CRIME DATA ANALYSIS

Hritish Chidambaram – AM.EN.U4CSE21164

In today's world, analysing the data not only helps us understand things but also helps us in predicting them. There are many examples such as weather prediction, stock market prediction, etc. But understanding situations may also help us prevent them. Let us take "Crime Data" as an example. Understanding how different factors are correlated and whether they are related at all can help us understand where we are lacking and how we can improve it to prevent the same situation from happening all over again.

All this is possible just by understanding the data. For that we use EDA which stands for Explanatory Data Analysis.

In this we first read and examine a dataset along with classifying the variables based on their type which can be either categorical or quantitative.

After categorizing the variables, we then encode the categorical variables to understand the data better.

After that, we perform various univariate and bivariate analysis to understand the inter-dependencies, if any.

Since the dataset may contain noise, missing values, outliers which can be a hinderance in understanding the data, we need to treat them.

After treating the dataset, we build many co-relations between features to derive insights from the data.

Now, let's perform all this on a dataset to understand and derive meaningful insights.

For this, we will create a dataset "crime_data" which will consist of the following features:

- Date (datetime64) – Ranging from 1st Jan, 2022 to 31st Jan, 2022
- Time (object) – Ranging from 00:00:00 to 23:23:23
- DayOfWeek (object) – Ranging from Monday to Sunday
- CrimeType (object) – Can range between burglary, assault, robbery, theft
- Severity (object) – Can range between low, medium and high
- WeaponUsed (object) – Can range between none, firearm, knife and other
- VictimAge (float64) – Ranging from 18 to 70
- VictimGender (object) – Ranging between Male and Female

- SuspectAge (float64) – Ranging from 18 to 70
- SuspectGender (object) – Ranging between Male and Female
- Location (object) – Ranging between Urban, Suburban, and Rural
- Temperature (float64) – Can range between 20F and 100F
- PopulationDensity (float64) – Can range between 50 and 1000
- PolicePatrolFrequency (object) - Can range between low, medium, high
- ResponseTime (float64) – Can range between 5 to 30 minutes
- CrimeRate (float64) – Ranges between 0 and 1

```
import pandas as pd
import numpy as np
from datetime import datetime, timedelta
import random

# Set random seed for reproducibility
np.random.seed(42)
random.seed(42)

# Generate synthetic data
num_records = 1000

# Generate random dates within a specific range
start_date = datetime(2022, 1, 1)
end_date = datetime(2022, 12, 31)
date_list = [start_date + timedelta(days=np.random.randint((end_date - start_date).days)) for _ in range(num_records)]

crime_data = pd.DataFrame({
    'Date': date_list,
    'Time': [datetime.strftime(datetime.strptime(str(random.randint(0, 23)), "%H"), "%I:%M %p") for _ in range(num_records)],
    'DayOfWeek': [date.strftime('%A') for date in date_list],
    'CrimeType': np.random.choice(['Burglary', 'Assault', 'Robbery', 'Theft'], size=num_records),
    'Severity': np.random.choice(['Low', 'Medium', 'High'], size=num_records),
    'WeaponUsed': np.random.choice(['None', 'Firearm', 'Knife', 'Other'], size=num_records),
    'VictimAge': np.random.randint(18, 70, size=num_records),
    'VictimGender': np.random.choice(['Male', 'Female'], size=num_records),
    'SuspectAge': np.random.randint(18, 70, size=num_records),
    'SuspectGender': np.random.choice(['Male', 'Female'], size=num_records),
    'Location': np.random.choice(['Urban', 'Suburban', 'Rural'], size=num_records),
    'Temperature': np.random.uniform(20, 100, size=num_records),
    'PopulationDensity': np.random.uniform(50, 1000, size= num_records),
    'PolicePatrolFrequency': np.random.choice(['Low', 'Medium', 'High'], size=num_records),
    'ResponseTime': np.random.uniform(5, 30, size=num_records),
    'CrimeRate': np.random.uniform(0, 1, size=num_records)  # This is a synthetic crime rate for illustration
})

# Display the generated dataset
print(crime_data.head())
```

The generated dataset should look like this:

```
        Date     Time  DayOfWeek CrimeType Severity WeaponUsed  VictimAge  \
0 2022-04-13  08:00 PM  Wednesday     Theft     High      Other         47
1 2022-12-15  03:00 AM   Thursday   Assault     High      Knife         61
2 2022-09-28  12:00 AM  Wednesday  Burglary   Medium      Knife         51
3 2022-04-17  11:00 PM     Sunday     Theft     High    Firearm         49
4 2022-03-13  08:00 AM     Sunday   Assault   Medium      Knife         20

  VictimGender  SuspectAge SuspectGender  Location  Temperature  \
0         Male          33          Male     Urban    27.735565
1       Female          62          Male  Suburban    77.636539
2         Male          64        Female     Urban    42.546000
3       Female          65        Female     Rural    34.744112
4       Female          44        Female     Urban    23.148950

   PopulationDensity PolicePatrolFrequency  ResponseTime  CrimeRate
0         968.261232                  High     21.587552   0.556881
1         479.604345                   Low     13.643256   0.278049
2         185.646355                   Low      9.284958   0.398566
3         419.962946                Medium      6.974515   0.170130
4         954.117286                   Low     22.332437   0.761887
```

In this scenario, we generate a synthetic dataset, but a real dataset may also contain some missing values. To understand the concept of missing values in a dataset, we will create some missing values.

```
import numpy as np
column_names = ['VictimAge', 'SuspectAge', 'Temperature', 'PopulationDensity', 'ResponseTime', 'CrimeRate']
for column in column_names:
    null_indices = np.random.choice(crime_data.index, size=120, replace=False)
    crime_data.loc[null_indices, column] = np.nan
```

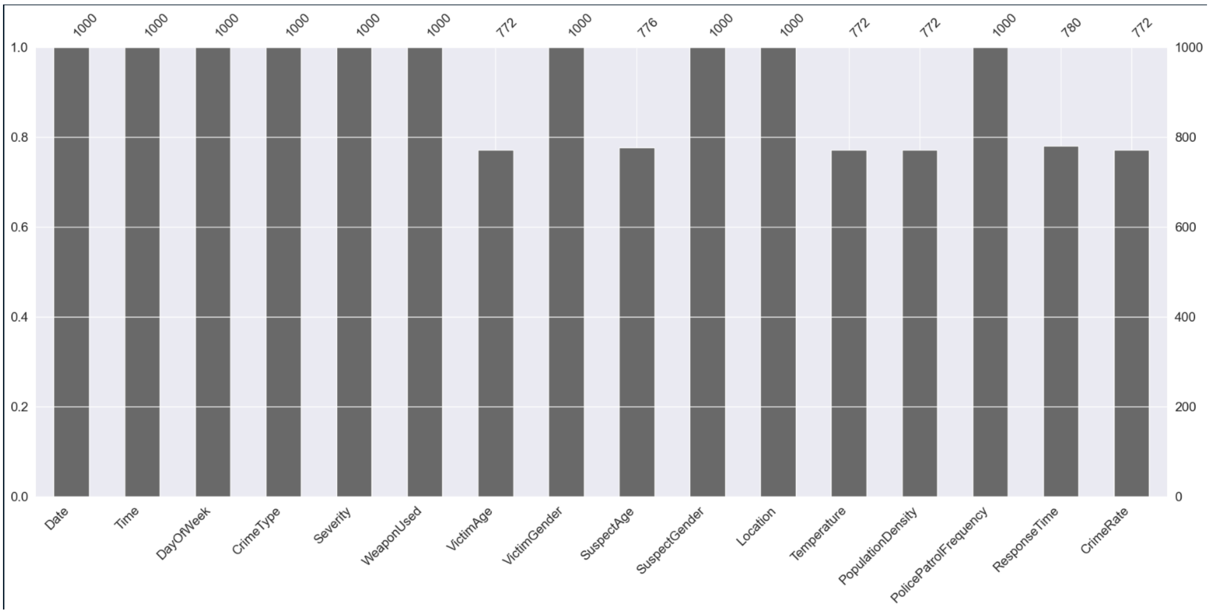
In this case we only have missing values in the categories consisting numerical values. In case we had missing values in object type features as well we will encode them in numeric format to use imputation methods to fill the missing values.

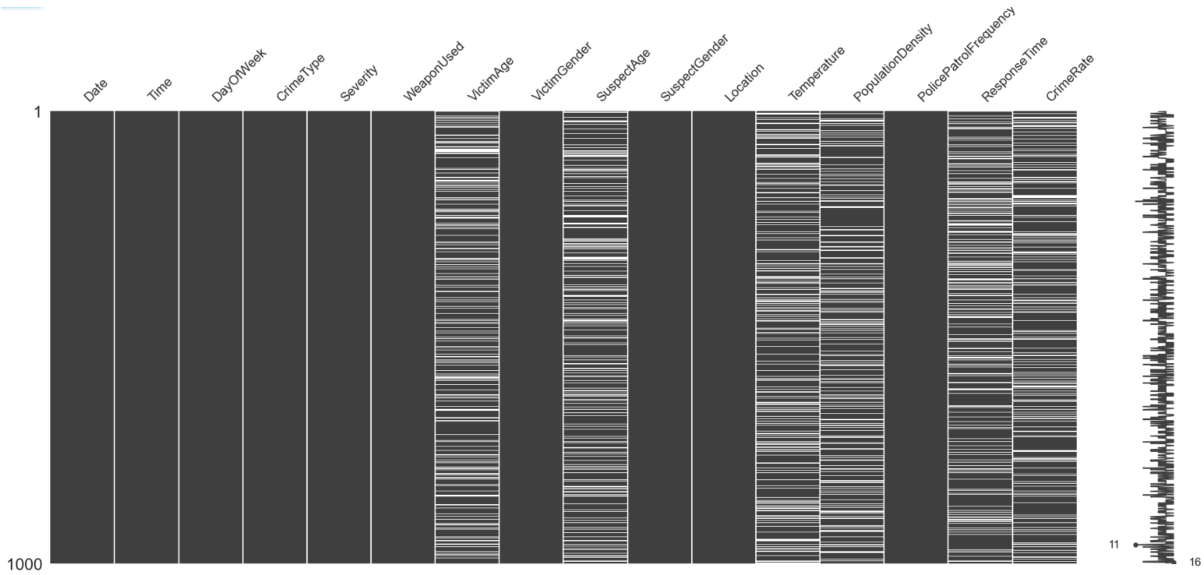
After creating missing values, let's check how many missing values do we have to deal with:

```
#Null values in each column
null_counts = crime_data.isnull().sum()
print("\nNull Value Counts:")
print(null_counts)

✓ 0.0s

Null Value Counts:
Date                      0
Time                      0
DayOfWeek                 0
CrimeType                 0
Severity                  0
WeaponUsed                0
VictimAge               399
VictimGender              0
SuspectAge              399
SuspectGender             0
Location                  0
Temperature             401
PopulationDensity       401
PolicePatrolFrequency     0
ResponseTime            399
CrimeRate               399
dtype: int64
```

We can also visualize the missing data by using MSNO bar and MSNO matrix.

```
import missingno as msno
msno.bar(crime_data)
plt.show()

msno.matrix(crime_data)
plt.show()
```





Also, to understand the size of the dataset we use:

```
#Details
print("Shape of crime_data:", crime_data.shape)
```

```
Shape of crime_data: (1000, 16)
```

After understanding the dataset that we are dealing with, it is time to optimize the dataset by first filling the missing values. For this we will use 3 methods, mean imputation, median imputation and KNN imputation.

Mean imputation:

First, we will make a copy of the original dataset and name it as df1.

```python
import pandas as pd
import numpy as np
df1 = crime_data.copy()
# Performing mean imputation on numerical columns
numeric_cols = df1.select_dtypes(include=np.number).columns
df1[numeric_cols] = df1[numeric_cols].fillna(df1[numeric_cols].mean())

print("\nDataFrame after mean imputation:")
print(df1)
```

✓ 0.0s

```
DataFrame after mean imputation:
           Date      Time DayOfWeek CrimeType Severity WeaponUsed  VictimAge  \
0    2022-04-13  08:00 PM Wednesday     Theft     High      Other  43.951747
1    2022-12-15  03:00 AM  Thursday   Assault     High      Knife  61.000000
2    2022-09-28  12:00 AM Wednesday  Burglary   Medium      Knife  51.000000
3    2022-04-17  11:00 PM    Sunday     Theft     High    Firearm  49.000000
4    2022-03-13  08:00 AM    Sunday   Assault   Medium      Knife  20.000000
..          ...       ...       ...       ...      ...        ...        ...
995  2022-06-10  06:00 PM    Friday  Burglary   Medium       None  36.000000
996  2022-06-27  09:00 AM    Monday   Assault     High      Knife  29.000000
997  2022-01-10  02:00 AM    Monday   Robbery      Low       None  33.000000
998  2022-09-18  07:00 AM    Sunday   Robbery      Low       None  60.000000
999  2022-05-18  03:00 AM Wednesday     Theft      Low    Firearm  43.951747

    VictimGender  SuspectAge SuspectGender  Location  Temperature  \
0           Male   44.202995          Male     Urban    27.735565
1         Female   44.202995          Male  Suburban    59.181331
2           Male   64.000000        Female     Urban    42.546000
3         Female   44.202995        Female     Rural    34.744112
4         Female   44.000000        Female     Urban    59.181331
..           ...         ...           ...       ...          ...
995       Female   44.202995        Female  Suburban    59.181331
996       Female   44.202995        Female  Suburban    36.600920
997       Female   53.000000        Female     Rural    65.505019
...
998         567.637602                Medium     25.634449   0.134798
999         319.331831                  High     29.677059   0.913670
```

Median Imputation:

Second, we will use median imputation by copying the original dataset and naming it as df2.

```
import pandas as pd
import numpy as np
df2 = crime_data.copy()
# Performing median imputation on numerical columns
numeric_cols = df2.select_dtypes(include=np.number).columns
df2[numeric_cols] = df2[numeric_cols].fillna(df2[numeric_cols].median())

print("\nDataFrame after median imputation:")
print(df2)
```

✓ 0.0s

```
DataFrame after median imputation:
           Date      Time  DayOfWeek CrimeType Severity WeaponUsed  VictimAge  \
0    2022-04-13  08:00 PM  Wednesday     Theft     High      Other       44.0
1    2022-12-15  03:00 AM   Thursday   Assault     High      Knife       61.0
2    2022-09-28  12:00 AM  Wednesday  Burglary   Medium      Knife       51.0
3    2022-04-17  11:00 PM     Sunday     Theft     High    Firearm       49.0
4    2022-03-13  08:00 AM     Sunday   Assault   Medium      Knife       20.0
..          ...       ...        ...       ...      ...        ...        ...
995  2022-06-10  06:00 PM     Friday  Burglary   Medium       None       36.0
996  2022-06-27  09:00 AM     Monday   Assault     High      Knife       29.0
997  2022-01-10  02:00 AM     Monday   Robbery      Low       None       33.0
998  2022-09-18  07:00 AM     Sunday   Robbery      Low       None       60.0
999  2022-05-18  03:00 AM  Wednesday     Theft      Low    Firearm       44.0

    VictimGender  SuspectAge SuspectGender  Location  Temperature  \
0           Male        44.0          Male     Urban    27.735565
1         Female        44.0          Male  Suburban    59.438617
2           Male        64.0        Female     Urban    42.546000
3         Female        44.0        Female     Rural    34.744112
4         Female        44.0        Female     Urban    59.438617
..           ...         ...           ...       ...          ...
995       Female        44.0        Female  Suburban    59.438617
996       Female        44.0        Female  Suburban    36.600920
997       Female        53.0        Female     Rural    65.505019
...
998         567.637602               Medium     25.634449    0.134798
999         319.331831                 High     29.677059    0.913670
```

KNN Imputation:

Lastly, we will use KNN imputation in a copied dataset named df3. In this we will also see how to decode a dataset to apply imputation methods on object type variables.

```python
from sklearn.impute import KNNImputer
from sklearn.preprocessing import LabelEncoder
import pandas as pd

df3 = crime_data.copy()
categorical_attributes = df3.select_dtypes(include=['object']).columns.tolist()
numeric_attributes = df3.select_dtypes(include=['number']).columns.tolist()

label_encoder = LabelEncoder()
for col in categorical_attributes:
    df3[col] = label_encoder.fit_transform(df3[col])

imputer = KNNImputer(n_neighbors=15)
data_imputed = imputer.fit_transform(df3[numeric_attributes])

data_imputed_df = pd.DataFrame(data_imputed, columns=numeric_attributes)

df_no_missing = pd.concat([df3[categorical_attributes], data_imputed_df], axis=1)

df3 = df_no_missing
df3
```

	Time	DayOfWeek	CrimeType	Severity	WeaponUsed	VictimGender	SuspectGender	Location	PolicePatrolFrequency	VictimAge	SuspectAge	Temperature	PopulationDensity	ResponseTime	CrimeRate
0	15	6	3	0	3	1	1	2	0	44.933333	48.266667	27.735565	441.528910	13.176582	0.556881
1	4	4	0	0	1	0	1	1	1	61.000000	50.266667	58.757976	479.604345	13.643256	0.454457
2	22	6	1	2	1	1	0	2	1	51.000000	64.000000	42.546000	482.035384	9.284958	0.548588
3	21	3	3	0	0	0	0	0	2	49.000000	43.133333	34.744112	419.962946	6.974515	0.170130
4	14	3	0	2	1	0	0	2	1	20.000000	44.000000	59.402461	954.117286	22.332437	0.761887
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
995	11	0	1	2	2	0	0	1	1	36.000000	48.933333	57.200268	641.421926	17.054584	0.522553
996	16	1	0	0	1	0	0	1	2	29.000000	50.133333	36.600920	478.284795	11.653960	0.475971
997	2	1	2	1	2	0	0	0	2	33.000000	53.000000	65.505019	504.021763	10.378015	0.632841
998	12	3	2	1	2	1	1	0	2	60.000000	42.000000	63.235081	567.637602	25.634449	0.134798
999	4	6	3	1	0	1	0	2	0	44.866667	49.000000	55.200781	319.331831	29.677059	0.913670

1000 rows × 15 columns

Now let us check if the imputed dataset contain any missing value.

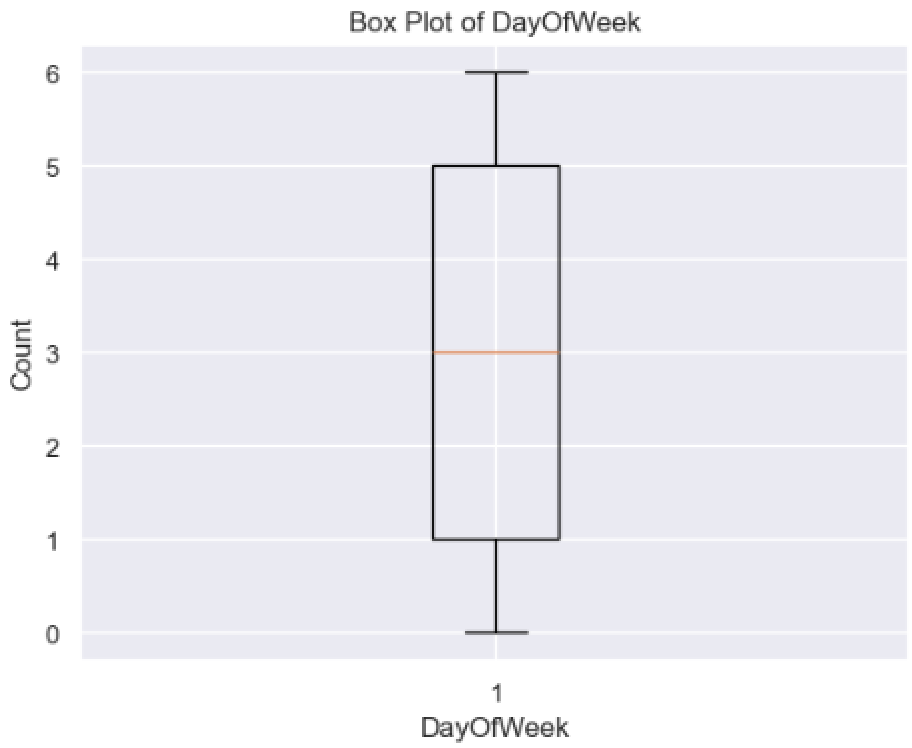
```python
#Null values in each column
null_counts = df3.isnull().sum()
print("\nNull Value Counts:")
print(null_counts)
```

✓ 0.0s

```
Null Value Counts:
Time                     0
DayOfWeek                0
CrimeType                0
Severity                 0
WeaponUsed               0
VictimGender             0
SuspectGender            0
Location                 0
PolicePatrolFrequency    0
VictimAge                0
SuspectAge               0
Temperature              0
PopulationDensity        0
ResponseTime             0
CrimeRate                0
dtype: int64
```

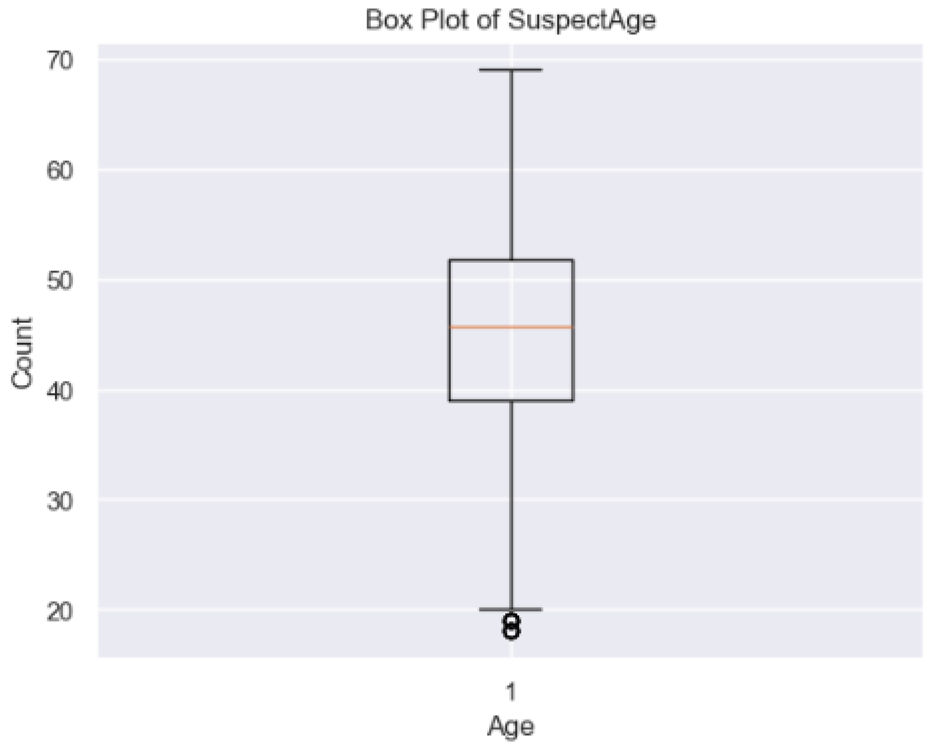
Now that we don't have any missing values, we can check for outliers, if any. For this we will use the box plot method.

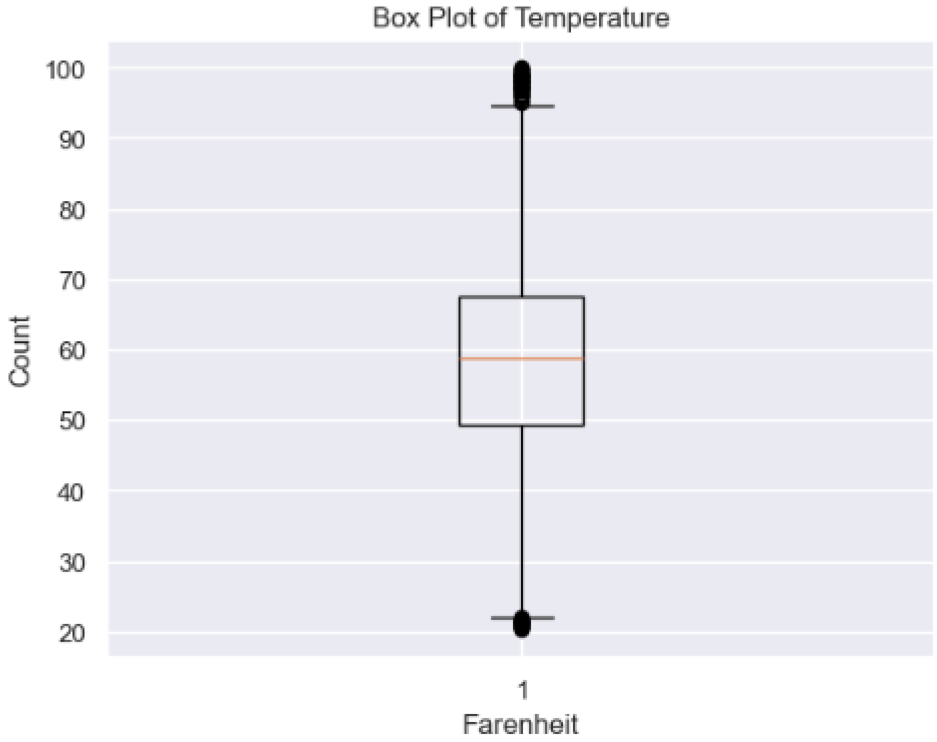
```
import matplotlib.pyplot as plt

plt.boxplot(df3["DayOfWeek"])
plt.xlabel("DayOfWeek")
plt.ylabel("Count")
plt.title("Box Plot of DayOfWeek")
plt.show()

plt.boxplot(df3["VictimAge"])
plt.xlabel("Age")
plt.ylabel("Count")
plt.title("Box Plot of VictimAge")
plt.show()

plt.boxplot(df3["SuspectAge"])
plt.xlabel("Age")
plt.ylabel("Count")
plt.title("Box Plot of SuspectAge")
plt.show()

plt.boxplot(df3["Temperature"])
plt.xlabel("Farenheit")
plt.ylabel("Count")
plt.title("Box Plot of SuspectAge")
plt.show()

plt.boxplot(df3["PopulationDensity"])
plt.xlabel("Population Density")
plt.ylabel("Count")
plt.title("Box Plot of Population Density")
plt.show()

plt.boxplot(df3["CrimeRate"])
plt.xlabel("Crime Rate")
plt.ylabel("Count")
plt.title("Box Plot of Crime Rate")
plt.show()

plt.boxplot(df3["ResponseTime"])
plt.xlabel("Response Time")
plt.ylabel("Count")
plt.title("Box Plot of Response Time")
plt.show()
```

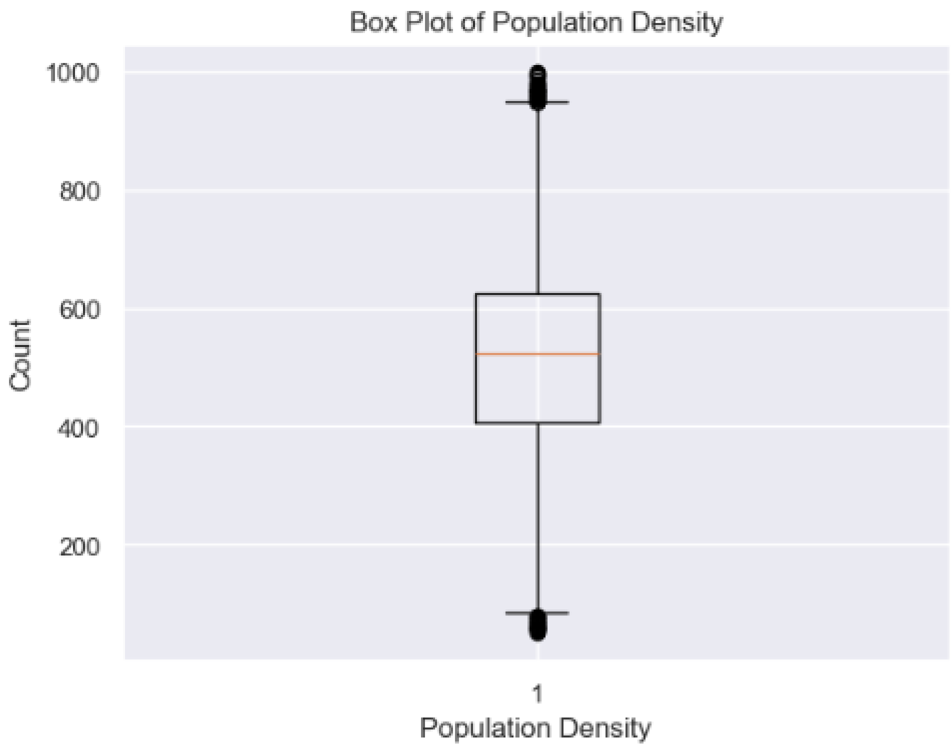


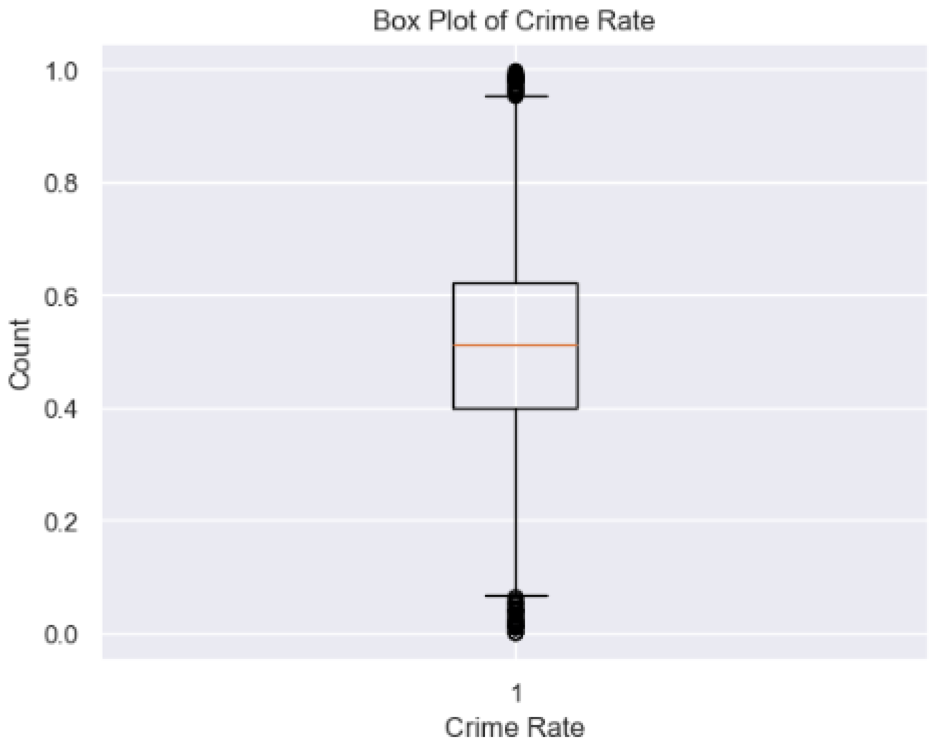
Box Plot of DayOfWeek
6
5
4
3
2
1
0
Count
1
DayOfWeek

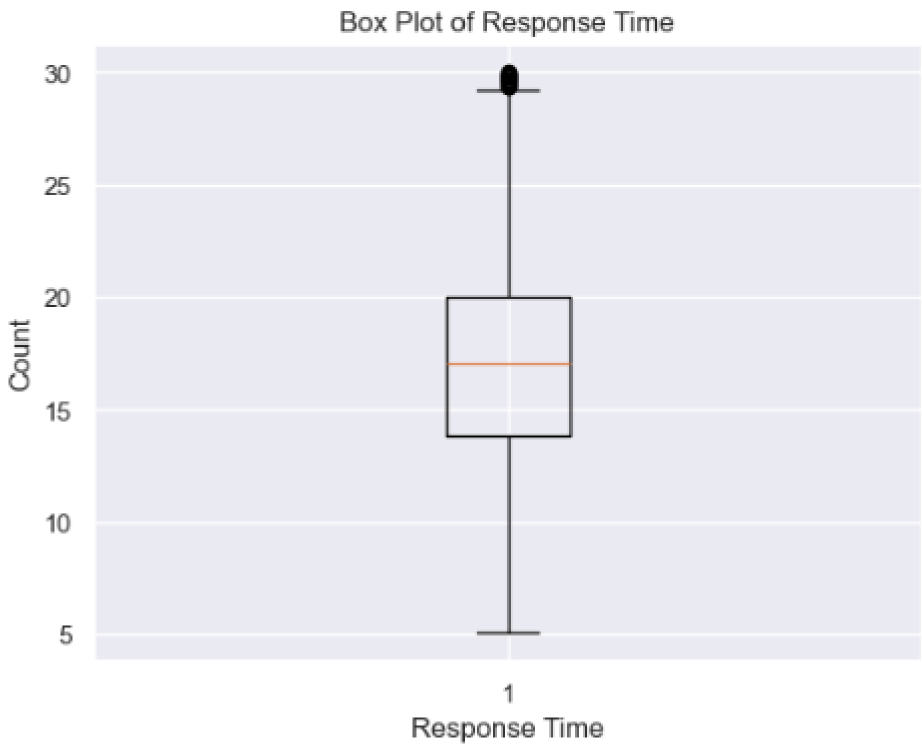
Box Plot of VictimAge
70
60
50
40
30
20
Count
1
Age


Box Plot of SuspectAge
Count
70
60
50
40
30
20
1
Age


Box Plot of Temperature
Count
100
90
80
70
60
50
40
30
20
1
Farenheit


Box Plot of Population Density
1000
800
600
400
200
Count
1
Population Density


Box Plot of Crime Rate
1.0
0.8
0.6
0.4
0.2
0.0
Count
1
Crime Rate

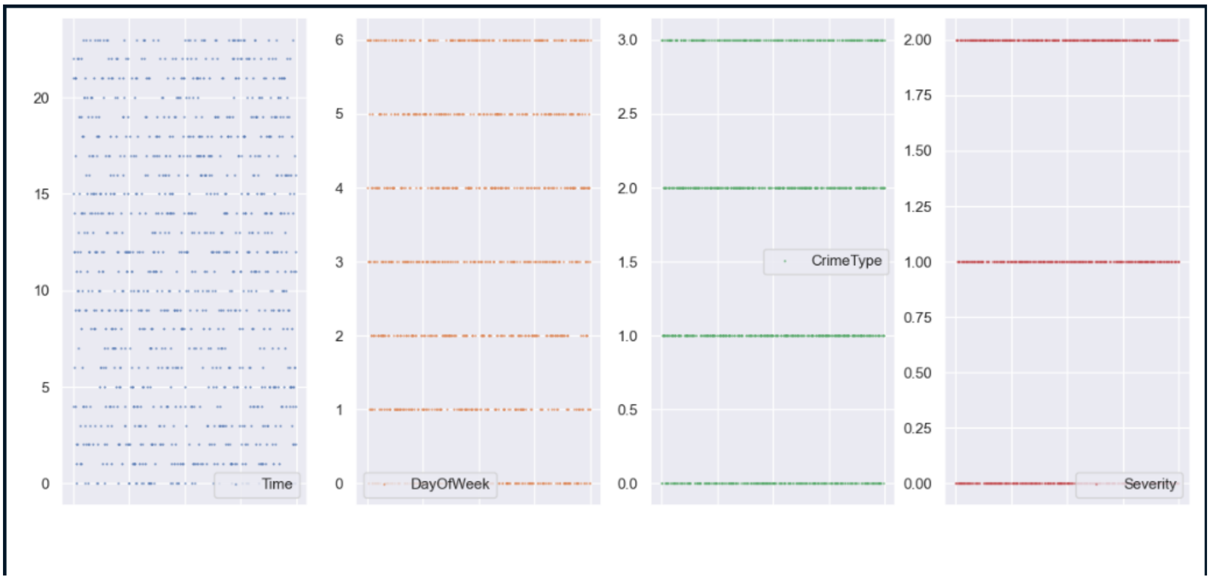


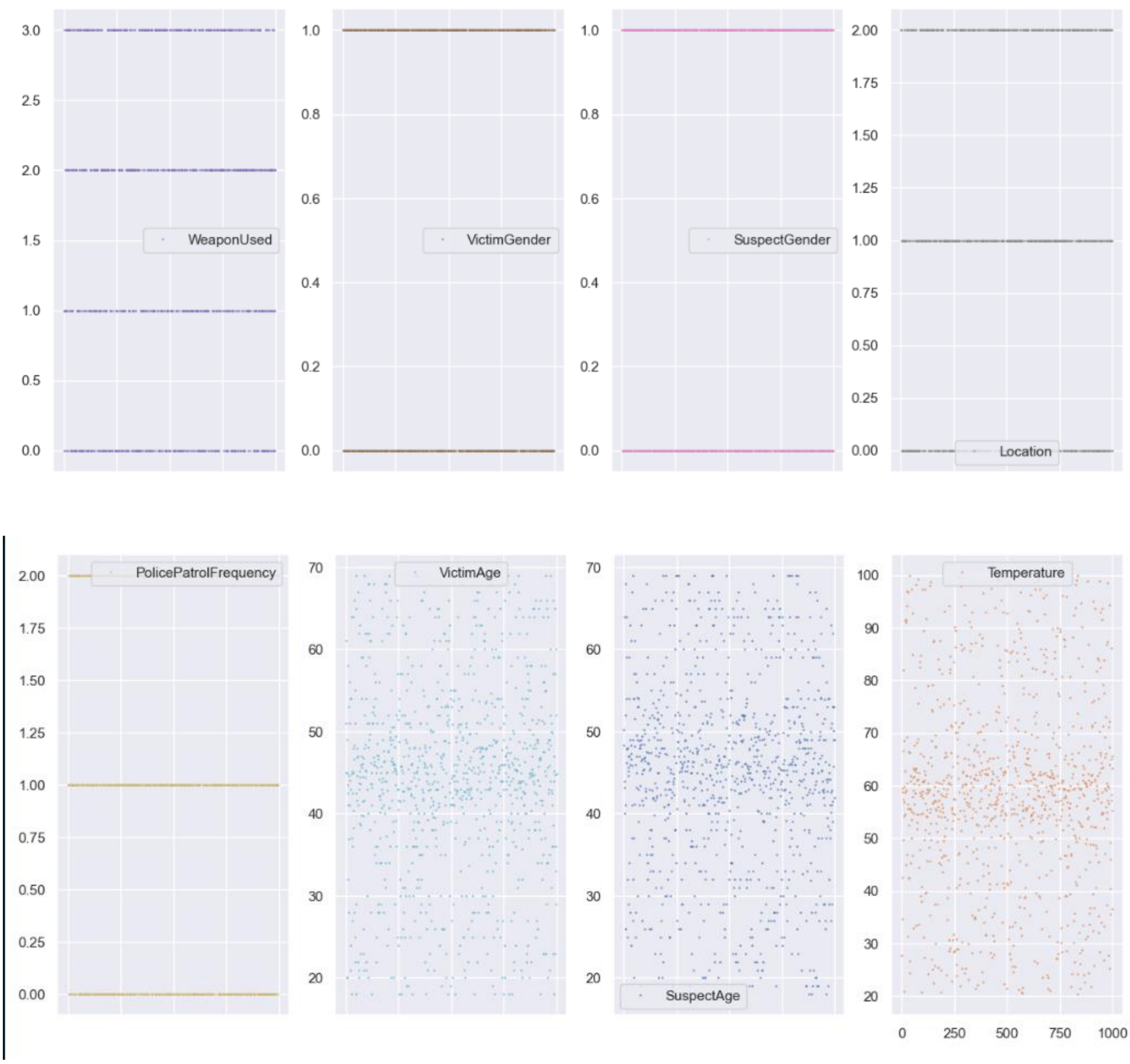
Through the box plot method, we can identify the features which have outliers in them. Once we find that, the next task is to remove those outliers. For this, we will use Z-Score method.

```
from scipy import stats
import numpy as np
z = np.abs(stats.zscore(df3))
print(z)

threshold = 3
print(np.where(z > 3))

df3 = df3[(z < 3).all(axis=1)]
df3.shape

df3.plot(lw=0, marker=".", subplots=True, layout=(-1, 4),
            figsize=(15, 30), markersize=1);
```

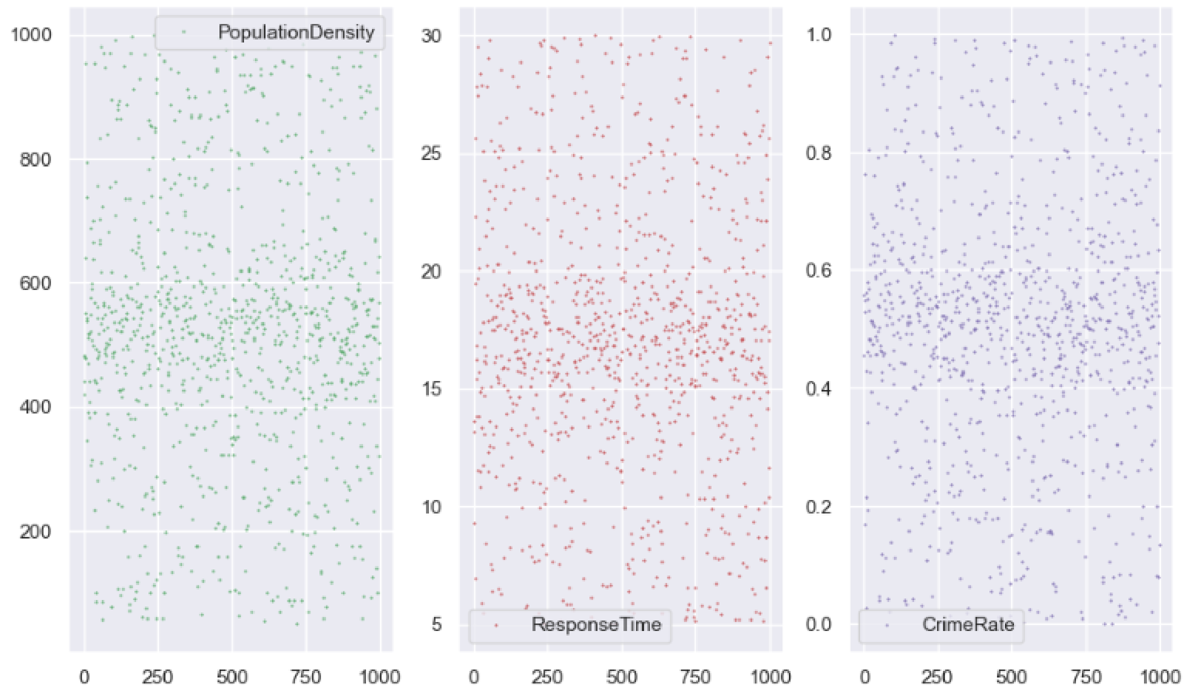
```
         Time  DayOfWeek  CrimeType  Severity  WeaponUsed  VictimGender  \
0    0.520489   1.462588   1.400805  1.288251    1.309555      0.988071
1    1.085057   0.471675   1.336924  1.288251    0.475796      1.012073
2    1.542200   1.462588   0.424348  1.184400    0.475796      0.988071
3    1.396241   0.023782   1.400805  1.288251    1.368471      1.012073
4    0.374530   0.023782   1.336924  1.184400    0.475796      1.012073
..        ...        ...        ...       ...         ...           ...
995  0.063346   1.510152   0.424348  1.184400    0.416879      1.012073
996  0.666448   1.014695   1.336924  1.288251    0.475796      1.012073
997  1.376975   1.014695   0.488228  0.051926    0.416879      1.012073
998  0.082613   0.023782   0.488228  0.051926    0.416879      0.988071
999  1.085057   1.462588   1.400805  0.051926    1.368471      0.988071

     SuspectGender  Location  PolicePatrolFrequency  VictimAge  SuspectAge  \
0         0.955011  1.233859               1.212105   0.042896    0.275007
1         0.955011  0.021338               0.055787   1.391553    0.443613
2         1.047108  1.233859               0.055787   0.552140    1.601374
3         1.047108  1.276536               1.323680   0.384258    0.157748
4         1.047108  1.233859               0.055787   2.050039    0.084686
..             ...       ...                    ...        ...         ...
995       1.047108  0.021338               0.055787   0.706979    0.331209
996       1.047108  0.021338               1.323680   1.294568    0.432373
997       1.047108  1.276536               1.323680   0.958803    0.674041
998       0.955011  1.276536               1.323680   1.307612    0.253292
999       1.047108  1.233859               1.212105   0.037300    0.336829
...
999     0.199769           0.923930      2.178690    1.825107
```

Once we remove the outliers let us realize the data





WeaponUsed
VictimGender
SuspectGender
Location
PolicePatrolFrequency
VictimAge
SuspectAge
Temperature



The more scattered they are, the more difficult it becomes to identify a pattern in them.

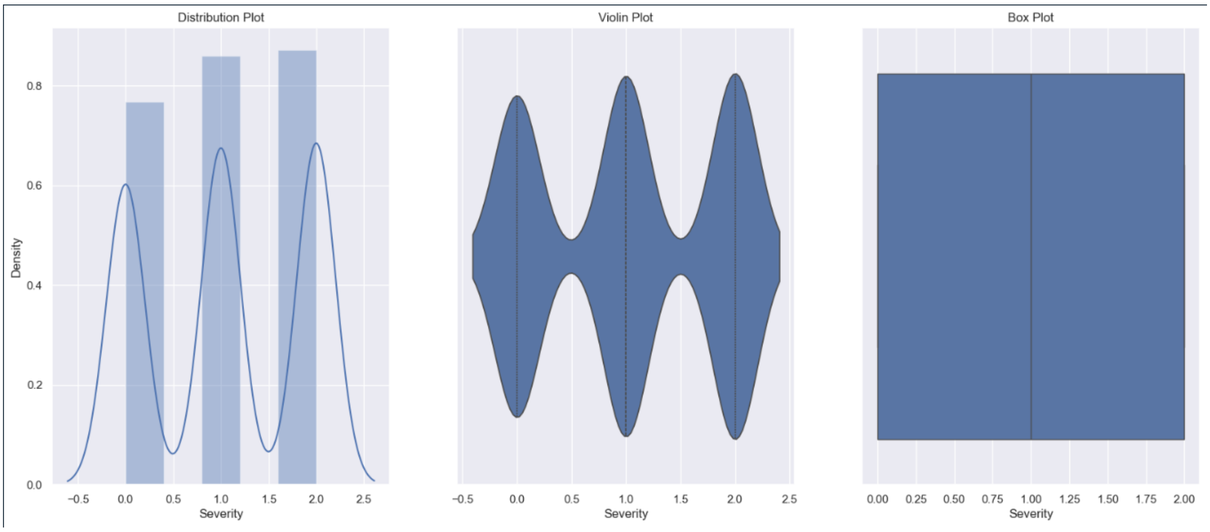
You can use many plotting techniques such as distribution plot, violin plot, etc. The technique that you use depends upon the features that you are dealing with.

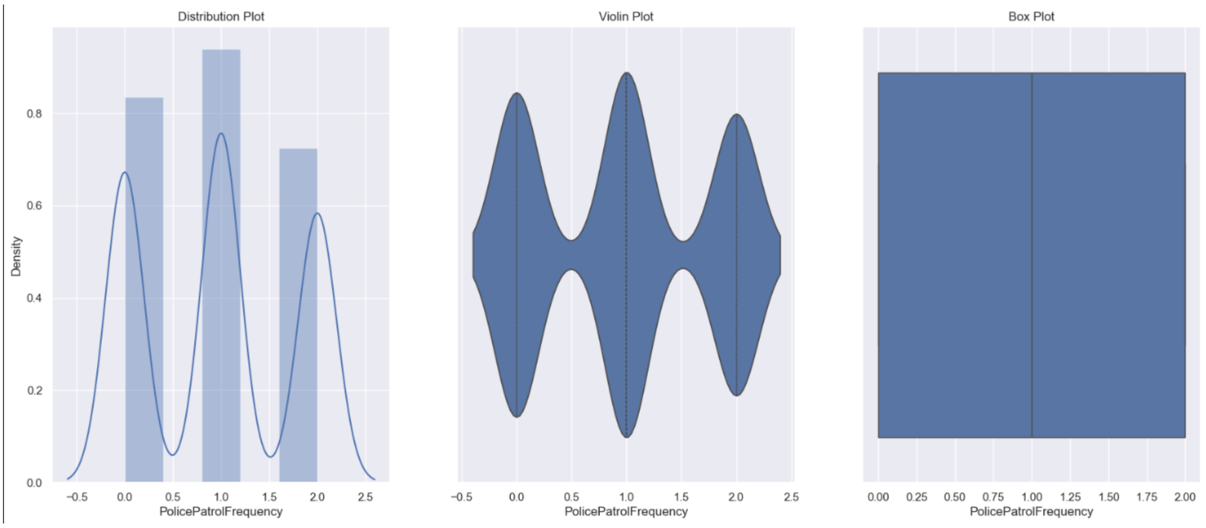
You can do univariate and bivariate analysis as well on the data.

```
def univariate(df,col,vartype,hue =None):
    '''
    Univariate function will plot parameter values in graphs.
    df      : dataframe name
    col     : Column name
    vartype : variable type : continuous or categorical
                Continuous(0)   : Distribution, Violin & Boxplot will be plotted.
                Categorical(1) : Countplot will be plotted.
    hue     : Only applicable in categorical analysis.
    '''
    sns.set(style="darkgrid")
    if vartype == 0:
        fig, ax=plt.subplots(nrows =1,ncols=3,figsize=(20,8))
        ax[0].set_title("Distribution Plot")
        sns.distplot(df[col],ax=ax[0])
        ax[1].set_title("Violin Plot")
        sns.violinplot(data =df, x=col,ax=ax[1], inner="quartile")
        ax[2].set_title("Box Plot")
        sns.boxplot(data =df, x=col,ax=ax[2],orient='v')
    if vartype == 1:
        temp = pd.Series(data = hue)
        fig, ax = plt.subplots()
        width = len(df[col].unique()) + 6 + 4*len(temp.unique())
        fig.set_size_inches(width , 7)
        ax = sns.countplot(data = df, x= col, order=df[col].value_counts().index,hue = hue)
        if len(temp.unique()) > 0:
            for p in ax.patches:
                ax.annotate('{:1.1f}%'.format((p.get_height()*100)/float(len(loan))), (p.get_x()+0.05, p.get_height()+20))
        else:
            for p in ax.patches:
                ax.annotate(p.get_height(), (p.get_x()+0.32, p.get_height()+20))
        del temp
    else:
        exit
```

```python
univariate(df=df3,col='Severity',vartype=0)
univariate(df=df3,col='PolicePatrolFrequency',vartype=0)
```

✓ 0.7s





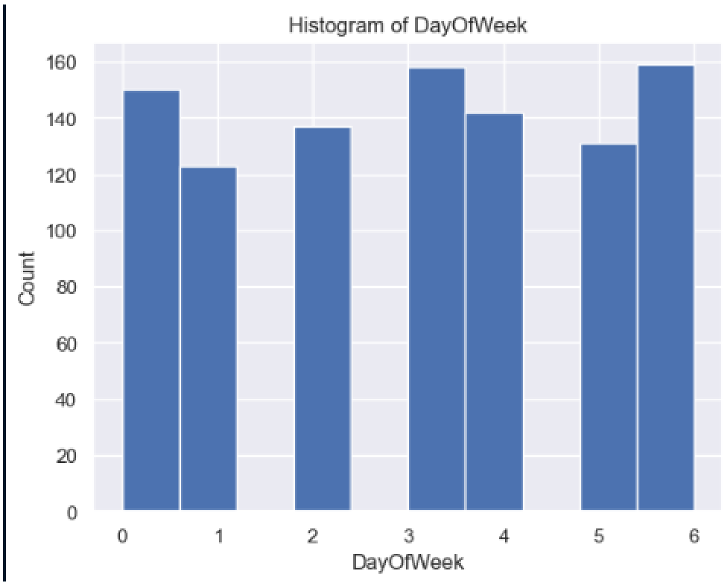
After removing the outliers, we need to remove the noise in the data. For this, we use binning. Lets use binning by mean as an example.

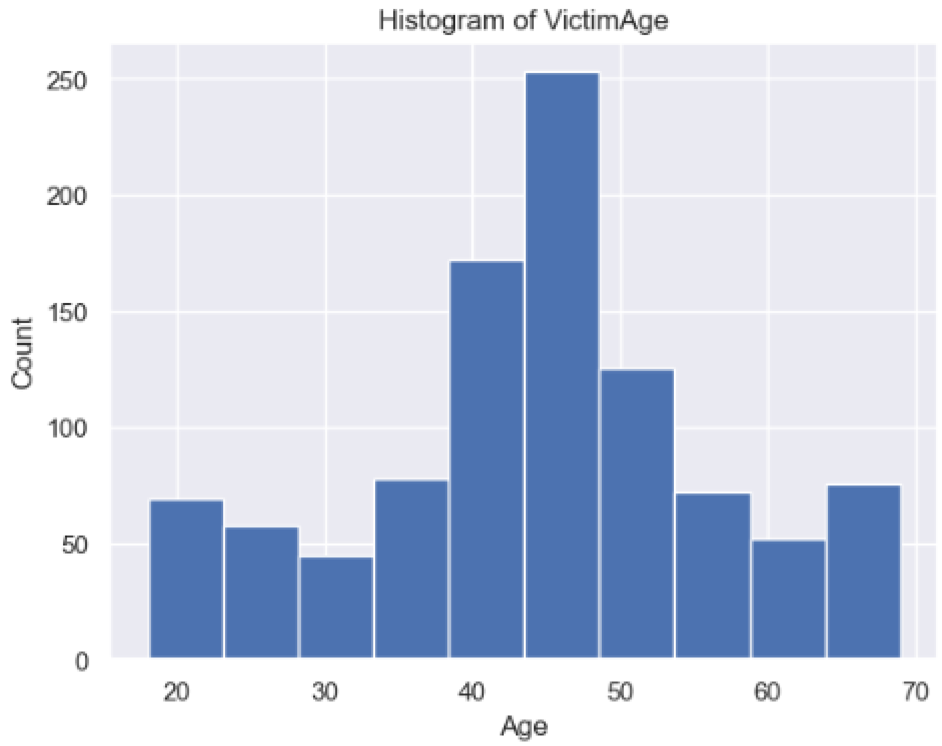
```
# Calculate the mean value for each bin
age_mean = df1.groupby('VictimAge')['VictimAge'].mean()
temperature_mean = df1.groupby('Temperature')['Temperature'].mean()
population_density_mean = df1.groupby('PopulationDensity')['PopulationDensity'].mean()
suspect_age_mean = df1.groupby('SuspectAge')['SuspectAge'].mean()
response_time_mean = df1.groupby('ResponseTime')['ResponseTime'].mean()

# Replace the bin labels with the mean values
df1['VictimAge'] = df1['VictimAge'].map(age_mean)
df1['Temperature'] = df1['Temperature'].map(temperature_mean)
df1['PopulationDensity'] = df1['PopulationDensity'].map(population_density_mean)
df1['SuspectAge'] = df1['SuspectAge'].map(suspect_age_mean)
df1['ResponseTime'] = df1['ResponseTime'].map(response_time_mean)

# Display the modified dataframe
print(df1)
```

```
          Date      Time  DayOfWeek CrimeType Severity WeaponUsed  VictimAge  \
0   2022-04-13  08:00 PM  Wednesday     Theft     High      Other  44.209845
1   2022-12-15  03:00 AM   Thursday   Assault     High      Knife  61.000000
2   2022-09-28  12:00 AM  Wednesday  Burglary   Medium      Knife  51.000000
3   2022-04-17  11:00 PM     Sunday     Theft     High    Firearm  49.000000
4   2022-03-13  08:00 AM     Sunday   Assault   Medium      Knife  20.000000
..         ...       ...        ...       ...      ...        ...        ...
995 2022-06-10  06:00 PM     Friday  Burglary   Medium       None  36.000000
996 2022-06-27  09:00 AM     Monday   Assault     High      Knife  29.000000
997 2022-01-10  02:00 AM     Monday   Robbery      Low       None  33.000000
998 2022-09-18  07:00 AM     Sunday   Robbery      Low       None  60.000000
999 2022-05-18  03:00 AM  Wednesday     Theft      Low    Firearm  27.000000

    VictimGender  SuspectAge SuspectGender  Location  Temperature  \
0           Male   33.000000          Male     Urban    27.735565
1         Female   43.976804          Male  Suburban    77.636539
2           Male   64.000000        Female     Urban    42.546000
3         Female   43.976804        Female     Rural    34.744112
4         Female   44.000000        Female     Urban    59.779762
..           ...         ...           ...       ...          ...
995       Female   43.976804        Female  Suburban    59.779762
996       Female   33.000000        Female  Suburban    36.600920
997       Female   53.000000        Female     Rural    65.505019
998         Male   42.000000          Male     Rural    59.779762
999         Male   49.000000        Female     Urban    59.779762
...
998         567.637602             Medium     25.634449    0.134798
999         319.331831               High     29.677059    0.913670
```

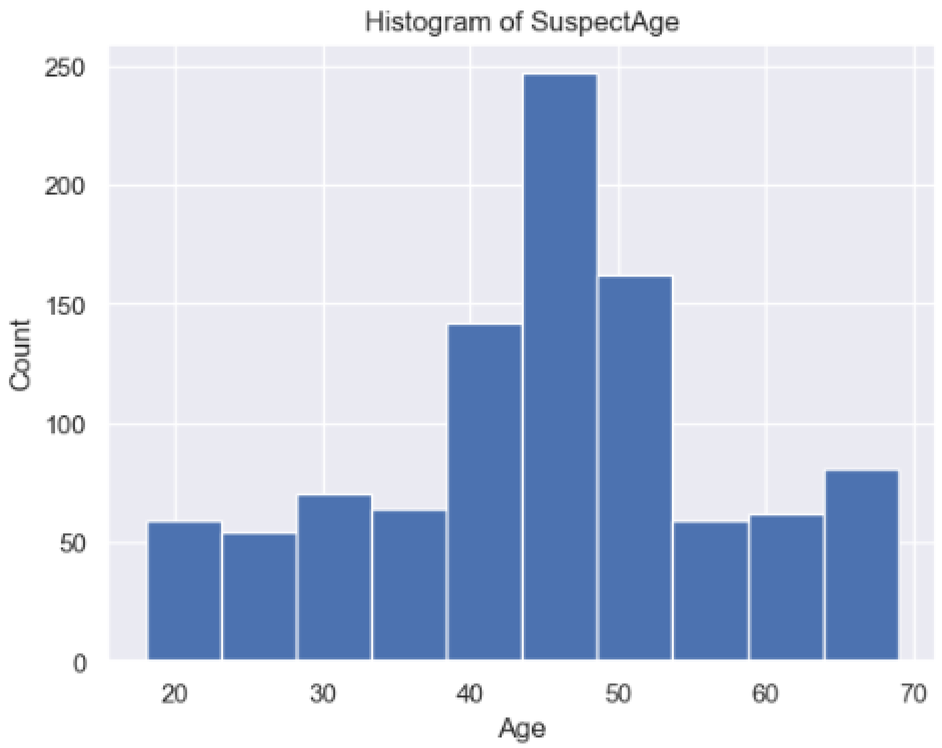
Now that we have removed the noise, let’s do a histography realisation of the features to understand on what values the crime is peaking.

```
plt.hist(df3["DayOfWeek"])
plt.xlabel("DayOfWeek")
plt.ylabel("Count")
plt.title("Histogram of DayOfWeek")
plt.show()

plt.hist(df3["VictimAge"])
plt.xlabel("Age")
plt.ylabel("Count")
plt.title("Histogram of VictimAge")
plt.show()

plt.hist(df3["SuspectAge"])
plt.xlabel("Age")
plt.ylabel("Count")
plt.title("Histogram of SuspectAge")
plt.show()

plt.hist(df3["Temperature"])
plt.xlabel("Farenheit")
plt.ylabel("Degrees")
plt.title("Histogram of Temperature")
plt.show()

plt.hist(df3["PopulationDensity"])
plt.xlabel("Population Density")
plt.ylabel("Count")
plt.title("Histogram of Population Density")
plt.show()

plt.hist(df3["ResponseTime"])
plt.xlabel("Response Time")
plt.ylabel("Count")
plt.title("Histogram of Response Time")
plt.show()
```

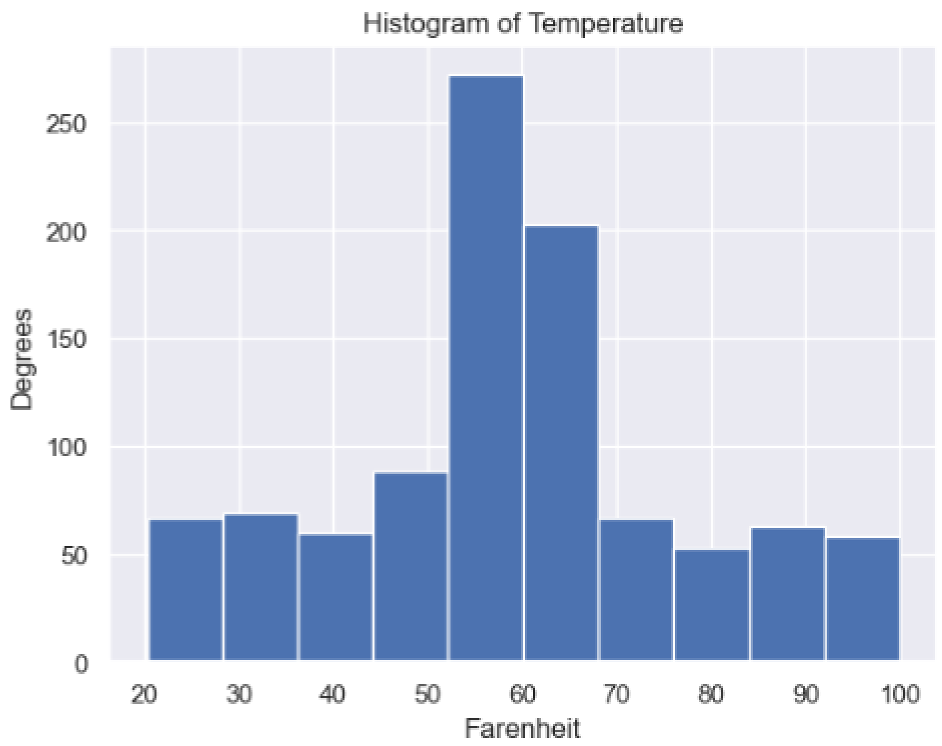


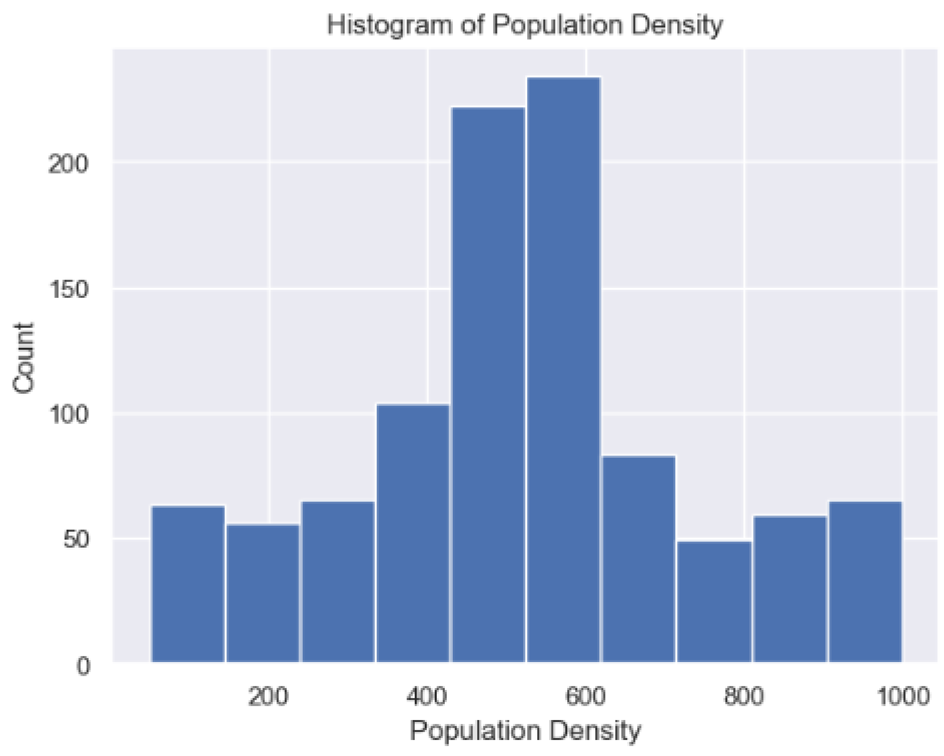


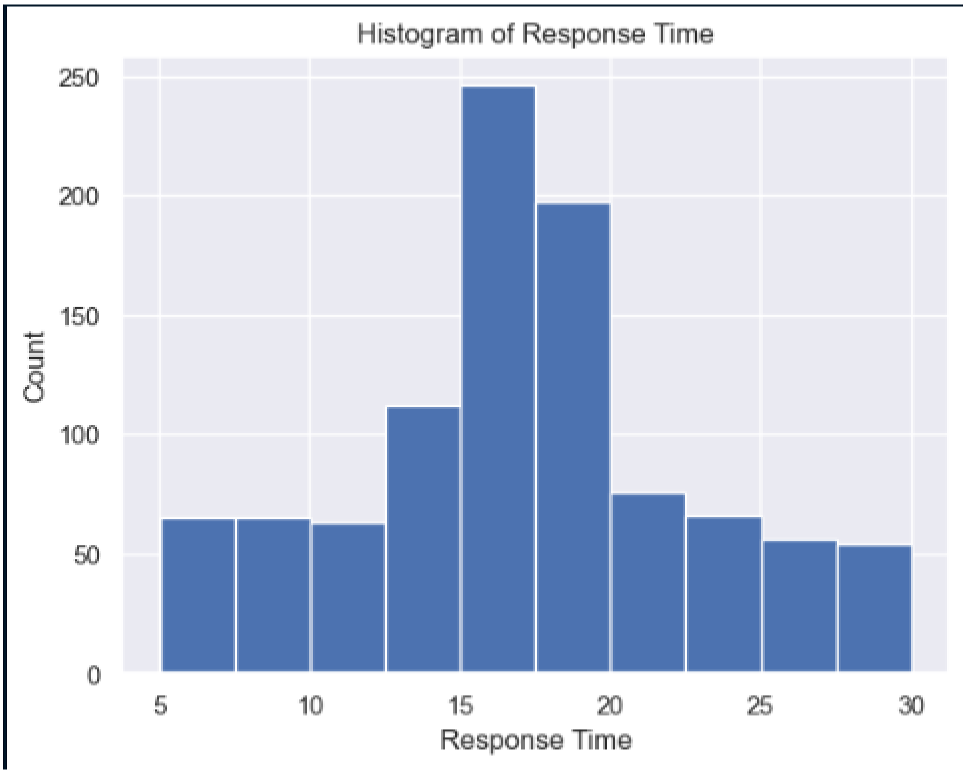
Histogram of VictimAge
Count
Age



Histogram of SuspectAge
Count
Age

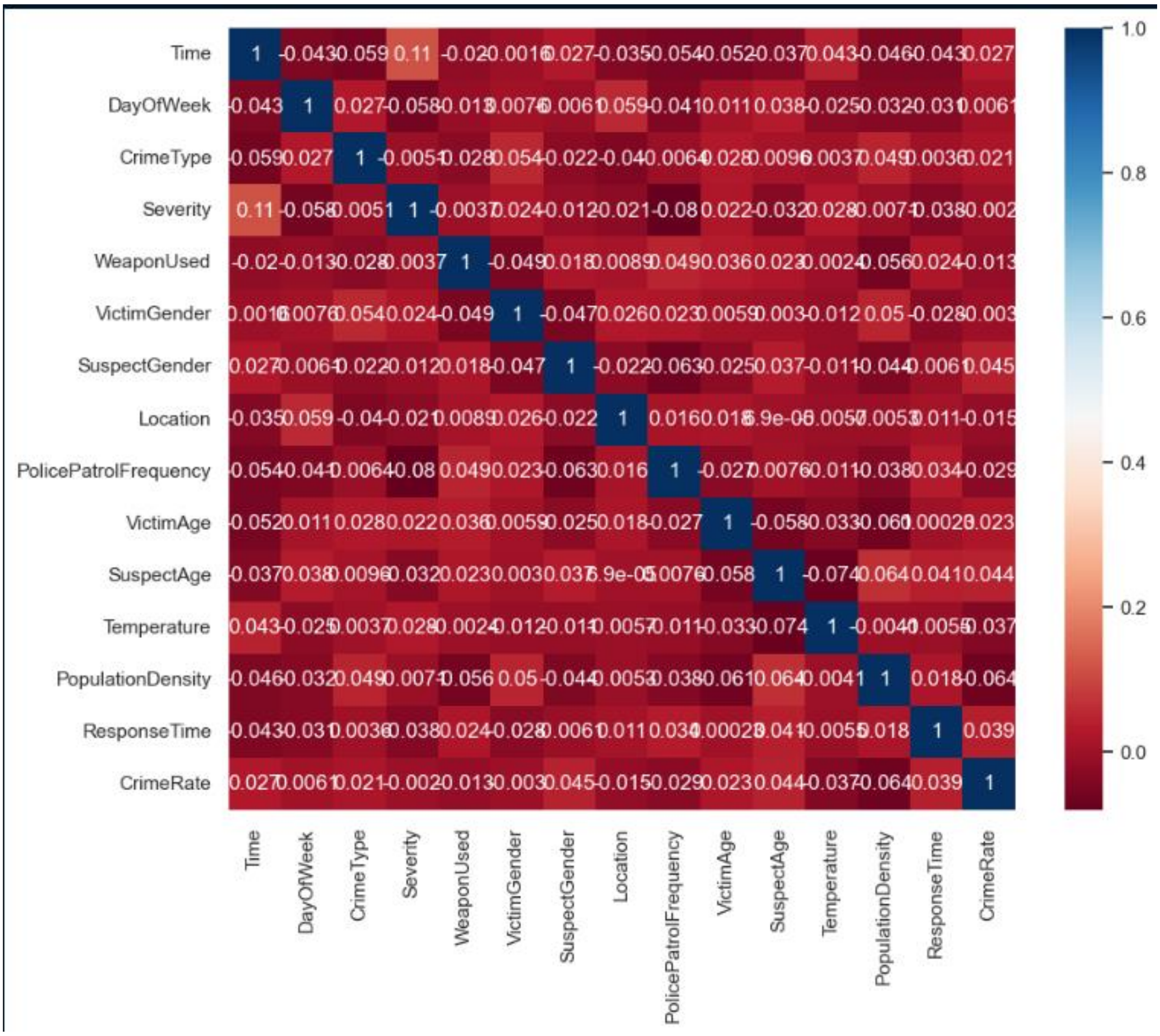

Histogram of Temperature
Degrees
250
200
150
100
50
0
20
30
40
50
60
70
80
90
100
Farenheit


Histogram of Population Density
Count
200
150
100
50
0
200
400
600
800
1000
Population Density



Now in order to find the which features are having correlations with other features we will draw a heatmap, in which, higher the value, higher the chance of correlation between those features.

```
import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 8))
sns.heatmap(df3.corr(), annot=True, cmap="RdBu")
plt.show()
```

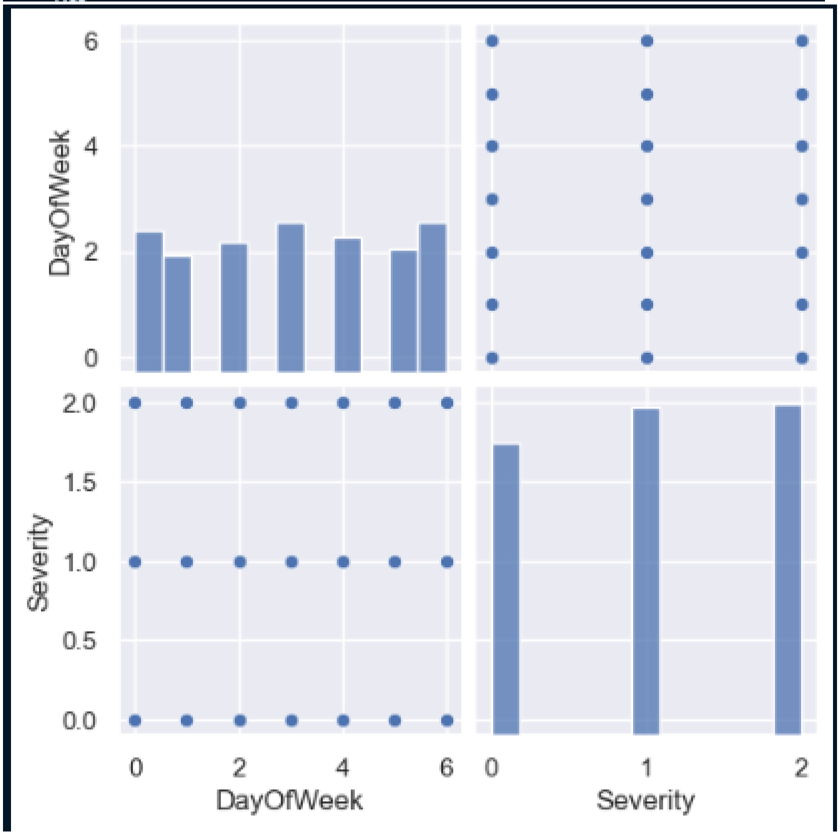


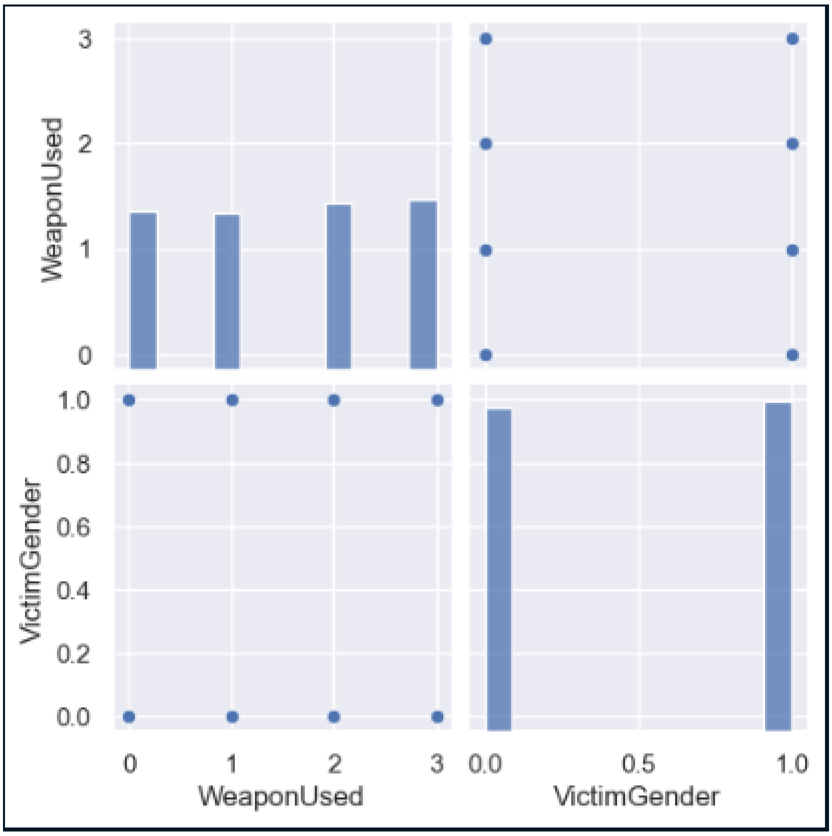
With the help of this heatmap, we can get a clearer idea of the features where correlations exist. Once we find that out, we can look deeper into those relations.

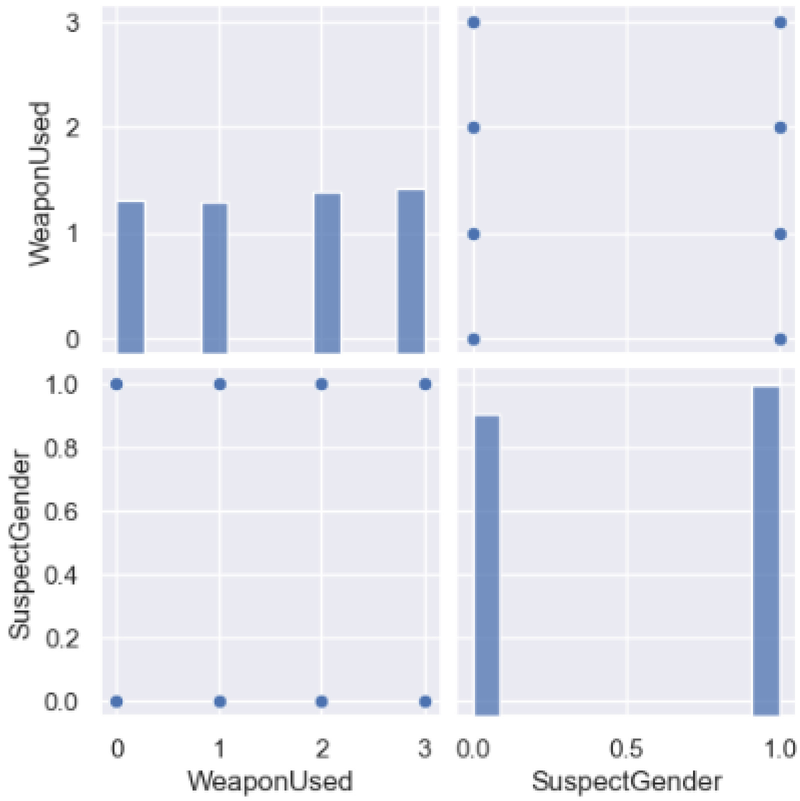
Once we realise this, we can visualize these relationships to gain insights on them. For this we will use the dataframe which was gained using KNN imputation since the object type features were encoded and we need them to find the relationships between other features.

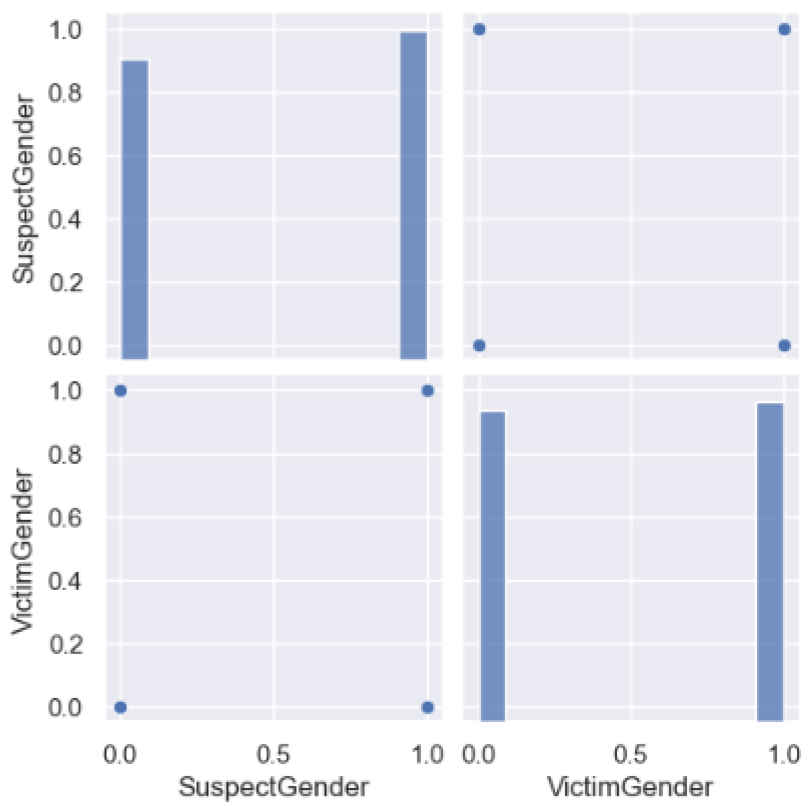
```
import seaborn as sns

sns.pairplot(df3[['DayOfWeek', 'Severity']])
sns.pairplot(df3[['WeaponUsed', 'VictimGender']])
sns.pairplot(df3[['WeaponUsed', 'SuspectGender']])
sns.pairplot(df3[['SuspectGender', 'VictimGender']])
sns.pairplot(df3[['SuspectAge', 'WeaponUsed']])
sns.pairplot(df3[['PolicePatrolFrequency', 'Severity']])
sns.pairplot(df3[['ResponseTime', 'Severity']])
sns.pairplot(df3[['ResponseTime', 'PolicePatrolFrequency']])
sns.pairplot(df3[['CrimeType', 'Location']])
sns.pairplot(df3[['VictimAge', 'SuspectAge']])
```

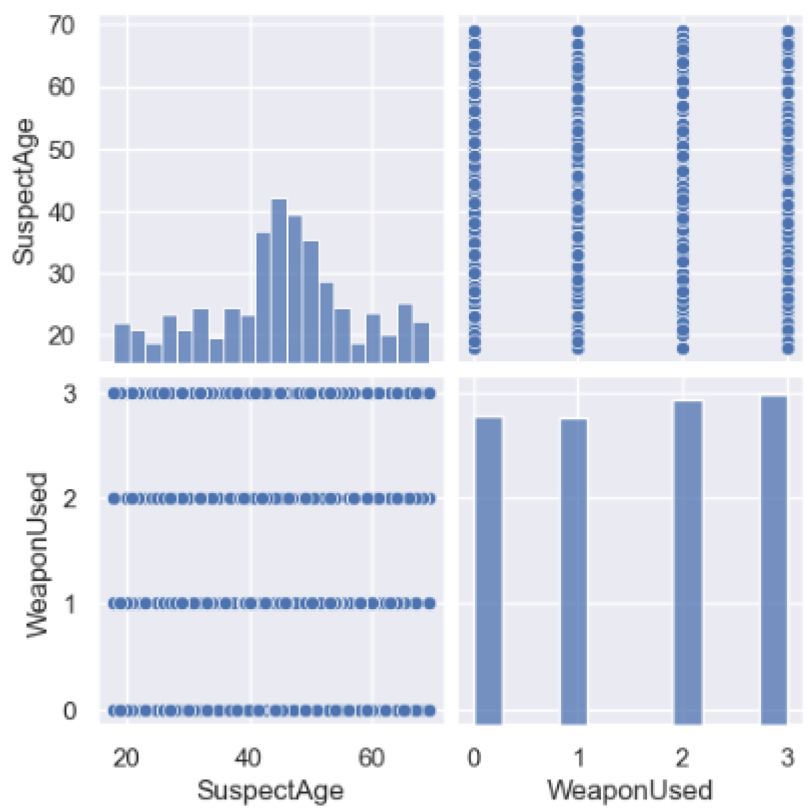
✓ 7.0s

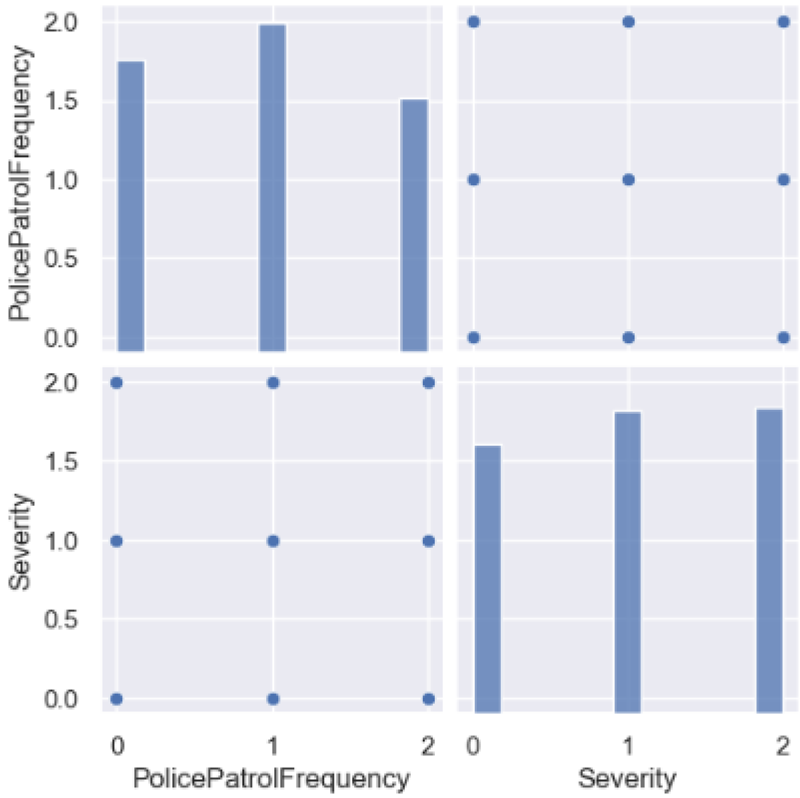


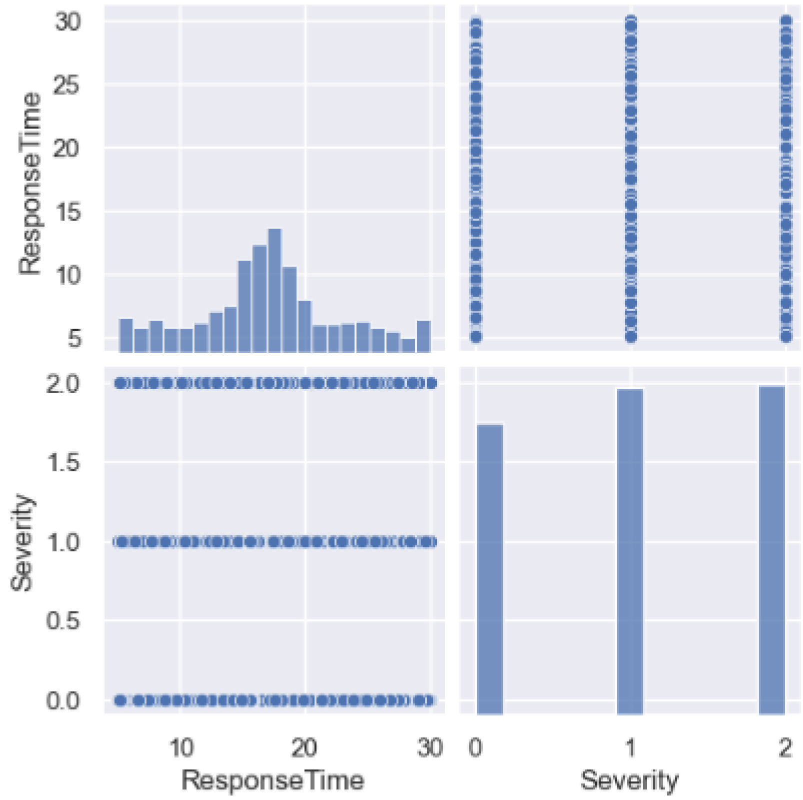

WeaponUsed
VictimGender
0
1
2
3
0.0
0.2
0.4
0.6
0.8
1.0
0.5
WeaponUsed
VictimGender

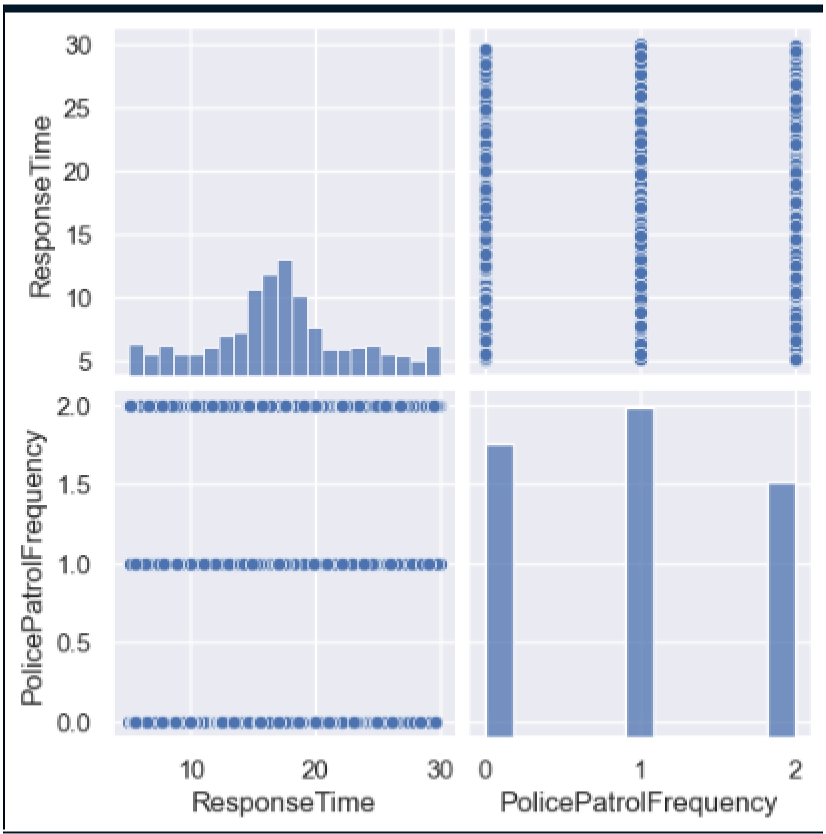

WeaponUsed
SuspectGender
0
1
2
3
0.0
0.2
0.4
0.6
0.8
1.0
0.5
WeaponUsed
SuspectGender


SuspectGender
VictimGender
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.5
1.0
SuspectGender
VictimGender

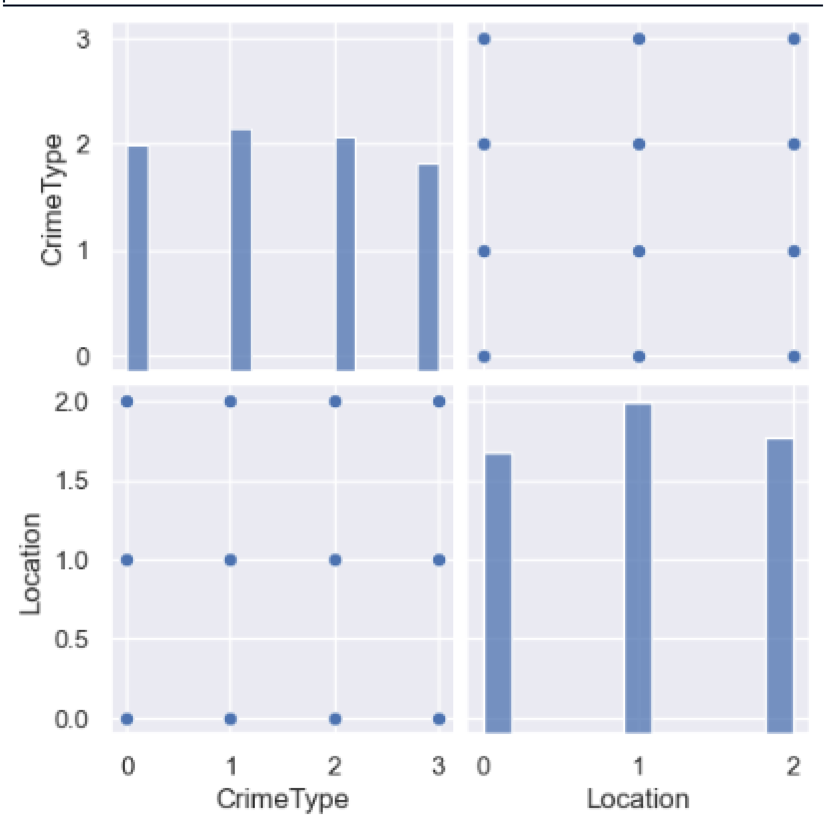

SuspectAge
WeaponUsed
70
60
50
40
30
20
3
2
1
0
20
40
60
SuspectAge
WeaponUsed


PolicePatrolFrequency
Severity
PolicePatrolFrequency
Severity

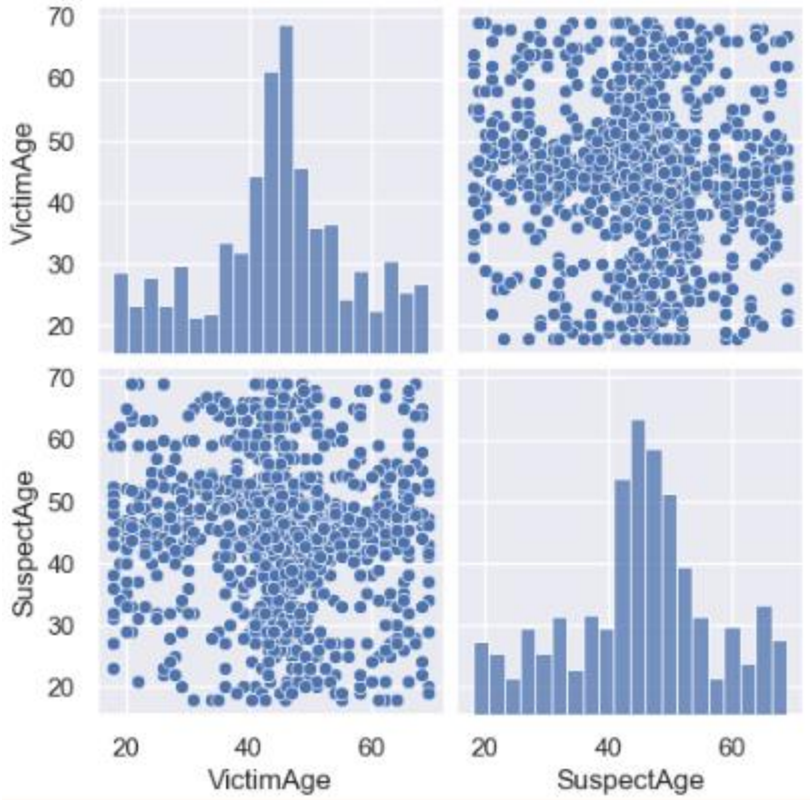

ResponseTime
Severity
ResponseTime
Severity



ResponseTime
PolicePatrolFrequency
ResponseTime
PolicePatrolFrequency



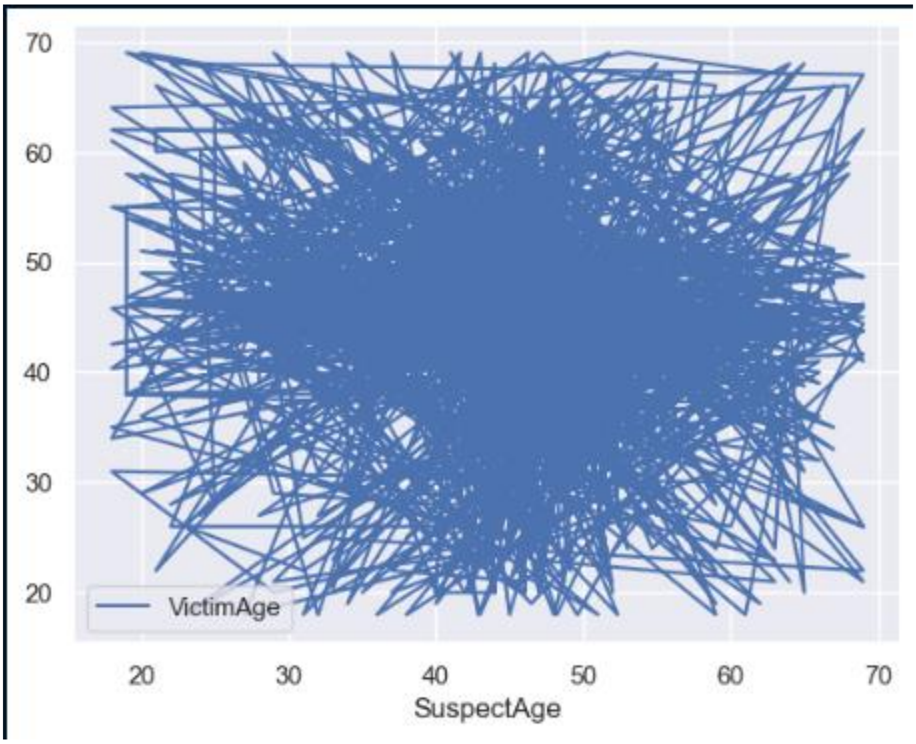
CrimeType
Location
CrimeType
Location



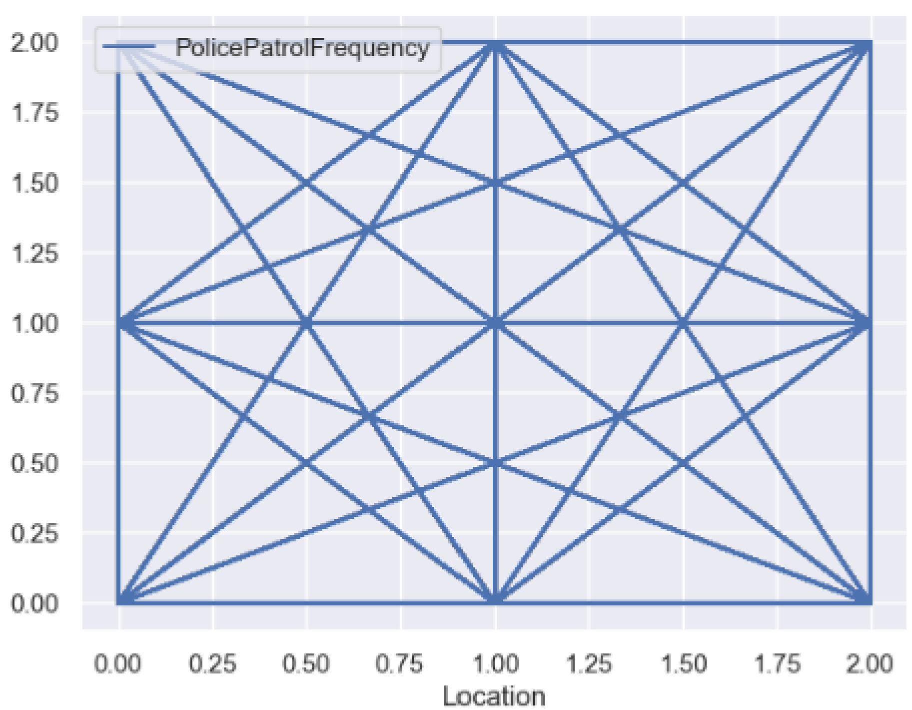
With the help of this, you can understand which of the features have relationships with other features. The more scattered and far they points are from each other, the less related they are to each other.

To verify this, you can also use line plot method.

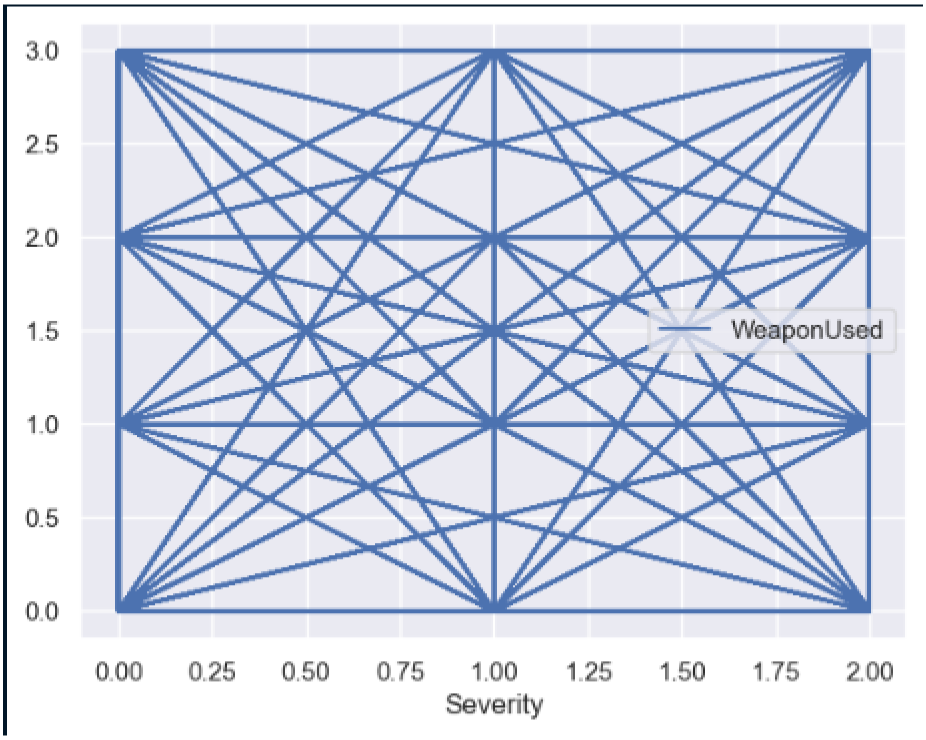
```
df3.plot(x='SuspectAge', y='VictimAge', kind='line')
df3.plot(x='Location', y='PolicePatrolFrequency', kind='line')
df3.plot(x='Severity', y='WeaponUsed', kind='line')
df3.plot(x='Severity', y='ResponseTime', kind='line')
df3.plot(x='PolicePatrolFrequency', y='Location', kind='line')
```

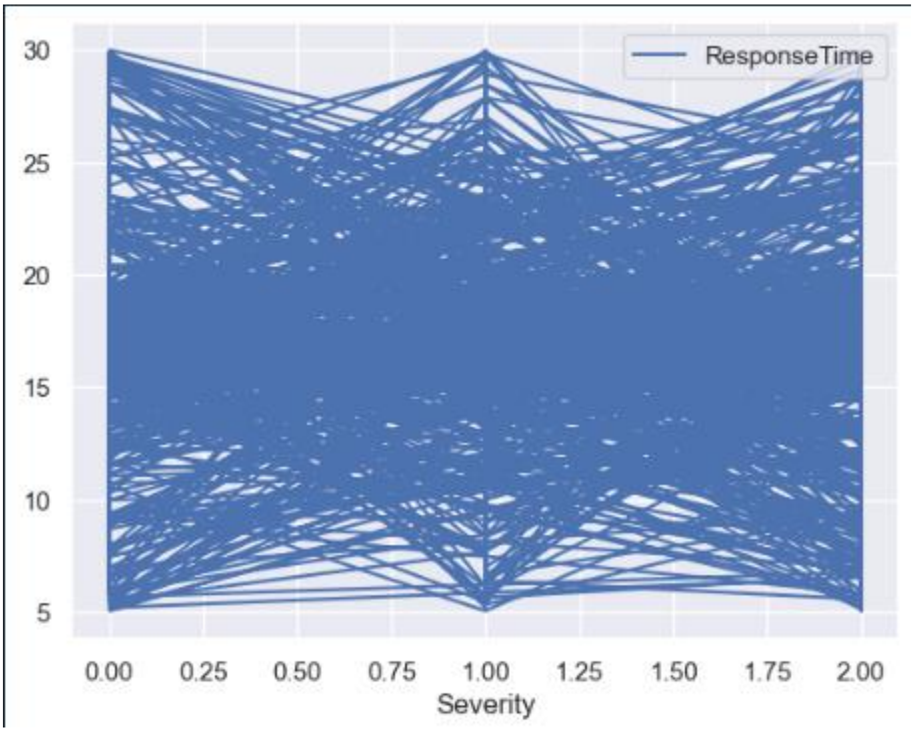


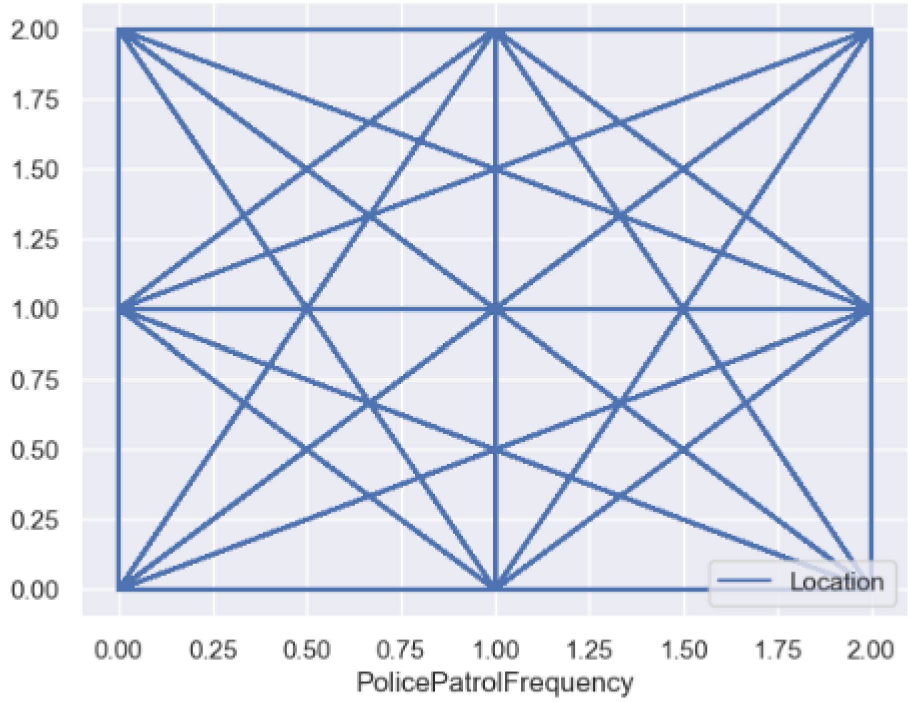
70
60
50
40
30
20
VictimAge
20
30
40
50
60
70
SuspectAge



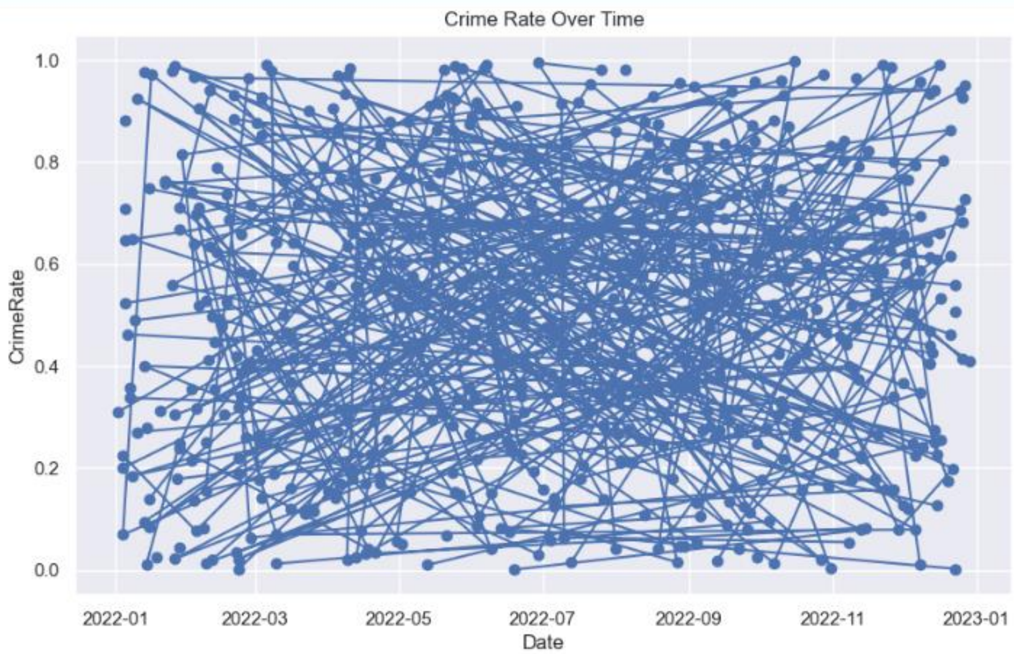
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
PolicePatrolFrequency
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
Location


3.0
2.5
2.0
1.5
1.0
0.5
0.0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
Severity
WeaponUsed


30
25
20
15
10
5
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
Severity
ResponseTime




```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
plt.plot(crime_data['Date'], crime_data['CrimeRate'], marker='o', linestyle='-')
plt.xlabel('Date')
plt.ylabel('CrimeRate')
plt.title('Crime Rate Over Time')
plt.grid(True)
plt.show()
```



In this, if you can find a definite pattern, then it means that there exists a relationship between those 2 features.

Once you gain enough insights on the data, you can make reasonable predictions. These predictions can help you reduce the crime rate and also help apprehend the suspects.