dataset.copy()
```

Describing the dataset

In [4]:
```python
#first we want to have a preview of our dataset
dataset
```

Out[4]:

	ID	Age	Gender	Education	Country	Ethnicity	Nscore	Escore	Oscore	Ascore	...	Ecstasy	Heroin	Ketamine
0	1	0.49788	0.48246	-0.05921	0.96082	0.12600	0.31287	-0.57545	-0.58331	-0.91699	...	CL0	CL0	CL0
1	2	-0.07854	-0.48246	1.98437	0.96082	-0.31685	-0.67825	1.93886	1.43533	0.76096	...	CL4	CL0	CL2
2	3	0.49788	-0.48246	-0.05921	0.96082	-0.31685	-0.46725	0.80523	-0.84732	-1.62090	...	CL0	CL0	CL0
3	4	-0.95197	0.48246	1.16365	0.96082	-0.31685	-0.14882	-0.80615	-0.01928	0.59042	...	CL0	CL0	CL2
4	5	0.49788	0.48246	1.98437	0.96082	-0.31685	0.73545	-1.63340	-0.45174	-0.30172	...	CL1	CL0	CL0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1880	1884	-0.95197	0.48246	-0.61113	-0.57009	-0.31685	-1.19430	1.74091	1.88511	0.76096	...	CL0	CL0	CL0
1881	1885	-0.95197	-0.48246	-0.61113	-0.57009	-0.31685	-0.24649	1.74091	0.58331	0.76096	...	CL2	CL0	CL0
1882	1886	-0.07854	0.48246	0.45468	-0.57009	-0.31685	1.13281	-1.37639	-1.27553	-1.77200	...	CL4	CL0	CL2
1883	1887	-0.95197	0.48246	-0.61113	-0.57009	-0.31685	0.91093	-1.92173	0.29338	-1.62090	...	CL3	CL0	CL0
1884	1888	-0.95197	-0.48246	-0.61113	0.21128	-0.31685	-0.46725	2.12700	1.65653	1.11406	...	CL3	CL0	CL0

1885 rows × 32 columns

In [5]:
```python
dataset.describe()
```

Out[5]:

	ID	Age	Gender	Education	Country	Ethnicity	Nscore	Escore	Oscore	As
count	1885.000000	1885.00000	1885.000000	1885.000000	1885.000000	1885.000000	1885.000000	1885.000000	1885.000000	1885.00
mean	945.294960	0.03461	-0.000256	-0.003806	0.355542	-0.309577	0.000047	-0.000163	-0.000534	-0.00
std	545.167641	0.87836	0.482588	0.950078	0.700335	0.166226	0.998106	0.997448	0.996229	0.99
min	1.000000	-0.95197	-0.482460	-2.435910	-0.570090	-1.107020	-3.464360	-3.273930	-3.273930	-3.46
25%	474.000000	-0.95197	-0.482460	-0.611130	-0.570090	-0.316850	-0.678250	-0.695090	-0.717270	-0.60
50%	946.000000	-0.07854	-0.482460	-0.059210	0.960820	-0.316850	0.042570	0.003320	-0.019280	-0.01
75%	1417.000000	0.49788	0.482460	0.454680	0.960820	-0.316850	0.629670	0.637790	0.723300	0.76
max	1888.000000	2.59171	0.482460	1.984370	0.960820	1.907250	3.273930	3.273930	2.901610	3.46

In [6]:
```python
#shape of the dataset
dataset.shape
```

Out[6]: (1885, 32)

In [7]:
```python
#columns of the dataset
dataset.columns
```

Out[7]:
```
Index(['ID', 'Age', 'Gender', 'Education', 'Country', 'Ethnicity', 'Nscore',
       'Escore', 'Oscore', 'Ascore', 'Cscore', 'Impulsive', 'SS', 'Alcohol',
       'Amphet', 'Amyl', 'Benzos', 'Caff', 'Cannabis', 'Choc', 'Coke', 'Crack',
       'Ecstasy', 'Heroin', 'Ketamine', 'Legalh', 'LSD', 'Meth', 'Mushrooms',
       'Nicotine', 'Semer', 'VSA'],
      dtype='object')
```

list of features : ['ID', 'Age', 'Gender', 'Education', 'Country', 'Ethnicity', 'Nscore', 'Escore', 'Oscore', 'Ascore', 'Cscore', 'Impulsive', 'SS', 'Alcohol', 'Amphet', 'Amyl', 'Benzos', 'Caff', 'Cannabis', 'Choc', 'Coke', 'Crack', 'Ecstasy', 'Heroin', 'Ketamine', 'Legalh', 'LSD', 'Meth', 'Mushrooms', 'Nicotine', 'Semer', 'VSA']

We need to discuss a bit about the variables :

```
- ID : number of record in original database
- Age : Age of the of participants
- Gender : Gender of the participant ( 0.48246 -> Female , -0.48246 -> Male)
- Education : Levele of education :
    -2.43591 Left school before 16 years
    -1.73790 Left school at 16 years
    -1.43719 Left school at 17 years
    -1.22751 Left school at 18 years
    -0.61113 Some college or university, no certificate or degree
    -0.05921 Professional certificate/ diploma
    0.45468 University degree
    1.16365 Masters degree
    1.98437 Doctorate degree
- Country: current residence country :
    -0.09765 Australia
    0.24923 Canada
    -0.46841 New Zealand
    -0.28519 Other
    0.21128 Republic of Ireland
    0.96082 UK
    -0.57009 USA
- Ethnicity : ethnicity of participants:
    -0.50212 Asian
    -1.10702 Black
    1.90725 Mixed-Black/Asian
    0.12600 Mixed-White/Asian
    -0.22166 Mixed-White/Black
    0.11440 Other
    -0.31685 White
-Nscore is NEO-FFI-R Neuroticism
-Escore is NEO-FFI-R Extraversion
-Oscore is NEO-FFI-R Openness to experience
-Ascore is NEO-FFI-R Agreeableness
-Cscore is NEO-FFI-R Conscientiousness
-Impulsive is impulsiveness measured by BIS-11
-SS is sensation seeing measured by ImpSS

for the 7 previous ones, features are nurmerical, they describes score. The higher your
score, the more you'll be.

for the last one the features are categoricals.
CL0 : Never Used
CL1 : Used over a Decade Ago
CL2 : Used in Last Decade
CL3 : Used in Last Year
CL4 : Used in Last Month
CL5 : Used in Last Week
CL6 : Used in Last Day
```

In [8]:
```python
#we want also to see the types of the features
dataset.dtypes
```

Out[8]:
```
ID             int64
Age          float64
Gender       float64
Education    float64
Country      float64
Ethnicity    float64
Nscore       float64
Escore       float64
Oscore       float64
Ascore       float64
Cscore       float64
Impulsive    float64
SS           float64
Alcohol       object
Amphet        object
Amyl          object
Benzos        object
Caff          object
Cannabis      object
Choc          object
Coke          object
Crack         object
Ecstasy       object
Heroin        object
Ketamine      object
Legalh        object
LSD           object
Meth          object
Mushrooms     object
Nicotine      object
Semer         object
VSA           object
dtype: object
```

In [9]:
```python
dataset.isna().sum()
```

Out[9]:
```
ID           0
Age          0
Gender       0
Education    0
Country      0
Ethnicity    0
Nscore       0
Escore       0
Oscore       0
Ascore       0
Cscore       0
Impulsive    0
SS           0
Alcohol      0
Amphet       0
Amyl         0
Benzos       0
Caff         0
Cannabis     0
Choc         0
Coke         0
Crack        0
Ecstasy      0
Heroin       0
Ketamine     0
Legalh       0
LSD          0
Meth         0
Mushrooms    0
Nicotine     0
Semer        0
VSA          0
dtype: int64
```

no missing values is columns -> we don't need to delete somes rows

Previously we saw we had some categorical variables by studying on the Dataset , let's convert them

In [10]:
```python
categoricals = ['Age', 'Gender', 'Education', 'Country'
               , 'Ethnicity', 'Alcohol'
               , 'Amphet', 'Amyl', 'Benzos', 'Caff', 'Cannabis'
               , 'Choc', 'Coke', 'Crack', 'Ecstasy', 'Heroin'
               , 'Ketamine', 'Legalh', 'LSD', 'Meth', 'Mushrooms', 'Nicotine', 'Semer', 'VSA']
numericals = list(set(dataset.columns)-set(categoricals))
numericals.remove('ID')
numericals
```

Out[10]: ['Impulsive', 'Escore', 'Cscore', 'Ascore', 'Nscore', 'SS', 'Oscore']

In [11]:
```python
dataset.shape
```

Out[11]: (1885, 32)

let's try to replace some values by their meanings

In [12]:
```python
dataset["Gender"].unique()
d = dict(zip(dataset["Gender"].unique(),["Female","Male"]))
dataset["Gender"].replace(d,inplace=True)
```

In [13]:
```python
#replacing age categories

temp=dict(zip(dataset["Age"].unique(),["35-44","25-34","18-24","65+","45-54","55-64"]))
dataset["Age"].replace(temp,inplace=True)
```

In [14]:
```python
data["Education"].unique()
```

Out[14]:
```
array([-0.05921,  1.98437,  1.16365, -1.22751, -1.7379 ,  0.45468,
       -0.61113, -2.43591, -1.43719])
```

In [15]:
```python
#Replacing education
temp=dict(zip(dataset["Education"].unique(),["Professional Deg","Doctorate Deg",
                                             "Master Deg","18 Left","16 Left","Uni Deg",
                                             "Uni No Deg","-16 Left","17 Left"]))
dataset["Education"].replace(temp,inplace=True)
```

In [16]:
```python
#replacing Countries
temp=dict(zip(dataset["Country"].unique(),["UK","Canada","USA","Other","Australia","Ireland" ,
                                           "NZ" ]))
dataset["Country"].replace(temp,inplace=True)
```

In [17]:
```python
dataset["Country"].value_counts()
# We can see we have the sames stats -> successfull replacement
```

Out[17]:
```
UK           1044
USA           557
Other         118
Canada         87
Australia      54
Ireland        20
NZ              5
Name: Country, dtype: int64
```

In [18]:
```python
#Replacing ethnicity
temp=dict(zip(dataset["Ethnicity"].unique(),
              ["White/Asian","White","Other","White/Black","Asian","Black","Black/Asian"]))
dataset["Ethnicity"].replace(temp,inplace=True)
```

In [19]:
```python
dataset[categoricals] = dataset[categoricals].astype("category")
dataset[numericals] = dataset[numericals].astype("float")
dataset.dtypes
```

Out[19]:
```
ID              int64
Age          category
Gender       category
Education    category
Country      category
Ethnicity    category
Nscore        float64
Escore        float64
Oscore        float64
Ascore        float64
Cscore        float64
Impulsive     float64
SS            float64
Alcohol      category
Amphet       category
Amyl         category
Benzos       category
Caff         category
Cannabis     category
Choc         category
Coke         category
Crack        category
Ecstasy      category
Heroin       category
Ketamine     category
Legalh       category
LSD          category
Meth         category
Mushrooms    category
Nicotine     category
Semer        category
VSA          category
dtype: object
```

In [20]:
```python
dataset["Ethnicity"].value_counts()
#successfull replacement too
```

Out[20]:
```
White          1720
Other            63
Black            33
Asian            26
White/Black      20
White/Asian      20
Black/Asian       3
Name: Ethnicity, dtype: int64
```

In [21]:
```python
dataset.nunique()
```

Out[21]:
```
ID           1885
Age             6
Gender          2
Education       9
Country         7
Ethnicity       7
Nscore         49
Escore         42
Oscore         35
Ascore         41
Cscore         41
Impulsive      10
SS             11
Alcohol         7
Amphet          7
Amyl            7
Benzos          7
Caff            7
Cannabis        7
Choc            7
Coke            7
Crack           7
Ecstasy         7
Heroin          7
Ketamine        7
Legalh          7
LSD             7
Meth            7
Mushrooms       7
Nicotine        7
Semer           5
VSA             7
dtype: int64
```

In [22]:
```python
dataset["Semer"].unique()
```

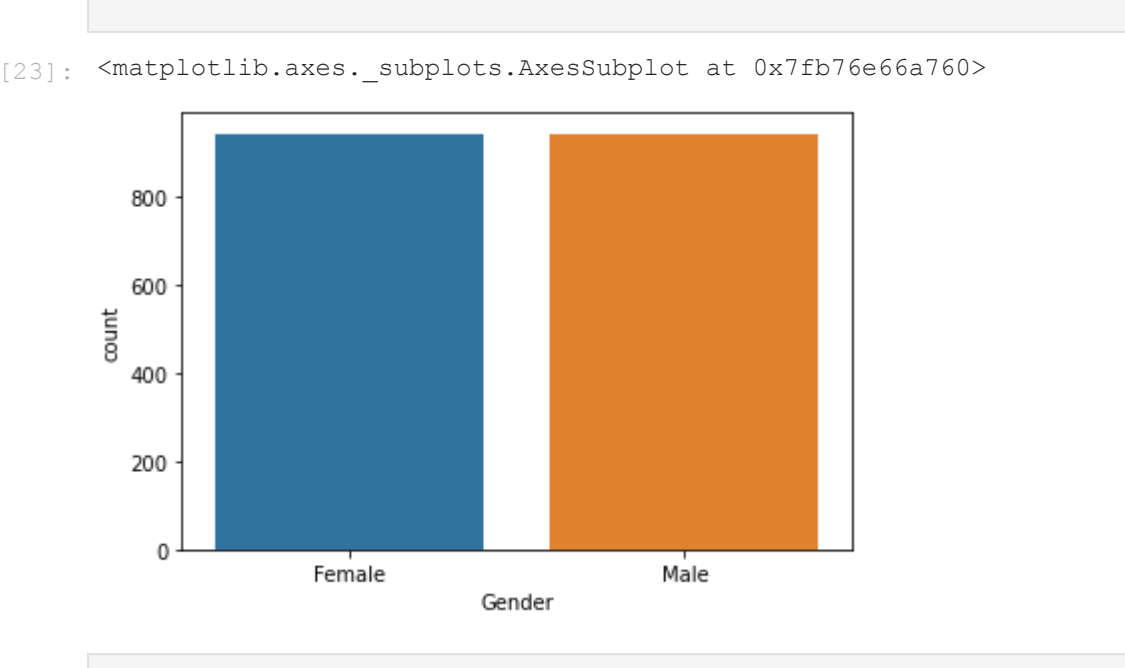
Out[22]:
```
[CL0, CL2, CL3, CL4, CL1]
Categories (5, object): [CL0, CL2, CL3, CL4, CL1]
```

Ok, so now let's try to visualise the datas

In [23]:
```python
# see proportions of the first 5 features

sns.countplot(dataset["Gender"])
# we have almost the same number of male and female in our dataset
```

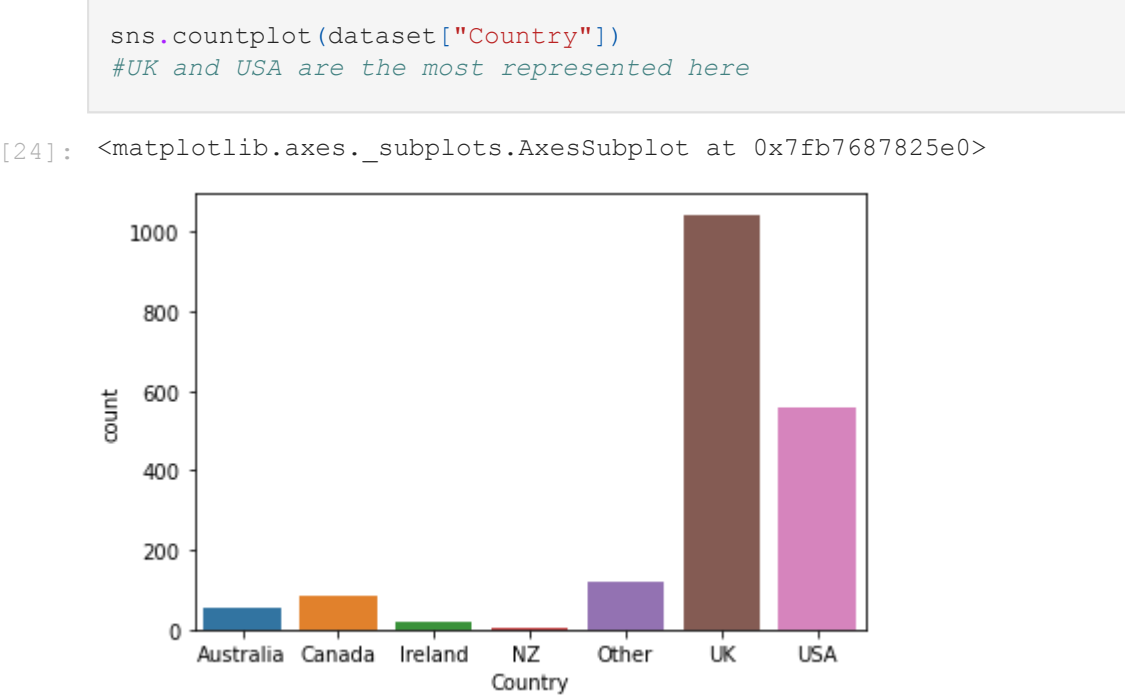
Out[23]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb76e6a760>



In [24]:
```python
# visualise the proportions of current residence country

sns.countplot(dataset["Country"])
#UK and USA are the most represented here
```

Out[24]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb7697825e0>

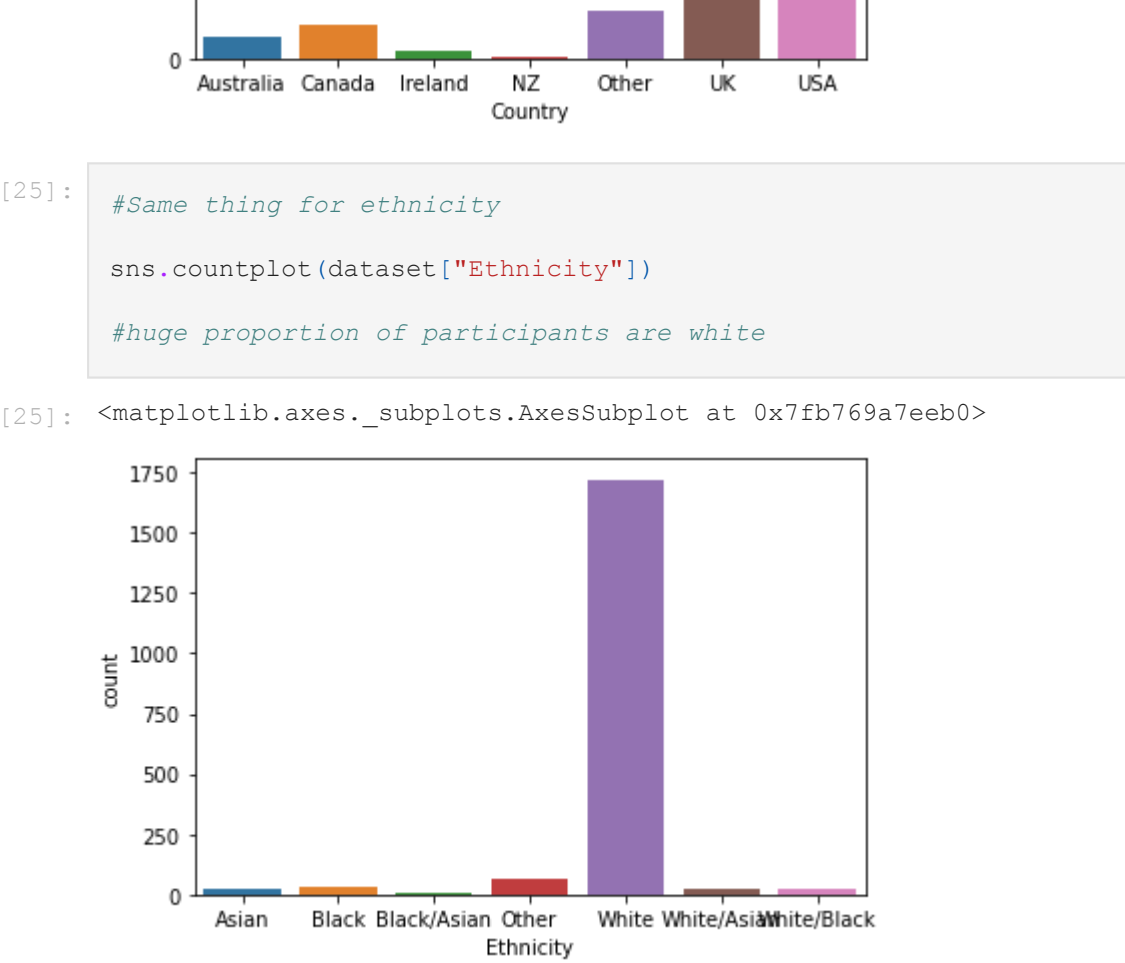


In [25]:
```python
#Same thing for ethnicity

sns.countplot(dataset["Ethnicity"])

#huge proportion of participants are white
```

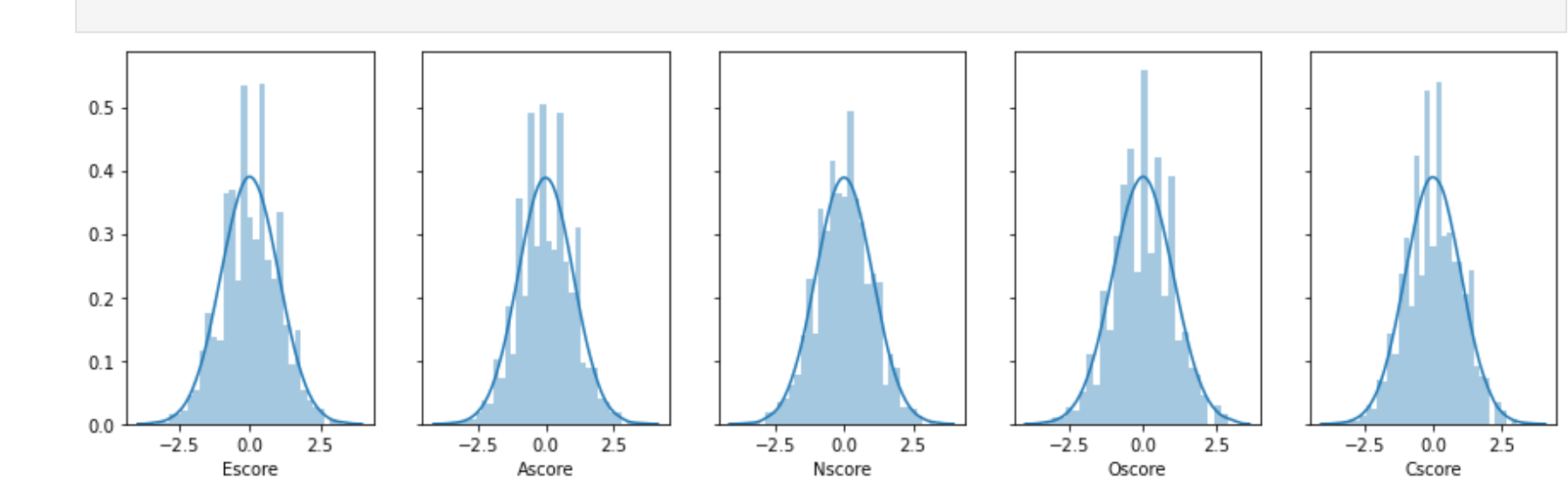
Out[25]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb769a7eeb0>



In [26]:
```python
# Now, let's try to visualise the distribution of dimensions of NEO Personality Inventory on our sample.

NEO = list(set(numericals)-set(['SS','Impulsive']))

n = len(NEO)
i=0
fig, axes = plt.subplots(1, n, figsize=(15, 4), sharey=True)
for element in NEO:
    sns.distplot(dataset[element],ax=axes[i])
    i+=1
```

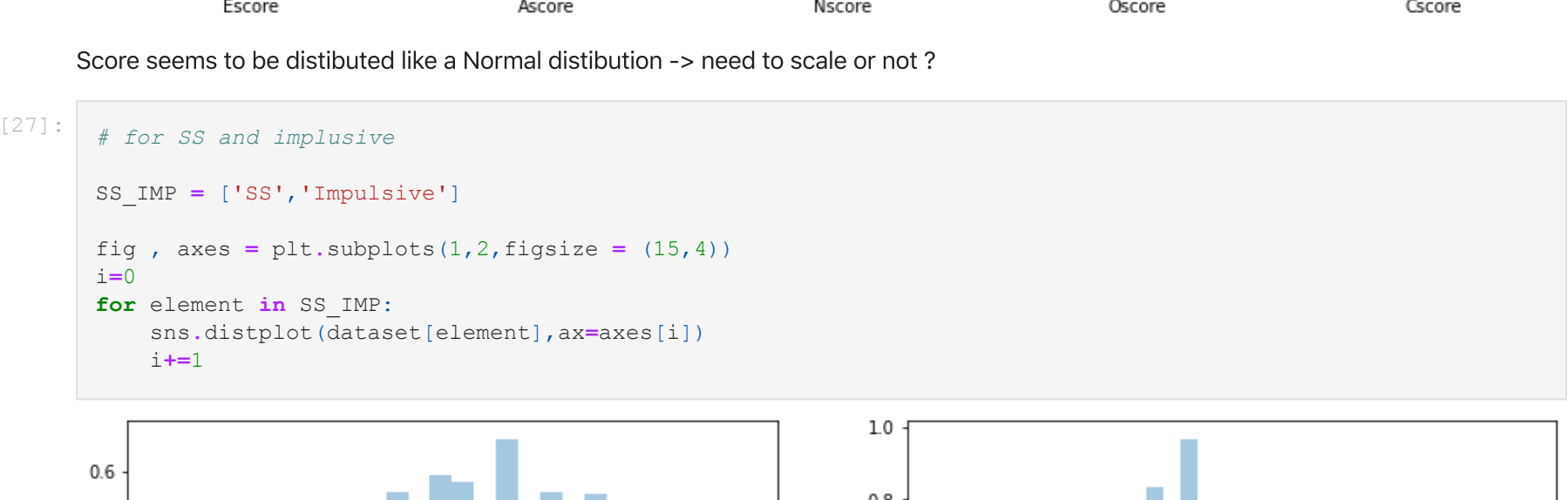


Score seems to be distibuted like a Normal distibution -> need to scale or not ?

In [27]:
```python
# for SS and implusive

SS_IMP = ['SS','Impulsive']

fig , axes = plt.subplots(1,2,figsize = (15,4))
i=0
for element in SS_IMP:
    sns.distplot(dataset[element],ax=axes[i])
    i+=1
```



In [28]:
```python
# visualition for the drugs

fig , axes = plt.subplots(6,3,figsize=(20,20))
drugs = ['Alcohol',
       'Amphet', 'Amyl', 'Benzos', 'Caff', 'Cannabis', 'Choc', 'Coke', 'Crack',
       'Ecstasy', 'Heroin', 'Ketamine', 'Legalh', 'LSD', 'Meth', 'Mushrooms',
       'Nicotine', 'Semer', 'VSA']
c=0
for i in range(6):
    for j in range(3):
        sns.countplot(dataset[drugs[c]],ax=axes[i][j])
        c+=1
```

we can see that "legal" or "almost-legal" drugs are highly use (caf,choc, nicotine, canibis,legalh, benzos)

May be we could try to classify drugs into two subparts such as legal an not legal ?

May be it could help to see the influence of the law about drug consumption s

May be we can visualise the proportions of user and non user for each drug too

In [29]:
```python
#user vs non user for each drug
fig , axes = plt.subplots(6,3,figsize=(20,20))
e =0
for i in range(6):
    for j in range(3):
        x = drugs[e]
        element = [ 'Not User' if e=="CL0" else 'User'for e in dataset[x] ]
        s = pd.DataFrame(element,columns = [x])
        sns.countplot(s[x],ax = axes[i,j])
        e+=1
```

We can see the that legals drugs have been a lot used at least one time. People seems to be reluctant to experience hard drugs such as meh , heroine, ketamine , crack.

In [30]:
```python
#proportion of user for each etnhicity and for each drug
fig , axes = plt.subplots(6,3,figsize=(30,30))
e=0
for i in range(6):
    for j in range(3):
        x = drugs[e]
        temp=dataset.groupby("Ethnicity")[x].apply(lambda x : x[x!="CL0"].count()/len(x) )
        sns.barplot(x=temp.index,y=temp.values,ax=axes[i,j])
        axes[i,j].set_title(x)
        e+=1
```

In [31]:
```python
dataset.Ethnicity.value_counts()
```

Out[31]:
```
White          1720
Other            63
Black            33
Asian            26
White/Black      20
White/Asian      20
Black/Asian       3
Name: Ethnicity, dtype: int64
```

we have a lot of white people here , even if we can visualize proportion I don't think they are a really relevant.

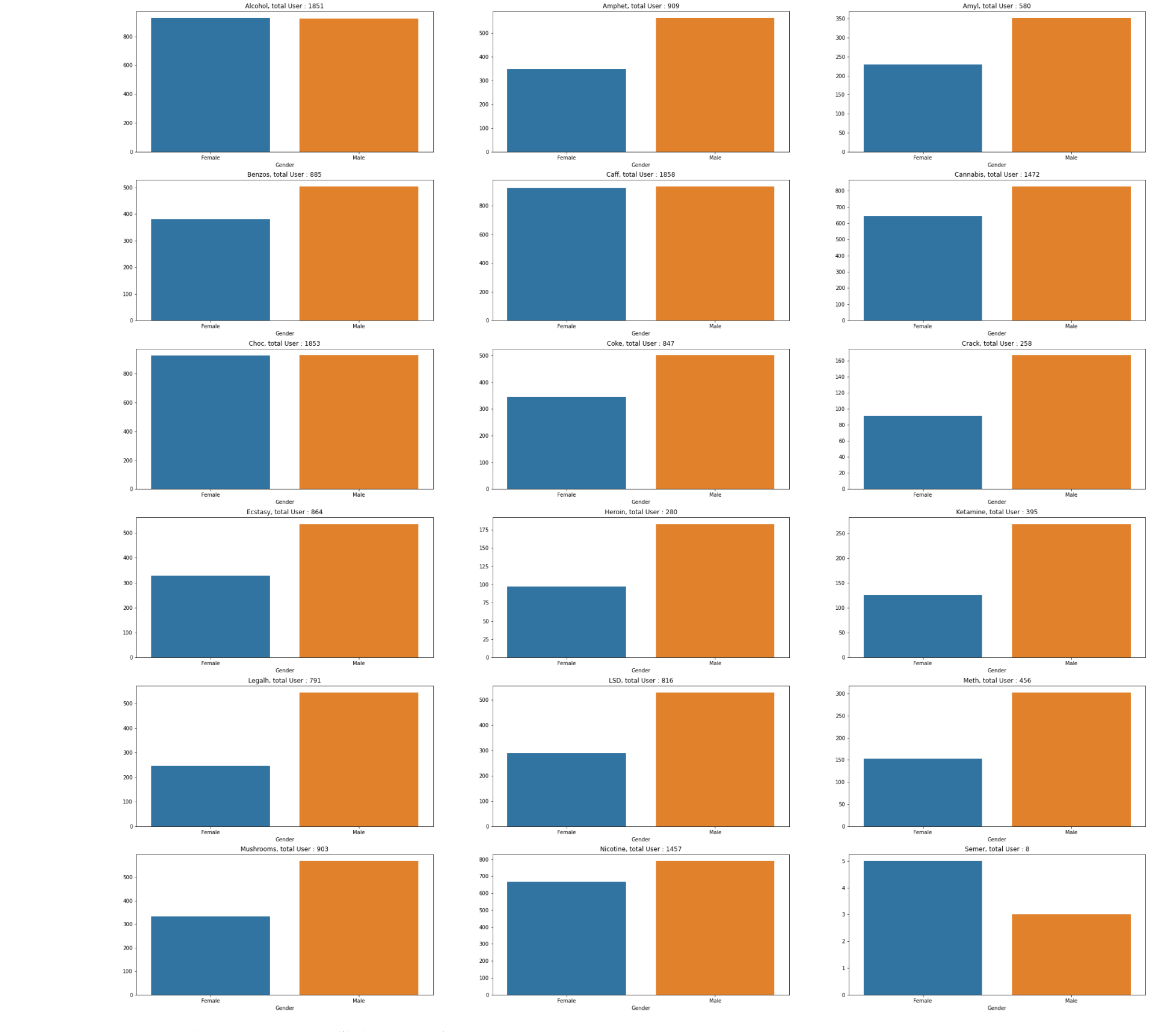
In [32]:
```python
#users eduaction repartition for each drug
fig , axes = plt.subplots(6,3,figsize=(35,35))
e =0
for i in range(6):
    for j in range(3):
        x = drugs[e]
        temp=dataset.groupby("Education")[x].apply(lambda x : x[x!="CL0"].count()/len(x) )
        sns.barplot(x=temp.index,y=temp.values,ax=axes[i,j])
        axes[i,j].set_title(x)
        e+=1
```

Ok , that's a nice visualtion to the the distibution of user study background of each drug

In [33]:
```python
dataset.Education.value_counts()
```

Out[33]:
```
Uni No Deg          506
Uni Deg             480
Master Deg          283
Professional Deg    270
18 Left             100
16 Left              99
Doctorate Deg        89
17 Left              30
-16 Left             28
Name: Education, dtype: int64
```

In [34]:
```python
fig , axes = plt.subplots(6,3,figsize=(35,35))
e =0
for i in range(6):
    for j in range(3):
        x = drugs[e]
        temp=dataset.groupby("Gender")[x].apply(lambda x : x[x!="CL0"].count())
        sns.barplot(x=temp.index,y=temp.values,ax=axes[i,j])
        axes[i,j].set_title(str(x)+", total User : {}".format(temp.values[1]+temp.values[0]))
        e+=1
```



It seems that mens are more likely to experiment drugs.

```python
from sklearn.preprocessing import LabelEncoder

col =list(dataset[drugs])
for c in col:
    le = LabelEncoder()
    data[c] = le.fit_transform( data[c])
```

```python
data.drop("ID",axis=1,inplace=True)
```

```python
data
```

	Age	Gender	Education	Country	Ethnicity	Nscore	Escore	Oscore	Ascore	Cscore	...	Ecstasy	Heroin
0	0.49788	0.48246	-0.05921	0.96082	0.12600	0.31287	-0.57545	-0.58331	-0.91699	-0.00665	...	0	0
1	-0.07854	0.48246	1.98437	0.96082	-0.31685	-0.67825	1.93886	1.43533	0.76096	-0.14277	...	4	0
2	0.49788	0.48246	-0.05921	0.96082	-0.31685	-0.46725	0.80523	-0.84732	-1.62090	-1.01450	...	0	0
3	-0.95197	0.48246	1.16365	0.96082	-0.31685	-0.14882	-0.80615	-0.01928	0.59042	0.58489	...	0	0
4	0.49788	0.48246	1.98437	0.96082	-0.31685	0.73545	-1.63340	-0.45174	-0.30172	1.30612	...	1	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
1880	-0.95197	-0.48246	-0.61113	-0.57009	-0.31685	-1.19430	1.74091	1.88511	0.76096	-1.13788	...	0	0
1881	-0.95197	-0.48246	-0.61113	-0.57009	-0.31685	-0.24649	1.74091	0.58331	0.76096	-1.51840	...	2	0
1882	-0.07854	-0.48246	0.45468	-0.57009	-0.31685	1.13281	-1.37639	-1.27553	-1.77200	-1.38502	...	4	0
1883	-0.95197	-0.48246	-0.61113	-0.57009	-0.31685	0.91093	-1.92173	0.29338	-1.62090	-2.57309	...	3	0
1884	-0.95197	-0.48246	-0.61113	0.21128	-0.31685	-0.46725	2.12700	1.65653	1.11406	0.41594	...	3	0

1885 rows × 31 columns

```python
corrmat = data.corr()

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

sns.heatmap(corrmat,cmap = "RdYlGn",annot=True,
            yticklabels = data.columns, xticklabels = data.columns)
plt.xticks(rotation=50)

plt.title("Correlation B/W Different Features",fontsize=18)
plt.show()
```

we can see there a strong correlation for some drugs, it could be expected because if you have experienced some hard drugs you are more likely to experienced others.

Now it's to time to build a prediction model to evaluate consommation of Cannabis knowing score and informations on a person.

KNN

```python
features = ['Age', 'Gender', 'Education', 'Country', 'Ethnicity', 'Nscore',
            'Escore', 'Oscore', 'Ascore', 'Cscore', 'Impulsive', 'SS']
target = ["Cannabis"]
```

```python
from sklearn.model_selection import train_test_split
X = data[features]
y = data[target].astype("category")
```

```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)
```

```python
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,precision_score
from sklearn.neighbors import KNeighborsClassifier
mod = KNeighborsClassifier()
mod.fit(X_train,y_train.values.ravel())
pred = mod.predict(X_test)
accuracy_score(y_test,pred)
```

Trying to know which is the best number of neighbours

```python
from sklearn.neighbors import KNeighborsClassifier


graph= dict()
for i in range(1,100):
    temp = KNeighborsClassifier(n_neighbors=i)
    temp.fit(X_train,y_train.values.ravel())
    pred_temp = temp.predict(X_test)

    graph[i]=accuracy_score(y_test,pred_temp)

x=list(graph.keys())
y=list(graph.values())
plt.plot(x,y)
sorted_d = sorted(graph.items(), key=lambda x: x[1],reverse=True)
sorted_d[0]
```

```python
from sklearn.metrics import confusion_matrix
knn = KNeighborsClassifier(sorted_d[0][0])
knn.fit(X_train,y_train.values.ravel())
final_pred = knn.predict(X_test)

sns.heatmap(confusion_matrix(y_test,final_pred),annot=True)
print(accuracy_score(y_test,final_pred))
```

we can see that our accuracy score is not really good. Let's try to create a new feature User_Cannabis

```python
element = [ 'Not User' if s=="CL0" else 'User'for s in dataset["Cannabis"] ]
element
```

```python
data["User_Canna"] = element
```

```python
data.replace({"User":True,"Not User":False},inplace=True)
```

```python
data.User_Canna.value_counts()
```

```python
#Scalling datas
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(data[features])
data[features]=scaler.transform(data[features])
```

```python
from sklearn.model_selection import train_test_split
X2 = data[features]
y2 = data["User_Canna"]
```

```python
X_train2, X_test2, y_train2, y_test2 = train_test_split(X2, y2, test_size=0.20)
```

influence of k for prediction

from sklearn.neighbors import KNeighborsClassifier

graph= dict() for i in range(1,100): temp = KNeighborsClassifier(n_neighbors=i) temp.fit(X_train2,y_train2.values.ravel()) pred_temp = temp.predict(X_test2)

    graph[i]=accuracy_score(y_test2,pred_temp)

x=list(graph.keys()) y=list(graph.values()) plt.plot(x,y) sorted_d = sorted(graph.items(), key=lambda x: x[1],reverse=True) sorted_d[0]

```python
kfinal = KNeighborsClassifier(sorted_d[0][0])
kfinal.fit(X_train2,y_train2.values.ravel())
yfinal = kfinal.predict(X_test2)
kscores= precision_score(y_test2,yfinal),accuracy_score(y_test2,yfinal)
kscores
```

```python
sns.heatmap(confusion_matrix(y_test2,yfinal),annot=True)
```

```python
print(kscores)
```

prediction is much better

Random Forrest

```python
from sklearn.ensemble import RandomForestClassifier
```

```python
rf = RandomForestClassifier()
rf.fit(X_train2,y_train2)
rf_predict=rf.predict(X_test2)
logscores = precision_score(y_test2,rf_predict), accuracy_score(y_test2,rf_predict)
logscores
```

searching the best number of tree to use like we did for knn

rep = dict() for i in range(10,1000, 10): rf = RandomForestClassifier(i) rf.fit(X_train2,y_train2.values.ravel()) rdpredict=rf.predict(X_test2) rep[i]= accuracy_score(rdpredict,y_test2) plt.plot(list(rep.keys()),list(rep.values()))

max_acc = sorted(rep.items(), key=lambda x: x[1],reverse=True)[0] max_acc[0]

rfinal = RandomForestClassifier(n_estimators=max_acc[0]) rf.fit(X_train2,y_train2.values.ravel()) rdpredict=rf.predict(X_test2) precision_score(rdpredict,y_test2)

Logistic Regression

```python
from sklearn.linear_model import LogisticRegression
Lr = LogisticRegression(max_iter=100)
Lr.fit(X_train2,y_train2)
y_rf = Lr.predict(X_test2)
rdscores = precision_score(y_test2,y_rf), accuracy_score(y_test2,y_rf)
rdscores
```

```python
from sklearn.ensemble import GradientBoostingClassifier
GBC = GradientBoostingClassifier()
GBC.fit(X_train2,y_train2)
pred_GBC = GBC.predict(X_test2)
GBCscores=precision_score(y_test2,pred_GBC),accuracy_score(y_test2,pred_GBC)
GBCscores
```

```python
models = ["KNN",
    "Random Forrest",
    "Logistic Regression",
    "Gradient Boosting"]
precisions = [kscores[0],
    rdscores[0],
    logscores[0],
    GBCscores[0]]
accuracies = [kscores[1],
    rdscores[1],
    logscores[1],
    GBCscores[1]]
df = pd.DataFrame(zip(models,precisions,accuracies),columns=["Model","Prec","Acc"])
```

```python
fig , axes = plt.subplots(1,2,figsize=(20,5
                                       ))
sns.barplot(x=df["Model"],
            y=df["Prec"],
            ax=axes[0],palette="Greens_d"
            )
axes[0].set_title("Precision comparaison for each model")

for ax in axes:

    for p in ax.patches:
        h = p.get_height()
        ax.text(p.get_x() + (p.get_width()/2), h*2.75/3, '{}'.format(round(h,3)), ha='center', color='black'
sns.barplot(x=df["Model"],
            y=df["Acc"],
            ax=axes[1]
            ,palette="Greens_d"
            )
axes[1].set_title("Accuracy comparaison for each model")


for ax in axes:

    for p in ax.patches:
        h = p.get_height()
        ax.text(p.get_x() + (p.get_width()/2), h*2.75/3, '{}'.format(round(h,3)), ha='center', color='black'
```

Random forrest seems to be the best model to deal with precision and accuracy

```python
sns.heatmap(confusion_matrix(rf_predict,y_test2),annot=True,cmap='Reds')
```

Let's try to improve our Random Forest regression model

```python
from sklearn.model_selection import GridSearchCV
```

```python
param_grid = {
    'bootstrap': [True],
    'criterion': ['gini','entropy'],
    'oob_score': [True],
    'max_depth': range(1,12),
    'max_features': ['sqrt'],
    'n_estimators': [75, 100, 200, 300, 500]
}
```

```python
grid_search = GridSearchCV(estimator = rf, scoring = ['accuracy','precision'], param_grid = param_grid,
                           cv = 5, n_jobs = -1, verbose = 2, refit='precision')
```

```python
grid_search.fit(X_train2, y_train2)
```

```python
best=grid_search.best_estimator_
```

```python
grid_search.best_score_
```

```python
params = grid_search.best_params_
params
```

```python
best.score(X_test2, y_test2)
```

```python
precision_score(y_test2,best.predict(X_test2))
```

from joblib import dump best.fit(X_train2,y_train2)

dump(best, 'best_model.joblib')