Predictive Modelling for Drug Consumption Risk

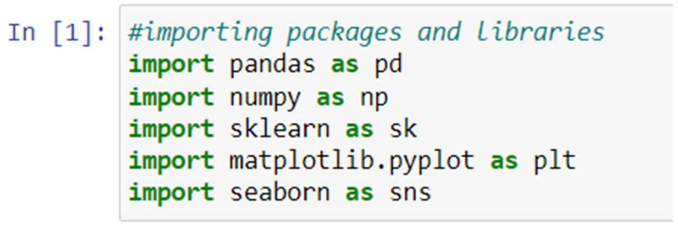
Dataset:

The dataset used for my report analysis is curated from the UCI Machine Learning Repository. it contains records for 1884 respondents, each characterized by 12 distinct attributes such as level of education, age, gender, country of residence, ethnicity, and personality measurements. such as BIS-11 (impulsivity), ImpSS (sensation seeking), and NEO-FFI-R (neuroticism, extraversion, openness to experience, agreeableness, and conscientiousness). Originally, each input attribute was quantified and categorical. Participants also answered questions about their use of 18 illicit and legal drugs (crack, cocaine, chocolate, cannabis, amphetamines, amyl nitrite, benzodiazepine, ecstasy, heroin, ketamine, legal highs, LSD, methadone, nicotine, mushrooms, and volatile substances), as well as one fictitious drug (Semeron), which was introduced to identify over-claimers. The following source link provides information about the dataset, including academic publications and other relevant details:

https://archive.ics.uci.edu/dataset/373/drug+consumption+quantified

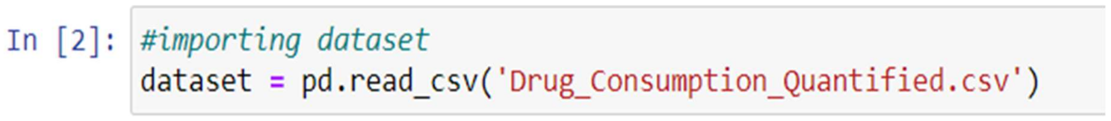
Explanation and preparation of dataset (Exploratory Data Analysis)

1. Launch Jupyter Notebook, either from the Anaconda Prompt or the Anaconda Navigator. Open a new notebook by clicking the New tab button and select Python 3 (ipykernel) to create a new notebook workspace.
2. Using the following codes, import all required libraries and packages:



```
In [1]: #importing packages and libraries
        import pandas as pd
        import numpy as np
        import sklearn as sk
        import matplotlib.pyplot as plt
        import seaborn as sns
```

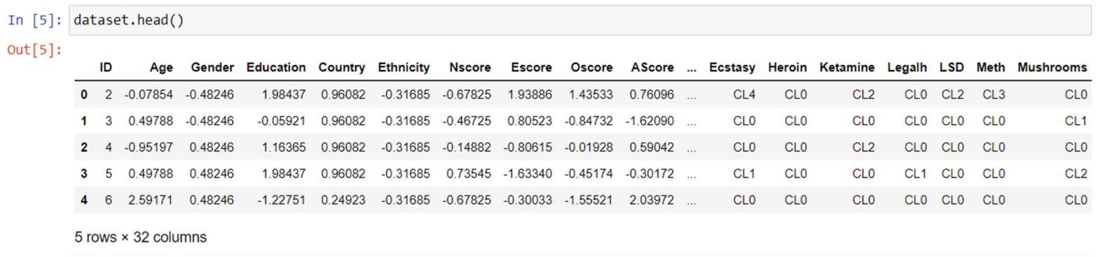
3. Import the dataset into the Data Frame using the read_csv() function in the pandas library. NB: The dataset should be in the same folder with jupyter notebook for easy reading.



```
In [2]: #importing dataset
        dataset = pd.read_csv('Drug_Consumption_Quantified.csv')
```

(a) dataset.head()

This code returns the header line and the first five rows of the dataset.



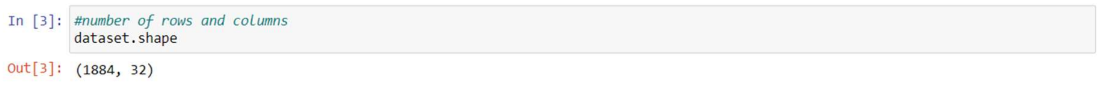
```
In [5]: dataset.head()
Out[5]:
```

	ID	Age	Gender	Education	Country	Ethnicity	Nscore	Escore	Oscore	AScore	...	Ecstasy	Heroin	Ketamine	Legalh	LSD	Meth	Mushrooms
0	2	-0.07854	-0.48246	1.98437	0.96082	-0.31685	-0.67825	1.93886	1.43533	0.76096	...	CL4	CL0	CL2	CL0	CL2	CL3	CL0
1	3	0.49788	-0.48246	-0.05921	0.96082	-0.31685	-0.46725	0.80523	-0.84732	-1.62090	...	CL0	CL0	CL0	CL0	CL0	CL0	CL1
2	4	-0.95197	0.48246	1.16365	0.96082	-0.31685	-0.14882	-0.80615	-0.01928	0.59042	...	CL0	CL0	CL2	CL0	CL0	CL0	CL0
3	5	0.49788	0.48246	1.98437	0.96082	-0.31685	0.73545	-1.63340	-0.45174	-0.30172	...	CL1	CL0	CL0	CL1	CL0	CL0	CL2
4	6	2.59171	0.48246	-1.22751	0.24923	-0.31685	-0.67825	-0.30033	-1.55521	2.03972	...	CL0	CL0	CL0	CL0	CL0	CL0	CL0

5 rows × 32 columns

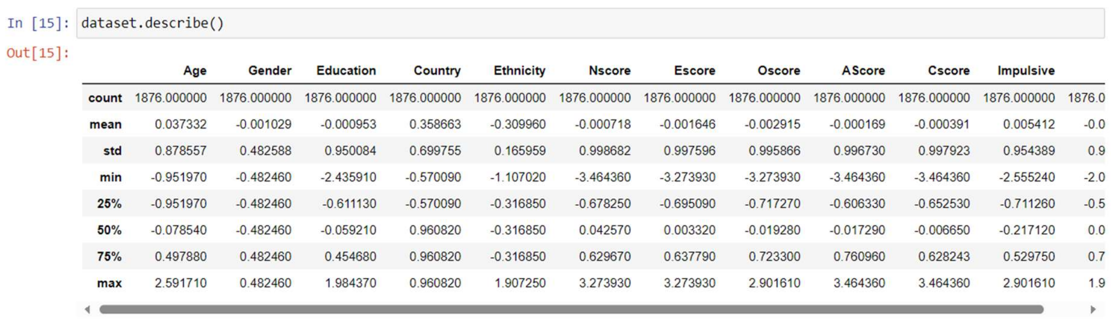
(b) dataset.shape

This code returns the number of rows and columns in the dataframe, which is useful for understanding its size and organisation.



```
In [3]: #number of rows and columns
        dataset.shape
Out[3]: (1884, 32)
```

(c) dataset.describe()



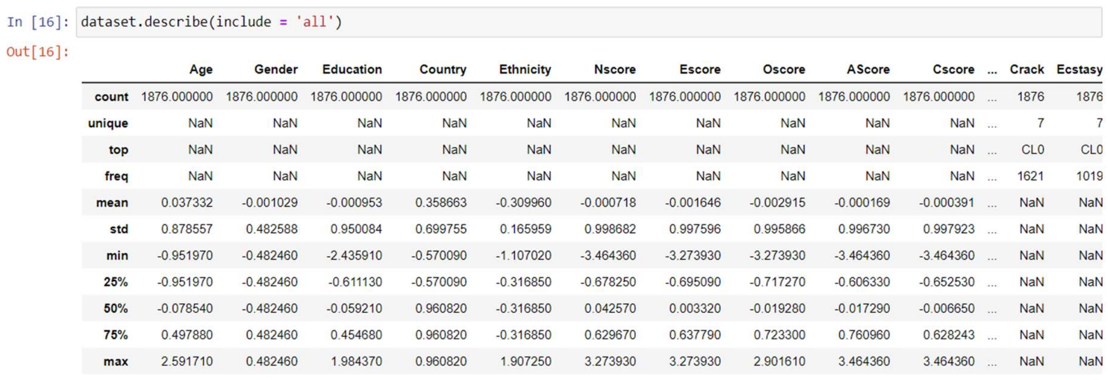
```
In [15]: dataset.describe()
Out[15]:
```

	Age	Gender	Education	Country	Ethnicity	Nscore	Escore	Oscore	AScore	Cscore	Impulsive	
count	1876.000000	1876.000000	1876.000000	1876.000000	1876.000000	1876.000000	1876.000000	1876.000000	1876.000000	1876.000000	1876.000000	1876.0
mean	0.037332	-0.001029	-0.000953	0.358663	-0.309960	-0.000718	-0.001646	-0.002915	-0.000169	-0.000391	0.005412	-0.0
std	0.878557	0.482588	0.950084	0.699755	0.165959	0.998682	0.997596	0.995866	0.996730	0.997923	0.954389	0.9
min	-0.951970	-0.482460	-2.435910	-0.570090	-1.107020	-3.464360	-3.273930	-3.273930	-3.464360	-3.464360	-2.555240	-2.0
25%	-0.951970	-0.482460	-0.611130	-0.570090	-0.316850	-0.678250	-0.695090	-0.717270	-0.606330	-0.652530	-0.711260	-0.5
50%	-0.078540	-0.482460	-0.059210	0.960820	-0.316850	0.042570	0.003320	-0.019280	-0.017290	-0.006650	-0.217120	0.0
75%	0.497880	0.482460	0.454680	0.960820	-0.316850	0.629670	0.637790	0.723300	0.760960	0.628243	0.529750	0.7
max	2.591710	0.482460	1.984370	0.960820	1.907250	3.273930	3.273930	2.901610	3.464360	3.464360	2.901610	1.9

A wide range of summary statistics is generated by this code, with NaN (Not a Number) values excluded. Included in the statistics are the following: count, mean, standard deviation (std), min, 25th, 50th, and 75th percentiles.

The results index for numerical data will consist of the following: "count," "mean," "std," "min," "25%," "50%," "75%," and "max." The results' index for object data (such as strings or timestamps) will contain the terms "count," "unique," "top," and "freq." It is most typical to have the value "top" and the frequency "freq" together.

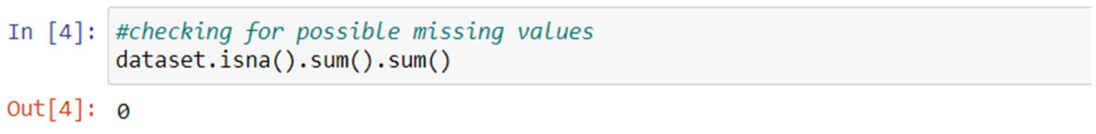
To return the above information for all columns of the dataset, use the following code: dataset.describe(include = 'all')



```
In [16]: dataset.describe(include = 'all')
Out[16]:
```

	Age	Gender	Education	Country	Ethnicity	Nscore	Escore	Oscore	AScore	Cscore	...	Crack	Ecstasy
count	1876.000000	1876.000000	1876.000000	1876.000000	1876.000000	1876.000000	1876.000000	1876.000000	1876.000000	1876.000000	...	1876	1876
unique	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	7	7
top	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	CL0	CL0
freq	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	1621	1019
mean	0.037332	-0.001029	-0.000953	0.358663	-0.309960	-0.000718	-0.001646	-0.002915	-0.000169	-0.000391	...	NaN	NaN
std	0.878557	0.482588	0.950084	0.699755	0.165959	0.998682	0.997596	0.995866	0.996730	0.997923	...	NaN	NaN
min	-0.951970	-0.482460	-2.435910	-0.570090	-1.107020	-3.464360	-3.273930	-3.273930	-3.464360	-3.464360	...	NaN	NaN
25%	-0.951970	-0.482460	-0.611130	-0.570090	-0.316850	-0.678250	-0.695090	-0.717270	-0.606330	-0.652530	...	NaN	NaN
50%	-0.078540	-0.482460	-0.059210	0.960820	-0.316850	0.042570	0.003320	-0.019280	-0.017290	-0.006650	...	NaN	NaN
75%	0.497880	0.482460	0.454680	0.960820	-0.316850	0.629670	0.637790	0.723300	0.760960	0.628243	...	NaN	NaN
max	2.591710	0.482460	1.984370	0.960820	1.907250	3.273930	3.273930	2.901610	3.464360	3.464360	...	NaN	NaN

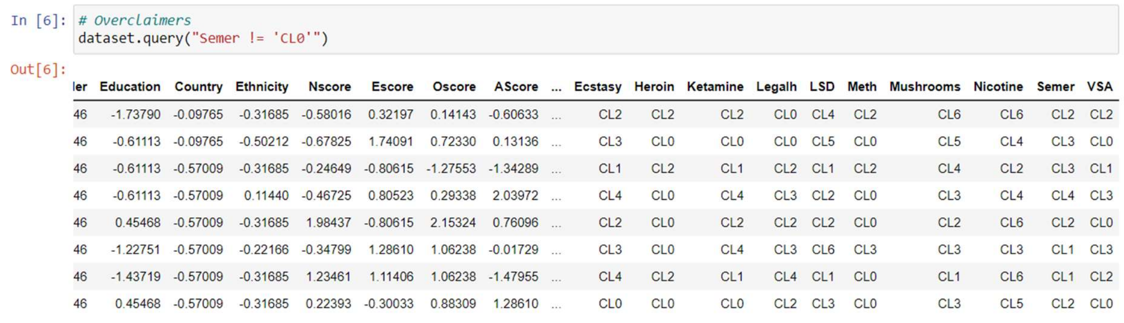
(d) dataset.isna().sum().sum()



```
In [4]: #checking for possible missing values
        dataset.isna().sum().sum()

Out[4]: 0
```

The above code is used to count the total number of missing values in a DataFrame. When collecting data, missing values are typically the most frequent problems. They may potentially cause major exceptions and errors when working with datasets. As a result, before we do any further data mining, we must manage them. Since our data collection method was an online survey, we didn't record any missing values in this instance.

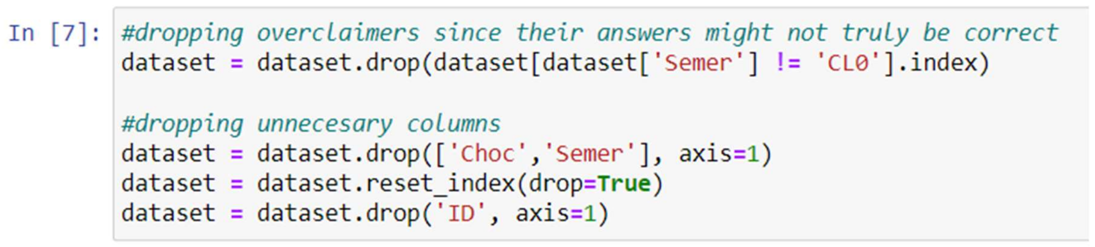
(e) dataset.query()



```
In [6]: # Overclaimers
        dataset.query("Semer != 'CL0'")

Out[6]:
ler  Education   Country  Ethnicity    Nscore    Escore    Oscore    AScore ... Ecstasy Heroin Ketamine Legalh LSD Meth Mushrooms Nicotine Semer VSA
46    -1.73790  -0.09765   -0.31685  -0.58016   0.32197   0.14143  -0.60633 ...     CL2    CL2      CL2    CL0 CL4  CL2       CL6      CL6   CL2 CL2
46    -0.61113  -0.09765   -0.50212  -0.67825   1.74091   0.72330   0.13136 ...     CL3    CL0      CL0    CL0 CL5  CL0       CL5      CL4   CL3 CL0
46    -0.61113  -0.57009   -0.31685  -0.24649  -0.80615  -1.27553  -1.34289 ...     CL1    CL2      CL1    CL2 CL1  CL2       CL4      CL2   CL3 CL1
46    -0.61113  -0.57009    0.11440  -0.46725   0.80523   0.29338   2.03972 ...     CL4    CL0      CL4    CL3 CL2  CL0       CL3      CL4   CL4 CL3
46     0.45468  -0.57009   -0.31685   1.98437  -0.80615   2.15324   0.76096 ...     CL2    CL0      CL2    CL2 CL2  CL0       CL2      CL6   CL2 CL0
46    -1.22751  -0.57009   -0.22166  -0.34799   1.28610   1.06238  -0.01729 ...     CL3    CL0      CL4    CL3 CL6  CL3       CL3      CL3   CL1 CL3
46    -1.43719  -0.57009   -0.31685   1.23461   1.11406   1.06238  -1.47955 ...     CL4    CL2      CL1    CL4 CL1  CL0       CL1      CL6   CL1 CL2
46     0.45468  -0.57009   -0.31685   0.22393  -0.30033   0.88309   1.28610 ...     CL0    CL0      CL0    CL2 CL3  CL0       CL3      CL5   CL2 CL0
```

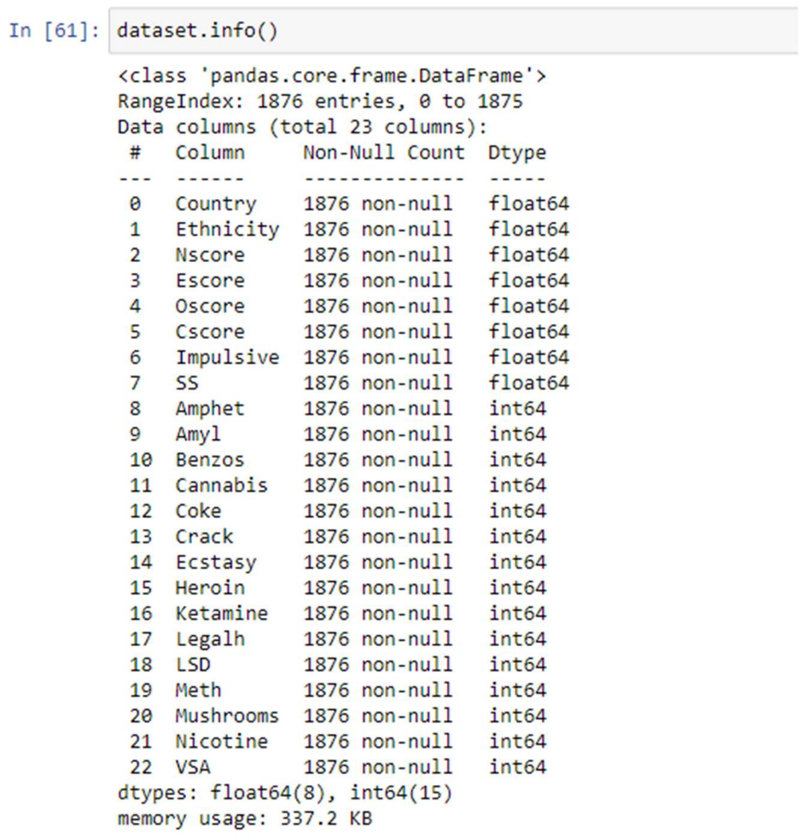
The above function is commonly used in querying data to retrieve specific information based on certain conditions or criteria. In our report, we used it to call up the fictitious drug “Semer”.

We learn from the dataset description that Semer is a fictitious drug used as a control. Those who claimed to have used Semer are considered overclaimers because it is not a legitimate drug. We shall drop "Semer" since we cannot be certain that these individuals have accurately reported their drug use, and other non-drug columns from the dataframe. We'll use the following code to accomplish this.



```
In [7]: #dropping overclaimers since their answers might not truly be correct
        dataset = dataset.drop(dataset[dataset['Semer'] != 'CL0'].index)

        #dropping unnecesary columns
        dataset = dataset.drop(['Choc','Semer'], axis=1)
        dataset = dataset.reset_index(drop=True)
        dataset = dataset.drop('ID', axis=1)
```

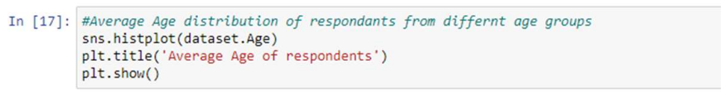
(f) Dataset.info()



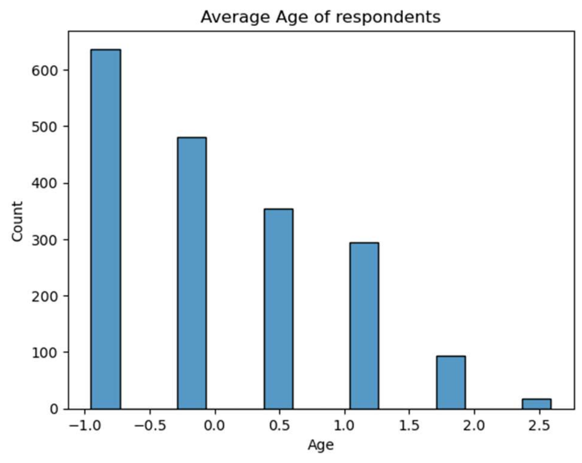
```
In [61]: dataset.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1876 entries, 0 to 1875
Data columns (total 23 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Country    1876 non-null   float64
 1   Ethnicity  1876 non-null   float64
 2   Nscore     1876 non-null   float64
 3   Escore     1876 non-null   float64
 4   Oscore     1876 non-null   float64
 5   Cscore     1876 non-null   float64
 6   Impulsive  1876 non-null   float64
 7   SS         1876 non-null   float64
 8   Amphet     1876 non-null   int64  
 9   Amyl       1876 non-null   int64  
 10  Benzos     1876 non-null   int64  
 11  Cannabis   1876 non-null   int64  
 12  Coke       1876 non-null   int64  
 13  Crack      1876 non-null   int64  
 14  Ecstasy    1876 non-null   int64  
 15  Heroin     1876 non-null   int64  
 16  Ketamine   1876 non-null   int64  
 17  Legalh     1876 non-null   int64  
 18  LSD        1876 non-null   int64  
 19  Meth       1876 non-null   int64  
 20  Mushrooms  1876 non-null   int64  
 21  Nicotine   1876 non-null   int64  
 22  VSA        1876 non-null   int64  
dtypes: float64(8), int64(15)
memory usage: 337.2 KB
```

A brief description of the dataframe is printed by the dataset.info() function. It also includes Datatypes, column names, number of rows, number of columns, number of non-null values in each column, and memory usage.

4. By Running the code below using Seaborn library, we can create a bar chart of the average age of respondents statistically from different age groups.

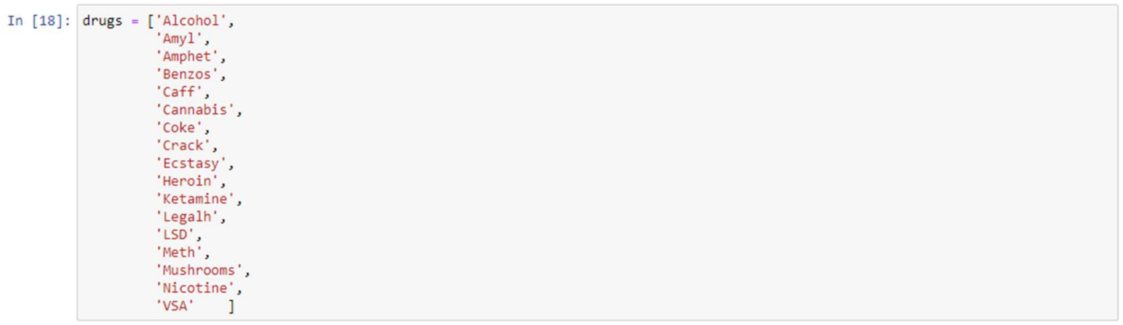


```
In [17]: #Average Age distribution of respondants from differnt age groups
         sns.histplot(dataset.Age)
         plt.title('Average Age of respondents')
         plt.show()
```



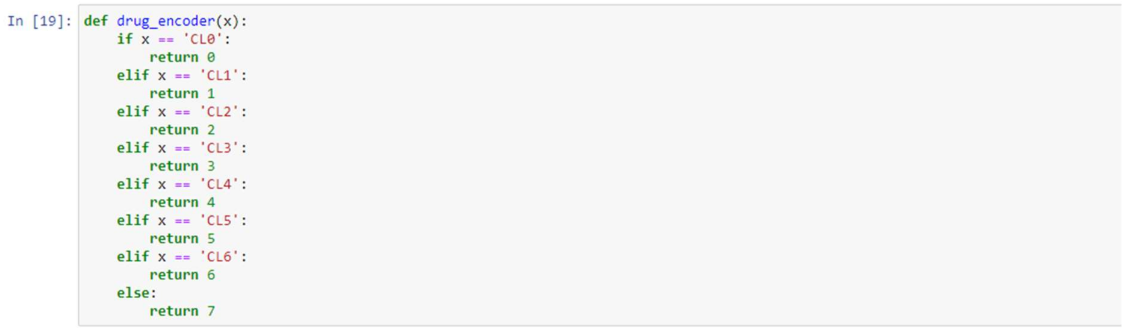
5. We can further explore some important patterns in the dataset. Using the code below, we can explore the Average use of each drug by those who responded.

   (a) First, we create a list that contains strings representing the selected drugs.



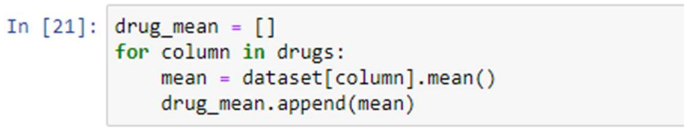
```
In [18]: drugs = ['Alcohol',
                  'Amyl',
                  'Amphet',
                  'Benzos',
                  'Caff',
                  'Cannabis',
                  'Coke',
                  'Crack',
                  'Ecstasy',
                  'Heroin',
                  'Ketamine',
                  'Legalh',
                  'LSD',
                  'Meth',
                  'Mushrooms',
                  'Nicotine',
                  'VSA'    ]
```

   (b) We now define a function called drug_encoder that takes a single argument x and returns an encoded numerical value based on the input string x. The encoding is done using a series of conditional statements.



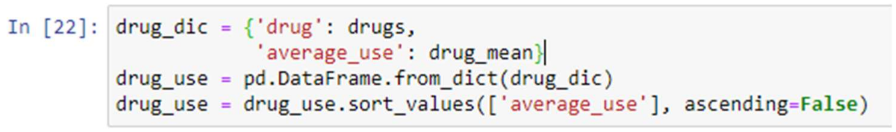
```
In [19]: def drug_encoder(x):
             if x == 'CL0':
                 return 0
             elif x == 'CL1':
                 return 1
             elif x == 'CL2':
                 return 2
             elif x == 'CL3':
                 return 3
             elif x == 'CL4':
                 return 4
             elif x == 'CL5':
                 return 5
             elif x == 'CL6':
                 return 6
             else:
                 return 7
```

   (c) Next, we calculate the mean (average) of each column in a dataset for a set of columns specified in the drugs variable.



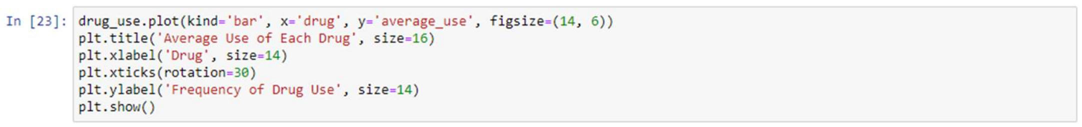
```
In [21]: drug_mean = []
         for column in drugs:
             mean = dataset[column].mean()
             drug_mean.append(mean)
```

   (d) We also created a code for sorting the Data Frame based on the 'average use' column in descending order.

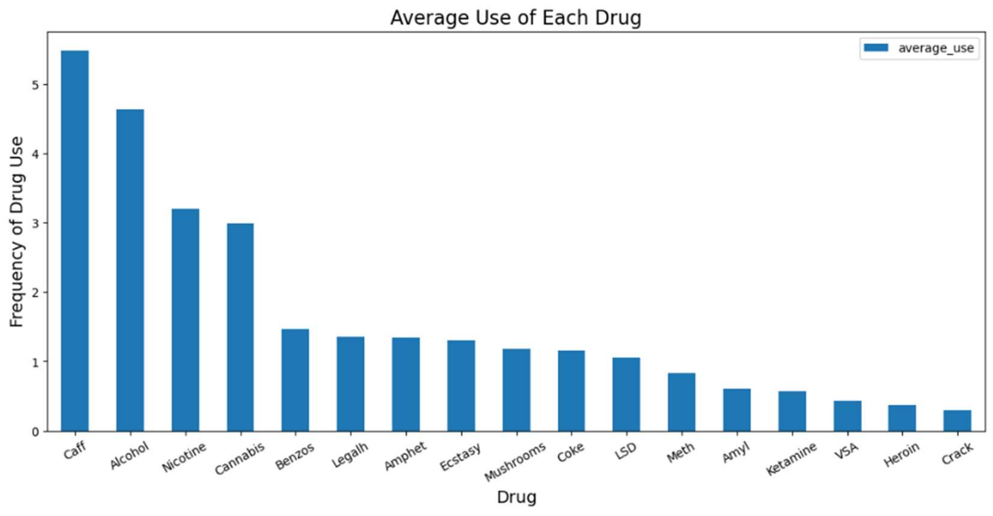


```
In [22]: drug_dic = {'drug': drugs,
                     'average_use': drug_mean}
         drug_use = pd.DataFrame.from_dict(drug_dic)
         drug_use = drug_use.sort_values(['average_use'], ascending=False)
```

   (e) Using the code below, we can now retrieve a bar plot using the ‘drug and average use column’. The plot visualizes the average use of each drug.



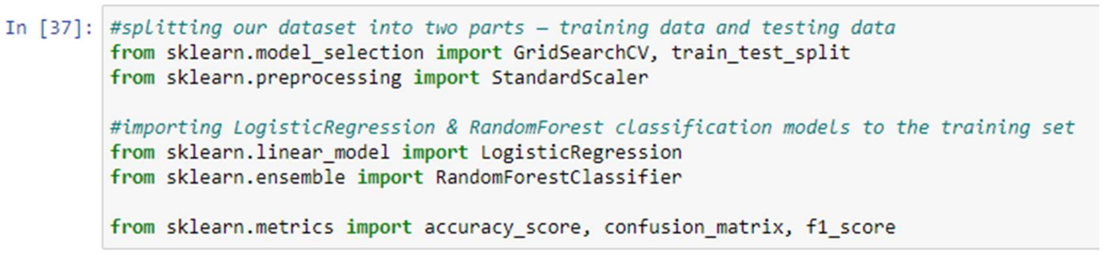
```
In [23]: drug_use.plot(kind='bar', x='drug', y='average_use', figsize=(14, 6))
         plt.title('Average Use of Each Drug', size=16)
         plt.xlabel('Drug', size=14)
         plt.xticks(rotation=30)
         plt.ylabel('Frequency of Drug Use', size=14)
         plt.show()
```



Implementation in Python

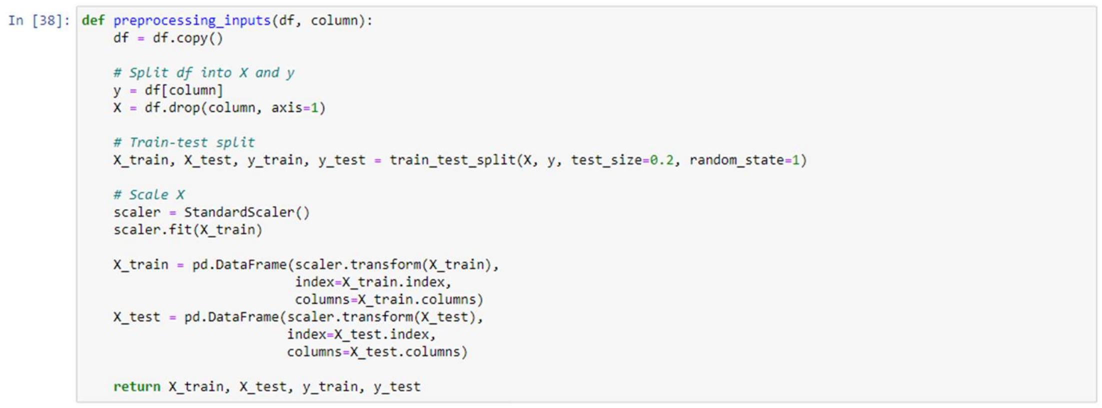
The risk of drug usage and abuse is categorised as a problem in our report. Our goal is to categorise our target market and forecast the typical drug use as well as the ages of the respondents. We'll use the scikit-learn library for classification in this part. We intend to employ two distinct classification techniques on our dataset: Random Forest and Logistic Regression Classifier.

Using the following codes, we'll import all required libraries for the classification and splitting of our dataset into two parts (training and testing data).



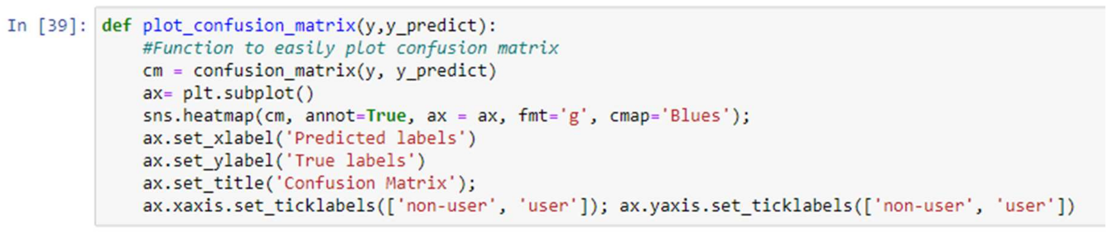
```
In [37]: #splitting our dataset into two parts – training data and testing data
         from sklearn.model_selection import GridSearchCV, train_test_split
         from sklearn.preprocessing import StandardScaler

         #importing LogisticRegression & RandomForest classification models to the training set
         from sklearn.linear_model import LogisticRegression
         from sklearn.ensemble import RandomForestClassifier

         from sklearn.metrics import accuracy_score, confusion_matrix, f1_score
```

In the classification process, our aim is to predict the sample class label (y, or output) based on its features (x, or input). Therefore, we should divide our data into input and output in the first phase by executing the following code.



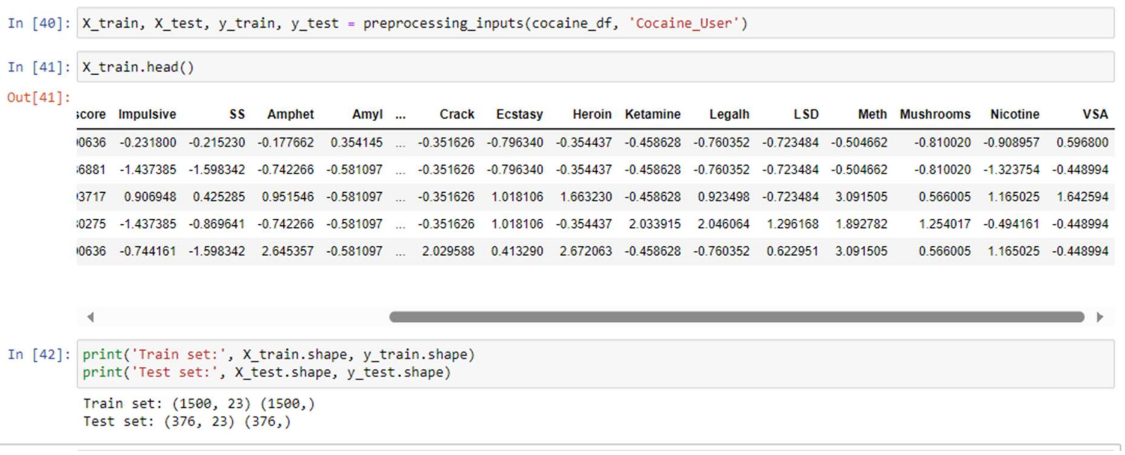
```
In [38]: def preprocessing_inputs(df, column):
             df = df.copy()

             # Split df into X and y
             y = df[column]
             X = df.drop(column, axis=1)

             # Train-test split
             X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)

             # Scale X
             scaler = StandardScaler()
             scaler.fit(X_train)

             X_train = pd.DataFrame(scaler.transform(X_train),
                                    index=X_train.index,
                                    columns=X_train.columns)
             X_test = pd.DataFrame(scaler.transform(X_test),
                                   index=X_test.index,
                                   columns=X_test.columns)

             return X_train, X_test, y_train, y_test
```

We should now create a function to visualise the performance of our classification models by plotting a confusion matrix as a heatmap with appropriate labels. To do this, execute the code below:



```
In [39]: def plot_confusion_matrix(y,y_predict):
             #Function to easily plot confusion matrix
             cm = confusion_matrix(y, y_predict)
             ax= plt.subplot()
             sns.heatmap(cm, annot=True, ax = ax, fmt='g', cmap='Blues');
             ax.set_xlabel('Predicted labels')
             ax.set_ylabel('True labels')
             ax.set_title('Confusion Matrix');
             ax.xaxis.set_ticklabels(['non-user', 'user']); ax.yaxis.set_ticklabels(['non-user', 'user'])
```

Now that we have the training and test library imported, we can proceed to evaluate and predict the "users and non-users" of the following drugs: cocaine users, meth users and heroin users. To do this, the following code.

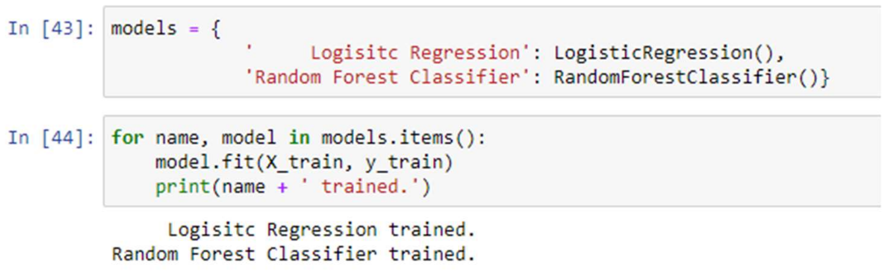


```
In [40]: X_train, X_test, y_train, y_test = preprocessing_inputs(cocaine_df, 'Cocaine_User')

In [41]: X_train.head()
Out[41]:
```

score	Impulsive	SS	Amphet	Amyl	...	Crack	Ecstasy	Heroin	Ketamine	Legalh	LSD	Meth	Mushrooms	Nicotine	VSA
0636	-0.231800	-0.215230	-0.177662	0.354145	...	-0.351626	-0.796340	-0.354437	-0.458628	-0.760352	-0.723484	-0.504662	-0.810020	-0.908957	0.596800
6881	-1.437385	-1.598342	-0.742266	-0.581097	...	-0.351626	-0.796340	-0.354437	-0.458628	-0.760352	-0.723484	-0.504662	-0.810020	-1.323754	-0.448994
3717	0.906948	0.425285	0.951546	-0.581097	...	-0.351626	1.018106	1.663230	-0.458628	0.923498	-0.723484	3.091505	0.566005	1.165025	1.642594
0275	-1.437385	-0.869641	-0.742266	-0.581097	...	-0.351626	1.018106	-0.354437	2.033915	2.046064	1.296168	1.892782	1.254017	-0.494161	-0.448994
0636	-0.744161	-1.598342	2.645357	-0.581097	...	2.029588	0.413290	2.672063	-0.458628	-0.760352	0.622951	3.091505	0.566005	1.165025	-0.448994

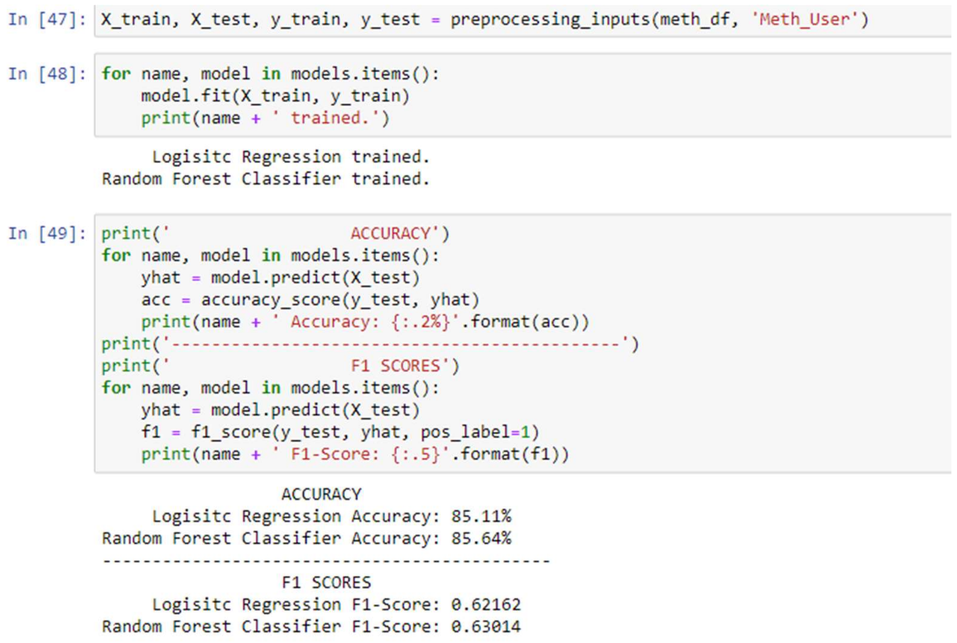
```
In [42]: print('Train set:', X_train.shape, y_train.shape)
         print('Test set:', X_test.shape, y_test.shape)

         Train set: (1500, 23) (1500,)
         Test set: (376, 23) (376,)
```

using the code below, we created a string for our classification models, after training each model, it prints a message indicating that the specific model has been trained.



```
In [43]: models = {
                 '     Logisitc Regression': LogisticRegression(),
                 'Random Forest Classifier': RandomForestClassifier()}

In [44]: for name, model in models.items():
             model.fit(X_train, y_train)
             print(name + ' trained.')

              Logisitc Regression trained.
         Random Forest Classifier trained.
```

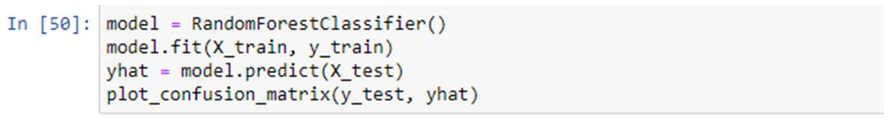
To evaluate the performance model for Methamphetamine users, execute the following code:



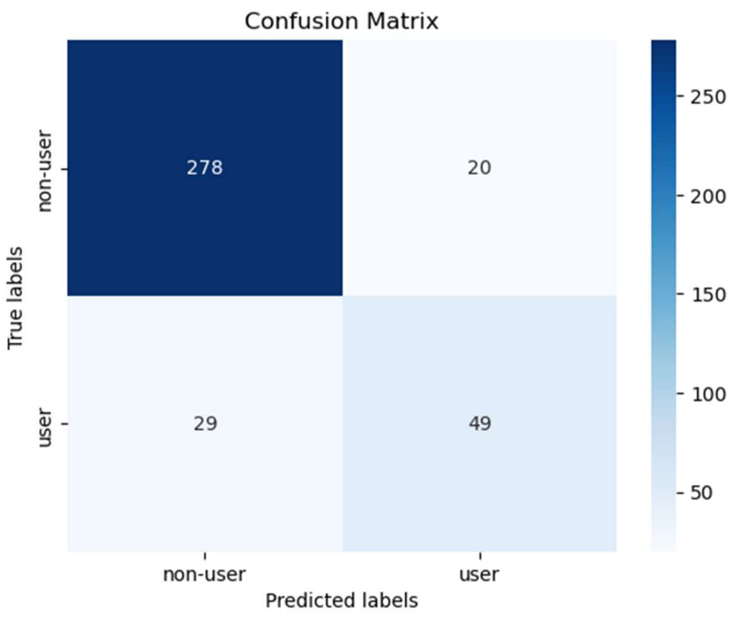
```
In [47]: X_train, X_test, y_train, y_test = preprocessing_inputs(meth_df, 'Meth_User')

In [48]: for name, model in models.items():
             model.fit(X_train, y_train)
             print(name + ' trained.')

              Logisitc Regression trained.
         Random Forest Classifier trained.

In [49]: print('                  ACCURACY')
         for name, model in models.items():
             yhat = model.predict(X_test)
             acc = accuracy_score(y_test, yhat)
             print(name + ' Accuracy: {:.2%}'.format(acc))
         print('---------------------------------------------')
         print('                  F1 SCORES')
         for name, model in models.items():
             yhat = model.predict(X_test)
             f1 = f1_score(y_test, yhat, pos_label=1)
             print(name + ' F1-Score: {:.5}'.format(f1))

                           ACCURACY
              Logisitc Regression Accuracy: 85.11%
         Random Forest Classifier Accuracy: 85.64%
         ---------------------------------------------
                           F1 SCORES
              Logisitc Regression F1-Score: 0.62162
         Random Forest Classifier F1-Score: 0.63014
```

Now we have the classification report for Methamphetamine users for Accuracy and F1-score for model evaluation.

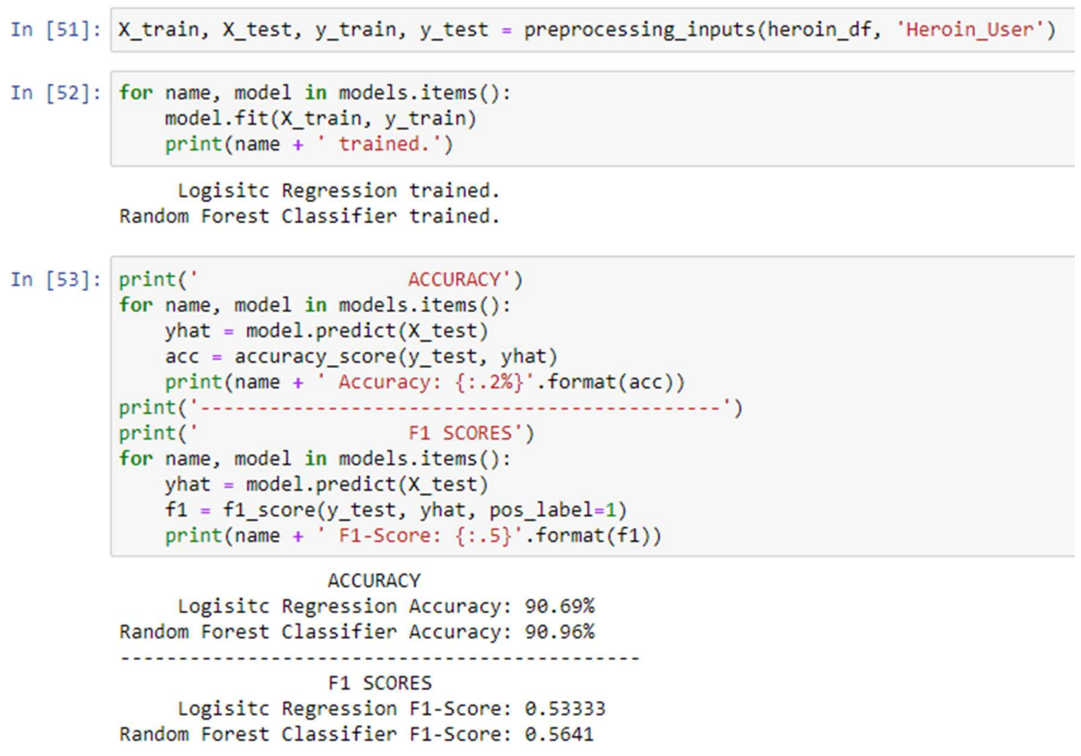
Also, we can use the seaborn heatmap to visualise the confusion matrix using this code below.



```
In [50]: model = RandomForestClassifier()
         model.fit(X_train, y_train)
         yhat = model.predict(X_test)
         plot_confusion_matrix(y_test, yhat)
```

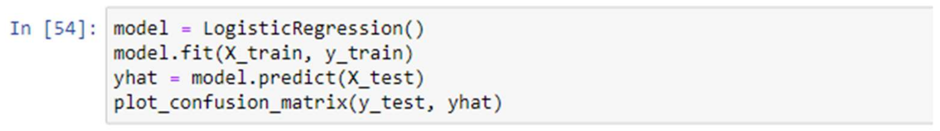


We can also evaluate the performance model for Heroin users, using the following code.

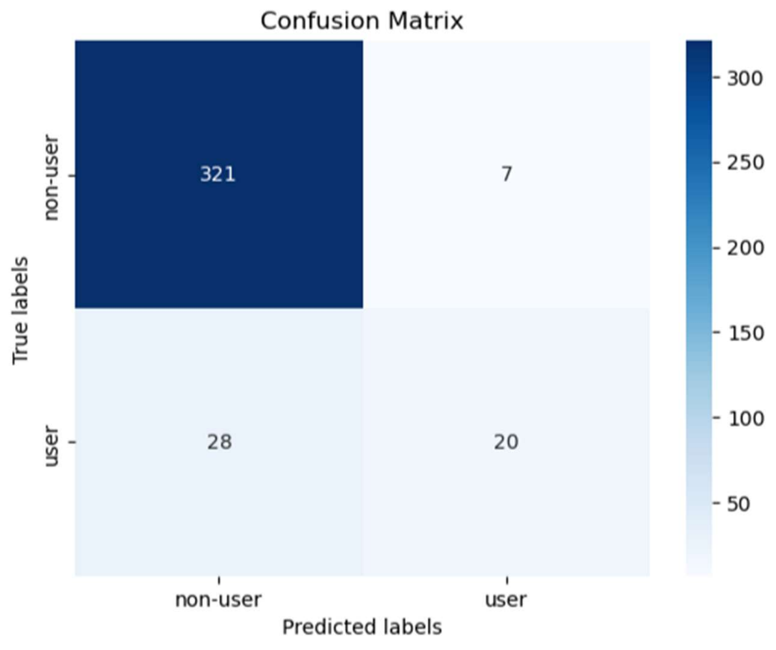


```
In [51]: X_train, X_test, y_train, y_test = preprocessing_inputs(heroin_df, 'Heroin_User')

In [52]: for name, model in models.items():
             model.fit(X_train, y_train)
             print(name + ' trained.')

              Logisitc Regression trained.
         Random Forest Classifier trained.

In [53]: print('                  ACCURACY')
         for name, model in models.items():
             yhat = model.predict(X_test)
             acc = accuracy_score(y_test, yhat)
             print(name + ' Accuracy: {:.2%}'.format(acc))
         print('---------------------------------------------')
         print('                  F1 SCORES')
         for name, model in models.items():
             yhat = model.predict(X_test)
             f1 = f1_score(y_test, yhat, pos_label=1)
             print(name + ' F1-Score: {:.5}'.format(f1))

                            ACCURACY
              Logisitc Regression Accuracy: 90.69%
         Random Forest Classifier Accuracy: 90.96%
         ---------------------------------------------
                            F1 SCORES
              Logisitc Regression F1-Score: 0.53333
         Random Forest Classifier F1-Score: 0.5641
```

To visualise using seaborn heatmap, run the following code.



```
In [54]: model = LogisticRegression()
         model.fit(X_train, y_train)
         yhat = model.predict(X_test)
         plot_confusion_matrix(y_test, yhat)
```



Results analysis and discussion

Overall, we can see that our Random Forest classifier performed the best compared to Logistic Regression. The model did well when classifying Methamphetamine. Although, this is probably due to the much larger sample size of methamphetamine users compared to Heroin users. Also, it is evident that an area of high risk is present for older men (shown in blue), but not for women. But compared to females with similar features, young males with the highest scores are notably less risky. Finally, Under a Creative Commons Attribution 4.0 International license, the dataset can be shared and adapted for any purpose as long as appropriate credit is given.

Conclusions

The result of this report shows a significant relationship between personality traits and drug usage risk. Drug-using individuals are more likely to have low scores when plotted in bar charts and higher values in evaluation models. We also conducted a thorough analysis of the average differences between the drug-using and non-using groups for two distinct drugs.

Additionally, we had some limitations because the sample was selected in a way that was unfair to the whole population, but it was still helpful in assessing drug consumption risk. We employed three distinct feature ranking methods. We removed nationality and ethnicity after inputting the feature rating. However, it was impossible to rule out the possibility that place of residence and ethnicity could be significant risk factors for drug consumption. but in our case, the dataset lacks sufficient data for most nations and ethnicities to demonstrate the usefulness of this information.