# Determining the Factors Contributing to the Cause of Death Due to Drug Overdose in the Most Affected Regions of the USA

Gundarapu Sai Varshith Reddy, Manasi Somasundaram, Ruchika Reddy Chimmula, Sadhveer Punuguti, Sai Sreya Tummala, Vaishali Lavangu

Indiana University-Purdue University, Indianapolis, IN 46202, USA,

gvreddy@iu.edu, masoma@iu.edu, ruchimmu@iu.edu, sapunu@iu.edu, tummalas@iu.edu, vlavangu@iu.edu

**Abstract**: The primary goal of this project is to analyze the leading cause of unintentional overdose in the most heavily afflicted areas of the United States. The major objective is to figure out how to forecast death from these causes so that preventative actions may be performed. We discovered the role and measure of each causal component in contributing to mortality using data analysis. We hope that by doing so, people will become more aware of the primary causes of mortality from drug overdose and will be able to use this as a preventative step.

**Keywords**: Drugs, Drug Overdose, Overdose, Mortality, Death, Data Analysis, Data Visualization, SQL, Python.

## 1. Project scope:

### 1.1. Introduction

Drug overdoses are a major concern in health care, public health, and society at large. Serious complications ending in death can arise from this drug overdosage. Overdoses can occur due to medical errors or active consumption of narcotic drugs like opioids. According to CDC data, in 2019, 70,630 drug overdose deaths occurred in the United States [1]. Children are at higher risk of accidental overdosage of drugs.

In the United States, drug overdose deaths are at an all-time high. Prescription opioids have long been the most common drug involved in overdose deaths, but heroin and synthetic opioids (most notably illicit fentanyl [2] are increasingly being linked to overdose deaths. Synthetic opioids are also becoming more common in illegal drug supply such as heroin, cocaine, methamphetamine, and counterfeit pharmaceuticals. Synthetic opioids' role in overdose deaths involving other substances is not well understood, which makes it difficult to develop effective clinical and public health solutions [3]. Identifying opportunities to intervene before an overdose death and putting in place evidence-based prevention policies, programs, and practices could help save lives. Overdoses using IMFs have unique characteristics, such as fast overdose progression and opioid and stimulant co-involvement, which should be addressed in strategies. Preventing the misuse of prescription opioids and stimulants, as well as illegal drug usage, should be a part of these efforts.

Historically, drug overdoses among men have much outweighed those among women. When looking at patterns of substance misuse, such as overall prevalence rates and substances of choice, national data consistently demonstrate that gender is a crucial factor to examine.

### 1.2. Aim:

The main purpose of this study is to know the prime cause of accidental overdosage in the most affected regions of the United States.

**Research questions:**

- To study the cause of death in terms of drug availability and mode of drug intake.
- Role of narcotic drugs contributing to death severity.

**Prediction:**

- Possibly predict the description of injury and pharmacological class of drug responsible for mortality.
- Implement clustering to allocate drugs based on their pharmacological class and assign appropriate clustered drugs to the test data inputs.

Below are the two hypotheses for our formulated research questions:

**Null Hypothesis:**

Mode of drug intake, drug availability, and role of narcotics do not affect the cause of death and death severity and the description of the injury and pharmacological class of drug responsible for mortality cannot be predicted.

**Alternative Hypothesis:**

Mode of drug intake, drug availability, and role of narcotics, affect the cause of death and death severity, and the description of the injury and a pharmacological class of drug responsible for mortality can be predicted.

### 1.3. Purpose:

There's been a tremendous increase in deaths due to drug overdosage in recent years. So, we would like to find the association between the available drugs in the region and their mode of consumption which can lead to death due to overdosage. We would like to possibly predict the description of the injury and pharmacological class of drug responsible for mortality. We are also interested to categorize drugs based on their pharmacological class to assign them into appropriate groups for data inputs. So, we would like to perform EDA on Hartford and New Haven and predictive modeling on the entire dataset. In addition, we would like to understand the role of narcotics (if any) in determining the death severity. This study becomes paramount in understanding the reasons and contributing factors for drug overdose to enlighten people about narcotics and their adverse effects of it.

## 2. Methodology:

### 2.1. Steps:

The ultimate goal of our project is to determine the cause of death in terms of drug availability and mode of drug intake and also to study the influence of narcotics in causing mortality if any. Based on the proposed factors contributing to death we would like to predict the Description of injury using SQL and Python, to achieve this we have used tools such as phpMyAdmin, and Python Jupyter notebook.

Our project's approach consists of four steps, which will be discussed in further detail later in the report.

1. Data Collection

2. Data Extraction and Storage

3. Data Analysis

4. Data Visualization

### 2.2. Proposed team members and their responsibilities

Team Members' Responsibilities: Below are the originally proposed team members' responsibilities which were assigned after a thorough discussion of each member's interests and strengths.

| Team Members | Responsibilities |
| --- | --- |
| | |

| | |
|---|---|
| 1. Gundarapu Sai Varshith Reddy<br>2. Vaishali Lavangu | • Data extraction using Python<br>• Normality test using python<br>• Project proposal |
| 3. Ruchika Reddy Chimmula<br>4. Sai Sreya Tummala | • Data Analysis<br>• Final Project Report<br>• ML modeling |
| 5. Manasi Somasundaram<br>6. Sadhveer Punugoti | • Data Analysis<br>• Data Visualization using Python Seaborn, Matplotlib.<br>• Statistical significance of results. |

Fig.1. Original responsibilities of team members

### 2.3. Actual contributions from individual Team members:

We initially identified the strengths of each member and divided the work accordingly. But as we were working through the project few changes to the team members' responsibilities were required as most of our team members are from a clinical background. Below is the updated table of team members' responsibilities.

| Team Members | Responsibilities |
|---|---|
| 1. Gundarapu Sai Varshith Reddy | Project presentation and proofreading |
| 2. Vaishali Lavangu | Data Collection and Cleaning |
| 3. Ruchika Reddy Chimmula | Data Analysis and proofreading |
| 4. Sai Sreya Tummala | Data Analysis and project report |
| 5. Manasi Somasundaram | Data Visualization and project presentation |
| 6. Sadhveer Punugoti | Data Cleaning and Visualization |

Fig.2. Revised responsibilities of team members

### 2.4. Project Challenges:

The very first challenge we encountered was with the dataset itself. We happened to choose this from a secondary source, i.e., Kaggle. This led to many missing columns and values. Also, this dataset was previously explored which limited our research questions and led to improper exploration of the data.

Data cleaning has turned out to be cumbersome as the data contained many null values. As our data is mostly categorical, converting it to numerical values took great amount of time.

We had done grouping of the columns to fit the data to perform modelling. This took a lot of effort as there are many unique values to each column. We even had to do Label encoding to convert categorical values to continuous data. We faced issues while balancing the data as model performance depends on class balance.

Above all, our group contains members are from clinical backgrounds and as this is our first technical project, we encountered problems exploring the data set at various stages.

## 3. Data Collection:

Our data was originally collected by the Connecticut Chief Medical Examiner on accidental overdose deaths between 2012 and 2018. Our source of data collection was from Kaggle.

**Reference link:**

https://www.kaggle.com/demonicknyght/drug-overdosedeaths-in-the-us

## 4. Data extraction and storage:

### 4.1. Data Import:

We have imported data in the form of CSV file into phpMyAdmin to a shared database named I501So22grp11_db. We have then established a connection between MySql and Python using the following script.

**MySql connection**

```
In [3]: myvars = {}
with open("tummalas-mysql-password") as myfile:
    for line in myfile:
        name, var = line.partition(":")[::2]
        myvars[name.strip()] = var.strip()

myvars.keys()

conn = MySQLdb.connect(host="localhost", user=myvars['DB username'], passwd=myvars['DB password'], db='I501Sp22grp11_db
cursor = conn.cursor()

cursor.execute('select * from DRUG_OVERDOSE');
rows = cursor.fetchall()
```

Fig.3. Python code to establish SQL connection

### 4.2. Data Extraction:

The selection of attributes from our dataset was the initial step in the data extraction and storage stage; there were roughly 30 different attributes in total, but we only selected about 9 columns, to investigate the cause of death in terms of drug availability and mode of drug intake as well as to possibly anticipate the type of injury based on age, gender, resident city, residence county, and geography.

The factors that contributed to our project are listed below:

1. Age
2. Sex
3. ResidenceCity_other
4. ResidenceCounty_others
5. Location
6. COD
7. DeathCity
8. PharmacologicalClass
9. DescriptionofInjury_others

The most important attributes in our data analysis are presented here.

- Description of injury_others
- Cause of death
- Death city

We ensured to study the dataset thoroughly because we encountered challenges in it, and as a result, we wound up using the description of injury to others by grouping the contents in that column and using it as a key attribute in order to meet our predictions and research questions.

Death city and description of injury were used to study the cause of death.

The pharmacological class was primarily utilized to implement clustering and distribute drugs based on their pharmacological class, as well as assign appropriate clustered drugs to test data inputs.

### 4.3. Data Cleaning:

The process of data cleaning involved steps such as selecting the attributes which are useful to answer the research questions and do the prediction. Our data contained a large number of null values. This process included the identification of columns containing null values. Columns containing more than fifty percent of null values were identified and dropped. As our data is completely categorical and also contained binary values. We have replaced the null values with mode except for the age column. The null values in the age column are replaced with the Mean as they contained continuous data. As our data is asymmetrical, we should have replaced the null values with median, but unknowingly we have replaced the null values in the age column with mean. The cleaned data is stored in the data frame.

```
In [8]: nullSeries = df.isna().sum(axis=0)
        print(nullSeries[nullSeries != 0])
        nullColumns = nullSeries[nullSeries != 0].index.values

Date                     2
DateType                 2
Age                      3
Sex                      6
Race                    13
ResidenceCity          173
ResidenceCounty        797
ResidenceState        1549
DeathCity                5
DeathCounty           1100
Location                24
LocationifOther       4515
DescriptionofInjury    780
InjuryPlace             66
InjuryCity            1756
InjuryCounty          2741
InjuryState           3681
OtherSignifican       4936
Other                 4670
MannerofDeath           10
ResidenceCityGeo        93
InjuryCityGeo           78
dtype: int64
```

Fig.4. Columns with null values before cleaning the data.

```
In [28]: df ['Age'] = df['Age'].fillna('41.96491571932575')
         df ['Race'] = df['Race'].fillna('White')
         df ['Sex'] = df['Sex'].fillna('Male')
         df ['ResidenceCity'] = df['ResidenceCity'].fillna('HARTFORD')
         df ['ResidenceCounty'] = df['ResidenceCounty'].fillna('HARTFORD')
         df ['ResidenceState'] = df['ResidenceState'].fillna('CT')
         df ['DeathCity'] = df['DeathCity'].fillna('HARTFORD')
         df ['DeathCounty'] = df['DeathCounty'].fillna('HARTFORD')
         df ['Location'] = df['Location'].fillna('Residence')
         df ['DescriptionofInjury'] = df['DescriptionofInjury'].fillna('Substance Abuse')
         df ['InjuryPlace'] = df['InjuryPlace'].fillna('Residence')
         df ['InjuryCity'] = df['InjuryCity'].fillna('HARTFORD')
         df ['InjuryCounty'] = df['InjuryCounty'].fillna('HARTFORD')
         df ['MannerofDeath'] = df['MannerofDeath'].fillna('Accident')
         df ['ResidenceCityGeo'] = df['ResidenceCityGeo'].fillna('HARTFORD, CT\n(41.765775, -72.673356)')
         df ['InjuryCityGeo'] = df['InjuryCityGeo'].fillna('CT\n(41.575155, -72.738288)')
```

Fig.5. Replacing the null values with mean and mode for the respective columns.

```
In [29]: df.isna().sum(axis=0)

Out[29]: ID                      0
         Age                     0
         Sex                     0
         Race                    0
         ResidenceCity           0
         ResidenceCounty         0
         ResidenceState          0
         DeathCity               0
         DeathCounty             0
         Location                0
         DescriptionofInjury     0
         InjuryPlace             0
         InjuryCity              0
         InjuryCounty            0
         COD                     0
         Heroin                  0
         Cocaine                 0
         Fentanyl                0
         Fentanyl_Analogue       0
         Oxycodone               0
         Oxymorphone             0
         Ethanol                 0
         Hydrocodone             0
         Benzodiazepine          0
         Methadone               0
         Amphet                  0
         Tramad                  0
         Morphine_NotHeroin      0
         Hydromorphone           0
         OpiateNOS               0
         AnyOpioid               0
         MannerofDeath           0
         DeathCityGeo            0
         ResidenceCityGeo        0
         InjuryCityGeo           0
         dtype: int64
```

Fig.6. Columns after cleaning the data.

# 5. Data analysis:

## 5.1. Correlation:

In order to answer our first research question, we checked the correlation between different attributes and plotted a heat map. After interpreting the heatmap we noted that the correlation between the comparing factors is not relatively significant.

```
In [69]: df_correlation = df_correlation_copy.corr(method = 'spearman')
         df_correlation
```

Out[69]:

| | Sex | DeathCity | Location | COD | Heroin | Cocaine | Fentanyl_Analogue | Oxycodone | Oxymorphone | Ethanol | ... | Benzodiazep |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Sex** | 1.000000 | 0.011318 | 0.015464 | -0.058426 | 0.104544 | 0.013236 | 0.031829 | -0.063198 | -0.021677 | 0.051431 | ... | -0.116 |
| **DeathCity** | 0.011318 | 1.000000 | 0.318960 | -0.018098 | -0.012572 | -0.078987 | 0.008619 | 0.093286 | 0.049758 | 0.009028 | ... | 0.078 |
| **Location** | 0.015464 | 0.318960 | 1.000000 | -0.004218 | -0.023087 | -0.067473 | -0.007989 | 0.081811 | 0.030875 | 0.008483 | ... | 0.066 |
| **COD** | -0.058426 | -0.018098 | -0.004218 | 1.000000 | -0.128955 | 0.222853 | 0.116868 | 0.088679 | -0.005140 | 0.187714 | ... | 0.191 |
| **Heroin** | 0.104544 | -0.012572 | -0.023087 | -0.128955 | 1.000000 | -0.002140 | -0.009908 | -0.224786 | -0.112995 | -0.025313 | ... | -0.104 |
| **Cocaine** | 0.013236 | -0.078987 | -0.067473 | 0.222853 | -0.002140 | 1.000000 | 0.038905 | -0.116250 | -0.063032 | -0.044395 | ... | -0.127 |
| **Fentanyl_Analogue** | 0.031829 | 0.008619 | -0.007989 | 0.116868 | -0.009908 | 0.038905 | 1.000000 | -0.053041 | -0.037092 | 0.006837 | ... | 0.007 |
| **Oxycodone** | -0.063198 | 0.093286 | 0.081811 | 0.088679 | -0.224786 | -0.116250 | -0.053041 | 1.000000 | 0.320290 | 0.005251 | ... | 0.136 |
| **Oxymorphone** | -0.021677 | 0.049758 | 0.030875 | -0.005140 | -0.112995 | -0.063032 | -0.037092 | 0.320290 | 1.000000 | 0.001961 | ... | 0.066 |
| **Ethanol** | 0.051431 | 0.009028 | 0.008483 | 0.187714 | -0.025313 | -0.044395 | 0.006837 | 0.005251 | 0.001961 | 1.000000 | ... | -0.004 |
| **Hydrocodone** | -0.072492 | 0.027708 | 0.049841 | 0.026576 | -0.087231 | -0.060304 | -0.024525 | 0.048206 | -0.004497 | 0.033909 | ... | 0.035 |
| **Benzodiazepine** | -0.116265 | 0.078505 | 0.066047 | 0.191725 | -0.104391 | -0.127564 | 0.007820 | 0.136506 | 0.066741 | -0.004199 | ... | 1.000 |
| **Methadone** | -0.054326 | -0.007498 | 0.031649 | 0.101364 | -0.125301 | -0.062315 | -0.028285 | -0.040371 | -0.004821 | -0.037363 | ... | 0.112 |
| **Amphet** | -0.017367 | 0.034390 | 0.012679 | 0.058797 | -0.003988 | 0.004012 | 0.025009 | -0.006639 | 0.004986 | -0.031072 | ... | 0.056 |
| **Tramad** | -0.032708 | 0.003570 | -0.001373 | 0.047039 | -0.038304 | -0.040055 | 0.005127 | 0.005926 | -0.006483 | 0.003603 | ... | 0.038 |
| **Hydromorphone** | -0.011352 | 0.029808 | 0.017736 | 0.028546 | -0.002160 | -0.021159 | -0.009571 | 0.034915 | 0.009186 | -0.026821 | ... | 0.028 |
| **OpiateNOS** | -0.031474 | -0.032735 | -0.049325 | 0.053892 | -0.131226 | -0.056662 | -0.038037 | 0.030393 | -0.009012 | -0.012245 | ... | 0.030 |
| **criptionofInjury_others** | 0.101669 | -0.041330 | -0.002957 | 0.094420 | 0.196038 | 0.174189 | 0.141972 | -0.224015 | -0.140050 | 0.013043 | ... | -0.050 |
| **counts** | -0.002802 | -0.784529 | -0.384237 | 0.029606 | 0.020339 | 0.104104 | -0.000529 | -0.096487 | -0.057943 | -0.005827 | ... | -0.083 |
| **ResidenceCity_others** | -0.009843 | 0.419897 | 0.079581 | -0.047020 | -0.018789 | -0.133426 | -0.024666 | 0.075364 | 0.044701 | -0.018836 | ... | 0.075 |
| **sidenceCounty_others** | 0.001030 | 0.287598 | 0.015358 | -0.003472 | -0.030425 | -0.010076 | 0.026097 | 0.016264 | -0.019388 | -0.030202 | ... | -0.021 |

ows × 21 columns

Fig.7. Python code for generating Spearman correlation between attributes

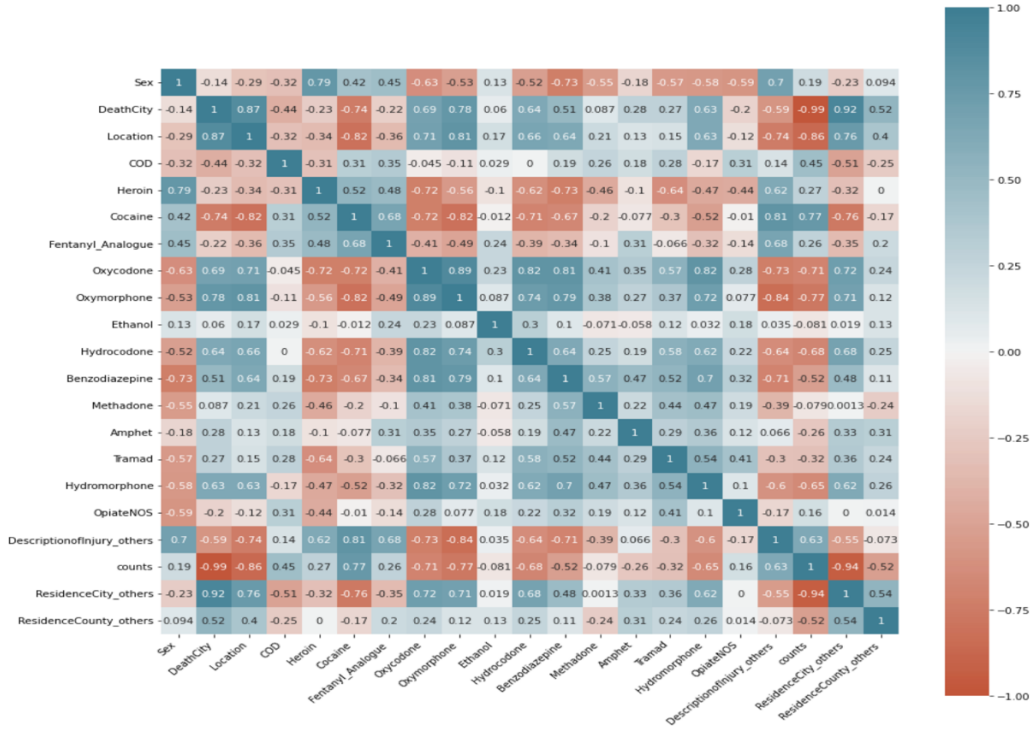


Fig.8. Heatmap showing correlation among attributes

Through the heatmap and the values of correlation, we identified that the cause of death (COD) is negatively correlated to Drug availability (DeathCity_Others), and it is positively correlated to the Mode of drug intake (DescriptionofInjury_others).

**5.2. Statistical tests:**

**Kruskal-Wallis Test:**

This statistical test perfectly fits our data set to answer the research question. We wanted to study the cause of death in terms of drug availability and mode of drug intake. This dependent variable is the Cause of Death, and the independent variables are Description of Injury and Death city. After doing this test we have found that the p-value is less than 0.05, and we rejected the null hypothesis.

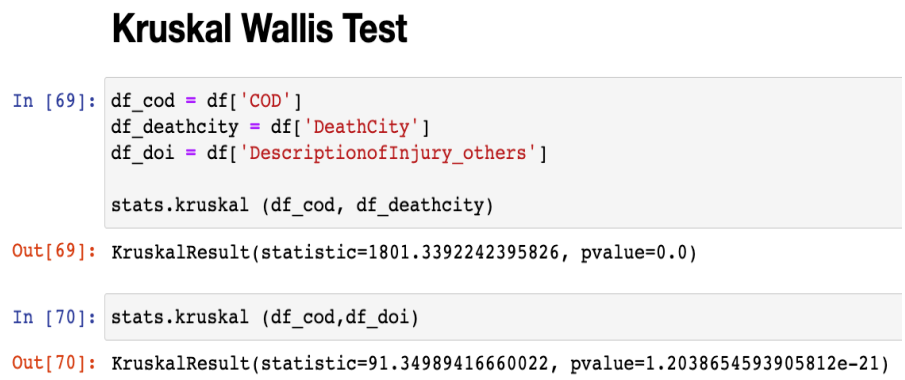


Fig.9. Python code to display Kruskal Wallis test among the variables

**Wilcoxon Signed Ranked Test:**

Wilcoxon Signed Ranked Test is the second statistical test we have performed. We have used Cause of death, description of injury, and death city as the variables. After doing the test we found that our p-value is less than our statistical value, so we reject the null hypothesis.

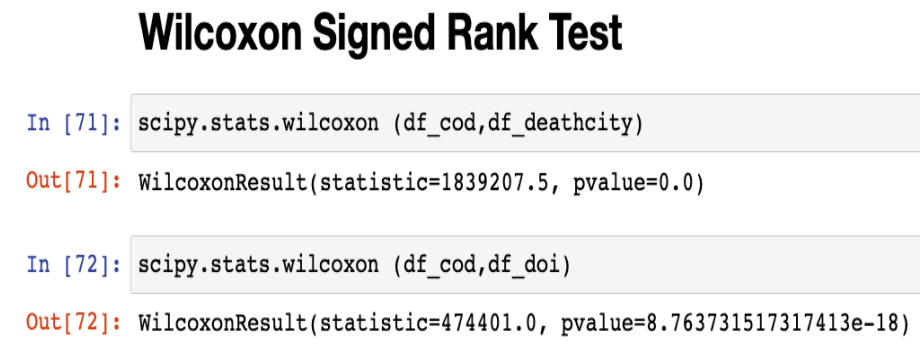


Fig.10. Python code to display Wilcoxon Signed Rank Test among the variables

**ANOVA Test:**

One way ANOVA is used is when the data is normally distributed. In our dataset, data is not normally distributed but we have more than 5000 rows where this test is applicable. We have used Cause of Death, Description of Injury,

and death city as variables. After performing the test, we found our p-value is less than 0.05, so we rejected the null hypothesis.

ANOVA

```
In [76]: scipy.stats.f_oneway(df_cod,df_deathcity,df_doi)

Out[76]: F_onewayResult(statistic=2456.4912688472705, pvalue=0.0)
```

Fig.11. Python code to display ANOVA test among the variables

### 5.3. Data Modeling:

**Logistic Regression:**

We built a logistic regression model using 'Description_of_injury_others' as y and 'Age','Sex','ResidenceCity_others','ResidenceCounty_others','Location','COD','DeathCity','PharmacologicalClass' as X to make the prediction. We have imported train_test_split from sklearn and split the data to X_train, X_test, y_train and y_test. We fit the model and then printed the score of it. To improve the performance of the model, we have done 10-fold cross-validation and also balanced the class weight.

```
In [461]: X = df[['Age', 'Sex','ResidenceCity_others','ResidenceCounty_others','Location','COD','DeathCity','PharmacologicalClass
          y = df['DescriptionofInjury_others']

In [462]: from sklearn.model_selection import train_test_split
          X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=0)
```

Fig.12. Python code to define the dependent and independent variables, import train_test_split and train the data

## Logistic regression

In [464]:
```
from sklearn.linear_model import LogisticRegression
logReg = LogisticRegression(class_weight="balanced")

from sklearn.model_selection import cross_validate
cv_results = cross_validate(logReg, X, y, cv=10)
cv_results

logReg.fit(X_train, y_train)

logReg_predicted = logReg.predict(X_test)
logReg_expected = y_test
score = logReg.score(X_test, y_test)
print(score)
```

```
0.24745497259201252
```

Fig.13. Python code for Logistic Regression model

In the same way, we have built **Random Forest Classifier**, **Support Vector Machine and Decision Trees.**

## Support Vector Machine

In [506]:
```
from sklearn.svm import SVC
svm_class = SVC(gamma='auto',class_weight="balanced", probability=True)

from sklearn.model_selection import cross_validate
cv_results = cross_validate(svm_class, X, y, cv=10)
cv_results

svm_class.fit(X_train, y_train)

svm_predicted = svm_class.predict(X_test)
svm_expected = y_test
score = svm_class.score(X_test, y_test)
print(score)
```

```
0.35160532498042285
```

Fig.14. Python code for Support Vector Machine

## Random Forest

In [469]:
```
from sklearn.ensemble import RandomForestClassifier
randomforest = RandomForestClassifier(max_depth=2, random_state=0, class_weight="balanced")

from sklearn.model_selection import cross_validate
cv_results = cross_validate(randomforest, X, y, cv=10)
cv_results

randomforest.fit(X_train, y_train)

randomforest_predicted = randomforest.predict(X_test)
randomforest_expected = y_test
score = randomforest.score(X_test, y_test)
print(score)
```

```
0.1096319498825372
```

Fig.15. Python code for Random Forest Classifier

## Decision Tree

In [474]:
```
from sklearn import tree
dtc = tree.DecisionTreeClassifier(class_weight="balanced")

from sklearn.model_selection import cross_validate
cv_results = cross_validate(dtc, X, y, cv=10)
cv_results

dtc.fit(X_train, y_train)


dtc_predicted = dtc.predict(X_test)
dtc_expected = y_test
score = dtc.score(X_test, y_test)
print(score)
```

```
0.6084573218480814
```

Fig.16. Python code for Decision Trees

To determine the efficiency of the performed models, metrics like ROC curve (Receiver Operating Characteristic curve) and classification report were displayed.

AUC-Area Under the Curve is the measure of the ability of the model to distinguish between the classes and used to summarize the ROC curve.

Precision, recall, PPV (Positive Predictive Value), NPV (Negative Predictive Value) and Accuracy. Out of all the models, decision trees is able to predict the instance by only 60%. F1 score owing to this model is highest among the other models making this the best fitting model to the proposed prediction.

**Classification Report of all the models**

```
In [478]: from sklearn import metrics
          print(metrics.classification_report(logReg_expected, logReg_predicted))
          print(metrics.classification_report(randomforest_expected, randomforest_predicted))
          print(metrics.classification_report(svm_expected, svm_predicted))
          print(metrics.classification_report(dtc_expected, dtc_predicted))
```

```
              precision    recall  f1-score   support

           0       0.12      0.09      0.10        99
           1       0.02      0.12      0.03        24
           2       0.14      0.44      0.21        41
           3       0.00      0.50      0.01         2
           4       0.09      0.13      0.10       102
           5       0.03      0.36      0.05        14
           6       0.84      0.27      0.41       995

    accuracy                           0.25      1277
   macro avg       0.18      0.27      0.13      1277
weighted avg       0.68      0.25      0.34      1277

              precision    recall  f1-score   support

           0       0.17      0.02      0.04        99
           1       0.05      0.12      0.07        24
           2       0.13      0.46      0.21        41
           3       0.00      0.00      0.00         2
           4       0.06      0.14      0.09       102
           5       0.02      0.57      0.03        14
           6       0.91      0.09      0.17       995

    accuracy                           0.11      1277
   macro avg       0.19      0.20      0.09      1277
weighted avg       0.73      0.11      0.15      1277

              precision    recall  f1-score   support

           0       0.14      0.34      0.20        86
           1       0.03      0.15      0.05        20
           2       0.11      0.23      0.15        44
           3       0.00      0.00      0.00         5
           4       0.07      0.20      0.11        91
           5       0.03      0.27      0.05        11
           6       0.82      0.39      0.53      1016

    accuracy                           0.36      1273
   macro avg       0.17      0.23      0.16      1273
weighted avg       0.67      0.36      0.45      1273

              precision    recall  f1-score   support

           0       0.18      0.23      0.20        99
           1       0.02      0.04      0.03        24
           2       0.23      0.22      0.22        41
           3       0.00      0.00      0.00         2
           4       0.11      0.16      0.13       102
           5       0.00      0.00      0.00        14
           6       0.81      0.73      0.77       995

    accuracy                           0.61      1277
   macro avg       0.19      0.20      0.19      1277
weighted avg       0.66      0.61      0.63      1277
```

Fig.17. Python code to display metrics of all the models deployed

### K-means clustering:

We have performed clustering to allocate drugs into two clusters (k = 2). We were able to plot the centroids but not the corresponding data points. As the data was binarily distributed, it wasn't possible to implement clustering on this data.

```
In [494]: from sklearn.cluster import KMeans
          Kmean = KMeans(n_clusters=2)
```

Fig.18. Importing K-means cluster and assigning 2 clusters

```
In [495]: df_cols = df[['Heroin', 'Cocaine', 'Fentanyl', 'Fentanyl_Analogue',
                        'Oxycodone', 'Oxymorphone', 'Ethanol', 'Hydrocodone', 'Benzodiazepine',
                        'Methadone', 'Amphet', 'Tramad', 'Morphine_NotHeroin', 'Hydromorphone',
                        'OpiateNOS', 'AnyOpioid']].values

In [496]: df_cols

Out[496]: array([[0, 0, '1', ..., 0, 0, '0'],
                 [0, 1, '0', ..., 0, 0, '0'],
                 [1, 1, '0', ..., 0, 0, '1'],
                 ...,
                 [1, 0, '1', ..., 0, 0, '1'],
                 [0, 0, '1', ..., 0, 0, '0'],
                 [1, 0, '0', ..., 0, 0, '1']], dtype=object)
```

Fig.19. Assigning all the drugs into one variable and printing its value

```
In [501]: Kmean.fit(df_cols)

Out[501]: KMeans(n_clusters=2)

In [502]: centers = Kmean.cluster_centers_
          centers

Out[502]: array([[5.12390925e-01, 2.83769634e-01, 9.99200722e-16, 3.49040140e-04,
                  1.66841187e-01, 3.24607330e-02, 2.45375218e-01, 3.38568935e-02,
                  2.81326353e-01, 1.22513089e-01, 2.75741710e-02, 2.82722513e-02,
                  1.22164049e-02, 5.58464223e-03, 2.51308901e-02, 3.60907504e-01],
                 [4.75292004e-01, 3.16262354e-01, 1.00000000e+00, 1.74303684e-01,
                  5.75022462e-02, 6.73854447e-03, 2.43036837e-01, 9.43396226e-03,
                  2.37646002e-01, 5.52560647e-02, 3.54896676e-02, 2.20125786e-02,
                  1.34770889e-03, 4.04312668e-03, 6.28930818e-03, 6.37915544e-01]])
```

Fig.20. Displaying the centroids of K-Means clusters

## 6. Data Visualization:

Python has a plethora of plugins and packages, such as seaborn and matplotlib, that we employed as a tool for visualization, allowing us to produce graphs and plots while also reducing the load of coding large programs. To visualize our data, we employed count plots, Receiver Operating Characteristic curves (ROC), and a confusion matrix.

### 6.1. Count plots:

Count plots are a type of histogram that is used to compare counts across multiple variables in categorical data. Count plots were employed to columns such as Description of injury_others, ResidenceCity_others, Sex, ResidenceCounty_others, Location, Cause of death, Death City. Using count plots, we found the most influential variable in the above-mentioned columns that causes death.

```
In [32]: import seaborn as sns
sns.countplot(df['DescriptionofInjury_others'])

plt.show()
```

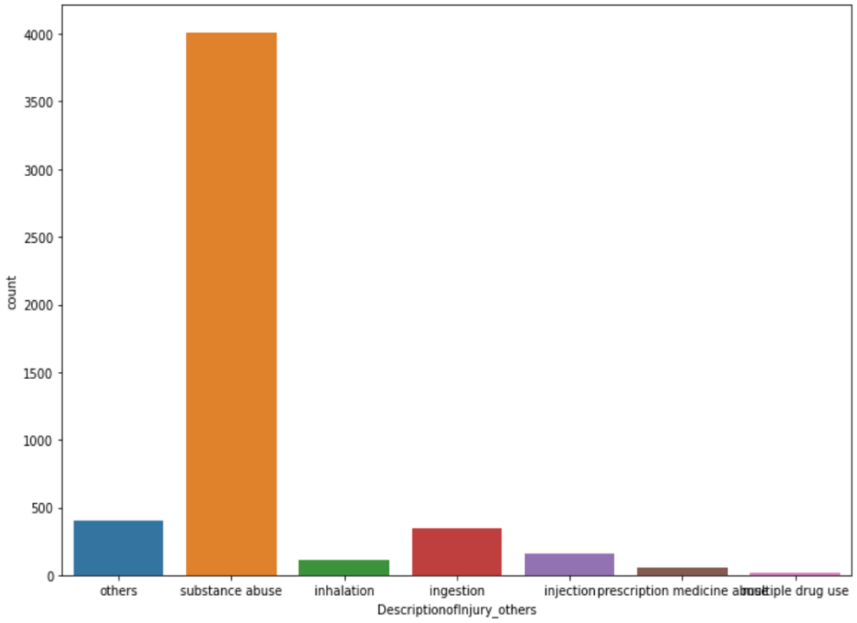


Fig.21. Python code to display counterplot of Description of Injury

```
In [37]: import seaborn as sns
sns.countplot(df['ResidenceCity_others'])


plt.show()
```

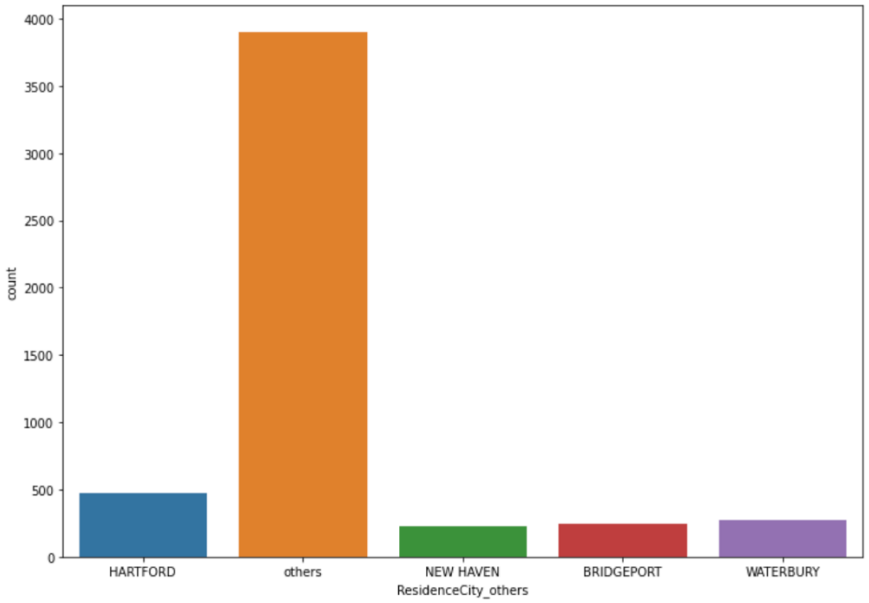


Fig.22. Python code to display counterplot of Residence City

```
In [41]: import seaborn as sns
         sns.countplot(df['Sex'])

         plt.show()
```

```
/home/students/tummalas/.local/lib/python3.8/site-packages/seaborn/_decorators.py:36: FutureWarning: Pass the followi
ng variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing ot
her arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

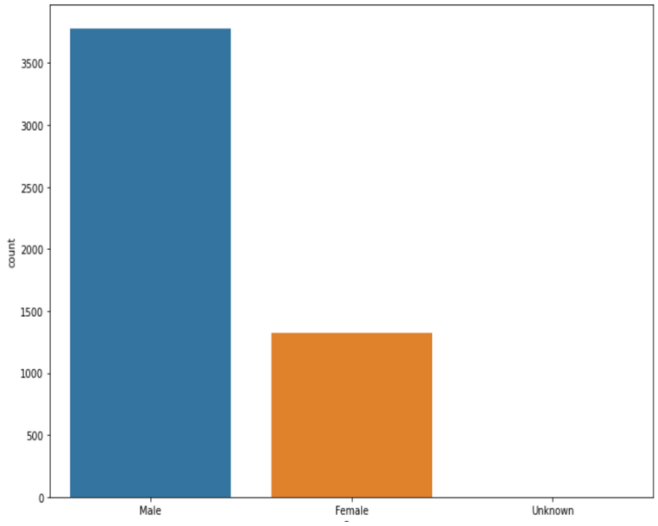


Fig.23.Python code to display counterplot of Gender

```
In [46]: import seaborn as sns
         sns.countplot(df['ResidenceCounty_others'])
         plt.show()
```

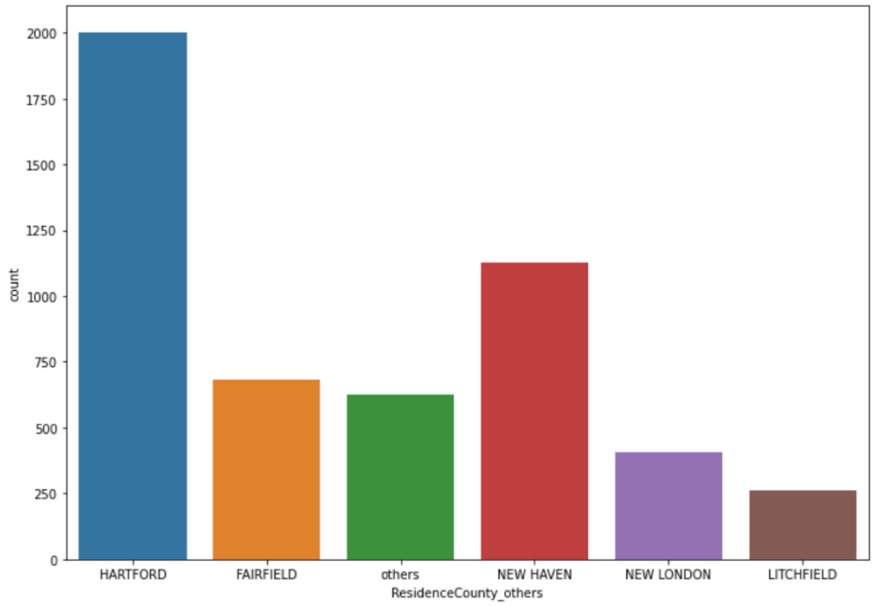


Fig.24.Python code to display counterplot of Residence County

```
In [50]: import seaborn as sns
         sns.countplot(df['Location'])
         plt.show()
```

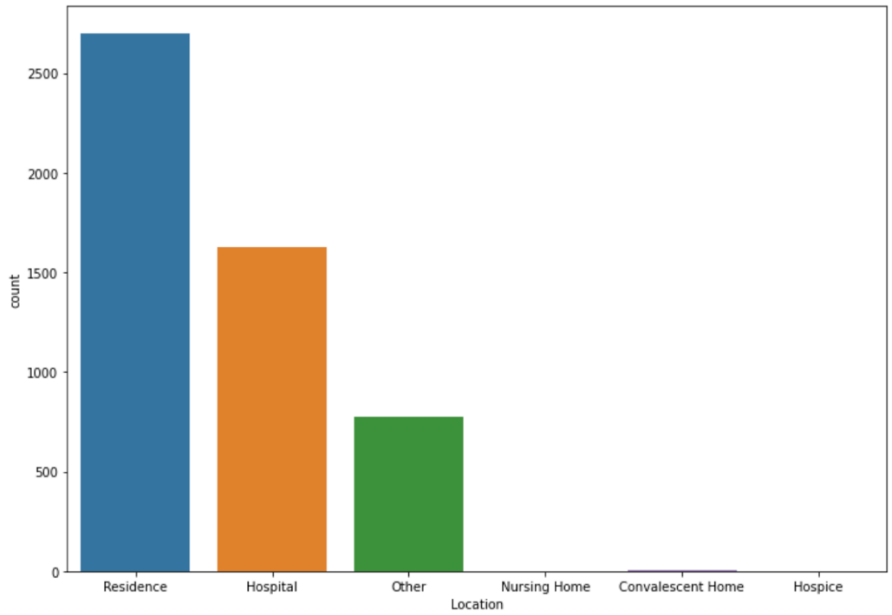


Fig.25. Python code to display counterplot of Location

```
In [55]: import seaborn as sns
         sns.countplot(df['COD'])
         plt.show()
```

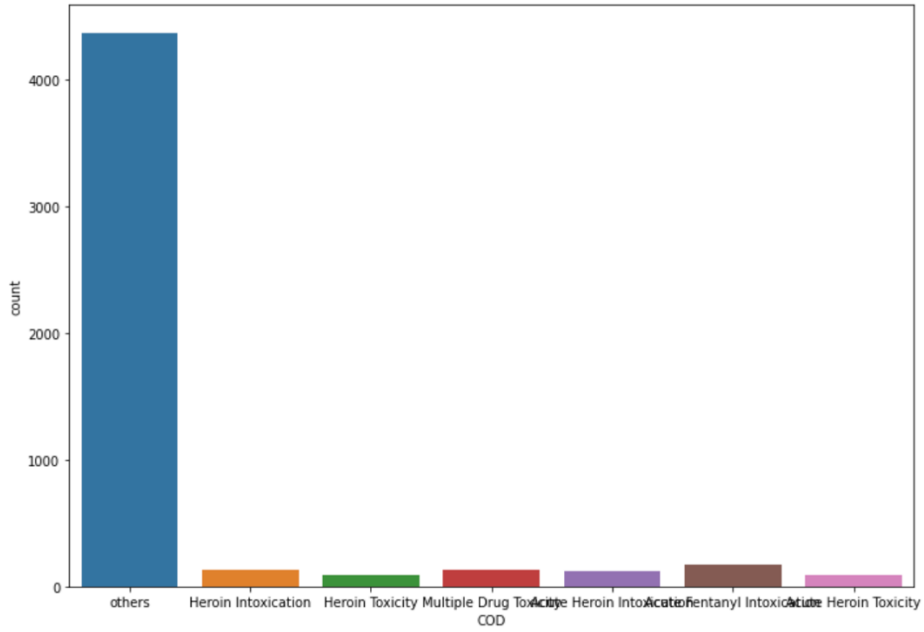


Fig.26.Python code to display counterplot of Cause of death

```
In [61]: import seaborn as sns
         sns.countplot(df['DeathCity'])
         plt.show()
```

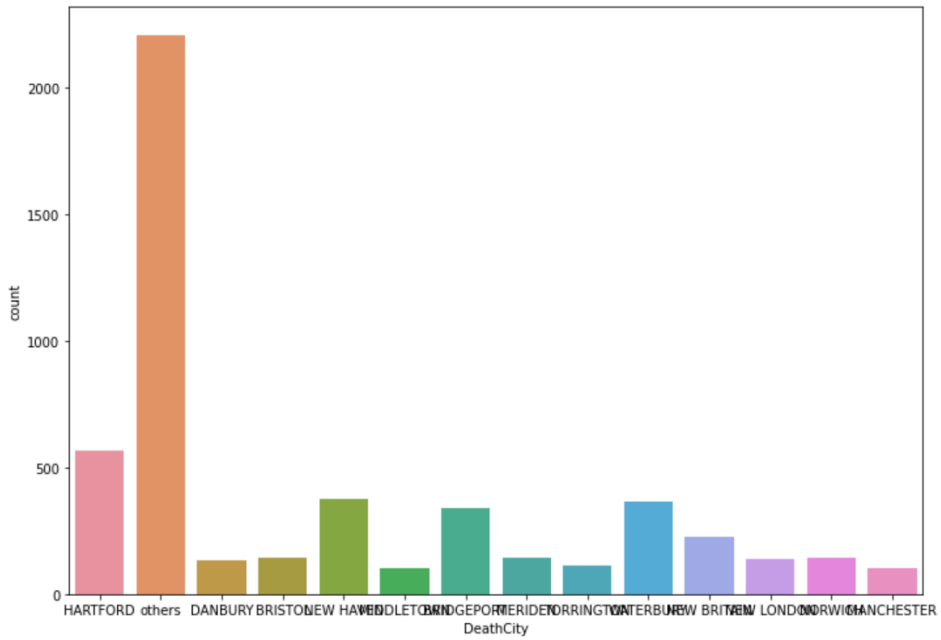


Fig.27.Python code to display counterplot of Death City

### 6.2. Receiver operating characteristic curves (ROC):

The performance of models and classification thresholds are represented using ROC curves. The highest performing classification model is evaluated using the ROC-AUC (Area under the curve) measure. Because our data was skewed, all the models' ROC curve characteristics, such as Micro-average roc curves, were greater than Macro-average roc curves. We found that the model "decision tree" is the best performing model based on the area under the curve measure and micro average roc curve.

```
In [467]: plt.rcParams['figure.figsize'] = [10, 10]

predicted_probas = logReg.predict_proba(X_test)
import matplotlib.pyplot as plt
import scikitplot as skplt
skplt.metrics.plot_roc(logReg_expected, predicted_probas)

plt.show()
```

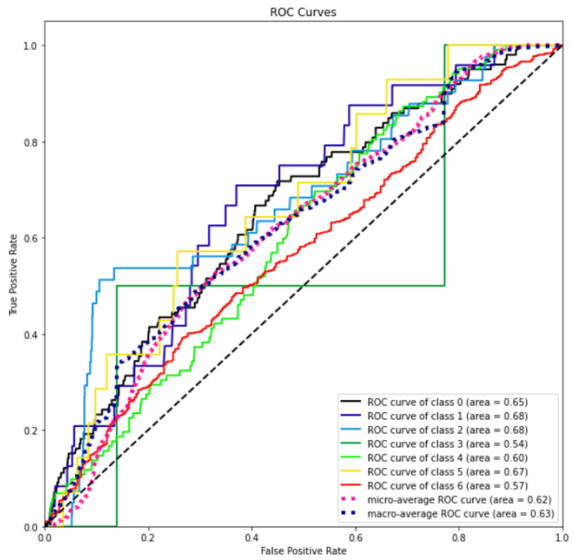


Fig.28. Python code displaying ROC curve of Logistic regression

```
In [471]: plt.rcParams['figure.figsize'] = [10, 10]

predicted_probas = randomforest.predict_proba(X_test)
import matplotlib.pyplot as plt
import scikitplot as skplt
skplt.metrics.plot_roc(randomforest_expected, predicted_probas)

plt.show()
```

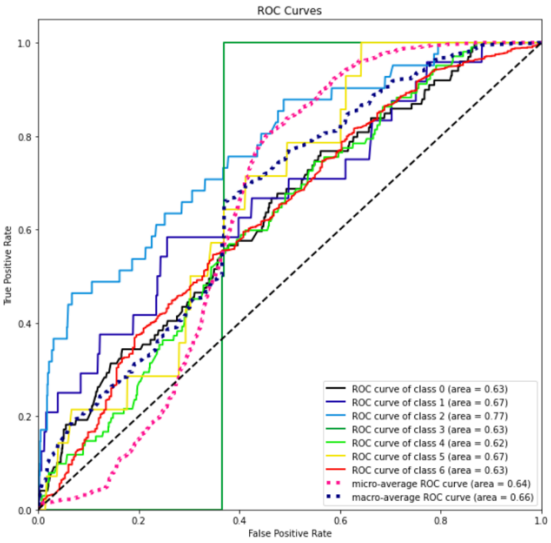


Fig.29. Python code displaying ROC curve of random Forest Classifier

```
In [508]: plt.rcParams['figure.figsize'] = [10, 10]

predicted_probas = svm_class.predict_proba(X_test)
import matplotlib.pyplot as plt
import scikitplot as skplt
skplt.metrics.plot_roc(svm_expected, predicted_probas)

plt.show()
```

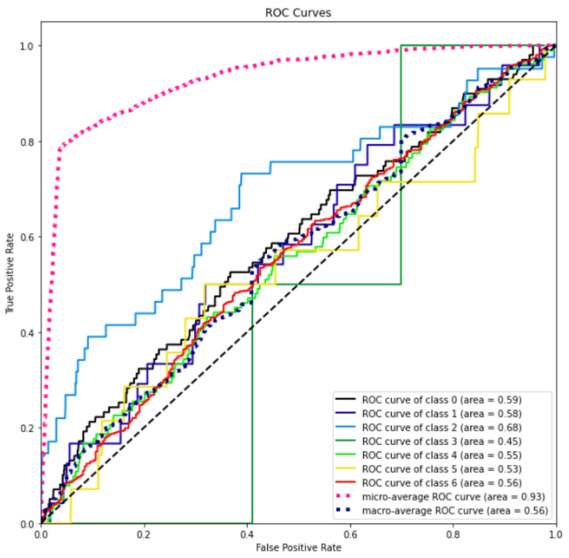


Fig.30. Python code displaying ROC curve of Support Vector Machine

```
In [476]: plt.rcParams['figure.figsize'] = [10, 10]

predicted_probas = dtc.predict_proba(X_test)
import matplotlib.pyplot as plt
import scikitplot as skplt
skplt.metrics.plot_roc(dtc_expected, predicted_probas)

plt.show()
```

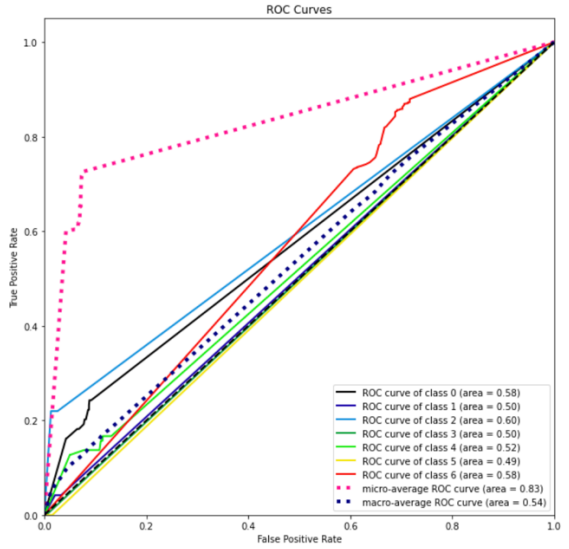


Fig.31. Python code displaying ROC curve of Decision Trees

### 6.3. Confusion matrix:

The confusion matrix was used solely to assess the models' performance. The interpretation of the confusion matrix was crucial in comparing actual and predicted values, which greatly aided us in analyzing false positives and false negatives and offered us a clear understanding of metrics like recall, sensitivity, specificity, and F1 score.

```
In [466]: import matplotlib.pyplot as plt
          import seaborn as sns
          from sklearn import metrics
          cm = metrics.confusion_matrix(y_test, logReg_predicted)
          plt.figure(figsize=(9,9))
          sns.heatmap(cm, annot=True, cmap='Blues_r');
          plt.ylabel('Actual label');
          plt.xlabel('Predicted label');
          all_sample_title = 'Accuracy Score: {0}'.format(score)
          plt.title(all_sample_title, size = 15);
```

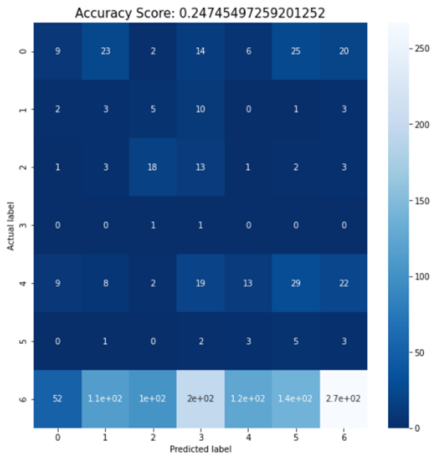


Fig.32. Python code displaying confusion matrix of Logistic Regression model

```
In [470]: import matplotlib.pyplot as plt
          import seaborn as sns
          from sklearn import metrics
          cm = metrics.confusion_matrix(y_test, randomforest_predicted)
          plt.figure(figsize=(9,9))
          sns.heatmap(cm, annot=True, cmap='Blues_r');
          plt.ylabel('Actual label');
          plt.xlabel('Predicted label');
          all_sample_title = 'Accuracy Score: {0}'.format(score)
          plt.title(all_sample_title, size = 15);
```

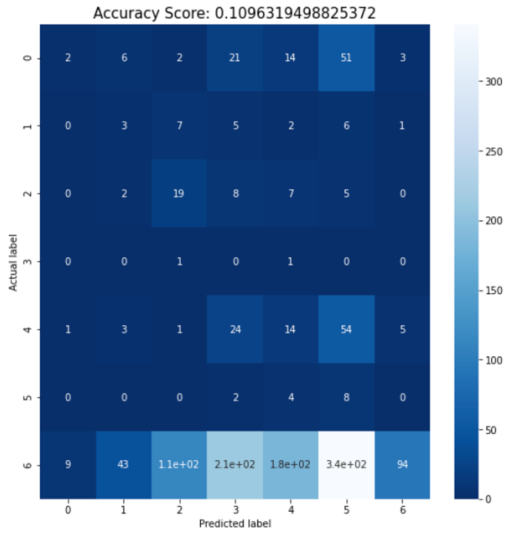


Fig.33. Python code displaying confusion matrix of random Forest Classifier

```
In [507]: import matplotlib.pyplot as plt
          import seaborn as sns
          from sklearn import metrics
          cm = metrics.confusion_matrix(y_test, svm_predicted)
          plt.figure(figsize=(9,9))
          sns.heatmap(cm, annot=True, cmap='Blues_r');
          plt.ylabel('Actual label');
          plt.xlabel('Predicted label');
          all_sample_title = 'Accuracy Score: {0}'.format(score)
          plt.title(all_sample_title, size = 15);
```

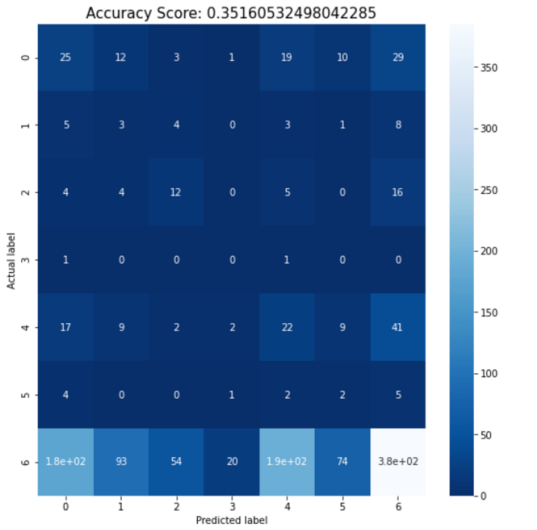


Fig.34.Python code displaying confusion matrix of Support Vector Machine

```
In [475]: import matplotlib.pyplot as plt
          import seaborn as sns
          from sklearn import metrics
          cm = metrics.confusion_matrix(y_test, dtc_predicted)
          plt.figure(figsize=(9,9))
          sns.heatmap(cm, annot=True, cmap='Blues_r');
          plt.ylabel('Actual label');
          plt.xlabel('Predicted label');
          all_sample_title = 'Accuracy Score: {0}'.format(score)
          plt.title(all_sample_title, size = 15);
```

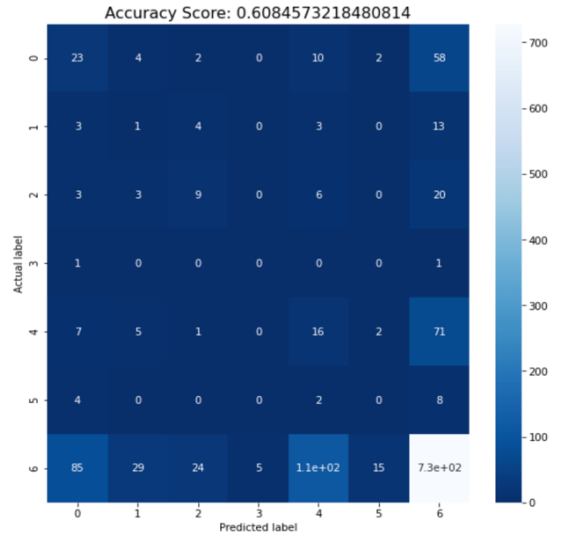


Fig.35. Python code displaying confusion matrix of Decision Trees

## 7. Summary of findings:

To answer the first research question, we have done a correlation test between Cause of death, description of injury, and death city. We did Spearman Correlation (which is a non-parametric test) as our data is categorical.

After performing the Correlation, we found that the Cause of Death is negatively correlated to Drug availability and positively correlated to the Mode of drug Intake.

Research question Two Findings:

To answer the second research question, we have classified the narcotic drugs into two classes which are opioids and non-opioids based on their harmfulness. The pharmacological class column is used in modeling to predict the description of injury. Which ultimately answers the research question.

## 8. Limitations:

Our dataset has limits, as previously indicated, and we were unable to explore the original dataset. We also planned to use clustering for the pharmacological class to forecast the description of the damage and the pharmacological class of medicine responsible for the death, but we were unable to do so. Because the Pharmacological class had only binary variables, clustering was unfeasible, but we were able to visualize the cluster centroids. We attempted to balance the data in order to execute the modeling appropriately. We used a 10-fold cross-validation method and balanced the class weights to achieve this. Nonetheless, there were differences in the classes.

## A. Appendix

### A.1 Grouping and Label Encoding of columns:

**Grouping of description of injury**

```
In [48]: df.DescriptionofInjury = df.DescriptionofInjury.str.lower()

         df['DescriptionofInjury_others'] = df.DescriptionofInjury
         counts = df.DescriptionofInjury.value_counts()
         idx = counts[counts.lt(11)].index

         df['counts'] = df['DescriptionofInjury'].map(counts)
         df.loc[df['DescriptionofInjury'].isin(idx), 'DescriptionofInjury_others'] = 'others'
         df.loc[df['DescriptionofInjury_others'].isin(['substance', 'drug use', 'drug abuse', 'used heroin', 'acute and chronic
         df.loc[df['DescriptionofInjury_others'].isin(['ingested medications', 'took medications', 'used medications']), 'Descri
         df.head()
```

Out[48]:

| | ID | Date | DateType | Age | Sex | Race | ResidenceCity | ResidenceCounty | ResidenceState | DeathCity | ... | Hydromorphone | Other | OpiateNOS | AnyOp |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 14-0273 | 6/28/14 0:00 | 1.0 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | 0 | NaN | 0 | |
| 1 | 13-0102 | 3/21/13 0:00 | 0.0 | 48.0 | Male | Black | NORWALK | NaN | NaN | NORWALK | ... | 0 | NaN | 0 | |
| 2 | 16-0165 | 3/13/16 0:00 | 0.0 | 30.0 | Female | White | SANDY HOOK | FAIRFIELD | CT | DANBURY | ... | 0 | NaN | 0 | |
| 3 | 16-0208 | 3/31/16 0:00 | 0.0 | 23.0 | Male | White | RYE | WESTCHESTER | NY | GREENWICH | ... | 0 | NaN | 0 | |
| 4 | 13-0052 | 2/13/13 0:00 | 0.0 | 22.0 | Male | Asian, Other | FLUSHING | QUEENS | NaN | GREENWICH | ... | 0 | NaN | 0 | |

5 rows × 43 columns

Fig.36.Python code to group Description of Injury column

## Label encoding of ResidenceCity

```
In [55]: df['ResidenceCity_others'] = df.ResidenceCity
         counts = df.ResidenceCity.value_counts()
         idx = counts[counts.lt(200)].index

         df['counts'] = df['ResidenceCity'].map(counts)
         df.loc[df['ResidenceCity'].isin(idx), 'ResidenceCity_others'] = 'others'

In [56]: df['ResidenceCity_others'].value_counts()

Out[56]: others         3902
         HARTFORD        296
         WATERBURY       269
         BRIDGEPORT      241
         NEW HAVEN       224
         Name: ResidenceCity_others, dtype: int64

In [58]: import seaborn as sns
         sns.countplot(df['ResidenceCity_others'])
         plt.title('Value counts of ResidenceCity')

         plt.show()

In [59]: from sklearn import preprocessing
         label_encoder = preprocessing.LabelEncoder()
         df['ResidenceCity_others']= label_encoder.fit_transform(df['ResidenceCity_others'])

         df.head()
```

Fig.37.Label Encoding of Residence City

## Label encoding of Gender column

```
In [60]: import seaborn as sns
         sns.countplot(df['Sex'])
         plt.title('Value counts of Sex')

         plt.show()

In [61]: from sklearn import preprocessing
         label_encoder = preprocessing.LabelEncoder()
         df['Sex']= label_encoder.fit_transform(df['Sex'])

         df.head()
```

Fig.38.Label Encoding of Gender

## Label encoding of Residence County

```
In [62]: df['ResidenceCounty'].value_counts()
```

```
Out[62]: HARTFORD       1205
         NEW HAVEN      1127
         FAIRFIELD       680
         NEW LONDON      406
         LITCHFIELD      262
                        ...
         MOULTRIE          1
         PLYMOUTH          1
         NEW CASTLE        1
         HIGHLANDS         1
         TIOGA             1
         Name: ResidenceCounty, Length: 84, dtype: int64
```

```
In [63]: df['ResidenceCounty_others'] = df.ResidenceCounty
         counts = df.ResidenceCounty.value_counts()
         idx = counts[counts.lt(250)].index

         df['counts'] = df['ResidenceCounty'].map(counts)
         df.loc[df['ResidenceCounty'].isin(idx), 'ResidenceCounty_others'] = 'others'
```

```
In [64]: df['ResidenceCounty_others'].value_counts()
```

```
Out[64]: HARTFORD       1205
         NEW HAVEN      1127
         FAIRFIELD       680
         others          628
         NEW LONDON      406
         LITCHFIELD      262
         Name: ResidenceCounty_others, dtype: int64
```

```
In [65]: import seaborn as sns
         sns.countplot(df['ResidenceCounty_others'])
         plt.title('Value counts of ResidenceCounty')

         plt.show()
```

```
In [66]: from sklearn import preprocessing
         label_encoder = preprocessing.LabelEncoder()
         df['ResidenceCounty_others']= label_encoder.fit_transform(df['ResidenceCounty_others'])

         df.head()
```

Fig.39.Label Encoding of Residence County

## Label encoding of Location column

```
In [49]: df['Location'].value_counts()
```

```
Out[49]: Residence            2701
         Hospital             1626
         Other                 773
         Convalescent Home       3
         Nursing Home            1
         Hospice                 1
         Name: Location, dtype: int64
```

```
In [67]: import seaborn as sns
         sns.countplot(df['Location'])
         plt.title('Value counts of Location')

         plt.show()
```

```
In [68]: from sklearn import preprocessing
         label_encoder = preprocessing.LabelEncoder()
         df['Location']= label_encoder.fit_transform(df['Location'])

         df.head()
```

Fig.40.Label Encoding of Location

## Label encoding of COD

```
In [69]: df['COD'] = df.COD
         counts = df.COD.value_counts()
         idx = counts[counts.lt(90)].index

         df['counts'] = df['COD'].map(counts)
         df.loc[df['COD'].isin(idx), 'COD'] = 'others'

In [70]: df['COD'].value_counts()

Out[70]: others                          4375
         Acute Fentanyl Intoxication      168
         Multiple Drug Toxicity           131
         Heroin Intoxication              130
         Acute Heroin Intoxication        116
         Heroin Toxicity                   95
         Acute Heroin Toxicity             90
         Name: COD, dtype: int64

In [71]: import seaborn as sns
         sns.countplot(df['COD'])
         plt.title('Value counts of COD')

         plt.show()

In [72]: from sklearn import preprocessing
         label_encoder = preprocessing.LabelEncoder()
         df['COD']= label_encoder.fit_transform(df['COD'])
```

Fig.41.Label Encoding of Cause of death

## Label encoding of DeathCity

```
In [58]: df['DeathCity'].value_counts()

Out[58]: HARTFORD          568
         NEW HAVEN         374
         WATERBURY         368
         BRIDGEPORT        341
         NEW BRITAIN       227
                          ...
         EAST HARTLAND       1
         BANTAM              1
         BRIDGEWATER         1
         W HAVEN             1
         NORFOLK             1
         Name: DeathCity, Length: 222, dtype: int64

In [59]: df['DeathCity'] = df.DeathCity
         counts = df.DeathCity.value_counts()
         idx = counts[counts.lt(100)].index

         df['counts'] = df['DeathCity'].map(counts)
         df.loc[df['DeathCity'].isin(idx), 'DeathCity'] = 'others'

In [60]: df['DeathCity'].value_counts()

Out[60]: others          2210
         HARTFORD         568
         NEW HAVEN        374
         WATERBURY        368
         BRIDGEPORT       341
         NEW BRITAIN      227
         MERIDEN          145
         BRISTOL          144
         NORWICH          144
         NEW LONDON       137
         DANBURY          131
         TORRINGTON       114
         MANCHESTER       102
         MIDDLETOWN       100
         Name: DeathCity, dtype: int64

In [61]: import seaborn as sns
         sns.countplot(df['DeathCity'])
         plt.show()

In [77]: from sklearn import preprocessing
         label_encoder = preprocessing.LabelEncoder()
         df['DeathCity']= label_encoder.fit_transform(df['DeathCity'])
         df.head()
```

Fig.42.Label Encoding of Residence City

## References:


1. https://www.cdc.gov/drugoverdose/deaths/index.html

2. Wilson, N., Kariisa, M., Seth, P., Smith, H., & Davis, N. L. (2020). Drug and Opioid-Involved Overdose Deaths — United States, 2017–2018. *MMWR. Morbidity and Mortality Weekly Report*, *69*(11), 290–297. https://doi.org/10.15585/mmwr.mm6911a4

3. Jones, C. M., Einstein, E. B., & Compton, W. M. (2018). Changes in Synthetic Opioid Involvement in Drug Overdose Deaths in the United States, 2010-2016. *JAMA*, *319*(17), 1819–1821. https://doi.org/10.1001/jama.2018.2844