7. Drug Analysis Based on Personality Traits

1. Analysis

Drugs. There are all sorts of them out there, most of them illegal. But does that keep people from accessing them? The number of overdose deaths and possession arrest suggests: no.

Australia's National Drug & Alcohol Research Centre promoted an article about potential drug reform policies, in which they noted that criminalization does not fully prevent drug abuse. Instead, it favors black markets and criminal networks associated with drug trade, and applies pressure on the criminal justice system.

Source: Lorenzo Rubiera, C. (2016) 'Australia’s recreational drug policies aren’t working, so what are the options for reform?', The Conversation. Available at: https://theconversation.com/australias-recreational-drug-policies-arent-working-so-what-are-the-options-for-reform-55493 (Accessed: 21 May 2024).

On top of that, drugs' illegal status fosters a stigma around drug abuse, making it harder for people to get educated on safe drug use and addiction prevention - as highlighted by an article by the Government of Canada.

Source: Government of Canada (2024) 'Stigma around drug use'. Available at: https://www.canada.ca/en/health-canada/services/opioids/stigma.html (Accessed: 21 May 2024).

So if there is a stigma around drug abuse and addiction, and drugs' illegal status does not keep people from accessing them, the likelihood of someone developing severe addictions and harmful behaviours increases.

But what if we could predict what drugs each person is likely to become a user of, *before* they become a user? This way, we could more early educate people on abuse and addiction prevention before the harm happens.

To better understand the drug use trends across different age groups, we found a paper describing Substance Use Disorders across ages 18-90, illustrating differences by gender and race:

Vasilenko, S.A. et al. (2017) 'Age trends in rates of substance use disorders across ages 18–90: Differences by gender and race/ethnicity'. Available at: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5757874/ (Accessed: 12 Apr 2024).

According to this source, the age range with the most prevalent substance abuse is that of people in their twenties - for both genders, but more significantly for men. For this reason, we cross our fingers that whatever data we get our hands on has a lot of data about young people, and that maybe there are a bit more Male respondents than Female, since these are at a higher risk of substance abuse.

If we had a data set featuring respondents' personality traits and frequency of consumption for 18 legal & illegal drugs, could we then find out which drugs a person is likely to become a user of, depending on their personality?

If applied correctly, these predictions could be used to more accurately spread awareness material for drug use prevention, or, at least, safer drug use, in case it cannot be prevented - therein reducing its harm.

For this problem, we have determined that the best performance metric is **recall**. Given that we are using the model to spread awareness to people who might be at risk for Substance Use Disorder, we would much rather spread too much awareness than too little - so we will prioritize "catching" all the positives.

Since we will have information about each person's frequency of consumption for the substances, which is what we will be predicting, this model will perform **supervised learning**.

2. Data Acquisition & Overview

We found the dataset "Drug Consumptions (UCI)" on Kaggle, provided by the user Khadija. They got it from a paper released in 2015 called "The Five Factor Model of personality and evaulation of drug consumption risk", by E. Fehrman, A. K. Muhammad, E. M. Mirkes, V. Egan and A. N. Gorban.

Let's break it down:

The dataset contains the following attributes:

- Age
- Gender (M or F)
- Education
- Country
- Ethnicity
- Score on different personality traits
    - Big-Five Scores (OCEAN):
        - Openness
        - Conscientiousness
        - Extraversion
        - Agreeableness
        - Neuroticism
    - Impulsiveness, measured by BIS-11
    - Sensation-Seeking, measured by ImpSS

It has, then, information about each individual's consumption of the following substances:

- Alcohol
- Amphetamines
- Amyl / Nitrite
- Benzodiazepine
- Caffeine
- Cannabis / Marijuana
- Chocolate
- Cocaine
- Crack Cocaine
- Ecstasy
- Heroin
- Ketamine
- Legal Highs
- LSD
- Methadone
- Magic Mushrooms
- Nicotine
- Semeron
  - Fictitious drug, for control, to spot overclaimers
- Volatile Substance Abuse / Inhalants
  - (for example: spray deodorantes, glue, lighter refills)

The rating for each individual's use of each substance can be one of the following:

- CL0: Never Used
- CL1: Used over a Decade Ago
- CL2: Used in the Last Decade
- CL3: Used in the Last Year
- CL4: Used in the Last Month
- CL5: Used in the Last Week
- CL6: Used in the Last Day

Personality Traits in Data Set

Now, a few more words on each of the aforementioned personality traits, just so we get a good picture of what we're looking at, including what to expect from each trait when a person's score is low or high in it.

Knowing each of these traits well might make us better at inferring these values for someone we want to make predictions for, as well as understanding our data better.

Openness To Experience:

This trait is characterized by active imagination, aesthetic sensitivity, attentiveness to inner feelings, preference for variety, and intellectual curiosity. Individuals with high openness seek novelty, creativity and intellectual pursuits, whereas people who score low on this trait thrive in procedure, compliance and routine.

Conscientiousness:

Conscientiousness relates to being dutiful, responsible and careful. Someone with a high score in this trait is likely to be very focused, thorough and self-disciplined - but, if extreme, potentially a workaholic. On the flip side, someone with a lower score could be described as laid-back, easy-going and more fun to be around. If extreme, however, a person with a very low conscientiousness may come off as careless or disorganized.

Extraversion:

Extraversion is framed as a function of stimulation - someone who is more sensitive to outside stimuli would score lower on extraversion, tending to prefer smaller crowds, quieter environments, and more focused, cognitively-demanding activities. In contrast, someone with a high score in extraversion is less sensitive to external stimuli and, as a result, seeks more of it. They enjoy large gatherings, and come off as more energetic and lively. Extraverted people have an easier time asserting themselves and do not mind being the center of attention.

Agreeableness:

Features of agreeableness include compassion, trust, honesty and politeness. Agreeable people are often popular, due to their inherent need for social harmony - they are more likely to "adapt to others", avoiding conflict and compromising in the favour of others. They are polite, affectionate and trusting. Conversely, people with a low Agreeableness score do not mind conflict, are better at standing their ground and do not care so much about what others think of them. For this reason, people who feel less need to be agreeable often make for excellent scientists, critics or soldiers.

Neuroticism:

Neuroticism refers to an individual's susceptibility to stress, and a persistent tendency to be in a negative mood state. On one hand, someone with a high neurotic state experiences negative emotions such as anxiety, sadness, frustration, fear and anger more frequently than most people. This may or may not be associated with mental health concerns. On the other hand, someone with a low neurotic state is more even tempered - they go through negative emotions less often and less intensely than most people - although virtually everyone experiences them every now and then.

Impulsiveness:

Impulsiveness is caracterized by a predisposition to rapid and unplanned decisions without adequate regard for the possible negative consequences. Impulsive individuals live under the "carpe diem" mindset, prioritizing spontaneity, whereas people with a lower impulsive score have a more strategic, planned, predictable and consistent approach to life, making them far less likely to make rash decisions or act without thinking. Less impulsive individuals keep the big picture in mind and are often more successful in social relationships and long-term goals due to their greater reliability, thoughtfulness and consistency.

Sensation Seeking:

The Sensation Seeking trait is characterized by the need for varied, novel, complex and intense situations and experiences. An individual with a strong SS trait may feel bored if they go a long period of time without nothing thrilling to do. These people are far more likely to do things that others might consider "crazy", and this can range all the way from climbing mountains to exploring recreational drugs, depending on the individual. In contrast, people with a low score on SS find higher leves of arousal unpleasant, and steer away from intense experiences, often enjoying routine and predictableness.

Now that we know, on a high level, what is in our dataset, it's a good idea to take a closer look at each feature so that we understand the format and distribution of our data for each column.

3. Data Exploration

To begin with, let's import the data and print out the first few columns just to get an overview.

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

dataRaw = pd.read_csv('Drug_Consumption.csv')

pd.set_option('display.max_columns', None)
dataRaw.head(5)
```

```
C:\Users\maxsz\AppData\Local\Temp\ipykernel_23052\2321357041.py:1: DeprecationWar
ning:
Pyarrow will become a required dependency of pandas in the next major release of
pandas (pandas 3.0),
(to allow more performant data types, such as the Arrow string type, and better i
nteroperability with other libraries)
but was not found to be installed on your system.
If this would cause problems for you,
please provide us feedback at https://github.com/pandas-dev/pandas/issues/54466

  import pandas as pd
```

Out[ ]:

	ID	Age	Gender	Education	Country	Ethnicity	Nscore	Escore	Oscore	ASc
0	2	25-34	M	Doctorate degree	UK	White	-0.67825	1.93886	1.43533	0.76
1	3	35-44	M	Professional certificate/ diploma	UK	White	-0.46725	0.80523	-0.84732	-1.62
2	4	18-24	F	Masters degree	UK	White	-0.14882	-0.80615	-0.01928	0.5
3	5	35-44	F	Doctorate degree	UK	White	0.73545	-1.63340	-0.45174	-0.30
4	6	65+	F	Left school at 18 years	Canada	White	-0.67825	-0.30033	-1.55521	2.03

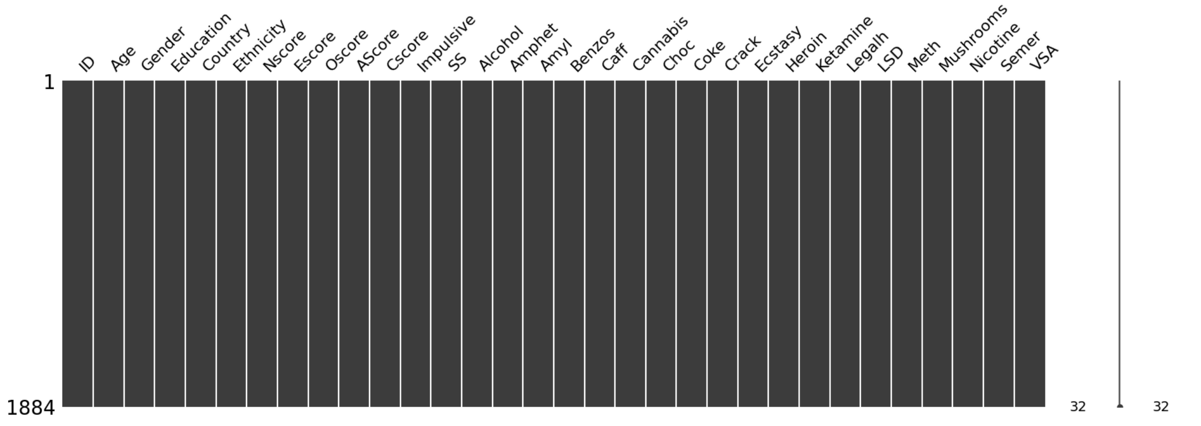
First observations:

- The Age is given as ranges and not specific values
- So far, the Gender is M or F. Maybe someone in the dataset didn't specify? We can take a look further on.
- Education is what you would expect.
- The Personality Trait scores look a lot like Standard Deviation values, given their observed range in these few rows. We will take a closer look later.
- The drug consumption is measured from CL0 to CL6, as specified before. This might require a transformation later.

One of the first things worth looking into is whether our data has missing values or not. In the data set description, it said there were none, but let's just see it with our own eyes.

In [ ]:
```python
import missingno as mno

mno.matrix(dataRaw, figsize = (20, 6))
```

Out[ ]:
```
<Axes: >
```



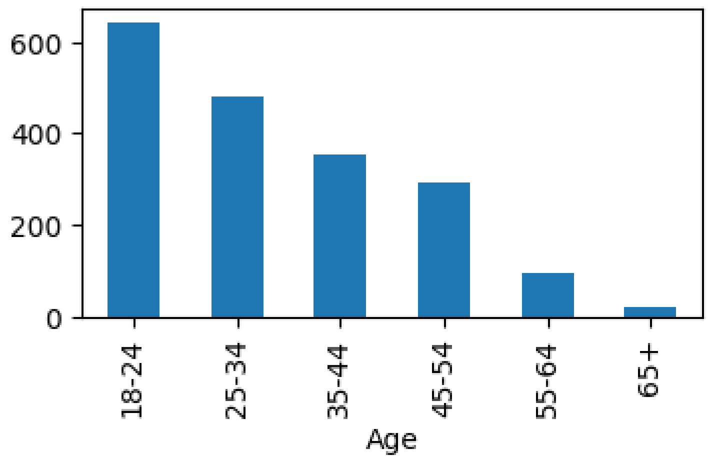
Nice, all filled out!

Age

Let's see what is the distribution of Age ranges in our data.

```
In [ ]: just_ages = dataRaw['Age']

        plt.figure(figsize=(4,2))
        just_ages.value_counts().plot(kind='bar')
```

```
Out[ ]: <Axes: xlabel='Age'>
```



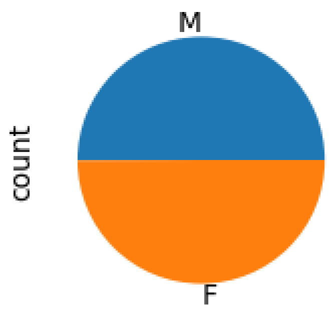
Okay, it looks like we have a considerable amount of young people in our data set! This is good because of what was discussed in Analysis - since people in their twenties are at a higher risk of Substance Use Disorder, they are the one who have probably done the most drugs recently, so each data point will be more valuable than one that is not a regular user of many substances.

Gender

Let's see the distribution of our data points in regards to gender, and whether or not we have a Gender value other than M or F.

```
In [ ]: just_gender = dataRaw['Gender']

        plt.figure(figsize=(4,2))
        just_gender.value_counts().plot(kind='pie')
```

```
Out[ ]: <Axes: ylabel='count'>
```



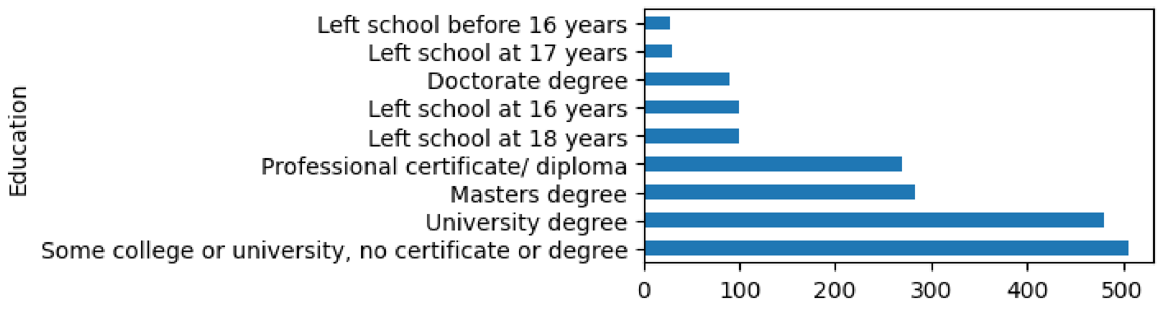
Oh my, that looks like a near-perfect split! And we don't have any value other than M or F, which makes things a bit simpler. This is good. We have a considerable sample size for both genders, meaning both are properly represented.

Education

How many people do we have for each Education level? Let's see.

```
In [ ]: just_education = dataRaw['Education']

        plt.figure(figsize=(4,2))
        just_education.value_counts().plot(kind='barh')
```

```
Out[ ]: <Axes: ylabel='Education'>
```



Okay okay! It looks like we have a pretty diverse dataset, ranging from people who are barely educated (all of those who left school), people who are fairly educated (got a diploma, or did some college/university), and people who are very educated (finished university, got a master's degree, or even a doctorate degree).

Since some of these categories are pretty close to each other, we could consider merging some of them - for example, in the way I described above.

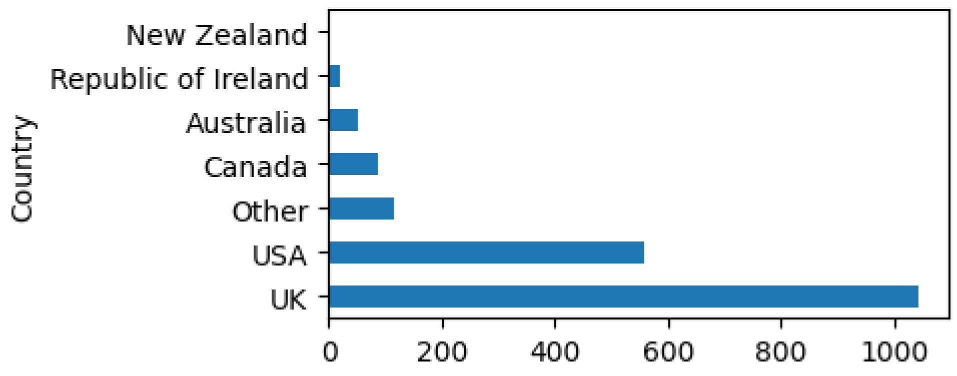
Either way, this diversity is good - by not having people of only one education level, our predictions can encompass a broader range of individuals.

Country

The people in our data set - from what countries are they from? Let's see.

```
In [ ]: just_country = dataRaw['Country']

        plt.figure(figsize=(4,2))
        just_country.value_counts().plot(kind='barh')
```

```
Out[ ]: <Axes: ylabel='Country'>
```



It looks like we mostly have people from English-speaking countries, featuring a large majority (over half of the data set) from the United Kingdom, and a whopping one third from the USA. We also have a couple data points from the continent of Oceania. We hope that the respondents from the UK will be a good representation of European citizens' tendencies.

Our data seems to cover at least three continents - with this much diversity, we hope that our model will be able to find patterns that go beyond the person's national origin.

One thing that is worth keeping in mind is the ease of access to illicit substances across different countries.

Given that our main countries are the UK and the USA, we did some research about the Drug Scene in these countries and found the following:

UK Drug Scene

According to the 2019 Report of the Drug Situation in the United Kingdom, issued by the UK Government itself , the most prevalent drugs in the UK are:

- Cannabis
- (Powder) Cocaine
- MDMA
- Ketamine
- Amphetamine
- Opioids
- Benzodiazepines

Access the report here: https://www.gov.uk/government/publications/united-kingdom-drug-situation-focal-point-annual-report/uk-drug-situation-2019-summary

It is also worth noting that opioids were impled in 80% of the illicit-drug-related deaths in the UK. For this reason, this is a very high-risk type of drug for which awareness should be raised. For this reason, we will keep a closer eye on opioids during our research. The opioids we have in our data set are Heroin and Methadone.

The UK Drug Situation Report reinforced the fact that young people are especially vulnerable to substance abuse.

The report also states that the UK is one of the countries in Europe with the highest level of drug problems. In a way, this is good for us, because since UK respondents make for such a high percentage of our data, we will get very representative information - given that our chosen method of measuring performance was Recall, having the UK, a country with a lot of drug use, will probably increase our odds of finding all the actual positives.

USA Drug Scene

According to *the 2020 SAMSHA National Survey on Drug Abuse and Mental Health*, here's what we found about the Illicit Drug Scene in the USA in recent years:

The most prevalent addictive substances in the USA are, in order:

- Alcohol
- Nicotine
- Marijuana
- Opioids
- Inhalants
- Cocaine
- Stimulants
- Benzodiazepines

Nicotine addictions are a serious problem, particularly among young people due to the recent rise and ease of access to vaping devices.

Cannabis use in the US is very concerning among young people, with nearly a third of last-year high-schoolers reporting having used marijuana in the past year.

Just like in the UK, opioids are extremely lethal and addictive.

After these findings about the two most representative countries in our data, these are the drugs we will keep a close eye on when moving forward with our analysis and predictions:

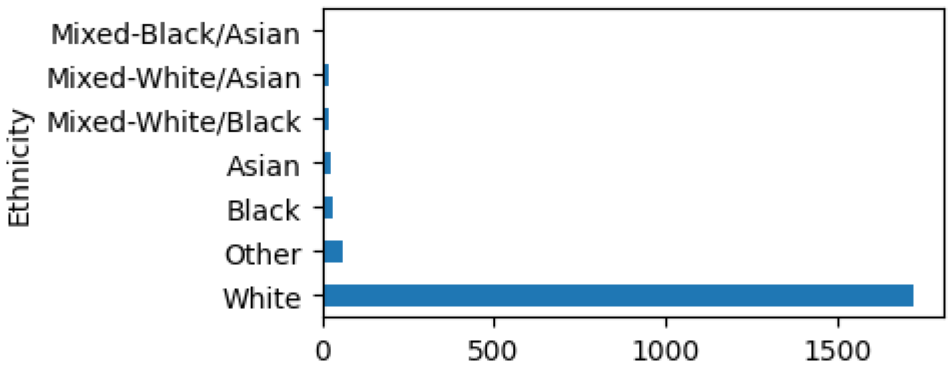
- **Opioids**: Because of the high letality of this type of drug, allied to its widespread use.
- **Cannabis**: Because of its consistently high use across virtually all countries, which is particularly concerning among young people due to the long-term effects in brains that are far from being fully developed.
- **Nicotine**: Young people are particularly vulnerable to nicotine addicitons because the younger one is first exposed to the drug, the harder it is to let go of an addiction. If it is truly possible to predict a nicotine addiction based on personality traits, it is crucial to do so and prevent it.
- **Cocaine**: Cocaine lethalities are on the rise, and addiction remains a problem. This drug is widely used and abused, so this is something to tackle.

Ethnicity

We have looked at the countries where our respondents live, but what about their ethnicity? Could this be a factor to consider as well when making predictions? Let's see what we're working with, and maybe do a little more research if deemed relevant.

```
In [ ]: just_ethnicity = dataRaw['Ethnicity']

        plt.figure(figsize=(4,2))
        just_ethnicity.value_counts().plot(kind='barh')
```

```
Out[ ]: <Axes: ylabel='Ethnicity'>
```



It looks like there is an overwhelming majority of white respondents, so we probably won't be able to tell whether or not ethnicity is a factor in drug abuse patterns - at least, not from our data. Given the little diversity in this column, it might be worth dropping.

Personality Traits

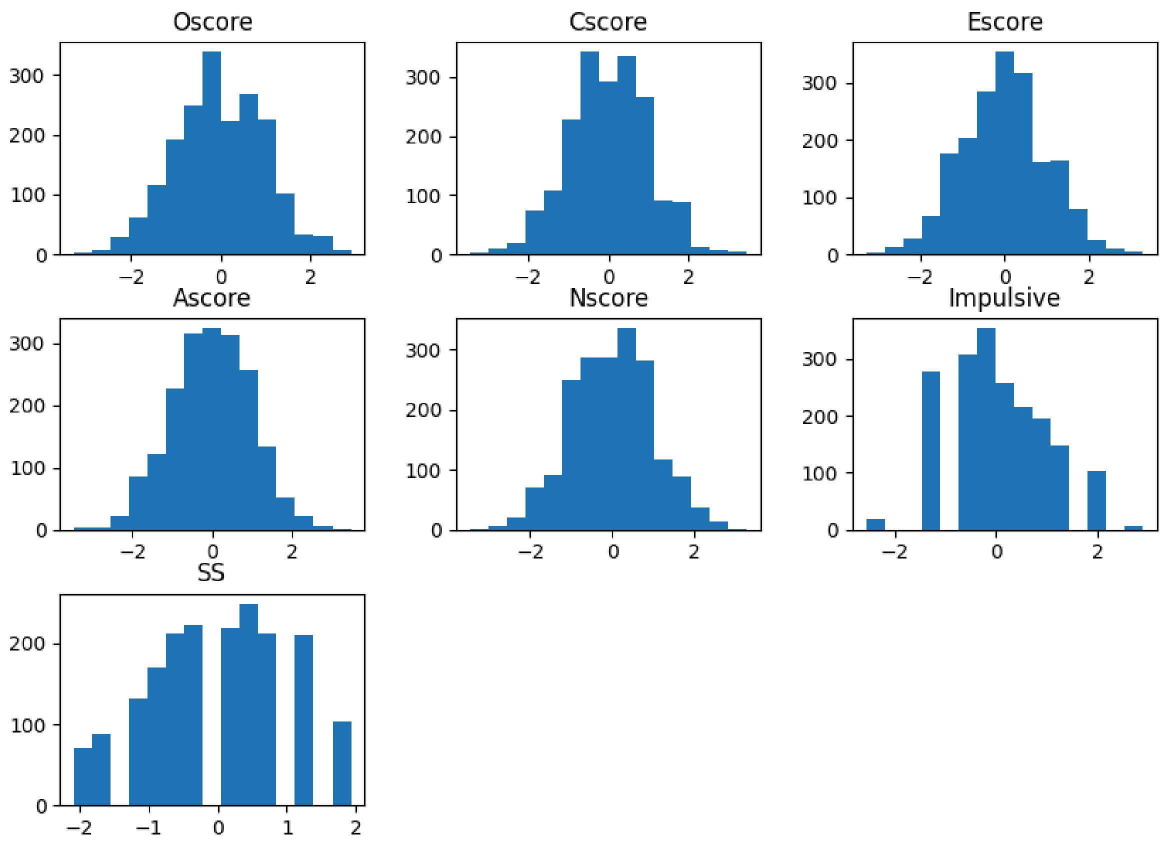
In the data set we took from the Internet, we saw no information about what scale the Personality Trait scores are on. So, we made a guess that they were standard deviation values - we imagined that whoever collected the data made a Personality Test for all respondents, got all their scores for each trait and transformed them into the std value they displayed in the context of the data set.

We can now test this by visualizing these columns - if they form a normal distribution curve, our guess can be deemed as correct. Let's see.

```
In [ ]: # I just noticed that all the Big-Five personality trait columns
        # have 'score' written in lowercase except for Agreeableness,
        # which has it capitalized, so I'll just correct that on the
        # original DataFrame for consistency.

        dataRaw.rename(columns={'AScore': 'Ascore'}, inplace='true')
```

```
In [ ]: personality_cols = ['Oscore', 'Cscore', 'Escore', 'Ascore', 'Nscore', 'Impulsive
        personality_only = dataRaw[personality_cols]

        personality_only.hist(bins=15, figsize=(10,7), grid=False)
```

```
Out[ ]: array([[<Axes: title={'center': 'Oscore'}>,
                <Axes: title={'center': 'Cscore'}>,
                <Axes: title={'center': 'Escore'}>],
               [<Axes: title={'center': 'Ascore'}>,
                <Axes: title={'center': 'Nscore'}>,
                <Axes: title={'center': 'Impulsive'}>],
               [<Axes: title={'center': 'SS'}>, <Axes: >, <Axes: >]], dtype=object)
```



Interesting! The Big-Five values seem pretty normally distributed, and while the scores for Impulsiveness and Sensation Seeking still look fairly equally distributed, they look a bit odd, not to mention they have a narrower range than the other personality features. This could be because they are not part of the same Personality Analysis theory, so they are measured differently. We will take a closer look in just a moment. First, let's also use the .describe() function on this subset of features to see what we can find out.

```
In [ ]: personality_only.describe()
```

Out[ ]:

	Oscore	Cscore	Escore	Ascore	Nscore	Impulsive	
count	1884.000000	1884.000000	1884.000000	1884.000000	1884.000000	1884.000000	1
mean	-0.000225	-0.000383	0.000143	0.000242	-0.000119	0.007335	
std	0.996402	0.997787	0.997625	0.997481	0.998345	0.954674	
min	-3.273930	-3.464360	-3.273930	-3.464360	-3.464360	-2.555240	
25%	-0.717270	-0.652530	-0.695090	-0.606330	-0.678250	-0.711260	
50%	-0.019280	-0.006650	0.003320	-0.017290	0.042570	-0.217120	
75%	0.723300	0.584890	0.637790	0.760960	0.629670	0.529750	
max	2.901610	3.464360	3.273930	3.464360	3.273930	2.901610	

Honestly, that's kind of beautiful. Here is some evidence that our Personality Traits data is normally distributed:

- The mean value of all features approximates to zero
- All features have approximately the same standard deviation value, even if some of their ranges differ

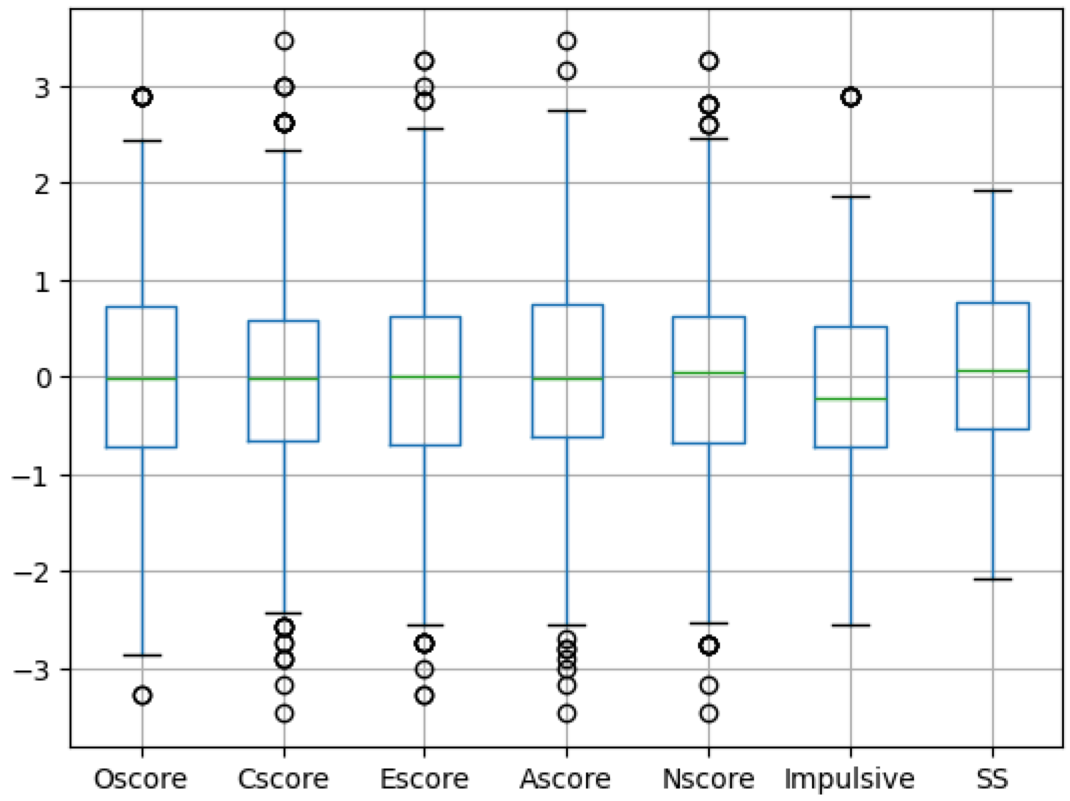
These are our (rounded) observed ranges for all Personality Trait subsets:

- Big-Five features: [-3.5, +3.5]
- Impulsiveness: [-2.5, +3.0]
- Sensation Seeking: [-2.0, +2.0]

Let's try putting those on a boxplot graph.

In [ ]:
```
personality_only.boxplot()
```

Out[ ]:
```
<Axes: >
```



Nice! This represents graphically what we said before that the Big Five traits are normally distributed as their mean is right in the middle and their quartiles are also a proof for that. However, we can see that there are some outliers in those traits. We decide to keep those as they are as we believe that they describe the actual situation in the world. Meaning that there are people who could and did score extremely on those metrics.

Moving to the Impulsive and SS columns, we can see that they are not so nicely distributed but we already mentioned that. We also decide to keep the outliers in the Impulsive columns.

Could it also be interesting to see how many unique values exist in each of these columns? Let's see.

In [ ]:
```
personality_only.nunique()
```

Out[ ]:
```
Oscore       35
Cscore       41
Escore       42
Ascore       41
Nscore       49
Impulsive    10
SS           11
dtype: int64
```

Well, that sure gives us a hint on why the distribution of the Impulsive and SS values looked different than the rest!

According to the printed output above, the Big-Five scores are measured a lot more continuously than the Impulsive and SS - in other words, the Big-Five contain "more boxes".

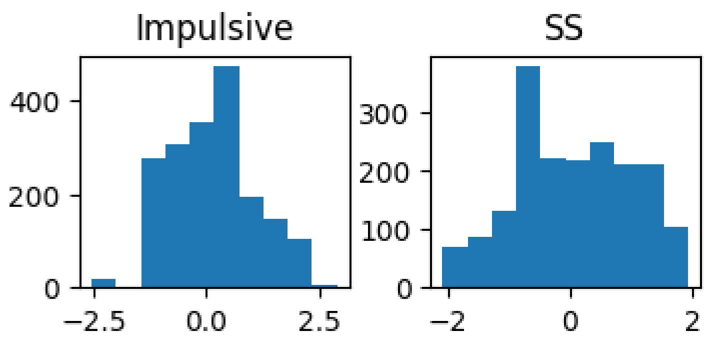
From this, we can imagine the Big-Five original scores to be, for example, all integer numbers between -20 and 20 (rounded for simplicity) and the Imp/SS scores to be something between -5 and 5.

Knowing this, let's plot Imp and SS again, but with even less "bins" to see if they look "more normal".

```
In [ ]: odd_cols = ['Impulsive', 'SS']

        imp_and_ss = dataRaw[odd_cols]
```

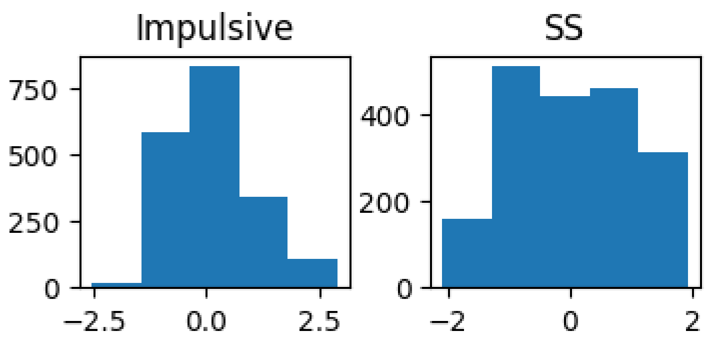
```
In [ ]: # 10 bins
        imp_and_ss.hist(bins=10, figsize=(4,1.5), grid=False)
```

```
Out[ ]: array([[<Axes: title={'center': 'Impulsive'}>,
                <Axes: title={'center': 'SS'}>]], dtype=object)
```



```
In [ ]: # 5 bins
        imp_and_ss.hist(bins=5, figsize=(4,1.5), grid=False)
```

```
Out[ ]: array([[<Axes: title={'center': 'Impulsive'}>,
                <Axes: title={'center': 'SS'}>]], dtype=object)
```



Okay - Impulsiveness looks pretty normally distributed when we give it the number of "bins" that its data was originally split in, but SS still stands out.

It looks like there is an unusually large amount of respondents scoring between -1.2 and -0.8. What now? Do we trust the data and assume that our respondents were just not very Sensation Seeking given the small sample size, or do we readjust it to have a normal distribution like the other features?

For now, we will leave it as it is.

Having observed all of this, we conclude that the Personality Trait scores follow a normal distribution, and can move on with our analysis.

Substances

Substances, substances... There are so many questions that can be asked with the information that we have. Let's break them down:

- Given that our desired label is "user" or "non-user" (as opposed to CL0-CL6 like we have now), when is it a good idea to transform our data into our desired label format?

(Let's first see the frequency tendencies we see with each substance and then go from there.)

- What drugs do people consume on a daily-basis?
- What drugs do people consume on a weekly-basis?
- What drugs have our respondents only consumed in the past year?
- And in the past decade?
- And in over a decade?
- How many people have never tried each drug?

(Now would be a good time to transform CL0-CL6 to user/non-user.)

- What drugs have the most and the least users?
- Are there any drugs in our data set that are so rare that they are not even worth adding to our predictions?

Maybe more questions will pop up as we visualize the data. Let's see.

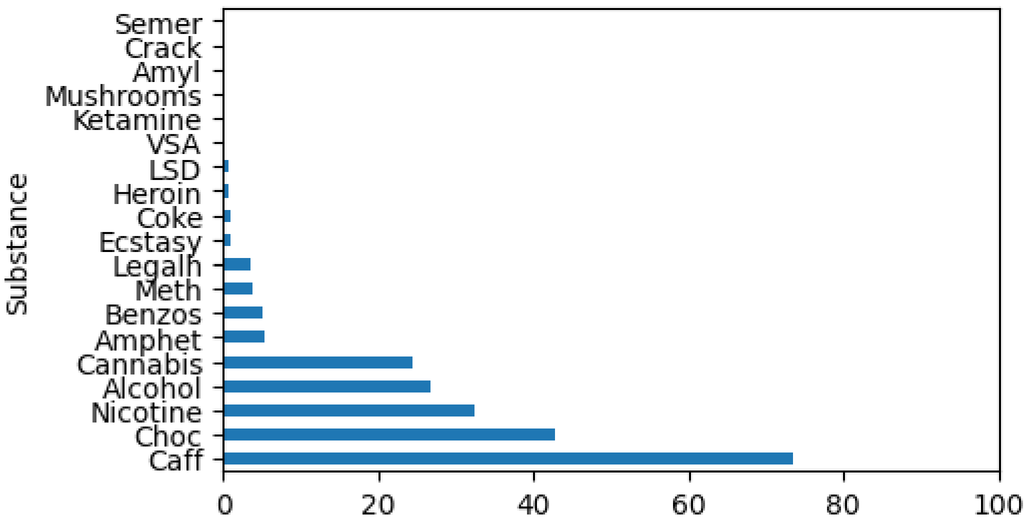
In [ ]:
```python
drugs_cols = ['Alcohol', 'Amphet', 'Amyl', 'Benzos', 'Caff', 'Cannabis', 'Choc',

drugs_only = dataRaw[drugs_cols]
```

In [ ]:
```python
# Define a function that prints out a certain substance's frequency

# Function that calculates the number of people that have consumed
# a certain drug for a certain given frequency.
# (fx. 'Alcohol', 'CL6' would output how many people consumed
# alcohol in the past day (before the survey).)
def calc_and_print_frequency(colname, freq):
    try:
        count = drugs_only[colname].value_counts()[freq]
    except:
        count = 0
    pct = round(count / 1884 * 100, 2)
    print_freq(colname,freq,count,pct)

# Support function for the function above, just to give the
# string output and print it in a readable manner.
def print_freq(colname, freq, count, pct):
    print(colname + " " + cl_to_words(freq) + str(count) + " \t(" + str(pct) + "
# Function that calculates the number & percentage of people
# that have consumed a certain drug for a certain given frequency.
# The "main" frequency is the first given in the array.
# If you give more than one frequency, this function
# will calculate, cumulatively, how many people have used this drug.
# For example: If you pass, as an argument:
    # the data set (obviously)
    # and freq = ['CL5', 'CL6']
    # Then the "main frequency" is CL5 (past week), so you'll
    # get the number of people who have only taken the drug
    # in the past week, but you will also get the CUMULATIVE
    # value, a.k.a. all the people with CL5 OR CL6 for this drug.
    # Because this sums how many did it in the past week but also
    # in the past day.
    # This is why there are columns "NumOfPeople"/"PctOfPeople"
    # (for the main frequency) - in our example, "past week"
    # and columns "NumOfPeopleCum/PctOfPeopleCum"
    # (for the cumulative frequency) - in our example, "past week" + "past day"


def make_cumulative_frequency_df(data, freq):
    pct_by_freq = pd.DataFrame({'Substance': [],
                                'NumOfPeople': [],
                                'NumOfPeopleCum': [],
                                'PctOfPeople': [],
                                'PctOfPeopleCum': []})
    i = 0
    for col in drugs_cols:
        try:
            count = 0
            cumCount = 0
            j = 0
            for f in freq:
                if (j == 0):
                    count = count + data[col].value_counts()[f]
                cumCount = cumCount + data[col].value_counts()[f]
                j = j+1

        except:
            count = 0
        pct = round(count / 1884 * 100, 2)
        pctCum = round(cumCount / 1884 * 100, 2)
        obj = {'Substance': col,
               'NumOfPeople': count,
               'NumOfPeopleCum': cumCount,
               'PctOfPeople': pct,
               'PctOfPeopleCum': pctCum}
        pct_by_freq.loc[i] = obj
        i = i + 1

    pct_by_freq.sort_values('PctOfPeople', ascending=False, inplace=True)
    pct_by_freq.set_index('Substance', inplace=True)
    return pct_by_freq

# Kind of dictionary function just to make the
# CL ratings readable to a human:)
def cl_to_words(cl):
    switcher = {
        'CL0': "Never Attempted: ",
        '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: "
    }
    return switcher.get(cl)
```

What substances have these people consumed the **DAY** before the survey?

In [ ]:
```
daily = make_cumulative_frequency_df(drugs_only, ['CL6'])

daily_pct = daily['PctOfPeople']

daily_pct.plot.barh(figsize=(5,3), xlim=[0,100])
```

Out[ ]:
```
<Axes: ylabel='Substance'>
```



In [ ]:
```
daily[['NumOfPeople', 'PctOfPeople']]
```

Out[ ]:

Substance	NumOfPeople	PctOfPeople
Caff	1384	73.46
Choc	807	42.83
Nicotine	610	32.38
Alcohol	505	26.80
Cannabis	463	24.58
Amphet	102	5.41
Benzos	95	5.04
Meth	73	3.87
Legalh	67	3.56
Ecstasy	21	1.11
Coke	19	1.01
Heroin	13	0.69
LSD	13	0.69
VSA	7	0.37
Ketamine	4	0.21
Mushrooms	4	0.21
Amyl	3	0.16
Crack	2	0.11
Semer	0	0.00

Very interesting numbers! Observations:

__In the 24 hours before the survey:__

Significant consumption:

- Three fourths **(75%)** of the people drank **coffee**.
- **42%** of the people ate **chocolate**.
- A third **(32%)** of the people consumed **nicotine**.
- A fourth **(25%)** of the people drank **alcohol**.
- Also, a fourth **(25%)** of the people smoked **cannabis**.

The most consumed substances on a daily basis are the legal ones, except for cannabis, which, despite being illegal in many places, still scored very high.

__Noticeable consumption:__

- Amphetamines (102 people, 5%)

- Benzodiazepines (95 people, 5%)
- Legal Highs (67 people, 3%)
- Methadone (73 people, 4%)

This was interesting to notice. Apart from legal highs, the second "level" of drugs that are being consumed on the daily basis are those that, despite being illegal recreationally, can be prescribed medically (amphetamines for ADHD, benzodiazepines for anxiety/insomnia, and methadone for pain relief and Opioid Use Disorder). Unfortunately, we have no data on how each respondent acquired each drug they have consumed, but we can assume with fair confidence that the trend for these drugs to be higher on the list is that they are being acquired in a medical context.

Low consumption:

- Ecstasy (21 people, 1.11%)
- Coke (19 people, 1.01%)
- LSD (13 people, 0.69%)
- Ketamine (13 people, 0.69%)
- Volatile Substances (7 people, 0.37%)
- Mushrooms (4 people, 0.21%)
- Amyl Nitrite (3 people, 0.16%)
- Heroin (3 people, 0.16%)
- Crack (2 people, 0.11%)

Although some of these substances are a bit hardcore, there was always at least someone who had used them in the past day. That's wild, but it means we got a good group of respondents. (Because the ideal group of respondents would range from super sober people to people who are straight up unhinged.)

No consumption at all:

- Semer (0 people)

It's very appropriate that no one answered that they had done Semer in the past day - since this drug is fictional. It would have been hilarious if someone had claimed that, but no. Maybe with a bigger sample size. Anyway. :)

What substances have these people consumed the **WEEK** before the survey?

When moving on from Day to Week, there is a question we must pose - will we look at the percentage of people who have cumulatively consumed it in the past week (past week + past day), or will we look at people who have used in the past week, but not in the past day?

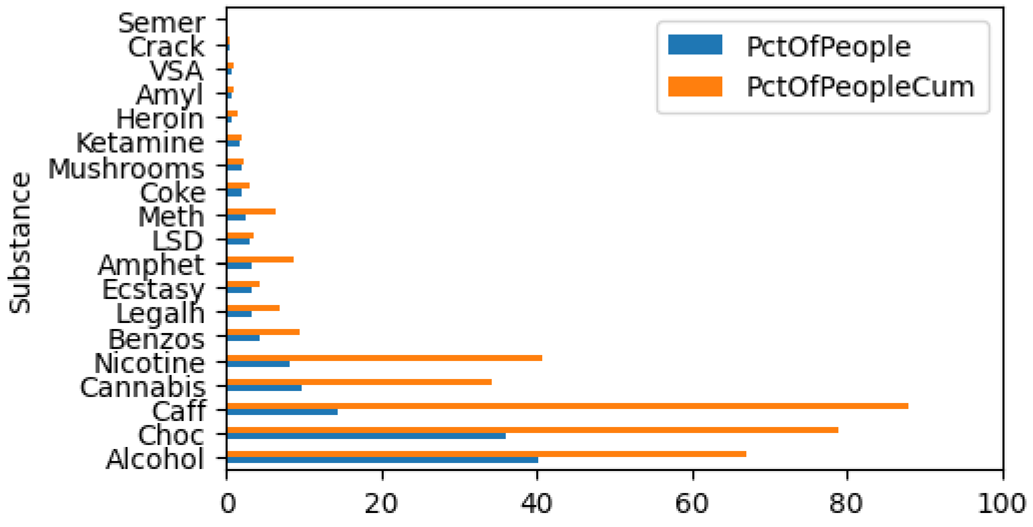
Both of these ways of looking at the data are interesting, so we modified our people-counting function to accomodate for this. In the following graphs, we will see two bars:

- One for only the given frequency (in this case, 'CL5')

- One for the cumulative frequency (in this case, 'CL5' + 'CL6').

In [ ]:
```
weekly = make_cumulative_frequency_df(drugs_only, ['CL5', 'CL6'])

weekly_pct = weekly[['PctOfPeople', 'PctOfPeopleCum']]

weekly_pct.plot.barh(figsize=(5,3), xlim=[0,100])
```

Out[ ]:
```
<Axes: ylabel='Substance'>
```



In [ ]:
```
weekly
```

Out[ ]:

	NumOfPeople	NumOfPeopleCum	PctOfPeople	PctOfPeopleCum
Substance				
Alcohol	758	1263	40.23	67.04
Choc	682	1489	36.20	79.03
Caff	273	1657	14.49	87.95
Cannabis	185	648	9.82	34.39
Nicotine	157	767	8.33	40.71
Benzos	84	179	4.46	9.50
Legalh	64	131	3.40	6.95
Ecstasy	63	84	3.34	4.46
Amphet	61	163	3.24	8.65
LSD	56	69	2.97	3.66
Meth	48	121	2.55	6.42
Coke	41	60	2.18	3.18
Mushrooms	40	44	2.12	2.34
Ketamine	33	37	1.75	1.96
Heroin	16	29	0.85	1.54
Amyl	14	17	0.74	0.90
VSA	14	21	0.74	1.11
Crack	9	11	0.48	0.58
Semer	0	0	0.00	0.00

Observations:

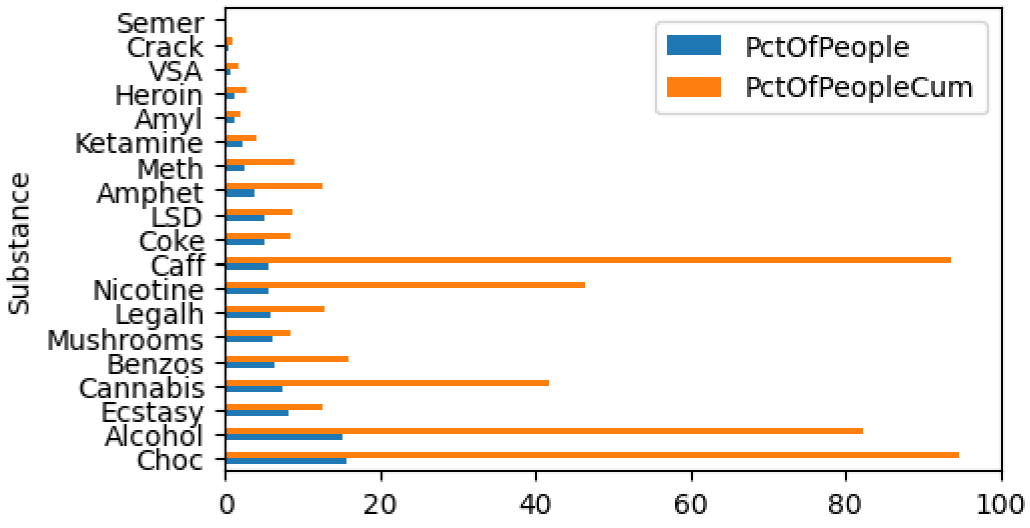
- Alcohol climbed to the top of the weekly table. We can infer from this that perhaps people are not consuming it on a daily basis, but rather on a weekly occasion - probably the weekend.
- Chocolate stayed strong at second place - it seems that a large chunk of our respondents have a sweet tooth and cannot resist this treat for longer than a week.
- Caffeine went down by two places, but cumulatively, the percentage of people who have consumed it either in the past day or week becomes (74 + 15 = 89%), making it, by far, the most regularly consumed substance.
- Cannabis cumulative consumption in the past week got bumped up to 34% - could it be that a third of our respondents get high regularly?
- The rest of the drugs got bumped up by just a little bit - between 84 (for Benzodiazepine) and 9 for Crack Cocaine.
  - It looks like 1 out of 10 of our respondents are taking Benzodiazepine on a weekly basis.

- Still, no one has claimed to have used Semeron in the past week. We are curious as to when someone will start making that claim.

What substances have these people consumed the **MONTH** before the survey?

```
In [ ]: monthly = make_cumulative_frequency_df(drugs_only, ['CL4', 'CL5', 'CL6'])

        monthly_pct = monthly[['PctOfPeople', 'PctOfPeopleCum']]

        monthly_pct.plot.barh(figsize=(5,3), xlim=[0,100])
```

```
Out[ ]: <Axes: ylabel='Substance'>
```



```
In [ ]: monthly
```

Out[ ]:

	NumOfPeople	NumOfPeopleCum	PctOfPeople	PctOfPeopleCum
Substance				
Choc	296	1785	15.71	94.75
Alcohol	287	1550	15.23	82.27
Ecstasy	156	240	8.28	12.74
Cannabis	140	788	7.43	41.83
Benzos	120	299	6.37	15.87
Mushrooms	115	159	6.10	8.44
Legalh	110	241	5.84	12.79
Nicotine	108	875	5.73	46.44
Caff	106	1763	5.63	93.58
Coke	99	159	5.25	8.44
LSD	97	166	5.15	8.81
Amphet	75	238	3.98	12.63
Meth	50	171	2.65	9.08
Ketamine	42	79	2.23	4.19
Amyl	24	41	1.27	2.18
Heroin	24	53	1.27	2.81
VSA	13	34	0.69	1.80
Crack	9	20	0.48	1.06
Semer	0	1	0.00	0.05

How interesting!!! Here are our findings so far:

Substances that almost everyone is consuming:

- Chocolate (95%)
  - 16% of people can keep it down to a few times a month
  - But ultimately, as much as 95% of our respondents have eaten chocolate in the past month!!
- Caffeine (93%)
  - Only 6% out of 93% drank coffee in the past month but not in the past week/day. This substance remains the most regularly consumed.
- Alcohol (82%)
  - 15% of people can keep it down to a few times a month
  - But 82% of our respondents have drank in the past month - that's 4 out of 5 people! Everyone is drinking!

Substances that people are taking not so much on a weekly basis, but rather monthly:

- Ecstasy
- Mushrooms
- Coke
- LSD

(By this, we mean the substances that got a significant bump (the percentage doubled or more, nearing 10%) from the CL4 (past month) score).

So far, the substances we must keep an eye on are:

- **Widely used drugs (> 40%)**
  - Alcohol (1550 people, 82%)
  - Nicotine (855 people, 46%)
  - Cannabis (788 people, 42%)

Caffeine and Chocolate don't pose as much risk to Substance Use Disorder so they do not make the list.

- **Prescription drugs & legal highs (9-16%)**
  - Benzodiazepines (299 people, 16%)
  - Legal Highs (241 people, 13%)
  - Amphetamines (238 people, 13%)
  - Methadone (171 people, 9%)
- **Party / psychedelic drugs (8-13%)**
  - Ecstasy (240 people, 13%)
  - LSD (166 people, 9%)
  - Mushrooms (159 people, 8%)
  - Coke (159 people, 8%)

These substances are being used fairly regularly on a monthly basis by our respondents so we have plenty of data on them. From our list, here is a harm assessment for the drugs that made it to our "watchlist", in order of risk:

- Alcohol
  - Direct effects: Cancer, liver damage, hypertension, heart disease, and fetal damage
  - Behavioral effects: alcohol abuse increases the risk of suicide, violence, and motor accidents
  - Combined with how widespread and socially accepted its use is, alcohol is deemed as a fairly dangerous substance.
- Methadone
  - Risk of overdose
  - Dangerous to combine with other substances (fx Benzodiazepines)
- Coke

    - Heart disease, hypertension, organ failure, respiratory distress, stroke, unhealthy weight loss, and seizures
    - Overdose deaths are common
- Benzodiazepines
    - Risk due to widespread availability
    - Sleeping problems, hallucinations, anxiety, and life-threatening seizures
    - Combining with other subsdtances increases the risk of a fatal overdose
- Nicotine
    - Extremely addictive
    - Tobacco in particular: harms nearly every organ in the body, can lead to premature death, and can increase the risk of stroke.
- Ecstasy
    - Often "cut" with other substances, which increases risk of overdosing substantially
    - Increase of heart rate, sharp rise in body temperature, kidney failure, death
    - Continued use can lead to psychological or physical dependence
- Amphetamines
    - Long-term use can damage the brain and the cardiovascular system and may lead to psychosis, malnutrition and erratic behaviour

Cannabis, LSD and Mushrooms are recreational drugs with a significantly lower risk than the substances aforementioned. Their toll on the body is far lower and their risk of physical addiction is far lower (David Nutt, 2015). That is not to say that the risk is non-existent - people have also gotten *psychologically* addicted to these substances, and cannabis, when consumed by young people, can produce effects such as cognitive deficiencies on a developing brain. (German Lopez, 2015) Still, they are far less likely to be life-threatening. If used with caution, the risk of these substances drops even lower.

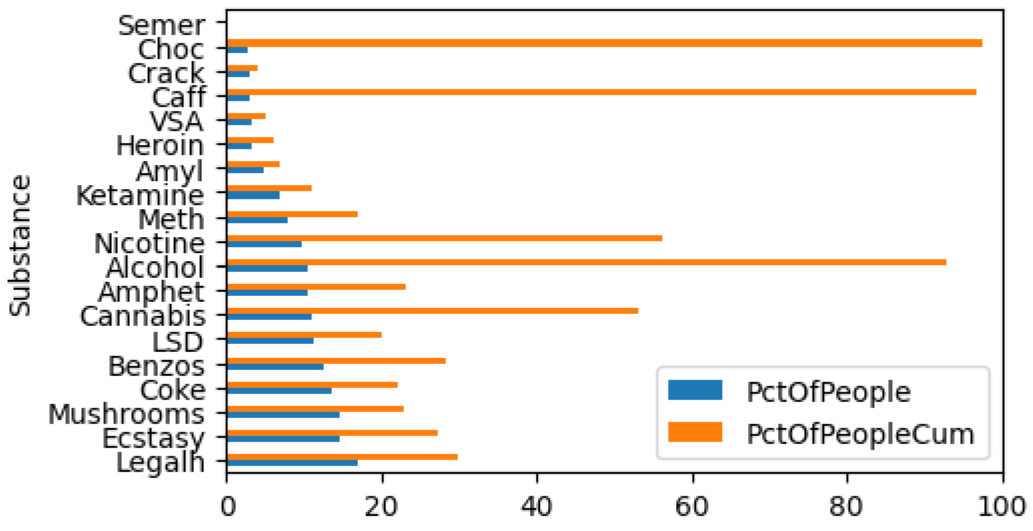
(Sources:

- https://www.addictioncenter.com/news/2019/08/15-most-dangerous-drugs/
- https://www.betterhealth.vic.gov.au/health/healthyliving/amphetamines
- https://www.vox.com/2015/2/24/8094759/alcohol-marijuana
- https://www.amazon.com/Drugs-without-hot-air-illegal/dp/0857844946)

What substances have these people consumed the **YEAR** before the survey?

In [ ]:
```
yearly = make_cumulative_frequency_df(drugs_only, ['CL3', 'CL4', 'CL5', 'CL6'])

yearly_pct = yearly[['PctOfPeople', 'PctOfPeopleCum']]

yearly_pct.plot.barh(figsize=(5,3), xlim=[0,100])
```

Out[ ]:
```
<Axes: ylabel='Substance'>
```



```
In [ ]: yearly
```

Out[ ]:

Substance	NumOfPeople	NumOfPeopleCum	PctOfPeople	PctOfPeopleCum
Legalh	323	564	17.14	29.94
Ecstasy	277	517	14.70	27.44
Mushrooms	275	434	14.60	23.04
Coke	258	417	13.69	22.13
Benzos	236	535	12.53	28.40
LSD	214	380	11.36	20.17
Cannabis	211	999	11.20	53.03
Amphet	198	436	10.51	23.14
Alcohol	198	1748	10.51	92.78
Nicotine	185	1060	9.82	56.26
Meth	149	320	7.91	16.99
Ketamine	129	208	6.85	11.04
Amyl	92	133	4.88	7.06
Heroin	65	118	3.45	6.26
VSA	61	95	3.24	5.04
Caff	60	1823	3.18	96.76
Crack	59	79	3.13	4.19
Choc	54	1839	2.87	97.61
Semer	0	3	0.00	0.16

What a significant bump for a lot of substances! Let's see:

A lot of people have not done these drugs in the past month, but have consumed them at least once in the past year:

(The list of substances below reflects those whose cumulative consumption more than doubled after considering the consumption in the past year. For example, if a substance went from 7% to 15% consumption, it makes the cut, but if it went from 30% to 40%, it would not make it.)

(The value between parenthesis is the cumulative % of people who have consumed this drug in year before the survey.)

- Legal Highs (30%)
- Ecstasy (27%)
- Mushrooms (23%)
- Coke (22%)
- LSD (20%)
- Ketamine (11%)
- Amyl Nitrite (7%)
- Heroin (6%)
- Volatile Substances (5%)

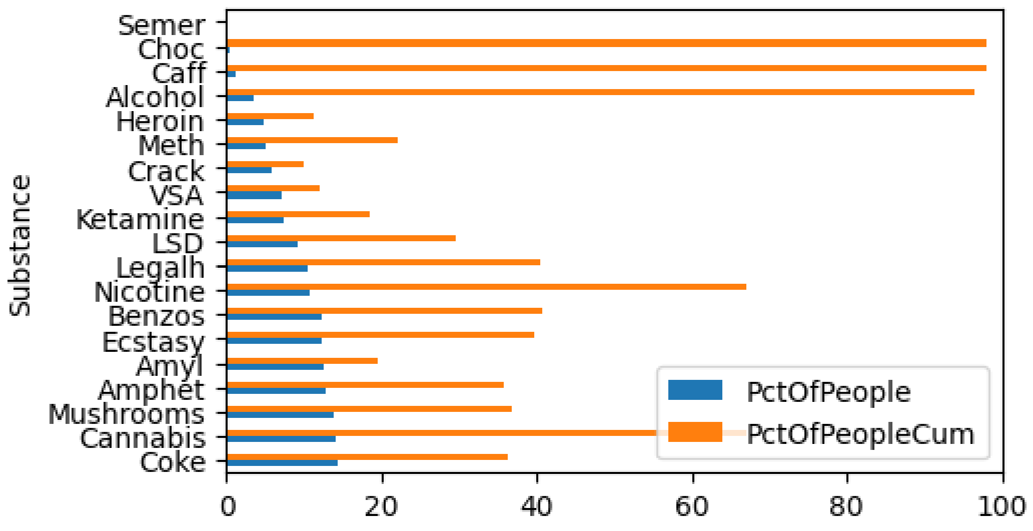
Here, we see a couple of "new faces", such as Ketamine, Amyl Nitrite, Heroin and Volatile Substances. These drugs are of a considerably high risk, particularly Heroin, so it's good that people are not doing them so often.

When we are done with analysing how many datapoints we have for each of these substances, we will weigh the amount of data we have versus the risk level of each drug, to determine if there is any substance that we will exclude from our model's predictions due to lack of sufficient data.

What substances have these people consumed the **DECADE** before the survey?

In [ ]:
```
decade = make_cumulative_frequency_df(drugs_only, ['CL2', 'CL3', 'CL4', 'CL5', '

decade_pct = decade[['PctOfPeople', 'PctOfPeopleCum']]

decade_pct.plot.barh(figsize=(5,3), xlim=[0,100])
```

Out[ ]:
```
<Axes: ylabel='Substance'>
```



```
In [ ]: decade
```

Out[ ]:

Substance	NumOfPeople	NumOfPeopleCum	PctOfPeople	PctOfPeopleCum
Coke	270	687	14.33	36.46
Cannabis	266	1265	14.12	67.14
Mushrooms	260	694	13.80	36.84
Amphet	242	678	12.85	35.99
Amyl	237	370	12.58	19.64
Ecstasy	234	751	12.42	39.86
Benzos	233	768	12.37	40.76
Nicotine	203	1263	10.77	67.04
Legalh	198	762	10.51	40.45
LSD	177	557	9.39	29.56
Ketamine	142	350	7.54	18.58
VSA	135	230	7.17	12.21
Crack	112	191	5.94	10.14
Meth	97	417	5.15	22.13
Heroin	94	212	4.99	11.25
Alcohol	68	1816	3.61	96.39
Caff	24	1847	1.27	98.04
Choc	10	1849	0.53	98.14
Semer	0	6	0.00	0.32

How interesting! For more than half of the drugs, 10% of the people have not used them in the past year, but did in the past decade.

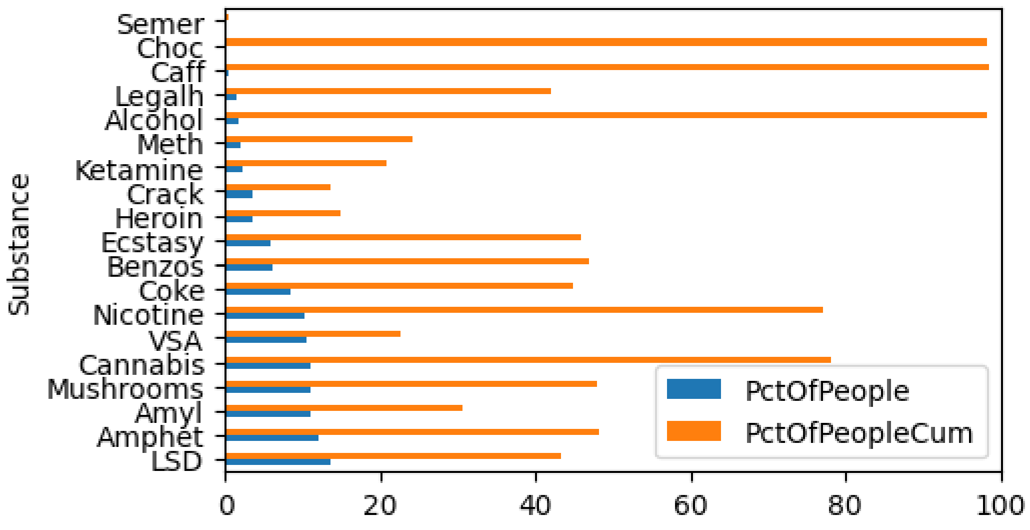
There are many ways in which we can interpret this, but for our analysis, we will say that someone who has done a drug in the past decade but not in the past year is not a user - a decade is a really long time. Imagine having tried Cocaine once at 18 and still being considered a user at 27? That doesn't sound right to us. Besides, trying a drug once isn't enough to be considered a user.

We still have no one who has claimed to have tried Semeron.

What substances have these people consumed **OVER A DECADE AGO** before the survey?

In [ ]:
```
over_decade = make_cumulative_frequency_df(drugs_only, ['CL1','CL2', 'CL3', 'CL4

over_decade_pct = over_decade[['PctOfPeople', 'PctOfPeopleCum']]

over_decade_pct.plot.barh(figsize=(5,3), xlim=[0,100])
```

Out[ ]:
```
<Axes: ylabel='Substance'>
```



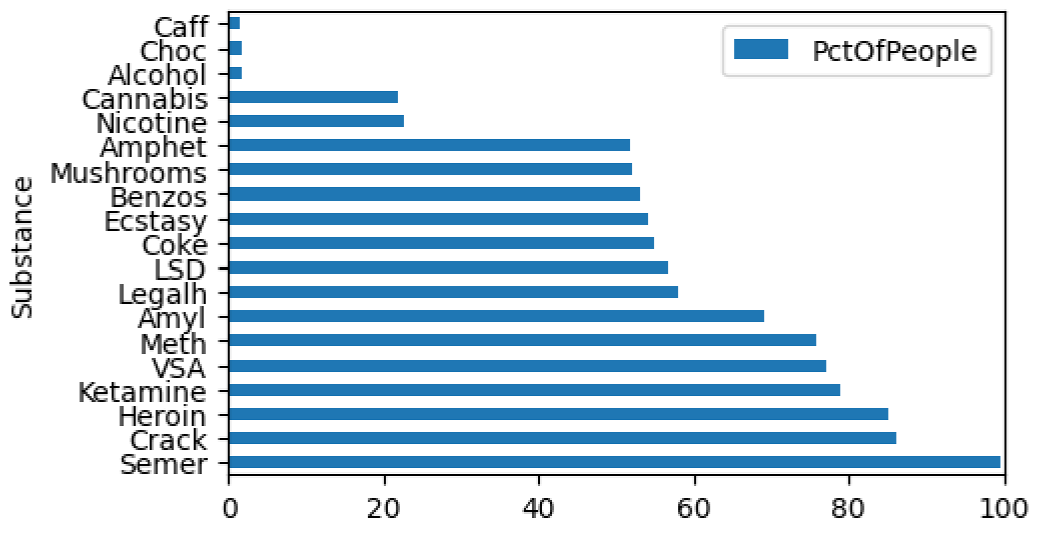
The same we said for the past decade applies here. Sure, a lot of people have tried Cocaine a long time ago, but that doesn't make them a user.

We must note that the substances that got the most significant cumulative increase from this "time slot" were Amyl Nitrite, Volatile Substances and Crack Cocaine - but since people haven't done them in a really long time, we don't deem it as a risk anymore.

What substances have these people **NEVER** consumed... before the survey?

In [ ]:
```
never = make_cumulative_frequency_df(drugs_only, ['CL0'])

never_pct = never[['PctOfPeople']]

never_pct.plot.barh(figsize=(5,3), xlim=[0,100])
```

Out[ ]:
```
<Axes: ylabel='Substance'>
```



```
In [ ]: never
```

Out[ ]:

Substance	NumOfPeople	NumOfPeopleCum	PctOfPeople	PctOfPeopleCum
Semer	1876	1876	99.58	99.58
Crack	1626	1626	86.31	86.31
Heroin	1604	1604	85.14	85.14
Ketamine	1489	1489	79.03	79.03
VSA	1454	1454	77.18	77.18
Meth	1428	1428	75.80	75.80
Amyl	1304	1304	69.21	69.21
Legalh	1093	1093	58.01	58.01
LSD	1068	1068	56.69	56.69
Coke	1037	1037	55.04	55.04
Ecstasy	1020	1020	54.14	54.14
Benzos	1000	1000	53.08	53.08
Mushrooms	981	981	52.07	52.07
Amphet	976	976	51.80	51.80
Nicotine	428	428	22.72	22.72
Cannabis	412	412	21.87	21.87
Alcohol	34	34	1.80	1.80
Choc	32	32	1.70	1.70
Caff	27	27	1.43	1.43

Yay, how interesting! Looking particularly at the graph, we can almost cluster the drugs into different categories based on how many people have never tried them before!

- Widespread Use:
  - Chocolate
  - Caffeine
  - Alcohol
- Common Use (4 out of 5 have tried):
  - Nicotine
  - Cannabis
- Half have tried:
  - Amphetamines
  - Mushrooms
  - Benzodiazepines
  - Ecstasy
  - Coke
  - LSD
- Most people haven't tried:
  - Amyl Nitrite
  - Methadone
  - Volatile Substances
  - Ketamine
  - Heroine
  - Crack Cocaine
- Fictitious - no one has tried :
  - Semeron

Users VS Non-Users

Yay, we've analysed the frequency with which each substance has been used by our respondents! Now, we just want to see the amount of users for each substance. Since "user/non-user" will be our label, seeing how many data points we have for this particular way of looking at the data will help us see if there is anything we should discard.

As mentioned before, we will only consider someone the user of a drug if they have used it in the past year. That means:

- CL6 (Past Day) - CL3 (Past Year): User (1)
- CL2 (Past Decade) - CL0 (Never): Non-User (0)

We will represent User as the integer 1 and Non-User as the integer 0.

In [ ]:
```
user_vals = ['CL6', 'CL5', 'CL4', 'CL3']
non_user_vals = ['CL2', 'CL1', 'CL0']

user_non_user = dataRaw

def transform_df(df):
    for drug in drugs_cols:
        for cat in user_vals:
            df.loc[user_non_user[drug] == cat, drug] = '1'
        for cat in non_user_vals:
            df.loc[user_non_user[drug] == cat, drug] = '0'
    return df

user_non_user = transform_df(user_non_user)

user_non_user.head(5)
```

Out[ ]:

	ID	Age	Gender	Education	Country	Ethnicity	Nscore	Escore	Oscore	Asc
0	2	25-34	M	Doctorate degree	UK	White	-0.67825	1.93886	1.43533	0.76
1	3	35-44	M	Professional certificate/ diploma	UK	White	-0.46725	0.80523	-0.84732	-1.62
2	4	18-24	F	Masters degree	UK	White	-0.14882	-0.80615	-0.01928	0.59
3	5	35-44	F	Doctorate degree	UK	White	0.73545	-1.63340	-0.45174	-0.30
4	6	65+	F	Left school at 18 years	Canada	White	-0.67825	-0.30033	-1.55521	2.03

Transformation successful.

What substances have the most and the least users?

In [ ]:
```
unu_drugs = user_non_user[drugs_cols]

users = make_cumulative_frequency_df(unu_drugs, ['1'])
users
```

Out[ ]:

	NumOfPeople	NumOfPeopleCum	PctOfPeople	PctOfPeopleCum
Substance				
Choc	1839	1839	97.61	97.61
Caff	1823	1823	96.76	96.76
Alcohol	1748	1748	92.78	92.78
Nicotine	1060	1060	56.26	56.26
Cannabis	999	999	53.03	53.03
Legalh	564	564	29.94	29.94
Benzos	535	535	28.40	28.40
Ecstasy	517	517	27.44	27.44
Amphet	436	436	23.14	23.14
Mushrooms	434	434	23.04	23.04
Coke	417	417	22.13	22.13
LSD	380	380	20.17	20.17
Meth	320	320	16.99	16.99
Ketamine	208	208	11.04	11.04
Amyl	133	133	7.06	7.06
Heroin	118	118	6.26	6.26
VSA	95	95	5.04	5.04
Crack	79	79	4.19	4.19
Semer	3	3	0.16	0.16

Are there any substances in our data set that are so rare that they are not even worth adding to our predictions?

Above, we see the number of users we have per substance according to the criteria we've determined (used in the past year), alongside their percentage in comparison to the whole dataset.

Very few people have done Crack Cocaine and Volatile Substances in the past year, so we might exclude them from our analysis given the small number of data points. Heroin is not that far off, at just 118 data points representing 6% of the dataset, but it's a drug that is so dangerous that we believe it's important to keep it in - again, since recall is our performance metric, we'd rather overshoot than undershoot when predicting who is at risk, especially when it's such an addicting and potentially harmful substance.

On the flip side, nearly everyone is a user of Chocolate, Caffeine and Alcohol. Chocolate and Caffeine do not pose lethal risks to our users, so we would have excluded them from our predictions anyway. But what about Alcohol? This is a substance that is very harmful, but also widely used and socially accepted. Due to nearly every data point of ours being considered a user, our predictions are also likely to say "yes, they are a user" to every new instance. This doesn't bring much value, so we might exclude Alcohol from our model as well - and, in a real-world scenario, simply attempt to educate everyone on the potential harm that can come with Alcohol despite it being legal and normalized.

Other than that, we have a considerable sample size for all other substances, so we'll gladly keep them in our analysis.

In summary, we will be excluding:

- Alcohol
- Caffeine
- Chocolate
- VSA
- Crack
- Semeron

It would also be wise to remove any data points that say they have done Semeron at some point, since there is a high chance of these people being overclaimers, making our data dirty.

4.Preprocessing

Okay, let's start by dropping columns that we don't need as we mentioned previously:

- Ethnicity
- Alcohol
- Caff
- Choc
- VSA
- Crack
- Semer

Before that let's remove people who stated that they used semer as they are overclaimers.
We will also remove ID as this not needed for the model.

Additionally, we will drop the column "nicotine" as it is widely used and accepted in the society. Our focus is to provide some educational resources for drug-users and nicotine is not as relevant as other drugs.

In [ ]:
```
clean_df = user_non_user.drop(user_non_user[user_non_user['Semer']==1].index)
clean_df = clean_df.drop(columns=['Ethnicity', 'Alcohol', 'Caff', 'Choc', 'VSA',
clean_df.head()
```

Out[ ]:

	Age	Gender	Education	Country	Nscore	Escore	Oscore	Ascore	Cscore
0	25-34	M	Doctorate degree	UK	-0.67825	1.93886	1.43533	0.76096	-0.14277
1	35-44	M	Professional certificate/ diploma	UK	-0.46725	0.80523	-0.84732	-1.62090	-1.01450
2	18-24	F	Masters degree	UK	-0.14882	-0.80615	-0.01928	0.59042	0.58489
3	35-44	F	Doctorate degree	UK	0.73545	-1.63340	-0.45174	-0.30172	1.30612
4	65+	F	Left school at 18 years	Canada	-0.67825	-0.30033	-1.55521	2.03972	1.63088

Now let's transform the Education to Education_Level:

value '0' will contain:

- Left school at/before 19 years old
- got a diploma
- started but did not finish univeristy/college

value '1' will contain:

- university degree
- master's
- doctorate

This is done to simplify the column and make it easier for the model to work.

In [ ]:
```
def categorize_education(df):
    higher_condition = df['Education'].str.contains('University degree|Masters d

    # Assign categories based on conditions
    df.loc[higher_condition, 'Education_Level'] = 1

    return df

df_edu = clean_df.copy()
df_edu['Education_Level'] = 0 #Setting the default education level to 0 so that
df_edu_transformed = categorize_education(df_edu.copy())
df_edu_transformed.drop(inplace=True, columns=['Education'])
df_edu_transformed.head()
```

Out[ ]:

	Age	Gender	Country	Nscore	Escore	Oscore	Ascore	Cscore	Impulsive	
0	25-34	M	UK	-0.67825	1.93886	1.43533	0.76096	-0.14277	-0.71126	-
1	35-44	M	UK	-0.46725	0.80523	-0.84732	-1.62090	-1.01450	-1.37983	
2	18-24	F	UK	-0.14882	-0.80615	-0.01928	0.59042	0.58489	-1.37983	-
3	35-44	F	UK	0.73545	-1.63340	-0.45174	-0.30172	1.30612	-0.21712	-
4	65+	F	Canada	-0.67825	-0.30033	-1.55521	2.03972	1.63088	-1.37983	-

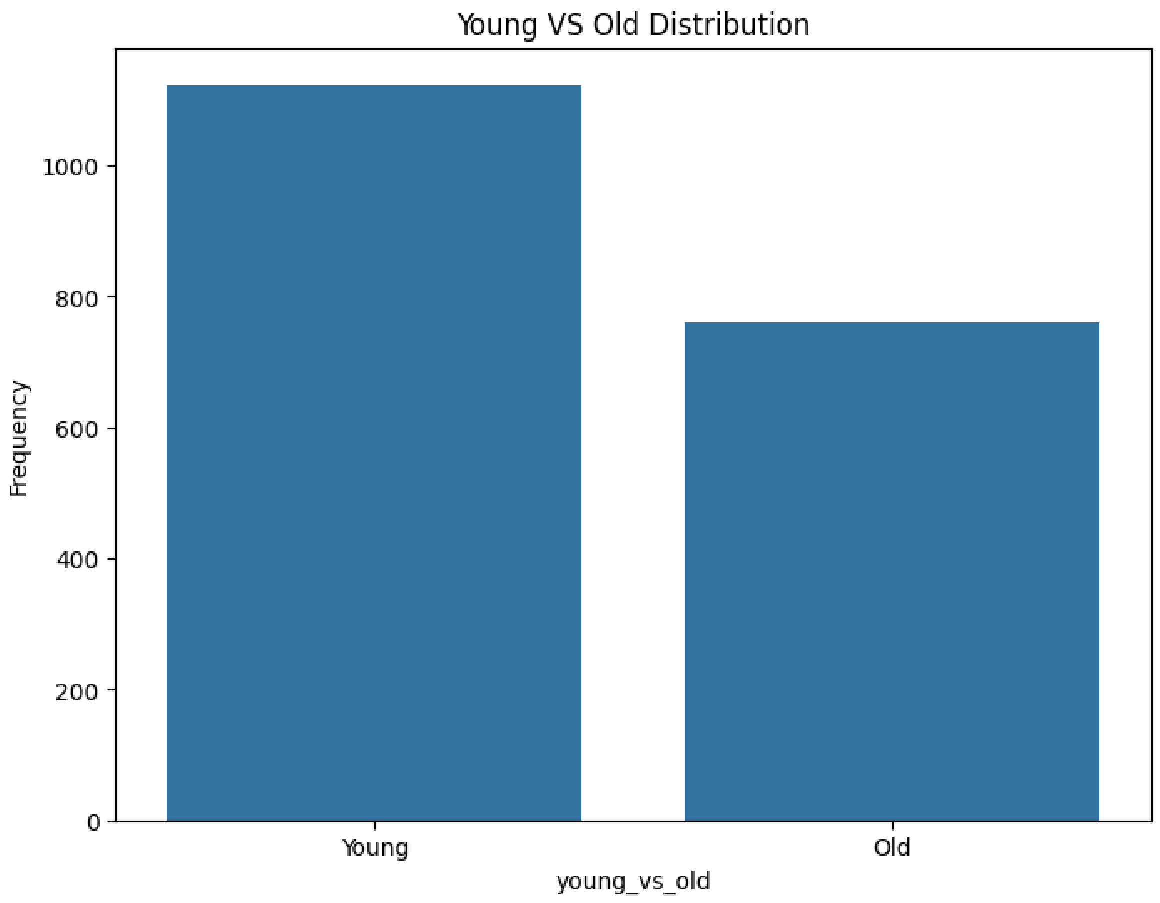
Let's do the same with age group. We will transform the column so that the young (ages 18-34) is represented as value '0' and old (ages 35+) is represented as '1'.

In [ ]:
```python
df_age = df_edu_transformed.copy()

def transform_age(df):
    old_condition = df['Age'].str.contains('35-44|45-54|55-64|65+', case=False)

    # Assign categories based on conditions
    df.loc[old_condition, 'young_vs_old'] = 1

    return df

df_age['young_vs_old'] = 0 #Setting the default to 0 so that we can change it
df_age_transformed = transform_age(df_age.copy())
df_age_transformed.drop(inplace=True, columns=['Age'])
df_age_transformed.head()
```

Out[ ]:

	Gender	Country	Nscore	Escore	Oscore	Ascore	Cscore	Impulsive	S
0	M	UK	-0.67825	1.93886	1.43533	0.76096	-0.14277	-0.71126	-0.2157
1	M	UK	-0.46725	0.80523	-0.84732	-1.62090	-1.01450	-1.37983	0.4014
2	F	UK	-0.14882	-0.80615	-0.01928	0.59042	0.58489	-1.37983	-1.1808
3	F	UK	0.73545	-1.63340	-0.45174	-0.30172	1.30612	-0.21712	-0.2157
4	F	Canada	-0.67825	-0.30033	-1.55521	2.03972	1.63088	-1.37983	-1.5485

In [ ]:
```python
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize=(8, 6))
sns.countplot(x='young_vs_old', data=df_age_transformed)
plt.title('Young VS Old Distribution')
plt.ylabel('Frequency')
plt.xticks(ticks=[0, 1], labels=['Young','Old'])
plt.show()
```



We can see that there is more data points which are set to be young, however, the number of old data points is signifcant enough.

We will perform the same transformation for Gender. Male will be encoded as '1' and female as '0'.

In [ ]:
```
df_gender = df_age_transformed.copy()

def transform_gender(df):
    male_condition = df['Gender'].str.contains('M', case=False)

    # Assign categories based on conditions
    df.loc[male_condition, 'Gender'] = "1"
    df.loc[male_condition==False, 'Gender'] = "0"

    return df

df_gender_transformed = transform_gender(df_gender.copy())
df_gender_transformed['Gender'] = df_gender_transformed['Gender'].astype(int)
df_gender_transformed.head()
```

Out[ ]:

	Gender	Country	Nscore	Escore	Oscore	Ascore	Cscore	Impulsive	S
0	1	UK	-0.67825	1.93886	1.43533	0.76096	-0.14277	-0.71126	-0.2157
1	1	UK	-0.46725	0.80523	-0.84732	-1.62090	-1.01450	-1.37983	0.4014
2	0	UK	-0.14882	-0.80615	-0.01928	0.59042	0.58489	-1.37983	-1.1808
3	0	UK	0.73545	-1.63340	-0.45174	-0.30172	1.30612	-0.21712	-0.2157
4	0	Canada	-0.67825	-0.30033	-1.55521	2.03972	1.63088	-1.37983	-1.5485

Perfect, now we are left with the "Country" column so let's encode that in the following manner in a binary columns:

- Country_UK
- Country_USA
- Country_Other

We do that as there a lot of people from USA and UK, howerver, the number of people not from those countries is small so we can group them to the 'Country_Other'.

In [ ]:

```
df_country = df_gender_transformed.copy()

def transform_country(df):
    uk_condition = df['Country'].str.contains('UK', case=False)
    usa_condition = df['Country'].str.contains('USA', case=False)
    # ~ bitwise NOT operator
    other_condition = (~uk_condition) & (~usa_condition)

    # Assign categories based on conditions
    df.loc[uk_condition, 'Country_UK'] = 1
    df.loc[usa_condition, 'Country_USA'] = 1
    df.loc[other_condition, 'Country_Other'] = 1

    return df

# Initialize new columns and set them to 0
df_country['Country_UK']=0
df_country['Country_USA']=0
df_country['Country_Other']=0

df_country_transformed = transform_country(df_country.copy())
df_country_transformed.drop(inplace=True, columns=['Country'])
df_country_transformed.head()
```

Out[ ]:

	Gender	Nscore	Escore	Oscore	Ascore	Cscore	Impulsive	SS	Amphe
0	1	-0.67825	1.93886	1.43533	0.76096	-0.14277	-0.71126	-0.21575	
1	1	-0.46725	0.80523	-0.84732	-1.62090	-1.01450	-1.37983	0.40148	
2	0	-0.14882	-0.80615	-0.01928	0.59042	0.58489	-1.37983	-1.18084	
3	0	0.73545	-1.63340	-0.45174	-0.30172	1.30612	-0.21712	-0.21575	
4	0	-0.67825	-0.30033	-1.55521	2.03972	1.63088	-1.37983	-1.54858	

In [ ]:

```
#Some cleaning up, reindexing Gender to be last
columns = ['Gender', 'Nscore', 'Escore', 'Oscore', 'Ascore', 'Cscore', 'Impulsiv
       'SS', 'Amphet', 'Amyl', 'Benzos', 'Cannabis', 'Coke', 'Ecstasy',
       'Heroin', 'Ketamine', 'Legalh', 'LSD', 'Meth', 'Mushrooms',
       'Education_Level', 'young_vs_old', 'Country_UK', 'Country_USA',
       'Country_Other']
columns.remove('Gender')
columns.append('Gender')
encoded_df = df_country_transformed.reindex(columns=columns)
# df_preprocessed.head()
```

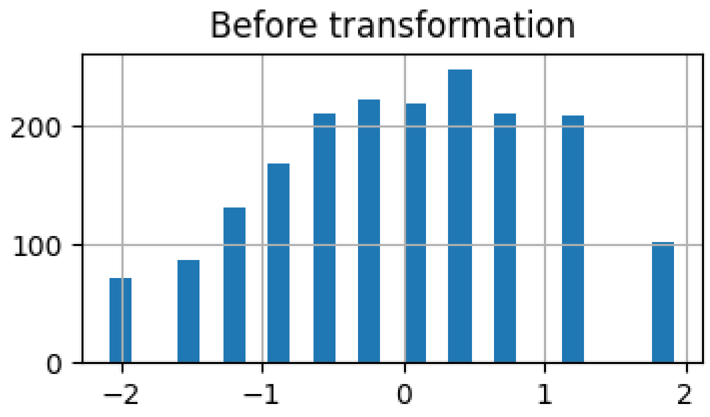
We will not discretize continous features as we believe that in this case it makes sense to keep the behaviour metrics as they are as they describe people. We believe that there is a lot of different personalities and their corresponding traits so we will keep it as it is to cover variations in our dataset.

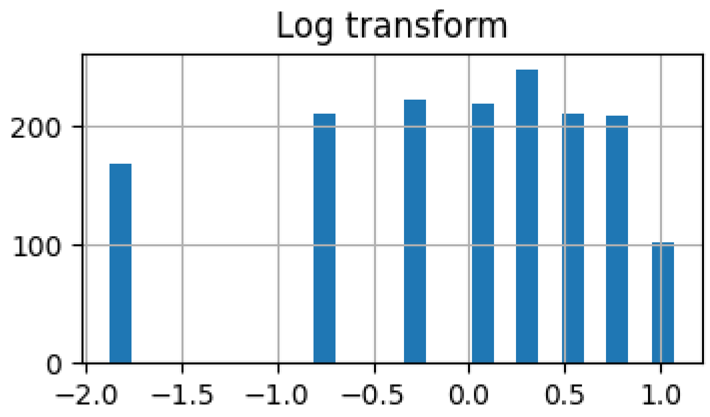
Let's take a look at transformations of features. As we previously mentioned, it might be a good idea to take a closer look at Impulsive and SS (Senstion seeking) columns as they were not normally distributed.

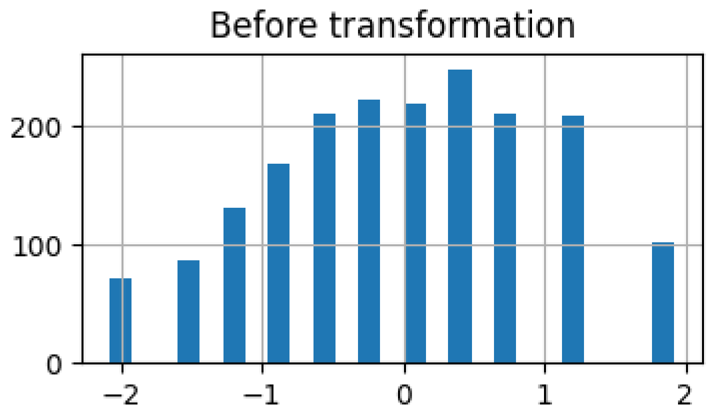
In [ ]:

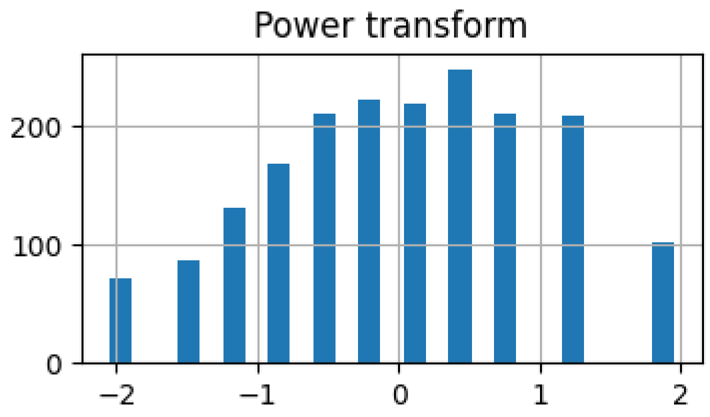
```
from sklearn.preprocessing import PowerTransformer
from sklearn.preprocessing import FunctionTransformer
import numpy as np

old_skew = encoded_df.skew().sort_values(ascending=False)

ss_data = encoded_df['SS']
pt = PowerTransformer(method='yeo-johnson', standardize=False)
def log_transform(x):
    return np.log(x+1)
transformer = FunctionTransformer(log_transform)

def transform(data, transformer, name):
    df = pd.DataFrame(data)
    transformed = transformer.fit_transform(df)
    df.hist(bins=25, figsize=(4,2))
    plt.title(label="Before transformation")
    transformed = pd.DataFrame(transformed, columns=df.columns)
    transformed.hist(bins=25, figsize=(4,2))
    plt.title(label=name)

transform(data=ss_data,transformer=transformer, name='Log transform')
transform(data=ss_data,transformer=pt, name='Power transform')
```

```
c:\Users\maxsz\AppData\Local\Programs\Python\Python312\Lib\site-packages\pandas\c
ore\internals\blocks.py:393: RuntimeWarning: invalid value encountered in log
  result = func(self.values, **kwargs)
```





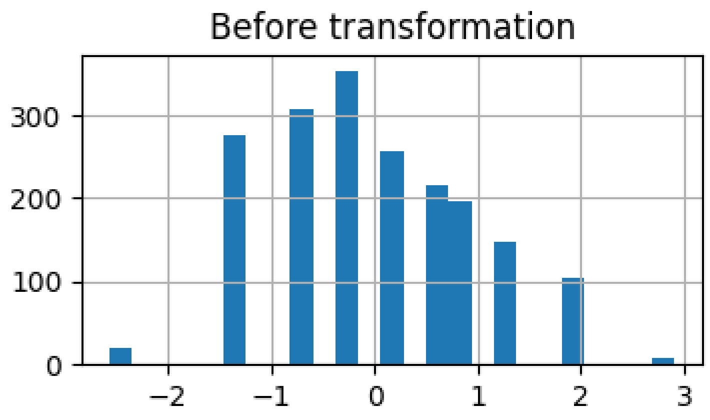


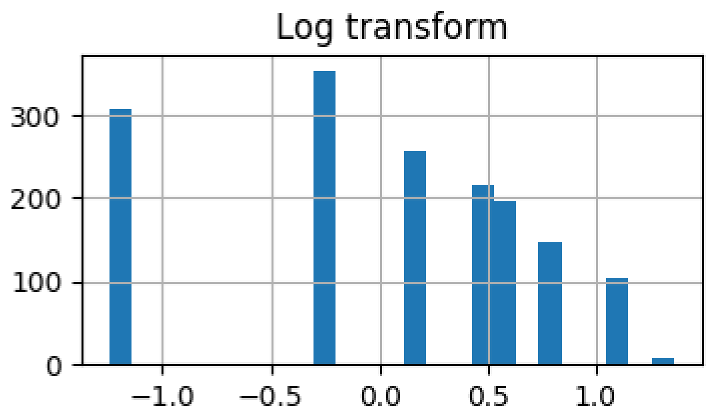


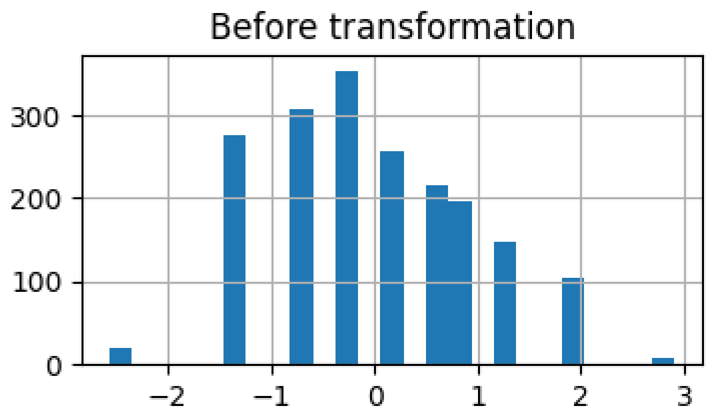
Oh okay, it seems like the transformation didn't give any reasonable result. Let's stay with the data we have. Let's take a look at the Implusive column. We believe that the results will be the same.

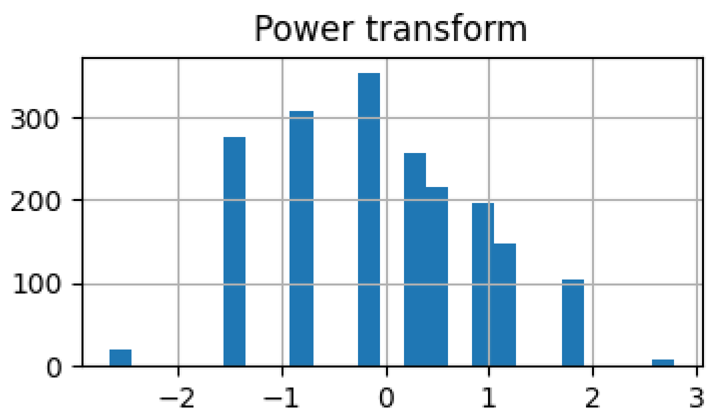
```
In [ ]: impulsive_data = encoded_df['Impulsive']

        transform(data=impulsive_data, transformer=transformer, name='Log transform')
        transform(data=impulsive_data, transformer=pt, name='Power transform')
```

```
c:\Users\maxsz\AppData\Local\Programs\Python\Python312\Lib\site-packages\pandas\c
ore\internals\blocks.py:393: RuntimeWarning: invalid value encountered in log
  result = func(self.values, **kwargs)
```







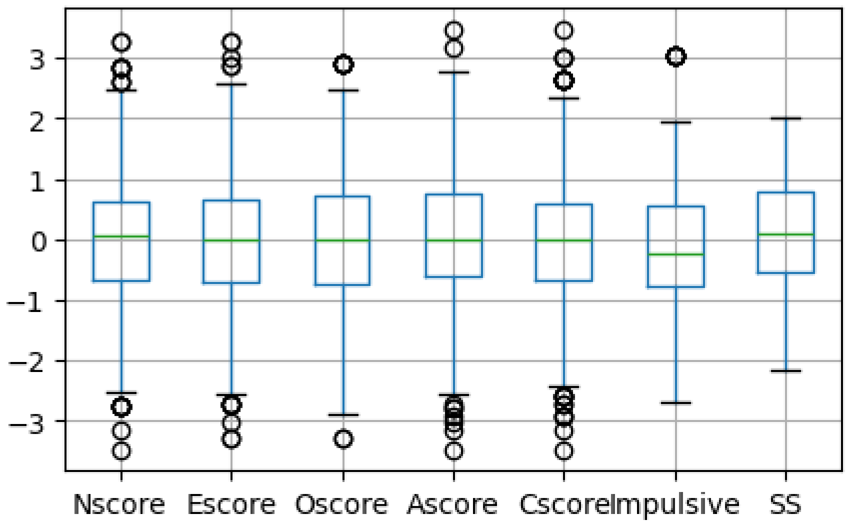


That doesn't look good. We will keep the data as it is. Now let's move to scaling the data. We will use the Standard scaler as we want to preserve the normal distribution of the data.

In [ ]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
continous_columns = ["Nscore","Escore","Oscore","Ascore","Cscore","Impulsive","S
to_scale = encoded_df[continous_columns]
scaled_data = scaler.fit_transform(to_scale)
scaled_df = pd.DataFrame(scaled_data, columns=continous_columns)
scaled_df.boxplot(figsize=(5,3))
scaled_df.head()
```

Out[ ]:

	Nscore	Escore	Oscore	Ascore	Cscore	Impulsive	SS
0	-0.679435	1.943848	1.441121	0.762842	-0.142741	-0.752913	-0.221197
1	-0.468029	0.807218	-0.850379	-1.625666	-1.016636	-1.453411	0.419536
2	-0.148987	-0.808426	-0.019129	0.591826	0.586726	-1.453411	-1.223036
3	0.736985	-1.637866	-0.453265	-0.302804	1.309747	-0.235174	-0.221197
4	-0.679435	-0.301268	-1.561014	2.045171	1.635314	-1.453411	-1.604778



Hmm that didn't change much, it could possibly be because the data was already scaled which would seem reasonable based on our previous observations. But to be sure that our model will perform good, we will keep scaled data.

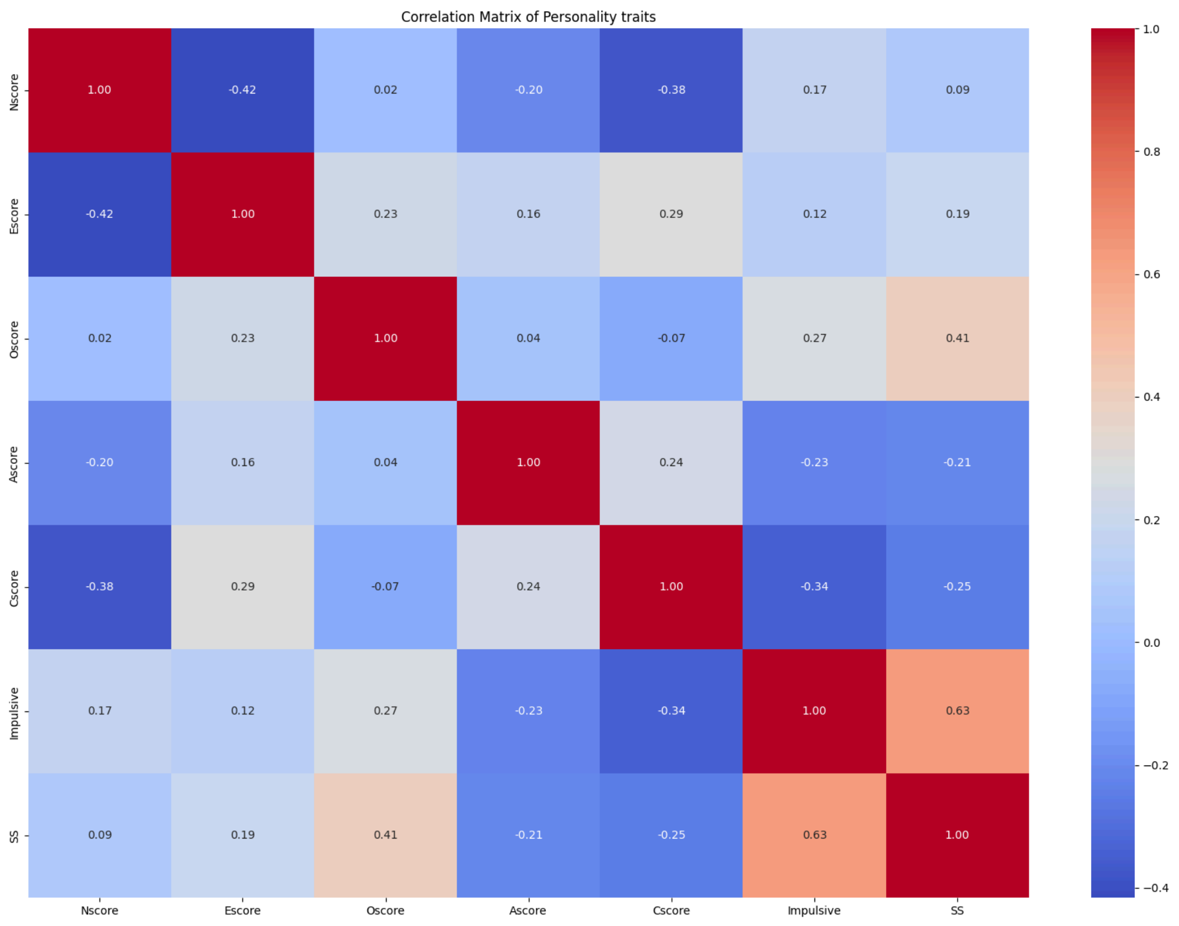
```
In [ ]: scaled_encoded_df = encoded_df.copy()
        scaled_encoded_df[continous_columns] = scaled_df
        scaled_encoded_df.head()
```

Out[ ]:

	Nscore	Escore	Oscore	Ascore	Cscore	Impulsive	SS	Amphet
0	-0.679435	1.943848	1.441121	0.762842	-0.142741	-0.752913	-0.221197	0
1	-0.468029	0.807218	-0.850379	-1.625666	-1.016636	-1.453411	0.419536	0
2	-0.148987	-0.808426	-0.019129	0.591826	0.586726	-1.453411	-1.223036	0
3	0.736985	-1.637866	-0.453265	-0.302804	1.309747	-0.235174	-0.221197	0
4	-0.679435	-0.301268	-1.561014	2.045171	1.635314	-1.453411	-1.604778	0

We couldn't see a correlation matrix previously as we had categorical values. Now that we have encoded them, let's have a look at the matrix!

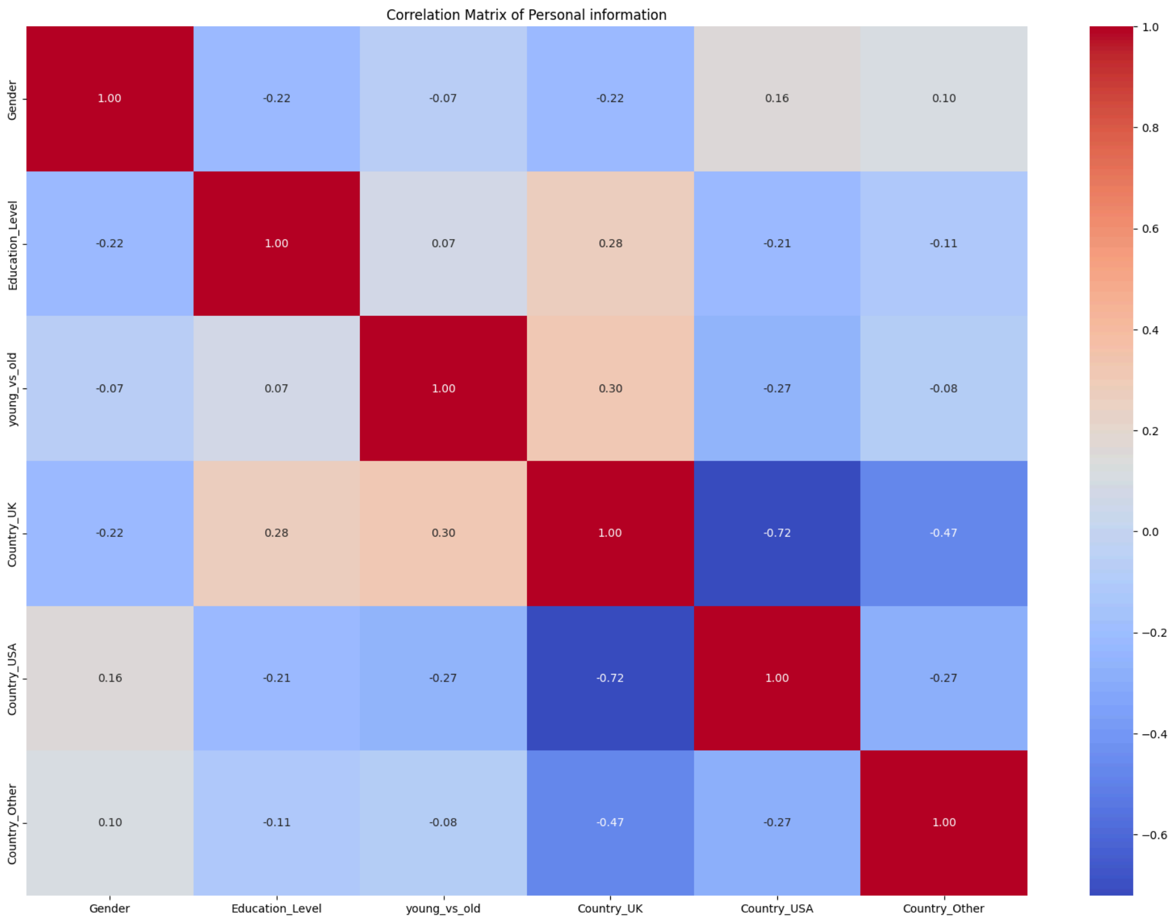
```
In [ ]: import seaborn as sns
        # correlation matrix define
        corr_matrix = scaled_encoded_df[continous_columns].corr(method='spearman')

        # plot it
        plt.figure(figsize=(20, 14))
        sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap='coolwarm')
        plt.title('Correlation Matrix of Personality traits')
        plt.show()
```



At the first glance, we can see that Impulsiveness and Sensation-Seeking are strongly correlated. Therefore, those two features might explain the same information. It is important to remove one of them. We have decided to remove the 'Impulsive' columns.

Let's take a look how are the other columns correlated.

```
In [ ]: other_columns = ["Gender","Education_Level","young_vs_old", "Country_UK","Countr
corr_matrix = scaled_encoded_df[other_columns].corr(method='spearman')

# plot it
plt.figure(figsize=(20, 14))
sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap='coolwarm')
plt.title('Correlation Matrix of Personal information')
plt.show()
```



Interesting! It seems like USA and UK columns are strongly negatively correlated, let's remove the USA column too. We can do that as USA will be represented when Country_UK=0 and Country_Other=0.

In [ ]:
```
scaled_encoded_df.drop(inplace=True, columns=['Impulsive','Country_USA'])
```

Okay now let's move to Prinicipal Component Analysis! We think it would be nice to reduce the dimensions of our dataset to make the model work faster. We might not use this, but it is a nice visualisation tool so we will create the PCA.

In [ ]:
```
import mglearn
from sklearn.decomposition import PCA

for_pca_df = scaled_encoded_df.copy()

labels_columns= ['Amphet',
       'Amyl', 'Benzos', 'Cannabis', 'Coke', 'Ecstasy', 'Heroin', 'Ketamine',
       'Legalh', 'LSD', 'Meth', 'Mushrooms']
labels = for_pca_df[labels_columns]
X_scaled = for_pca_df.drop(columns=labels_columns)

pca = PCA(n_components = 8)
pca.fit(X_scaled)
X_pca = pca.transform(X_scaled)
print("Original shape: {}".format(str(X_scaled.shape)))
print("Reduced shape: {}".format(str(X_pca.shape)))
```

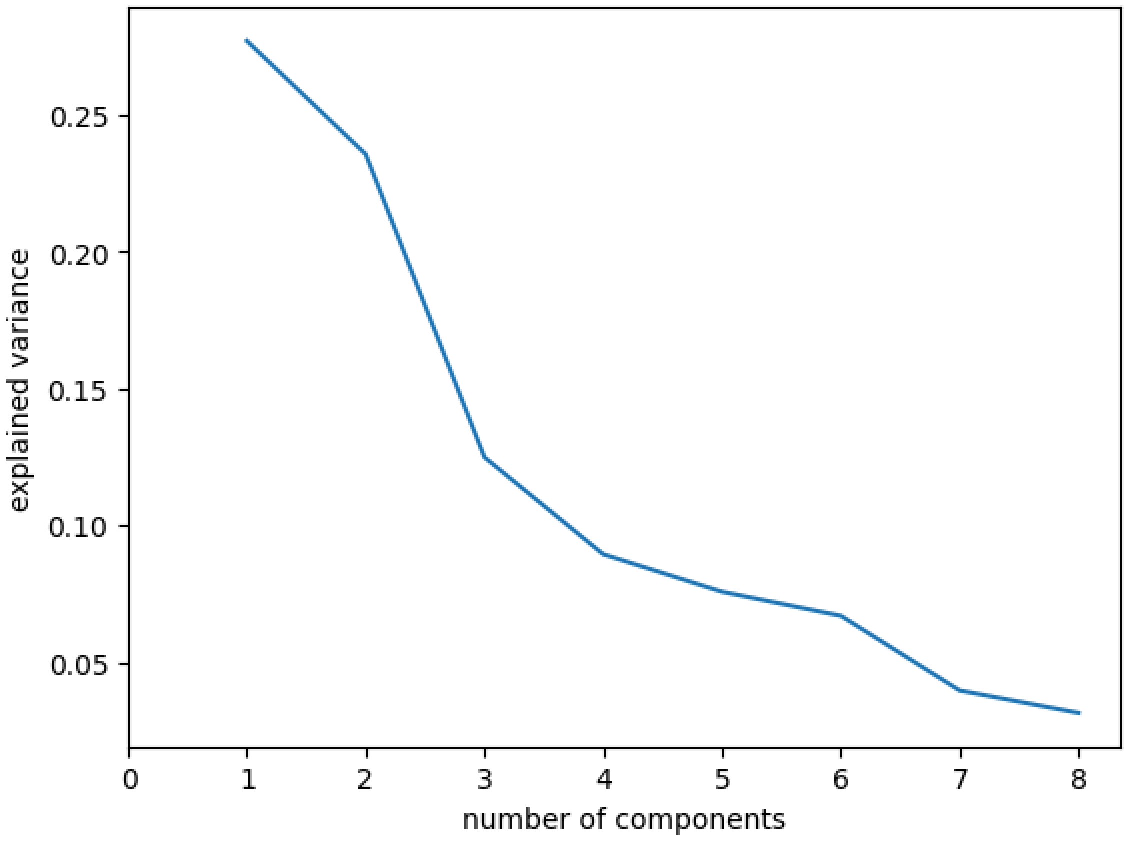
```
Original shape: (1884, 11)
Reduced shape: (1884, 8)
```

In [ ]:
```
import numpy
print(f"The ratio of 8 components: {[round(x, 4) for x in pca.explained_variance
print(f"This is how much information is explained in those ten components: {roun
```

```
The ratio of 8 components: [0.277, 0.2356, 0.1249, 0.0894, 0.0757, 0.0669, 0.039
6, 0.0314]
This is how much information is explained in those ten components: 0.9405
```

WOW, 94% of variance explained in 8 components, that's is pretty good! But let's investigate more as we have only reduced from 11 features to 8 so that is not much.

In [ ]:
```
plt.plot(list(range(1, 1 + len(pca.explained_variance_ratio_))), pca.explained_v
plt.xlabel('number of components')
plt.ylabel('explained variance')
plt.xticks(np.arange(0, 1 + len(pca.explained_variance_ratio_), step=1))
plt.show()
```



Hmmm, interesting, that's not a very sharp elbow. However, we can still see that the first 3 components are vertical thus they explain a lot of information. The next three components are more horizontal but they still explain around 7% of the variance so we think that it makes sense to keep them. Let's try PCA with 5 components.

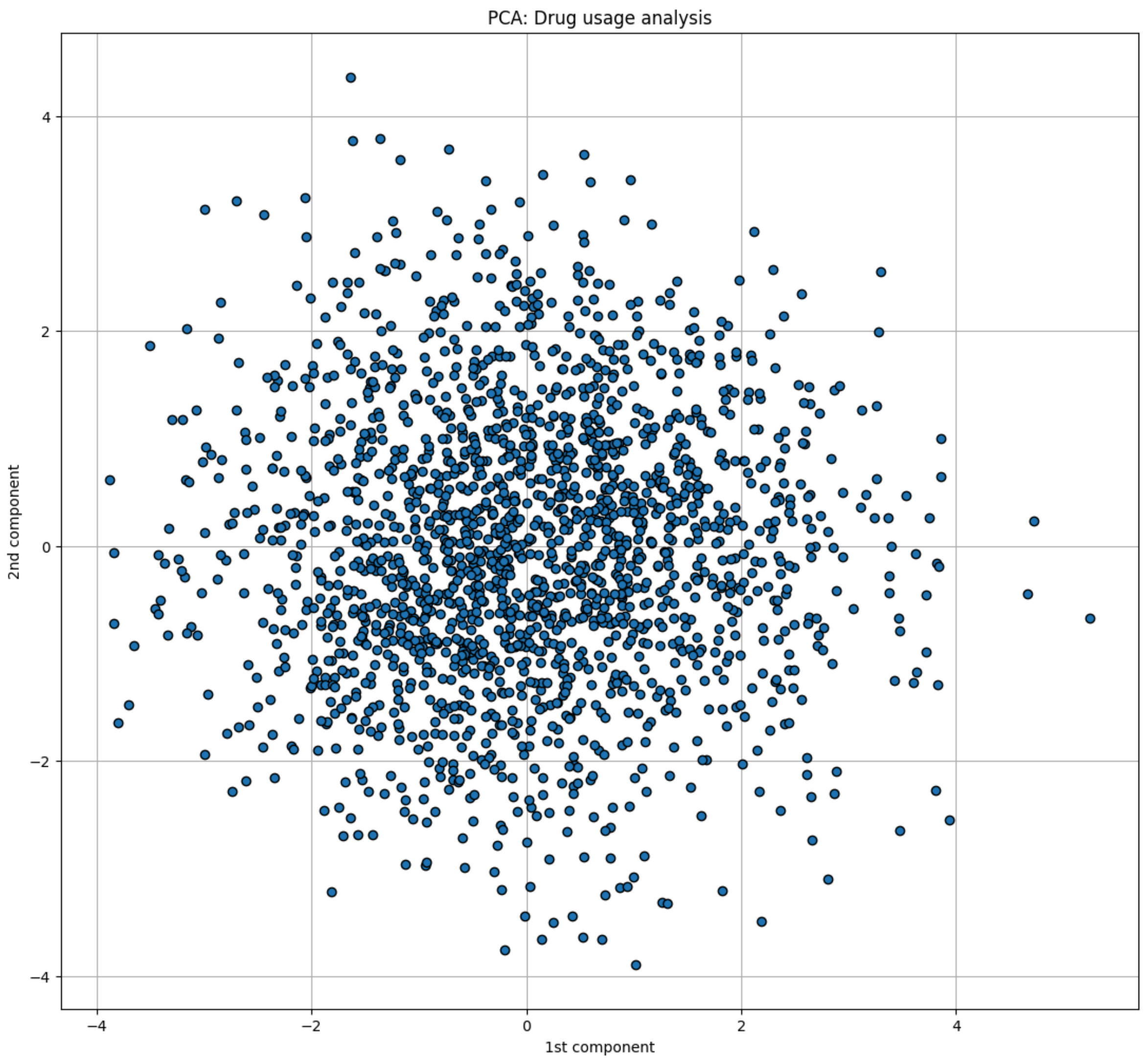
In [ ]:
```
import numpy
def pcaN(n):
    pca5 = PCA(n_components = n)
    pca5.fit(X_scaled)
    X_pca5 = pca5.transform(X_scaled)
    print(f"The ratio of {n} components: {[round(x, 4) for x in pca5.explained_v
    print(f"This is how much information is explained in those {n} components: {

pcaN(5)
```

```
The ratio of 5 components: [0.277, 0.2356, 0.1249, 0.0894, 0.0757]
This is how much information is explained in those 5 components: 0.8026
```

In [ ]:
```
pcaN(7)
```

```
The ratio of 7 components: [0.277, 0.2356, 0.1249, 0.0894, 0.0757, 0.0669, 0.039
6]
This is how much information is explained in those 7 components: 0.9091
```

In [ ]:
```
plt.figure(figsize=(15, 12))
mglearn.discrete_scatter(X_pca[:, 0], X_pca[:, 1], s=6, markers='o')
plt.gca().set_aspect("equal")
plt.xlabel("1st component")
plt.ylabel("2nd component")
plt.title("PCA: Drug usage analysis")
plt.grid(True)
```



Cool! The dots are placed all around the place and there are no evident clusters. That could suggest that there is not enough data to perform certain predictions, but we will mention that again later. For now, let's try to figure out what the first two components mean.

In [ ]:
```
components = pca.components_
explained_variance = pca.explained_variance_
feature_names = X_scaled.columns
```

```
# Print the contribution of each feature to the components
important_features = []
num_features = X_scaled.shape[1]
for i in range(2):
    print(f"Principal Component {i+1}:")
    component_features = [(feature_names[j], components[i][j]) for j in range(nu
    component_features.sort(key=lambda x: abs(x[1]), reverse=True)
    top_5_features = component_features[:5]
    important_features.append(top_5_features)
    for feature, coefficient in top_5_features:
        print(f"\t{feature}: {coefficient}")
    print("-"*50)
```

```
Principal Component 1:
        Cscore: -0.5430368687510547
        Nscore: 0.5266690157532106
        Escore: -0.41809575061341375
        Ascore: -0.3982494329979861
        SS: 0.238729134078417
--------------------------------------------------
Principal Component 2:
        SS: 0.6100888947589481
        Oscore: 0.6002582742767257
        Escore: 0.44792425276300624
        Nscore: -0.13937366444937746
        Country_UK: -0.1350976454479822
--------------------------------------------------
```

First of all, we can clearly see that those components are highly influenced by the personality traits which we have suspected earlier. Conscientiousness and Neuroticism seem to be the most important features for the 1st component, whereas Sensation Seeking and Openness to Experience play a vital role for the 2nd component.

Let's try to understand what that means.

- The first component could suggest that if someone is more laid-back, has less self-discipline and is facing some mental issues might be more prone to taking drugs. On the other hand, someone who is not so neurotic, enjoys being focused and thorough, might not be so interested in doing drugs.
- The second component is more self explainatory as it describes the person's drive to experience new things, try unknown things which might bring sensations and unexpected feelings.

5. Preparing for model creation

Let's start with developing binary classification model that predicts wheter a peson is a drug user or non-drug user based on their personality test as well as basic information such as:

- country
- age group
- education

In [ ]:
```
df_6 = scaled_encoded_df
df_6.head()
```

Out[ ]:

	Nscore	Escore	Oscore	Ascore	Cscore	SS	Amphet	Amyl	Ben
0	-0.679435	1.943848	1.441121	0.762842	-0.142741	-0.221197	0	0	
1	-0.468029	0.807218	-0.850379	-1.625666	-1.016636	0.419536	0	0	
2	-0.148987	-0.808426	-0.019129	0.591826	0.586726	-1.223036	0	0	
3	0.736985	-1.637866	-0.453265	-0.302804	1.309747	-0.221197	0	0	
4	-0.679435	-0.301268	-1.561014	2.045171	1.635314	-1.604778	0	0	

Creating a binary column "Any_Drug_Use".

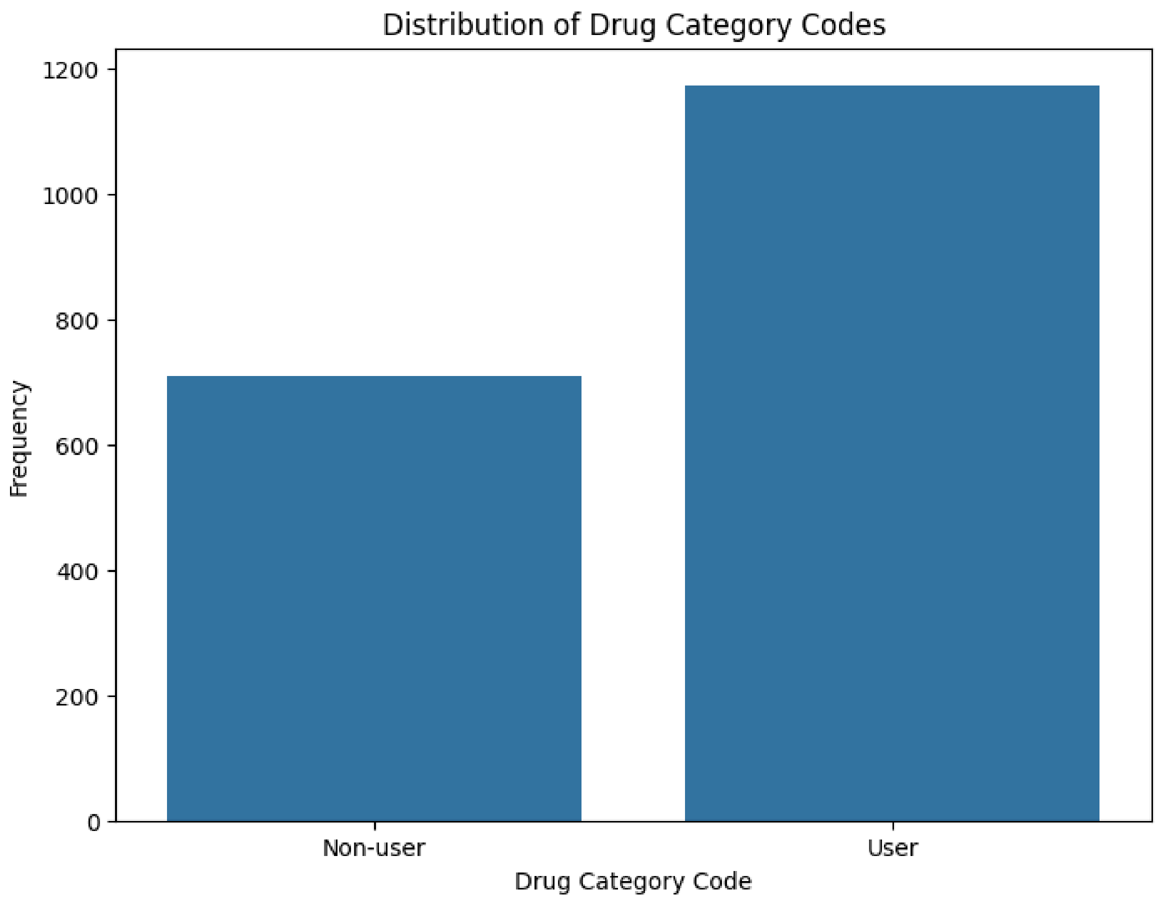
1 : Drug User; 0 : Non-Drug-User

In [ ]:
```
drug_columns_without_nicotine = ['Amphet', 'Amyl', 'Benzos', "Cannabis",
       'Coke', 'Ecstasy', 'Heroin', 'Ketamine', 'Legalh', 'LSD', 'Meth',
       'Mushrooms']

df_6[drug_columns_without_nicotine] = df_6[drug_columns_without_nicotine].apply(
df_6['Any_Drug_Use'] = df_6[drug_columns_without_nicotine].sum(axis=1)
df_6['Any_Drug_Use'] = (df_6['Any_Drug_Use'] > 0).astype(int)

non_users_count1 = (df_6['Any_Drug_Use'] == 0).sum()
users_count1 = (df_6['Any_Drug_Use'] == 1).sum()
total_count1 = len(df_6)

print("Without Nicotine:")
print("-------")
print(f"Number of non-users: {non_users_count1}")
print(f"Number of users: {users_count1}")
print(f"Total individuals: {total_count1}")
print(f"Percentage of non-users: {non_users_count1 / total_count1 * 100:.2f}%")
```

```
Without Nicotine:
-------
Number of non-users: 709
Number of users: 1175
Total individuals: 1884
Percentage of non-users: 37.63%
```

Visual representation of the binary drug category:

In [ ]:
```
import seaborn as sns
plt.figure(figsize=(8, 6))
sns.countplot(x='Any_Drug_Use', data=df_6)
plt.title('Distribution of Drug Category Codes')
plt.xlabel('Drug Category Code')
plt.ylabel('Frequency')
plt.xticks(ticks=[0, 1], labels=['Non-user','User'])
plt.show()
```



In [ ]:
```
df_no_drugs = df_6.drop(columns=['Amphet','Amyl', 'Benzos', 'Coke', 'Ecstasy', '
              'Legalh', 'LSD', 'Meth', 'Mushrooms', 'Cannabis'])
```

In [ ]:
```
df_ML = df_no_drugs.copy()
df_ML.describe()
```

Out[ ]:

	Nscore	Escore	Oscore	Ascore	Cscore	
count	1884.000000	1.884000e+03	1.884000e+03	1.884000e+03	1884.000000	1884.000
mean	0.000000	-2.640021e-17	-3.017167e-17	-3.582885e-17	0.000000	0.000
std	1.000265	1.000265e+00	1.000265e+00	1.000265e+00	1.000265	1.000
min	-3.470905	-3.282738e+00	-3.286397e+00	-3.474271e+00	-3.472580	-2.154
25%	-0.679435	-6.970726e-01	-7.198249e-01	-6.082646e-01	-0.653767	-0.543
50%	0.042772	3.185856e-03	-1.912881e-02	-1.758056e-02	-0.006283	0.085
75%	0.631001	6.393350e-01	7.263301e-01	7.628416e-01	0.586726	0.797
max	3.280348	3.282452e+00	2.913085e+00	3.473787e+00	3.473348	1.997

Let's take a look at the algorithms that can be used. The problem of a binary classifications seemed at the beginning very straighforward and thus, we wanted to start with some basic models to see how they perform. Classifying whether a person is a user or non-user of drugs is a supervised classification problem and these are the investigated models:

- Logistic Regression: easy to interpret and implement - good starting point of binary classification.
- Random Forest - can model complex, non-linear relationships.
- Supported Vector Machine - using kernel tricks that can catch complex relationships.
- K-Nearest Neighbour - easy to implement and understand, can be effective on small datasets.

In [ ]:
```
# importing all of the ML modules
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import GradientBoostingClassifier
import xgboost as xgb
from sklearn.metrics import precision_recall_fscore_support
```

In [ ]:
```
# defining the features X and target Y
X = df_ML.drop(['Any_Drug_Use'], axis=1)
y = df_ML['Any_Drug_Use']

# Splitting the data 70/15/15
X_train_temp, X_test, y_train_temp, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y)

X_train, X_val, y_train, y_val = train_test_split(
    X_train_temp, y_train_temp, test_size=0.5, random_state=40, stratify=y_train
```

In [ ]:
```
# Evaluate models
def evaluate_model(model, model_name):

    # fitting the model
    model.fit(X_train, y_train)

    # prediction
    y_train_pred = model.predict(X_train)
    y_val_pred = model.predict(X_val)

    # metrics
    train_metrics = precision_recall_fscore_support(y_train, y_train_pred, avera
    val_metrics = precision_recall_fscore_support(y_val, y_val_pred, average='bi

    # displaying results
    print(f"{model_name} Training - Avg Recall: {train_metrics[1]:.3f}, Avg Prec
    print(f"{model_name} Validation - Avg Recall: {val_metrics[1]:.3f}, Avg Prec
    print("-" * 80)

# models
models = {
    'Logistic Regression': LogisticRegression(random_state=413),
    'Random Forest': RandomForestClassifier(random_state=441, max_depth=10, min_
    'Support Vector Machine': SVC(random_state=47, kernel='poly'),
    'K-Nearest Neighbors': KNeighborsClassifier(),
}
```

```
for name, model in models.items():
    evaluate_model(model, name)
```

```
Logistic Regression Training - Avg Recall: 0.844, Avg Precision: 0.872, Avg F1: 
0.858
Logistic Regression Validation - Avg Recall: 0.842, Avg Precision: 0.892, Avg F1: 
0.866
--------------------------------------------------------------------------------
Random Forest Training - Avg Recall: 0.927, Avg Precision: 0.927, Avg F1: 0.927
Random Forest Validation - Avg Recall: 0.866, Avg Precision: 0.866, Avg F1: 0.866
--------------------------------------------------------------------------------
Support Vector Machine Training - Avg Recall: 0.915, Avg Precision: 0.843, Avg F
1: 0.877
Support Vector Machine Validation - Avg Recall: 0.881, Avg Precision: 0.819, Avg 
F1: 0.849
--------------------------------------------------------------------------------
K-Nearest Neighbors Training - Avg Recall: 0.854, Avg Precision: 0.882, Avg F1: 
0.868
K-Nearest Neighbors Validation - Avg Recall: 0.808, Avg Precision: 0.849, Avg F1: 
0.828
--------------------------------------------------------------------------------
```

All of the models perform pretty decends but the ones that sparkle the most are SVM and Random Forest. here, they overfit a little but let's carry on with them and execute hyper-tuning.

Hypertuning SVM:

In [ ]:
```
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import recall_score, make_scorer, f1_score

param_distributions = {
    'C': np.linspace(1.8, 2.3, 50),
    'gamma': np.linspace(0.08, 0.12, 50),
    'degree': [4, 5, 6],
    'coef0': [0, 0.1],
    'kernel': ['poly']
}

# recall the most important metric (for us)
recall_scorer = make_scorer(recall_score)

random_search = RandomizedSearchCV(
    SVC(random_state=47),
    param_distributions=param_distributions,
    n_iter=100,
    scoring=recall_scorer,
    cv=6,
    verbose=2,
    random_state=42,
    n_jobs=-1
)

random_search.fit(X_val, y_val)
print("Best parameters found:", random_search.best_params_)
print("Best recall obtained:", random_search.best_score_)

# variable for the best model
best_svm = random_search.best_estimator_
```

```
Fitting 6 folds for each of 100 candidates, totalling 600 fits
Best parameters found: {'kernel': 'poly', 'gamma': 0.0816326530612245, 'degree':
6, 'coef0': 0, 'C': 2.289795918367347}
Best recall obtained: 0.9635905086672351
```

This incredible result was retrieved by adjusting the hyperparameters several times after each iterations to achieve the best possible ones.

Hypertuning RandomForest:

In [ ]:
```python
# parameters for the random forest model
param_distributions = {
    'n_estimators': np.arange(100, 1001, 100),
    'max_depth': [None, 20, 30, 40, 50],
    'min_samples_split': [2, 4, 6, 8, 10],
    'min_samples_leaf': [1, 2, 3, 4],
    'max_features': ['sqrt', 'log2', 0.3, 0.5]
}


# recall the most important metric (for us)
recall_scorer = make_scorer(recall_score)

# My laptop couldn't cope with GridSearch and thus, we stay with RandomizedSearc
random_search_rf = RandomizedSearchCV(
    RandomForestClassifier(random_state=441),
    param_distributions=param_distributions,
    n_iter=70,
    scoring=recall_scorer,
    cv=3,
    verbose=2,
    random_state=42,
    n_jobs=-1
)

random_search_rf.fit(X_val, y_val)
print("Best parameters found:", random_search_rf.best_params_)
print("Best recall obtained:", random_search_rf.best_score_)

# variable for the best model
best_rf = random_search_rf.best_estimator_
```

```
Fitting 3 folds for each of 70 candidates, totalling 210 fits
Best parameters found: {'n_estimators': 100, 'min_samples_split': 10, 'min_sample
s_leaf': 4, 'max_features': 'log2', 'max_depth': 50}
Best recall obtained: 0.8442822384428222
```

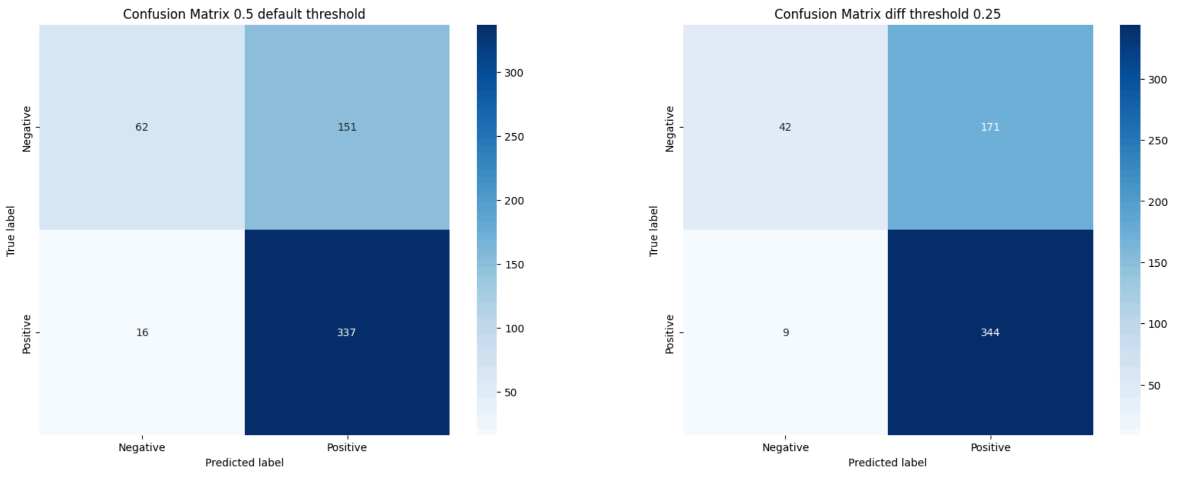
By changing parameters of random forest several times it can't give me anything greater that 0.86. therefore, I will stick to the SVM that performs really well.

In [ ]:
```python
# Final evaluation on the test data
y_test_pred = best_svm.predict(X_test)
test_recall = recall_score(y_test, y_test_pred)
print(f"Test Recall with Best SVM: {test_recall:.3f}")


def tune_prediction_threshold(threshold:float, pred_proba:np.ndarray):
    """
    Only for binary
    """
    pred_proba_positive = pred_proba[:,1]
    return [1 if prob >= threshold else 0 for prob in pred_proba_positive]



# Tune threshold
# y_test_pred_proba = best_svm.predict_proba(X_test)


svc_clf = SVC(random_state=47, probability=True, **random_search.best_params_)
svc_clf.fit(X_train_temp, y_train_temp)

y_test_pred_proba = svc_clf.predict_proba(X_test)

#threshold can be adjusted according to the final result of recall I want to obt
threshold = 0.25

preds_low_threshold = tune_prediction_threshold(threshold, y_test_pred_proba)

test_recall = recall_score(y_test, preds_low_threshold)
test_f1 = f1_score(y_test, preds_low_threshold)

print(f"Test Recall for {threshold}: {test_recall}")
print(f"Test f1-score for {threshold}: {test_f1}")
```

```
Test Recall with Best SVM: 0.955
Test Recall for 0.25: 0.9745042492917847
Test f1-score for 0.25: 0.7926267281105991
```

This is a recall that is very satisfactory for the analyis. Now, by displaying confusion matrixes and balancing the precision-recall curve: let's look what are our confusion matrixes elements and if it is worth it to minimize the false negatives at all costs.

In [ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Defining and displaying confusion matrixes to have a detailed classification i

conf_matrix = confusion_matrix(y_test, y_test_pred)
# print("Confusion Matrix 0.5 threshold:")
# print(conf_matrix)

conf_matrix_diff_threshold = confusion_matrix(y_test, preds_low_threshold)
# print("Confusion Matrix diff threshold:")
# print(conf_matrix_diff_threshold)

fig, ax = plt.subplots(1, 2, figsize=(20, 7))

# Heatmap for confusion matrix 0.5 threshold
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap='Blues', square=True,
            xticklabels=['Negative', 'Positive'], yticklabels=['Negative', 'Posi
            ax=ax[0])
ax[0].set_title('Confusion Matrix 0.5 default threshold')
ax[0].set_ylabel('True label')
ax[0].set_xlabel('Predicted label')

# Heatmap for confusion matrix 0.25 threshold
sns.heatmap(conf_matrix_diff_threshold, annot=True, fmt="d", cmap='Blues', squar
            xticklabels=['Negative', 'Positive'], yticklabels=['Negative', 'Posi
            ax=ax[1])
ax[1].set_title('Confusion Matrix diff threshold 0.25')
ax[1].set_ylabel('True label')
ax[1].set_xlabel('Predicted label')

plt.show()
```



Results on the adjsuted threshold:

- All Test Cases: 566
- True Negatives (TN): 42
- False Positives (FP): 171
- False Negatives (FN): 9
- True Positives (TP): 344

As the main objective of classifying a drug-user is to provide educational resources, it is okay to keep the adjusted threshold classifications. As we do not possess the knowledge of what kind of drugs that person could takes, it could be wise to notify and educate.

Multi-Labelling Attempt:

Based on same personality traits features, let's try to classify what kind of drugs a person takes.

Output Example: A person with certain personality is likely to take LSD, Mushroom and Coke.

Dropping:

- previous target: any_drug_use - as it could introduce data leakage.
- columns with least drug users (Amyl, Heroin, Ketamine)

In [ ]:
```
df_multi = df_6.drop(columns=['Any_Drug_Use', 'Amyl', 'Heroin', 'Ketamine'])
```

In [ ]:
```
# X2 - features; y2 - target labels
X2 = df_multi.drop(columns=['Amphet', 'Benzos', 'Ecstasy',
        'Legalh', 'LSD', 'Meth', 'Mushrooms', 'Cannabis', 'Coke'])
```

```
y2 = df_multi[['Amphet', 'Benzos', 'Ecstasy',
        'Legalh', 'LSD', 'Meth', 'Mushrooms','Cannabis', 'Coke']]

# making a 80/20 split, later the NN algorithms takes 20% of the training data f
X2_train, X2_test, y2_train, y2_test = train_test_split(X2, y2, test_size=0.2, r
```

This problem seems much complicated and thus, we decided to use a neural network to perform this task.

In [ ]:
```
# Cell with all of the neccesarry modules for NN
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.layers import Dropout
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.layers import BatchNormalization
np.random.seed(0)
tf.random.set_seed(0)
from tensorflow.keras.layers import LeakyReLU
```

In [ ]:
```
# Defining a number of classes
num_classes = y2_train.shape[1]
# 64 input neurons - Hidden Layers 48 and 42; all with relu activation
model = Sequential([
    Dense(64, activation='relu', input_dim=X2_train.shape[1]),
    Dense(48, activation='relu'),
    Dense(32, activation='relu'),
    Dense(num_classes, activation='sigmoid')
])
# Compiling the model with Adam optimizer - "binery_crossentropy" is good for mu
model.compile(optimizer=Adam(learning_rate=0.001), loss='binary_crossentropy', m

# Anti overfitting - monitors "val_loss" if it does not improve for 50 epochs
early_stopping = EarlyStopping(monitor='val_loss', patience=50, restore_best_wei

# training the model
history = model.fit(
    X2_train, y2_train,
    epochs=85,
    batch_size=5,
    verbose=1,
    validation_split=0.2,
    callbacks=[early_stopping]
)
```

```
Epoch 1/85
```

```
c:\Users\maxsz\AppData\Local\Programs\Python\Python312\Lib\site-packages\keras\sr
c\layers\core\dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim`
argument to a layer. When using Sequential models, prefer using an `Input(shape)`
object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
241/241 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - loss: 0.5512 - recall: 0.2789 - val_lo
ss: 0.4507 - val_recall: 0.3490
Epoch 2/85
241/241 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4364 - recall: 0.3923 - val_lo
ss: 0.4412 - val_recall: 0.4254
Epoch 3/85
241/241 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.4246 - recall: 0.4641 - val_lo
ss: 0.4403 - val_recall: 0.4340
Epoch 4/85
241/241 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.4187 - recall: 0.5022 - val_lo
ss: 0.4396 - val_recall: 0.4538
Epoch 5/85
241/241 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.4143 - recall: 0.5168 - val_lo
ss: 0.4389 - val_recall: 0.4871
Epoch 6/85
241/241 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.4113 - recall: 0.5272 - val_lo
ss: 0.4393 - val_recall: 0.4908
Epoch 7/85
241/241 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.4076 - recall: 0.5369 - val_lo
ss: 0.4402 - val_recall: 0.4846
Epoch 8/85
241/241 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.4038 - recall: 0.5419 - val_lo
ss: 0.4410 - val_recall: 0.4932
Epoch 9/85
241/241 ━━━━━━━━━━━━━━━━━━━━ 0s 928us/step - loss: 0.4000 - recall: 0.5508 - val_
loss: 0.4417 - val_recall: 0.5043
Epoch 10/85
241/241 ━━━━━━━━━━━━━━━━━━━━ 0s 927us/step - loss: 0.3966 - recall: 0.5630 - val_
loss: 0.4438 - val_recall: 0.5018
Epoch 11/85
241/241 ━━━━━━━━━━━━━━━━━━━━ 0s 943us/step - loss: 0.3933 - recall: 0.5704 - val_
loss: 0.4454 - val_recall: 0.5117
Epoch 12/85
241/241 ━━━━━━━━━━━━━━━━━━━━ 0s 983us/step - loss: 0.3896 - recall: 0.5828 - val_
loss: 0.4480 - val_recall: 0.5117
Epoch 13/85
241/241 ━━━━━━━━━━━━━━━━━━━━ 0s 994us/step - loss: 0.3861 - recall: 0.5920 - val_
loss: 0.4494 - val_recall: 0.5142
Epoch 14/85
241/241 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.3826 - recall: 0.5915 - val_lo
ss: 0.4520 - val_recall: 0.5179
Epoch 15/85
241/241 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3789 - recall: 0.5905 - val_lo
ss: 0.4536 - val_recall: 0.5290
Epoch 16/85
241/241 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.3755 - recall: 0.5946 - val_lo
ss: 0.4556 - val_recall: 0.5302
Epoch 17/85
241/241 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.3725 - recall: 0.6071 - val_lo
ss: 0.4575 - val_recall: 0.5302
Epoch 18/85
241/241 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.3692 - recall: 0.6144 - val_lo
ss: 0.4600 - val_recall: 0.5290
Epoch 19/85
241/241 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.3658 - recall: 0.6205 - val_lo
ss: 0.4625 - val_recall: 0.5364
Epoch 20/85
241/241 ━━━━━━━━━━━━━━━━━━━━ 0s 982us/step - loss: 0.3624 - recall: 0.6250 - val_
loss: 0.4650 - val_recall: 0.5450
Epoch 21/85
```

```
241/241 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.3595 - recall: 0.6262 - val_lo
ss: 0.4677 - val_recall: 0.5475
Epoch 22/85
241/241 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.3565 - recall: 0.6341 - val_lo
ss: 0.4727 - val_recall: 0.5425
Epoch 23/85
241/241 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.3531 - recall: 0.6455 - val_lo
ss: 0.4754 - val_recall: 0.5487
Epoch 24/85
241/241 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.3496 - recall: 0.6522 - val_lo
ss: 0.4790 - val_recall: 0.5536
Epoch 25/85
241/241 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.3465 - recall: 0.6565 - val_lo
ss: 0.4830 - val_recall: 0.5573
Epoch 26/85
241/241 ━━━━━━━━━━━━━━━━━━━━ 0s 977us/step - loss: 0.3431 - recall: 0.6632 - val_
loss: 0.4870 - val_recall: 0.5549
Epoch 27/85
241/241 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.3399 - recall: 0.6674 - val_lo
ss: 0.4911 - val_recall: 0.5610
Epoch 28/85
241/241 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.3367 - recall: 0.6716 - val_lo
ss: 0.4949 - val_recall: 0.5536
Epoch 29/85
241/241 ━━━━━━━━━━━━━━━━━━━━ 0s 988us/step - loss: 0.3337 - recall: 0.6760 - val_
loss: 0.4979 - val_recall: 0.5610
Epoch 30/85
241/241 ━━━━━━━━━━━━━━━━━━━━ 0s 943us/step - loss: 0.3303 - recall: 0.6836 - val_
loss: 0.5016 - val_recall: 0.5635
Epoch 31/85
241/241 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.3269 - recall: 0.6838 - val_lo
ss: 0.5068 - val_recall: 0.5573
Epoch 32/85
241/241 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.3239 - recall: 0.6908 - val_lo
ss: 0.5103 - val_recall: 0.5660
Epoch 33/85
241/241 ━━━━━━━━━━━━━━━━━━━━ 0s 948us/step - loss: 0.3206 - recall: 0.6981 - val_
loss: 0.5136 - val_recall: 0.5660
Epoch 34/85
241/241 ━━━━━━━━━━━━━━━━━━━━ 0s 966us/step - loss: 0.3181 - recall: 0.6969 - val_
loss: 0.5187 - val_recall: 0.5610
Epoch 35/85
241/241 ━━━━━━━━━━━━━━━━━━━━ 0s 960us/step - loss: 0.3151 - recall: 0.7048 - val_
loss: 0.5228 - val_recall: 0.5721
Epoch 36/85
241/241 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.3121 - recall: 0.7112 - val_lo
ss: 0.5255 - val_recall: 0.5672
Epoch 37/85
241/241 ━━━━━━━━━━━━━━━━━━━━ 0s 936us/step - loss: 0.3095 - recall: 0.7138 - val_
loss: 0.5310 - val_recall: 0.5660
Epoch 38/85
241/241 ━━━━━━━━━━━━━━━━━━━━ 0s 952us/step - loss: 0.3068 - recall: 0.7192 - val_
loss: 0.5357 - val_recall: 0.5598
Epoch 39/85
241/241 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.3041 - recall: 0.7202 - val_lo
ss: 0.5412 - val_recall: 0.5524
Epoch 40/85
241/241 ━━━━━━━━━━━━━━━━━━━━ 0s 922us/step - loss: 0.3006 - recall: 0.7245 - val_
loss: 0.5448 - val_recall: 0.5536
Epoch 41/85
```

```
241/241 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.2978 - recall: 0.7270 - val_lo
ss: 0.5514 - val_recall: 0.5438
Epoch 42/85
241/241 ━━━━━━━━━━━━━━━━━━━━ 0s 937us/step - loss: 0.2956 - recall: 0.7274 - val_
loss: 0.5551 - val_recall: 0.5487
Epoch 43/85
241/241 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.2935 - recall: 0.7326 - val_lo
ss: 0.5637 - val_recall: 0.5450
Epoch 44/85
241/241 ━━━━━━━━━━━━━━━━━━━━ 0s 932us/step - loss: 0.2902 - recall: 0.7376 - val_
loss: 0.5678 - val_recall: 0.5425
Epoch 45/85
241/241 ━━━━━━━━━━━━━━━━━━━━ 0s 967us/step - loss: 0.2873 - recall: 0.7410 - val_
loss: 0.5747 - val_recall: 0.5364
Epoch 46/85
241/241 ━━━━━━━━━━━━━━━━━━━━ 0s 929us/step - loss: 0.2852 - recall: 0.7406 - val_
loss: 0.5804 - val_recall: 0.5376
Epoch 47/85
241/241 ━━━━━━━━━━━━━━━━━━━━ 0s 948us/step - loss: 0.2823 - recall: 0.7429 - val_
loss: 0.5862 - val_recall: 0.5327
Epoch 48/85
241/241 ━━━━━━━━━━━━━━━━━━━━ 0s 917us/step - loss: 0.2794 - recall: 0.7491 - val_
loss: 0.5915 - val_recall: 0.5351
Epoch 49/85
241/241 ━━━━━━━━━━━━━━━━━━━━ 0s 927us/step - loss: 0.2774 - recall: 0.7539 - val_
loss: 0.5959 - val_recall: 0.5240
Epoch 50/85
241/241 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.2746 - recall: 0.7547 - val_lo
ss: 0.6031 - val_recall: 0.5253
Epoch 51/85
241/241 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.2728 - recall: 0.7581 - val_lo
ss: 0.6085 - val_recall: 0.5277
Epoch 52/85
241/241 ━━━━━━━━━━━━━━━━━━━━ 0s 923us/step - loss: 0.2708 - recall: 0.7612 - val_
loss: 0.6130 - val_recall: 0.5216
Epoch 53/85
241/241 ━━━━━━━━━━━━━━━━━━━━ 0s 935us/step - loss: 0.2685 - recall: 0.7666 - val_
loss: 0.6189 - val_recall: 0.5154
Epoch 54/85
241/241 ━━━━━━━━━━━━━━━━━━━━ 0s 952us/step - loss: 0.2655 - recall: 0.7651 - val_
loss: 0.6289 - val_recall: 0.5105
Epoch 55/85
241/241 ━━━━━━━━━━━━━━━━━━━━ 0s 950us/step - loss: 0.2639 - recall: 0.7719 - val_
loss: 0.6291 - val_recall: 0.5191
```

In [ ]:

```python
# test set evaluations
test_loss, test_recall = model.evaluate(X2_test, y2_test)
print(f'Test Loss: {test_loss}')
print(f'Test Recall: {test_recall}')
```

```
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.4406 - recall: 0.5247
Test Loss: 0.43989574909210205
Test Recall: 0.5089379549026489
```

It is a failure of an attemp of implementing a neural network model to perform multilabelling classifications on our dataset. After trying executing the model with diffrent parameter setting the highest obtained recall was 0.64. which is pretty bad.

There are a few arguments listed, why the model was not managed to

- The model starts to overfit the data at some point.
- the dataset is too small to classify several labels properly.
- there is no pattern that the algorithm can find to distinguish diffrent drugs labels based on personality features.

7. Video presentation

Here is the link to the video presentation. We hope that you enjoy it!