CRIME_DATA_ANALYSIS

September 16, 2024

1 INSTALLING AND IMPORTING LIBRARIES

[1]:
```
#shifting from cpu to gpu
import torch
device = torch.device("cuda") if torch.cuda.is_available() else torch.
↪device("cpu")
device
```

[1]:
```
device(type='cuda')
```

[2]:
```
#installing some extra required libraries
%pip install fuzzywuzzy[speedup]
%pip install statsmodels
%pip install pmdarima
```

```
Collecting fuzzywuzzy[speedup]
  Downloading fuzzywuzzy-0.18.0-py2.py3-none-any.whl.metadata (4.9 kB)
Collecting python-levenshtein>=0.12 (from fuzzywuzzy[speedup])
  Downloading python_Levenshtein-0.25.1-py3-none-any.whl.metadata (3.7 kB)
Collecting Levenshtein==0.25.1 (from python-
levenshtein>=0.12->fuzzywuzzy[speedup])
  Downloading Levenshtein-0.25.1-cp310-cp310-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (3.3 kB)
Collecting rapidfuzz<4.0.0,>=3.8.0 (from Levenshtein==0.25.1->python-
levenshtein>=0.12->fuzzywuzzy[speedup])
  Downloading rapidfuzz-3.9.7-cp310-cp310-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (12 kB)
Downloading python_Levenshtein-0.25.1-py3-none-any.whl (9.4 kB)
Downloading
Levenshtein-0.25.1-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
(177 kB)
                         177.4/177.4 kB
5.3 MB/s eta 0:00:00
Downloading fuzzywuzzy-0.18.0-py2.py3-none-any.whl (18 kB)
Downloading
rapidfuzz-3.9.7-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (3.4
MB)
                         3.4/3.4 MB
```

```
43.5 MB/s eta 0:00:00
Installing collected packages: fuzzywuzzy, rapidfuzz, Levenshtein, python-
levenshtein
Successfully installed Levenshtein-0.25.1 fuzzywuzzy-0.18.0 python-
levenshtein-0.25.1 rapidfuzz-3.9.7
Requirement already satisfied: statsmodels in /usr/local/lib/python3.10/dist-
packages (0.14.2)
Requirement already satisfied: numpy>=1.22.3 in /usr/local/lib/python3.10/dist-
packages (from statsmodels) (1.26.4)
Requirement already satisfied: scipy!=1.9.2,>=1.8 in
/usr/local/lib/python3.10/dist-packages (from statsmodels) (1.13.1)
Requirement already satisfied: pandas!=2.1.0,>=1.4 in
/usr/local/lib/python3.10/dist-packages (from statsmodels) (2.1.4)
Requirement already satisfied: patsy>=0.5.6 in /usr/local/lib/python3.10/dist-
packages (from statsmodels) (0.5.6)
Requirement already satisfied: packaging>=21.3 in
/usr/local/lib/python3.10/dist-packages (from statsmodels) (24.1)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.10/dist-packages (from pandas!=2.1.0,>=1.4->statsmodels)
(2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-
packages (from pandas!=2.1.0,>=1.4->statsmodels) (2024.2)
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-
packages (from pandas!=2.1.0,>=1.4->statsmodels) (2024.1)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages
(from patsy>=0.5.6->statsmodels) (1.16.0)
Collecting pmdarima
  Downloading pmdarima-2.0.4-cp310-cp310-
manylinux_2_17_x86_64.manylinux2014_x86_64.manylinux_2_28_x86_64.whl.metadata
(7.8 kB)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.10/dist-
packages (from pmdarima) (1.4.2)
Requirement already satisfied: Cython!=0.29.18,!=0.29.31,>=0.29 in
/usr/local/lib/python3.10/dist-packages (from pmdarima) (3.0.11)
Requirement already satisfied: numpy>=1.21.2 in /usr/local/lib/python3.10/dist-
packages (from pmdarima) (1.26.4)
Requirement already satisfied: pandas>=0.19 in /usr/local/lib/python3.10/dist-
packages (from pmdarima) (2.1.4)
Requirement already satisfied: scikit-learn>=0.22 in
/usr/local/lib/python3.10/dist-packages (from pmdarima) (1.3.2)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-
packages (from pmdarima) (1.13.1)
Requirement already satisfied: statsmodels>=0.13.2 in
/usr/local/lib/python3.10/dist-packages (from pmdarima) (0.14.2)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.10/dist-
packages (from pmdarima) (2.0.7)
Requirement already satisfied: setuptools!=50.0.0,>=38.6.0 in
/usr/local/lib/python3.10/dist-packages (from pmdarima) (71.0.4)
```

```
Requirement already satisfied: packaging>=17.1 in
/usr/local/lib/python3.10/dist-packages (from pmdarima) (24.1)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.10/dist-packages (from pandas>=0.19->pmdarima) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-
packages (from pandas>=0.19->pmdarima) (2024.2)
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-
packages (from pandas>=0.19->pmdarima) (2024.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.22->pmdarima)
(3.5.0)
Requirement already satisfied: patsy>=0.5.6 in /usr/local/lib/python3.10/dist-
packages (from statsmodels>=0.13.2->pmdarima) (0.5.6)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages
(from patsy>=0.5.6->statsmodels>=0.13.2->pmdarima) (1.16.0)
Downloading pmdarima-2.0.4-cp310-cp310-
manylinux_2_17_x86_64.manylinux2014_x86_64.manylinux_2_28_x86_64.whl (2.1 MB)
                         2.1/2.1 MB
19.0 MB/s eta 0:00:00
Installing collected packages: pmdarima
Successfully installed pmdarima-2.0.4
```

[39]:
```
#libraries data procurement and wrangling
import csv
import requests
import pandas as pd
import numpy as np
import datetime
from fuzzywuzzy import process
import io
from sklearn.impute import KNNImputer

#libraies for cleanig and visualization
import missingno as msno

#libraries data visualization
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.pylab import rcParams
import plotly.graph_objects as go
import plotly.express as px
import folium
from folium.plugins import HeatMap
import plotly.io as pio
pio.renderers.default = 'colab'
pd.options.plotting.backend = "plotly"
```

```
#libraries for time series analysis and prediction
from statsmodels.tsa.stattools import adfuller
import pmdarima as pm

#libraries for ignoring unwanted error warnings
import warnings
warnings.filterwarnings("ignore")
sns.set()
```

2 GETTING DATA USING API

[4]:
```
%%time

CSV_URL = 'https://data.montgomerycountymd.gov/resource/icn6-v9z3.csv'
chunk_size = 1000 #API sends only 1000 rows in one call

chunks = []
offset = 0 #add chuck_size into it after every loop tp get next 1000 rpws

while True:
    # API URL with offset and limit parameters
    api_url = f"{CSV_URL}?$offset={offset}&$limit={chunk_size}"

    response = requests.get(api_url)
    if response.status_code != 200:
        break  # Stop if there was an error in the request

    chunk = pd.read_csv(io.StringIO(response.text))

    if chunk.empty:
        break  # Break the loop if no data is returned

    chunks.append(chunk)

    # Increment the offset for the next request
    offset += chunk_size

raw_data = pd.concat(chunks, ignore_index=True)
```

```
CPU times: user 5.9 s, sys: 1.22 s, total: 7.12 s
Wall time: 3min 32s
```

[137]:
```
df=raw_data.copy()
print(df.shape)
df.head(3)
```

```
(319492, 30)
```

[137]:

```
   incident_id  offence_code  case_number                     date  \
0    201495777          1301    240043886  2024-09-15T02:13:50.000
1    201495771          9199    240043881  2024-09-15T00:15:50.000
2    201495776          3512    240043882  2024-09-14T23:55:26.000

                start_date                 end_date nibrs_code  victims  \
0  2024-09-15T02:13:00.000                      NaN        13A        1
1  2024-09-15T00:15:00.000                      NaN        90Z        1
2  2024-09-14T23:55:00.000  2024-09-15T00:30:00.000        35A        1

              crimename1                crimename2  …  pra address_number  \
0   Crime Against Person        Aggravated Assault  …  348         3500.0
1  Crime Against Society        All Other Offenses  …  208         8600.0
2  Crime Against Society  Drug/Narcotic Violations  …  295         1100.0

  street_prefix_dir address_street street_suffix_dir street_type  latitude  \
0               NaN      PEAR TREE               NaN          CT  39.08919
1               NaN      SPLIT OAK               NaN         CIR  38.99742
2               NaN       CRAWFORD               NaN          DR  39.07809

  longitude police_district_number                  geolocation
0  -77.0695                     4D  \n,  \n(39.0892, -77.0695)
1  -77.1658                     2D  \n,  \n(38.9974, -77.1658)
2  -77.1331                     1D  \n,  \n(39.0781, -77.1331)

[3 rows x 30 columns]
```

[6]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 319492 entries, 0 to 319491
Data columns (total 30 columns):
 #   Column                  Non-Null Count   Dtype
---  ------                  --------------   -----
 0   incident_id             319492 non-null  int64
 1   offence_code            319492 non-null  int64
 2   case_number             319492 non-null  int64
 3   date                    272575 non-null  object
 4   start_date              319492 non-null  object
 5   end_date                156437 non-null  object
 6   nibrs_code              319492 non-null  object
 7   victims                 319492 non-null  int64
 8   crimename1              319492 non-null  object
 9   crimename2              319492 non-null  object
 10  crimename3              319492 non-null  object
 11  district                318864 non-null  object
```

```
 12  location                 290433 non-null  object
 13  city                     319480 non-null  object
 14  state                    312132 non-null  object
 15  zip_code                 317003 non-null  float64
 16  agency                   319492 non-null  object
 17  place                    319492 non-null  object
 18  sector                   319492 non-null  object
 19  beat                     319492 non-null  object
 20  pra                      319486 non-null  object
 21  address_number           290544 non-null  float64
 22  street_prefix_dir        14526 non-null   object
 23  address_street           318863 non-null  object
 24  street_suffix_dir        4677 non-null    object
 25  street_type              318539 non-null  object
 26  latitude                 319492 non-null  float64
 27  longitude                319492 non-null  float64
 28  police_district_number   319492 non-null  object
 29  geolocation              319492 non-null  object
dtypes: float64(4), int64(4), object(22)
memory usage: 73.1+ MB
```

3 DATA PREPROCESSING

[7]:
```
#Used this code to download an excel sheet with every unique crime-name for␣
↪editing manually as per requirment
#its jumbled, so rearrange accordingly
'''crime_3=df['Crime Name3'].unique()
crime_3 = pd.DataFrame(crime_3)
# Specify the file path where you want to save the Excel file
excel_file_path = 'C:\\Users\\saadu\\Desktop\\crime_cat_3.xlsx'  # Change this␣
↪to your desired file name and location
# Save the DataFrame to an Excel file
crime_3.to_excel(excel_file_path, index=False)'''

'''unique_crime_names_1= df.groupby('crimename3')['crimename1'].unique()
ucn_dict_1=dict(unique_crime_names_1)
ucn_dict_1
c3['Crime_1']=c3['Crime'].map(ucn_dict_1)
unique_crime_names= df.groupby('crimename3')['crimename2'].unique()
ucn_dict=dict(unique_crime_names)
ucn_dict

df.groupby('crimename3')['crimename2'].unique().reset_index()
excel_file_path = 'crime_3_revised_2.xlsx'  # Change this to your desired file␣
↪name and location


# Save the DataFrame to an Excel file
```

```
c3.to_excel(excel_file_path, index=False)
df.groupby('crimename1')['crimename3'].unique().loc[('Crime Against Person')]'''
```

[7]:

```
"unique_crime_names_1= df.groupby('crimename3')['crimename1'].unique()\nucn_dict
_1=dict(unique_crime_names_1)\nucn_dict_1\n\nc3['Crime_1']=c3['Crime'].map(ucn_d
ict_1)\n\nunique_crime_names= df.groupby('crimename3')['crimename2'].unique()\nu
cn_dict=dict(unique_crime_names)\nucn_dict\n\ndf.groupby('crimename3')['crimenam
e2'].unique().reset_index()\n\nexcel_file_path = 'crime_3_revised_2.xlsx'  #
Change this to your desired file name and location\n\n# Save the DataFrame to an
Excel file\nc3.to_excel(excel_file_path,
index=False)\n\ndf.groupby('crimename1')['crimename3'].unique().loc[('Crime
Against Person')]"
```

3.0.1 CRIME CATEGORY RECATEGORIZATION

[8]:

```
%%time
#All crimename1 and crimename2 categories, just for reference
for j in df['crimename1'].unique():
  print(j)
  count=0
  for i in df['crimename2'][df['crimename1']==j].unique():
    print(i)
    count=count+1
  print(count)
  print('\n')
```

```
Crime Against Person
Aggravated Assault
Simple Assault
Forcible Fondling
Forcible Rape
Murder and Nonnegligent Manslaughter
Statuory Rape
Sexual Assault With An Object
Intimidation
Kidnapping/Abduction
Forcible Sodomy
Human Trafficking, Commercial Sex Acts
Incest
Purchasing Prostitution
Negligent Manslaughter
Human Trafficking, Involuntary Servitude
15


Crime Against Society
All Other Offenses
```

Drug/Narcotic Violations
Disorderly Conduct
Driving Under the Influence
Liquor Law Violations
Trespass of Real Property
Pornography/Obscene Material
Drug Equipment Violations
Weapon Law Violations
Family Offenses, NonViolent
Prostitution
Assisting or Promoting Prostitution
Animal Cruelty
Curfew/Loitering/Vagrancy Violations
Operating/Promoting/Assisting Gambling
15

Crime Against Not a Crime
Runaway
Justifiable Homicide
2

Crime Against Property
Destruction/Damage/Vandalism of Property
Robbery
Theft from Building
Shoplifting
Burglary/Breaking and Entering
Embezzlement
False Pretenses/Swindle/Confidence Game
Theft From Motor Vehicle
Motor Vehicle Theft
Theft of Motor Vehicle Parts or Accessories
Purse-snatching
All other Larceny
Identity Theft
Pocket/picking
Counterfeiting/Forgery
Credit Card/Automatic Teller Machine Fraud
Wire Fraud
Impersonation
Extortion/Blackmail
Arson
From Coin/Operated Machine or Device
Stolen Property Offenses
Hacking/Computer Invasion
Welfare Fraud

```
Bribery
25


Crime Against Person, Property, or Society
All Other Offenses
1


CPU times: user 169 ms, sys: 510 µs, total: 170 ms
Wall time: 168 ms
```

[9]:
```
#storing crimename2 in a diff column coz its based on nibrs code and I wanted␣
↪to edit it for reduction of categories and easier meaningful EDA
df['nibrs_crime']=df['crimename2']
```

[13]:
```
#Reading the revised crime categories
#Reduced crimename2 and crimename3 catregories, added columns for weapon, age,␣
↪sex and intensity(physical harm)
c3=pd.read_csv('crime_3_revised_2.csv')
c3.head(3)
```

[13]:
```
                    Crime_1       Crime_1_Revised                 Crime_2  \
0  ['Crime Against Person']  CRIME AGAINST PERSON  ['Aggravated Assault']
1  ['Crime Against Person']  CRIME AGAINST PERSON  ['Aggravated Assault']
2  ['Crime Against Person']  CRIME AGAINST PERSON  ['Aggravated Assault']

  Crime_2_Revised                                          Crime_3  \
0         ASSAULT                          ASSAULT - AGGRAVATED - OTHER
1         ASSAULT  ASSAULT - AGGRAVATED - NON-FAMILY-OTHER WEAPON
2         ASSAULT        ASSAULT - AGGRAVATED - FAMILY-STRONG-ARM

      Crime_3_Revised     Weapon Type  Sex Age Category      Intensity
0  AGGRAVATED ASSAULT       No Weapon  NaN          NaN  Physical Harm
1  AGGRAVATED ASSAULT  Unknown Weapon  NaN          NaN     Endangered
2  AGGRAVATED ASSAULT      Strong Arm  NaN          NaN  Physical Harm
```

[11]:
```
#Function to implement the changes
def crime_mapping(a,b):  # a = column in revised crime-category data  that we␣
↪want to map, b = column name in the main datasets that will be affected/
↪replaced/added

  # Create the mapping dictionary
  crime_to_column_mapping = dict(zip(c3['Crime_3'], c3[a]))

  df[b] = df['crimename3'].map(crime_to_column_mapping)
```

```
    # Handle missing cases while mapping
    df[b].fillna('No Entry', inplace=True)
```

[14]:
```
#Calling Function to implement the desired changes
crime_mapping('Weapon Type','Weapon Type')
crime_mapping('Sex', 'Sex')
crime_mapping('Age Category','Age Category')
crime_mapping('Intensity', 'Intensity')
crime_mapping('Crime_2_Revised','crimename2')
crime_mapping('Crime_1_Revised','crimename1')
crime_mapping('Crime_3_Revised','crimename3')

#Creating other columns just for whether weapon was used or not
weapon={'No Weapon':False, 'Unknown Weapon':True, 'Gun':True, 'Knife':True,␣
 ↪'Explosive':True, 'Incendiary Device':True, 'Strong Arm':False}
df['Weapon'] = df['Weapon Type'].map(weapon)

#Found one entry/row which was "No Entry", removiong it
df=df[df['crimename1']!='No Entry']
df.reset_index(drop=True,inplace=True)
```

[15]:
```
%%time
#All crimename1 and crimename2 categories, just for reference and comparison
for j in df['crimename1'].unique():
  print(j)
  count=0
  for i in df['crimename2'][df['crimename1']==j].unique():
    print(i)
    count=count+1
  print(count)
  print('\n')
```

```
CRIME AGAINST PERSON
ASSAULT
SEX OFFENCE
HOMICIDE
INTIMIDATION
KIDNAPPING/ABDUCTION
5


CRIME AGAINST PERSON, PROPERTY, OR SOCIETY
ALL OTHER OFFENSES
OBSTRUCT GOVERNMENT/POLICE
FRAUD
3
```

```
CRIME AGAINST SOCIETY
DRUG/NARCOTIC VIOLATIONS
ALL OTHER OFFENSES
SEX OFFENCE
DRIVING UNDER THE INFLUENCE
LIQUOR LAW VIOLATIONS
TRESPASSING
PORNOGRAPHY/OBSCENE MATERIAL
WEAPON LAW VIOLATIONS
FAMILY OFFENSE
DISORDERLY CONDUCT
ANIMAL CRUELTY
CURFEW/LOITERING/VAGRANCY VIOLATION
GAMBLING
13


CRIME AGAINST NOT A CRIME
RUNAWAY
HOMICIDE
2


CRIME AGAINST PROPERTY
DESTRUCTION/DAMAGE/VANDALISM
LARCENY
BURGLARY/BREAKING AND ENTERING
FRAUD
DISORDERLY CONDUCT
EXTORT/BLACKMAIL
ARSON
STOLEN PROPERTY OFFENSES
BRIBERY
9


CPU times: user 147 ms, sys: 593 µs, total: 147 ms
Wall time: 146 ms
```

[16]:
```python
# Final dataframe of all unique crime categories

#Use code in strings to print the entire list
'''pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)'''

df.groupby(['crimename1','crimename2','crimename3'])['crimename3'].unique()
```

[16]:
```
crimename1                  crimename2              crimename3
CRIME AGAINST NOT A CRIME   HOMICIDE                JUSTIFIABLE HOMICIDE
[JUSTIFIABLE HOMICIDE]
                            RUNAWAY                 JUVENILE RUNAWAY
[JUVENILE RUNAWAY]
CRIME AGAINST PERSON        ASSAULT                 2ND DEGREE ASSAULT
[2ND DEGREE ASSAULT]
                                                    AGGRAVATED ASSAULT
[AGGRAVATED ASSAULT]
                                                    SIMPLE ASSAULT
[SIMPLE ASSAULT]
                                     …
CRIME AGAINST SOCIETY       WEAPON LAW VIOLATIONS   WEAPON CONCEALED
[WEAPON CONCEALED]
                                                    WEAPON FIRING
[WEAPON FIRING]
                                                    WEAPON POSSESSION
[WEAPON POSSESSION]
                                                    WEAPON SELLING
[WEAPON SELLING]
                                                    WEAPON
TRANSPORTATION/TRAFFICKING    [WEAPON TRANSPORTATION/TRAFFICKING]
Name: crimename3, Length: 191, dtype: object
```

[17]:
```
#Use this code to reset pandas to default, else notebook might hang if the␣
↪dataset if too large
'''pd.reset_option('display.max_rows')
pd.reset_option('display.max_columns')'''
```

[17]:
```
"pd.reset_option('display.max_rows')\npd.reset_option('display.max_columns')"
```

3.0.2 CITY NAME CLEANING

[18]:
```
#Get all unique cities in the dataset
df.city.unique()
```

[18]:
```
array(['SILVER SPRING', 'BETHESDA', 'ROCKVILLE', 'GERMANTOWN',
       'MONTGOMERY VILLAGE', 'DAMASCUS', 'OLNEY', 'GAITHERSBURG',
       'TAKOMA PARK', 'POOLESVILLE', 'BROOKEVILLE', 'POTOMAC', 'DERWOOD',
       'CHEVY CHASE', 'CLARKSBURG', 'BURTONSVILLE', 'KENSINGTON',
       'CABIN JOHN', 'BOYDS', 'ASHTON', 'LAYTONSVILLE', 'MCLEAN',
       'SANDY SPRING', 'WASHINGTON', 'DICKERSON', 'BELTSVILLE',
       'SPENCERVILLE', 'WASHINGTON GROVE', 'MT AIRY', 'BARNESVILLE',
       'BEALLSVILLE', 'LANHAM', 'LAUREL', 'BRINKLOW', 'FALLS CHURCH',
       'NORTH BETHESDA', 'GARRETT PARK', 'HYATTSVILLE', 'GLEN ECHO',
       'BEHESDA', 'WHEATON', 'ASPEN HILL', 'GAITHERBURG', 'BEHTESDA',
       'BEALSVILLE', 'NORTH POTOMAC', 'HIGHLAND', 'COLESVILLE', 'HERNDON',
```

```
'WOODBINE', 'DARNESTOWN', 'GRMANTOWN', 'BETHEDA', 'GAIHTERSBURG',
'CAPITOL HEIGHTS', 'SILVERS SPRING', 'FRIENDSHIP HEIGHTS',
'CLAEKSBURG', 'MOUNT AIRY', 'TACOMA PARK', 'GAUTHERSBURG',
'REDLAND', 'WEHATON', 'SILVE SPRING', 'KENSTINGTON', 'BARNSVILLE',
'SILVER SPRIN G', 'GAITHERSSBURG', 'DEERWOOD', '4', 'KENSINGTNO',
'2', 'SILER SPRING', 'GAITHERESBURG', 'TAKOMA', '1', 'HYATTTOWN',
'GERMANTWN', 'GIATHERSBURG', 'ADELPHI', 'GATIHERSBURG',
"ROCKVILLE'", 'WHITE OAK', 'MT. AIRY', 'OXON HILL', 'GAITHERSBRUG',
'SILVER APRING', 'GREENBELT', 'SILVER SPING', 'ROCKVILLLE', nan,
'BOWIE', 'SILVER SPRIG', 'GERMATOWN', 'GAITEHRSBURG', 'FREDERICK',
'GAITHERSGURG', 'GERMANTNOWN', 'CLARKESBURG', 'ROCKVIILLE', '6',
'BURTSONVILLE', 'GEMANTOWN', 'DISTRICT OF COLUMBIA',
'GAITHERSBUIRG', 'GAITHERSBURT', 'CLARSBURG', 'FRIENDHSIP HEIGHTS',
'GERMANTOWM', 'GERMANTOWNMD', 'MONTGOMERY VILLLAGE',
'SILVR SPRING', 'WEATON', 'GAITHESBURG', 'BUTINSVILLE',
'GAITHERSRBURG', 'COLUMBIA', 'APENCERVILLE', 'N BETHESDA',
'GAITHRERSBURG', 'SILVERSPRING', 'SILVER', 'NORTH POTOAMC', 'RO',
'TP', 'SILVER SPRIN', 'GA', 'N POTOMAC', 'BETHSDA', 'CLARSKBURG',
'MONTGOMERY VILAGE', 'SILVER SRPING', 'COMUS', 'ROCKIVLLE',
'MONTGOMERY COUNTY', 'GITHERSBURG', 'PO', 'SILVER SPRIND',
'N. POTOMAC', 'TAKOMS PARK', 'GLEN ECHO`', 'GERMANTOOWN',
'GAIHERSBURG', 'GERMANTOW', 'KE', 'GERMANTOEN', 'BETHESA',
'ROCKVILE', 'GAITERSBURG', 'ONLEY', 'CEHVY CHASE', 'GERMANTONW',
'20877', 'GERMANTIWN', 'SIVLER SPRING', 'CHEVY CHASE #4',
'VALLEYWOOD', 'SILVER SPSRING', 'BARNESVIILE', 'SILVER SRING',
'MONTGOMRY VILLAGE', 'KENSIGNTON', 'GEERMANTOWN',
'CHEVY CHASE VILLAGE', 'NOTRTH POTOMAC', 'BURTOSNVILLE',
'SILVER SPRNG', 'GERMNATOWN', 'ROCKILLE', 'MCG', 'ROCVILLE',
'GAITHERSBYRG', 'ROCVKILLE', '3', 'BROOKVILLE', 'GAITHERBSURG',
'SILVER  SPRING', 'LATONSVILLE', 'MOTGOMERY VILLAGE',
'GAITHESRBURG', 'POTIMAC', 'ROOCKVILLE', 'CHVEY CHASE',
'ROCKIVILLE', 'KENNSINGTON', 'GAITHERSBRG', 'FOREST HEIGHTS',
'GERAMNTOWN', 'DANASCUS', 'GERMANTWON', 'ROCKVIILE', 'GERMAN4TOWN',
'MONGTOMERY VILLAGE', 'ONEY', 'ROKVILLE', 'GERRMANTOWN', 'LA',
'COLLEGE PARK', 'SLIVER SPRING', 'POOLSVILLE', 'SILVER SPIRNG',
'ROCKVLLE', 'BALTIMORE', 'SIVER SPRING', 'BETESDA', 'BETHESDAS',
'GLENMONT', 'ROCKVILL', 'NORTH CHEVY CHASE', 'GATHERSBURG',
'ROCKVILEE', 'GERMNTOWN', 'MONTOMGERY VILLAGE', 'CALARKSBURG',
'MONTGOMERY', 'MONTGGOMERY VILLAGE', 'N. BETHESDA',
'MCGGAITHERSBURG', 'CHEVY CHASE VIEW', 'HYATTSTOWN', 'KENSINGTOWN',
'KENSONGTON', 'NORTH BEHTESDA', 'NORTH BETHSDA', 'SILVER SPRINGQ',
'CHEVY CHASE #3', 'N BETHESDAQ', 'GAISTHERSBURG', 'GAITHRESBURG',
'SANDY SPPRING', 'MONT VILLAGE', 'MONTGOMERY VILLAE',
'SILVE4R SPRING', 'GERANTOWN', 'ROCKVILLE,', 'SILVER SPRING`',
'MOMTGOMERY VILLAGE', 'SILVER SPRNIG', 'GAITHERSURG',
'SLVER SPRING', 'RCKVILLE', 'GAITHERSBUG', 'SILVER SPRIING',
'CLARKSURG', 'MARYLAND', 'SILVER SPRIGN', 'MD'], dtype=object)
```

```
[19]: #Create Dataset of unique city names
city_data = {'city_name': df.city.unique()}
city_df = pd.DataFrame(city_data)

# List of correct city names from Montgomery County(Google Search)
correct_city_names = ['ASHTON', 'BARNESVILLE', 'BEALLSVILLE', 'BETHESDA',␣
 ↪'BOYDS','BRINKLOW', 'BROOKEVILLE', 'BURTONSVILLE', 'CABIN JOHN',
                       'CHEVY CHASE', 'CLARKSBURG', 'DAMASCUS', 'DERWOOD',␣
 ↪'DICKERSON','GAITHERSBURG', 'GARRETT PARK', 'GERMANTOWN', 'GLEN ECHO',
                       'KENSINGTON', 'MONTGOMERY VILLAGE', 'OLNEY',␣
 ↪'POOLESVILLE','POTOMAC', 'ROCKVILLE', 'SANDY SPRING', 'SILVER SPRING',
                       'SPENCERVILLE', 'TAKOMA PARK', 'WASHINGTON␣
 ↪GROVE','MCG','WHEATON','ASPEN HILL','DARNESTOWN','FRIENDSHIP HEIGHTS',
                       'MOUNT AIRY','REDLAND','WHITE␣
 ↪OAK','WHEATON','COMUS','LAYTONSVILLE','GLENMONT','HYATTSTOWN']

# Function to find the best match for each city name in the dataset usinf␣
 ↪FuzzyWuzz
def correct_city_name(city, correct_city_names):
    if isinstance(city, str):  # Check if the city name is a string
        best_match, score = process.extractOne(city, correct_city_names)
        return best_match if score > 70 else city  # Adjust the threshold as␣
 ↪needed
    else:
        return city  # Return the original value if it's not a string

# Calling and applying function and storing corrected values in another column
city_df['corrected_city_name'] = city_df['city_name'].apply(correct_city_name,␣
 ↪args=(correct_city_names,))

print(city_df)
```

```
              city_name corrected_city_name
0         SILVER SPRING       SILVER SPRING
1              BETHESDA            BETHESDA
2             ROCKVILLE           ROCKVILLE
3            GERMANTOWN          GERMANTOWN
4    MONTGOMERY VILLAGE  MONTGOMERY VILLAGE
..                  …                   …
242      SILVER SPRIING       SILVER SPRING
243           CLARKSURG          CLARKSBURG
244            MARYLAND            MARYLAND
245       SILVER SPRIGN       SILVER SPRING
246                  MD                  MD

[247 rows x 2 columns]
```

[20]:
```
#Did some manual research, tallied the locations and pin code to work on the␣
↪city names that FuzzyWuzz could not understand
city_list = ['LA', 'PO', 'GA', 'KE', 'RO', 'TP', 'DISTRICT OF COLUMBIA',␣
↪'WASHINGTON']
city_corr_list = ['LAYTONSVILLE', 'POTOMAC', 'GAITHERSBURG', 'KENSINGTON',␣
↪'ROCKVILLE', 'TAKOMA PARK', 'WASHINGTON DC', 'WASHINGTON']
city_corr_dict = dict(zip(city_list, city_corr_list))

# Apply the mapping
#City names not found in city_list above are replaced by Nan, hence, fill NaN␣
↪values with the original city names
city_df['corrected_city_name'] = city_df['city_name'].map(city_corr_dict).
↪fillna(city_df['corrected_city_name'])
```

[21]:
```
#Create dictionary of the initial and corrected city name
city_dict=dict(zip(city_df.city_name,city_df.corrected_city_name))
df['city'] = df['city'].map(city_dict)  # map it to original dataset

#drop all cities outside montgomery county, maryland and few entires which were␣
↪just numbers
#there were only 126 such entries as of August 26, 2024
df=df[df['city'].isin(correct_city_names)]
df.reset_index(drop=True,inplace=True)

#You can keep them if you want and further correct them
#Used this to see if the remaining cities had any significance
'''remaining = city_df[~city_df['corrected_city_name'].
↪isin(correct_city_names)]['corrected_city_name'].unique()
(df[df['city'].isin(remaining)].groupby('crimename3')['incident_id'].
↪count())*100/df.groupby('crimename3')['incident_id'].count()>5'''
df.shape
```

[21]: (319318, 36)

3.0.3 CONVERT DESIRED COLUMNS TO INTEGER AND FLOAT

[22]:
```
#replace missing values/empty strings with NaN
columns_to_convert=['incident_id','offence_code', 'case_number','victims',␣
↪'zip_code', 'pra','latitude', 'longitude']
df[columns_to_convert].replace('', np.nan, inplace=True)
df[columns_to_convert].replace(' ', np.nan, inplace=True)

# Convert columns to numeric, coercing errors to NaN
df[columns_to_convert] = df[columns_to_convert].apply(pd.to_numeric,␣
↪errors='coerce')
```

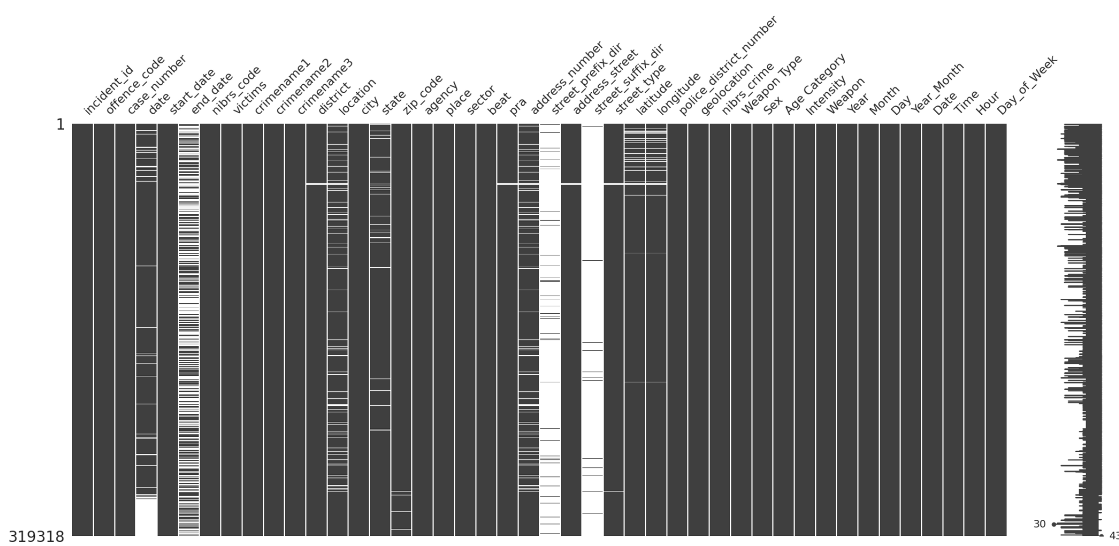
```
#replace 0 in location data with NaN
df.latitude.replace(0, np.nan, inplace=True)
df.longitude.replace(0, np.nan, inplace=True)
```

3.0.4 TIME DATA WRANGLING

[23]:
```
# Convert the 'start_date' column to datetime format
df['start_date'] = pd.to_datetime(df['start_date'], format="%Y-%m-%dT%H:%M:%S.%f", errors='coerce')

# Create the required columns using vectorized operations
df['Year'] = df['start_date'].dt.year
df['Month'] = df['start_date'].dt.month
df['Day'] = df['start_date'].dt.day
df['Year_Month'] = df['start_date'].dt.to_period('M').astype(str)
df['Date'] = df['start_date'].dt.date#strftime("%m/%d/%Y")
df['Time'] = df['start_date'].dt.strftime("%I:%M:%S %p")
df['Hour'] = df['start_date'].dt.hour
df['Day_of_Week'] = df['start_date'].dt.day_name()

# If needed, you can drop or handle rows where the 'start_date' could not be parsed (NaT values)
#df.dropna(subset=['start_date'], inplace=True)
```

3.0.5 DEALING WITH MISSING VALUES AND UNWANTED COLUMNS

[24]:
```
#Visual Representation of missing values
msno.matrix(df)
plt.show()
```



[25]:
```
#Did backgroud search about each columns, decided which ones to drop
df.drop(columns=['end_date','date','location','state',␣
↪'address_number','street_prefix_dir', 'street_suffix_dir',␣
↪'geolocation','police_district_number','zip_code'],inplace=True)

#Handling certain minfits/wrong data entries
df.sector.replace('w', np.nan, inplace=True)
df.beat.replace('own', np.nan, inplace=True)
```

[26]:
```
#Create a function to impute categorical values
#Can also use KNN Imputation, I might use it later while updating the project
def col_imputer(col1, col2, b=True):
  if b:
    miss_index= df[df[col1].isin(df[col1][df[col2].isna()].unique())].
↪groupby(df[col1])[col2].agg(pd.Series.mode).index
    miss_value= df[df[col1].isin(df[col1][df[col2].isna()].unique())].
↪groupby(df[col1])[col2].agg(pd.Series.mode).values
    miss_dict=dict(zip(miss_index,miss_value))
    df[col2][df[col2].isna()] = df[col1][df[col2].isna()].map(miss_dict)
  else:
    miss_index= df[df[col1].isin(df[col1][df[col2].isna()].unique())].
↪groupby(df[col1])[col2].agg(pd.Series.median).index
    miss_value= df[df[col1].isin(df[col1][df[col2].isna()].unique())].
↪groupby(df[col1])[col2].agg(pd.Series.median).values
    miss_dict=dict(zip(miss_index,miss_value))
    df[col2][df[col2].isna()] = df[col1][df[col2].isna()].map(miss_dict)

col_imputer('city', 'district')
col_imputer('city', 'sector')
col_imputer('sector', 'beat')
col_imputer('beat', 'pra')
col_imputer('pra', 'address_street')
col_imputer('address_street', 'street_type')

#Need to come up with a ML model for these imputations in the future
col_imputer('address_street', 'latitude', False)
col_imputer('address_street', 'longitude', False)

col_imputer('pra', 'latitude', False)
col_imputer('pra', 'longitude', False)
```

[27]:
```
#Might want this code in the future
'''zip_miss_city= df[df.city.isin(df.city[df.zip_code.isna()].unique())].
↪groupby(df.city)['zip_code'].agg(pd.Series.mode).index
```

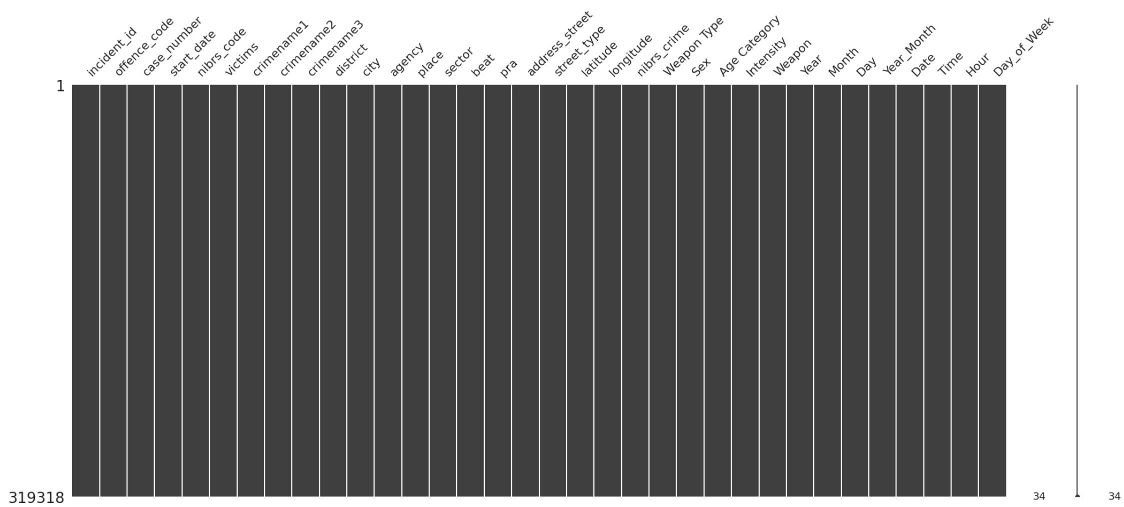
```
zip_miss_zip= df[df.city.isin(df.city[df.zip_code.isna()].unique())].groupby(df.
↪city)['zip_code'].agg(pd.Series.mode).values
zip_miss_dict=dict(zip(zip_miss_city,zip_miss_zip))
df.zip_code[df.zip_code.isna()] = df.city[df.zip_code.isna()].
↪map(zip_miss_dict)'''
```

[27]:

```
"zip_miss_city= df[df.city.isin(df.city[df.zip_code.isna()].unique())].groupby(d
f.city)['zip_code'].agg(pd.Series.mode).index\nzip_miss_zip= df[df.city.isin(df.
city[df.zip_code.isna()].unique())].groupby(df.city)['zip_code'].agg(pd.Series.m
ode).values\nzip_miss_dict=dict(zip(zip_miss_city,zip_miss_zip))\ndf.zip_code[df
.zip_code.isna()] = df.city[df.zip_code.isna()].map(zip_miss_dict)"
```

[28]:

```
#Visual Representation of missing values to make sure there are no missing␣
↪values of unwanted coolumns
msno.matrix(df)
plt.show()
```



4 FINALIZING DATASET

[136]:

```
#Finalizing the column names, everything to lower case
column=['id', 'offence_code', 'case_number', 'start_date','nibrs_code',␣
↪'victims', 'crime_category', 'crime_type', 'crime_detail',
       'district', 'city', 'agency', 'place', 'sector', 'beat',␣
↪'pra','address_street', 'street_type', 'latitude', 'longitude',␣
↪'nibrs_crime',
       'weapon_type', 'sex', 'age', 'intensity', 'weapon', 'year','month',␣
↪'day', 'year_month', 'date', 'time', 'hour', 'day_of_week']
#df.set_axis(column, axis=1)
df.columns=column
```

```
df.sort_values(by='start_date',inplace=True)
df.reset_index(drop=True,inplace=True)
df.head(3)
```

[136]:

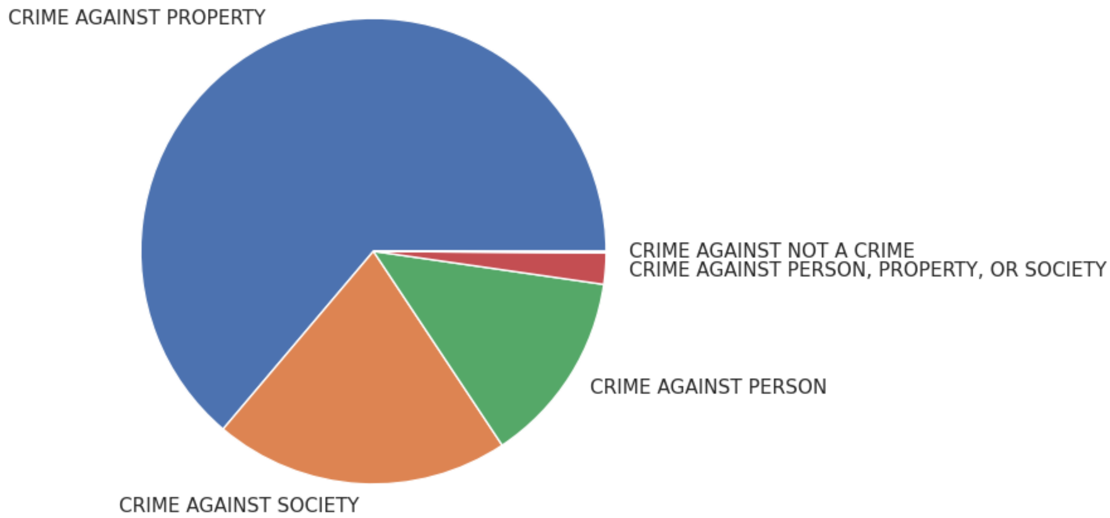
```
          id  offence_code  case_number start_date nibrs_code  victims  \
0  201222351          1199    180049421 2016-07-01        11D        1
1  201100330          2589     16048345 2016-07-01        250        1
2  201130906          2308    170503555 2016-07-01        23D        1

           crime_category   crime_type          crime_detail  \
0    CRIME AGAINST PERSON  SEX OFFENCE              FONDLING
1  CRIME AGAINST PROPERTY        FRAUD      DESCRIBE OFFENSE
2  CRIME AGAINST PROPERTY      LARCENY  THEFT FROM BUILDING

               district  … intensity weapon  year month day  year_month  \
0  MONTGOMERY VILLAGE  …  No Entry  False  2016     7   1     2016-07
1           ROCKVILLE  …  No Entry  False  2016     7   1     2016-07
2       SILVER SPRING  …  No Entry  False  2016     7   1     2016-07

         date         time  hour  day_of_week
0  2016-07-01  12:00:00 AM     0       Friday
1  2016-07-01  12:00:00 AM     0       Friday
2  2016-07-01  12:00:00 AM     0       Friday

[3 rows x 34 columns]
```

5 EXPLORATORY DATA ANALYSIS

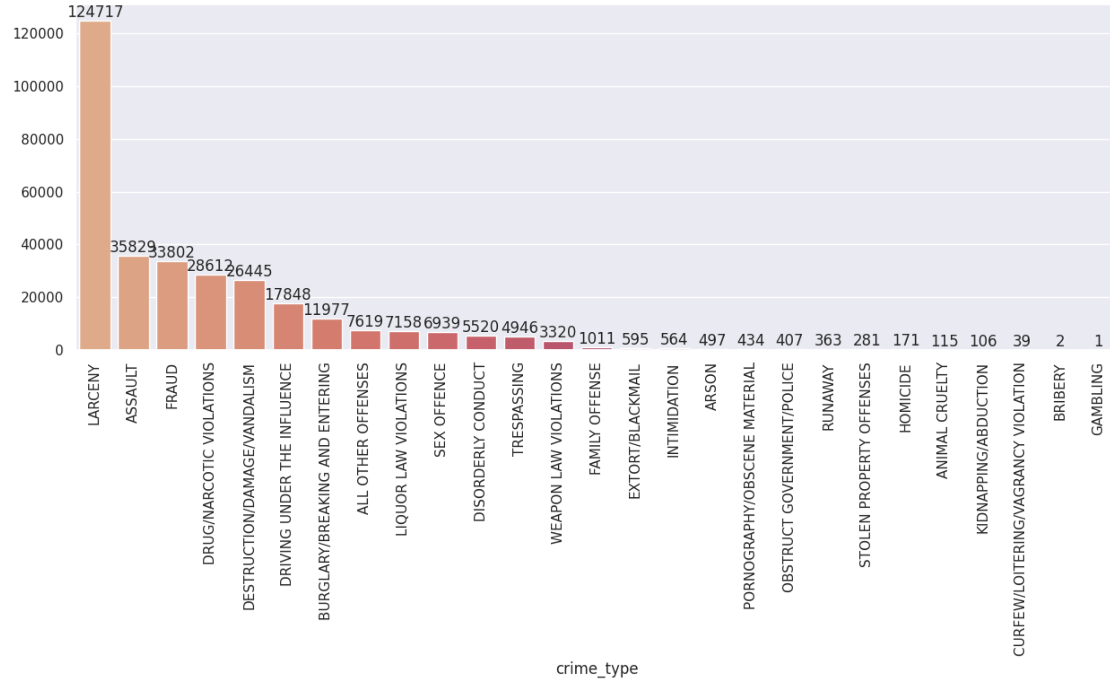
5.0.1 ANALYSIS BASED ON CRIME CATEGORIES

[134]:

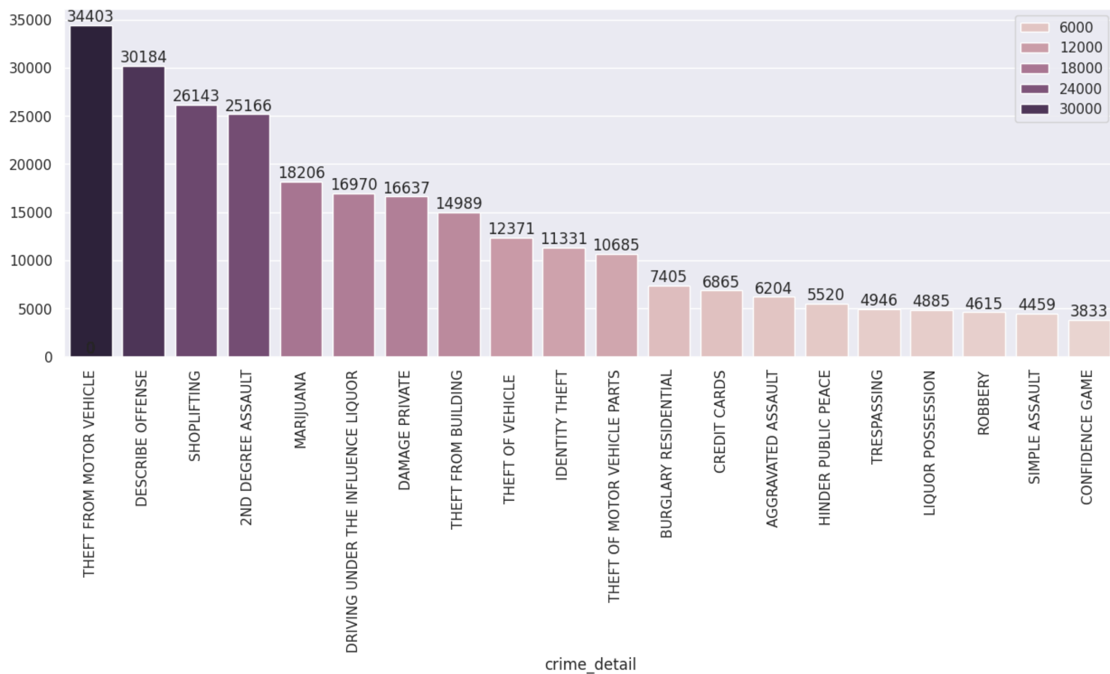
```
#Using Matplotlib for plotting pie chart for crime_category(crime_name1)
x=df.crime_category.value_counts().index
y=df.crime_category.value_counts().values
plt.figure(figsize=(8,6))
plt.pie(y,labels=x)
plt.xticks(rotation=90)
plt.show()
```



[131]:
```
#Using Seaborn for plotting bar chart for crime_type(crime_name2)
x=df.crime_type.value_counts().index
y=df.crime_type.value_counts().values

plt.figure(figsize=(15,5))
ax = sns.barplot(x=x, y=y, palette = "flare")

# Iterate over the patches (bars) in the Axes
for bar in ax.patches:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, yval, int(yval), ha='center',␣
 ↪va='bottom')


plt.xticks(rotation=90)
plt.show()
```

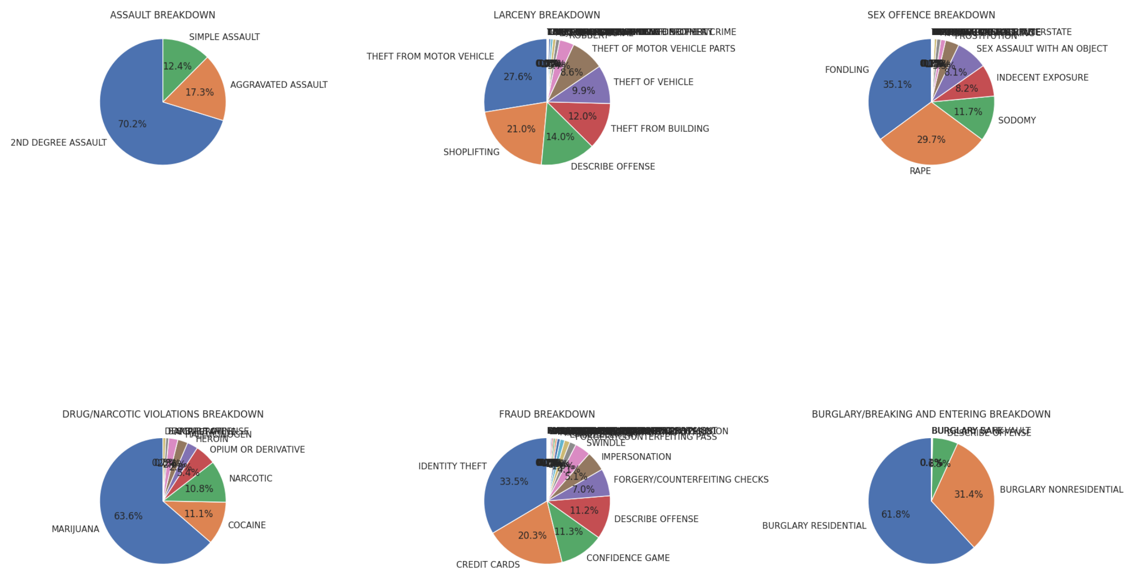


[132]:
```
#Using Seaborn for plotting bar chart for crime_details(crime_name3)
x=df.crime_detail.value_counts().head(20).index
y=df.crime_detail.value_counts().head(20).values

plt.figure(figsize=(15,5))
ax = sns.barplot(x=x, y=y,hue=y)

# Iterate over the patches (bars) in the Axes
for bar in ax.patches:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, yval, int(yval), ha='center',␣
 ↪va='bottom')


plt.xticks(rotation=90)
plt.show()
```



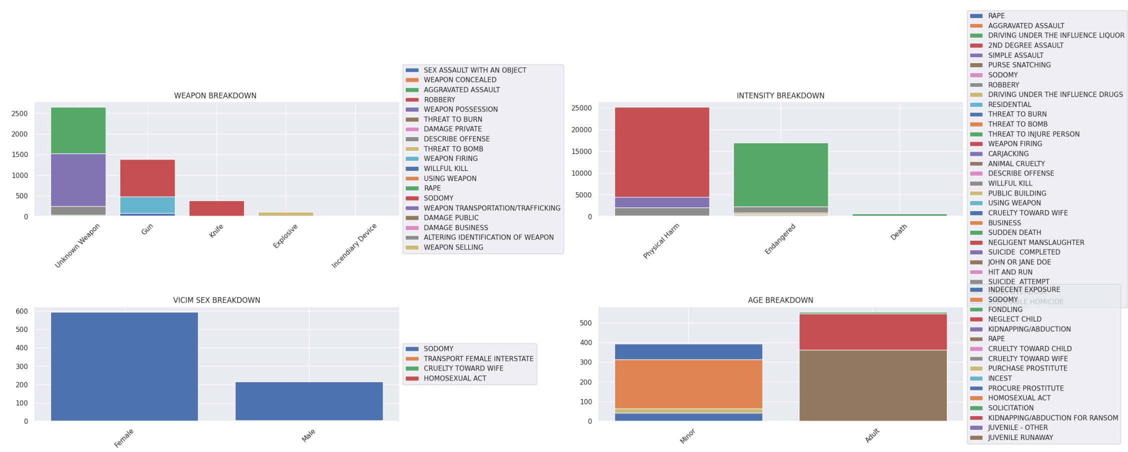
[129]:

```
# Comparing the major crime_types(crime_name2) commited using matplotlib
crime_types = ['ASSAULT', 'LARCENY', 'SEX OFFENCE', 'DRUG/NARCOTIC VIOLATIONS', 'FRAUD', 'BURGLARY/BREAKING AND ENTERING']


fig, axes = plt.subplots(2, 3, figsize=(20, 15))  # 2 rows, 3 columns of subplots
# Flatten the axes array for easy iteration
axes = axes.flatten()

# Plot each pie chart in the corresponding subplot
for i, crime_type in enumerate(crime_types):
    x = df.crime_detail[df.crime_type == crime_type].value_counts().index
    y = df.crime_detail[df.crime_type == crime_type].value_counts().values

    # Plot the pie chart with percentage
    axes[i].pie(y, labels=x, autopct='%1.1f%%', startangle=90)
    axes[i].set_title(f'{crime_type} BREAKDOWN')

# Adjust layout to prevent overlap
plt.tight_layout()
plt.show()
```

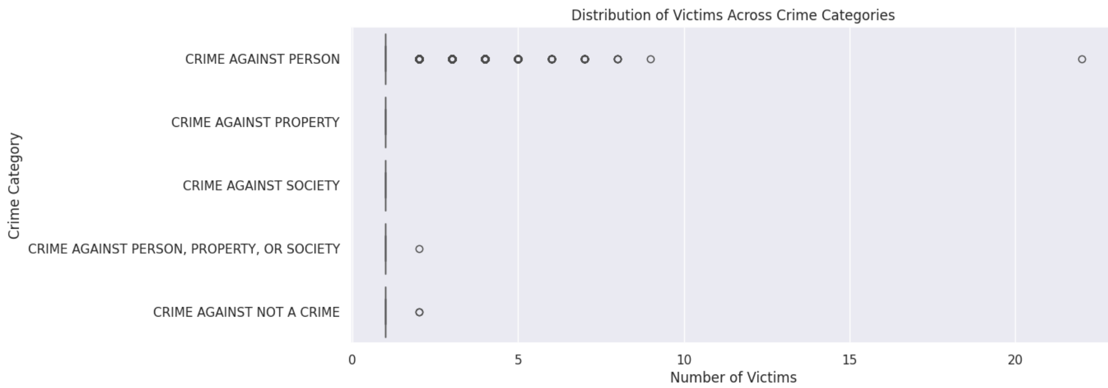


[127]:
```python
#Creating 4 different stacked bar plot for:-
# 1. weapon_type,
#2. weapon used or not,
#3. intensity and
#4. victim age
#the original  data does not have details for these columns but this is my
↪attempt to paint a vague picture
fig, axes = plt.subplots(2,2, figsize=(26,10))#2 rows 2 columns
axes = axes.flatten()

#creating a fucntion to be called for each subplot
def subplotting(df_t,col,type_count,ax,title):
  for i,types in enumerate(type_count):
    x=df_t[col][df_t.crime_detail==types].value_counts().index
    y=df_t[col][df_t.crime_detail==types].value_counts().values
    axes[ax].bar(x,y)
  axes[ax].set_title(title)
  axes[ax].legend(type_count,loc='center left', bbox_to_anchor=(1, 0.5))
  axes[ax].set_xticklabels(df_t[col].value_counts().index,rotation=45)

#calling function for each subplot
df_w=df[df.weapon]
type_count=df_w.crime_detail.unique()
subplotting(df_w,'weapon_type',type_count,0,'WEAPON BREAKDOWN ')

df_i=df[df.intensity != 'No Entry']
type_count=df_i.crime_detail.unique()
subplotting(df_i,'intensity',type_count,1,'INTENSITY BREAKDOWN')
```

```
df_s=df[df.sex != 'No Entry']
type_count=df_s.crime_detail.unique()
subplotting(df_s,'sex',type_count,2,'VICIM SEX BREAKDOWN')

df_a=df[df.age != 'No Entry']
type_count=df_a.crime_detail.unique()
subplotting(df_a,'age',type_count,3,'AGE BREAKDOWN')

# Adjust layout to prevent overlap
plt.tight_layout()
plt.show()

#This can be done more easily by using df and plot together as show further␣
↪down in the code
```



[126]:
```
#creating a box plot for number of victims and crime_category

plt.figure(figsize=(12, 5))
sns.boxplot(x='victims', y='crime_category', data=df)
plt.title('Distribution of Victims Across Crime Categories')
plt.xlabel('Number of Victims')
plt.ylabel('Crime Category')
plt.show()
```



[125]:

```
#Using seaborn to map number of victims
x=df.victims.value_counts().index
y=df.victims.value_counts().values

plt.figure(figsize=(10, 4))
ax=sns.barplot(x=x,y=y, palette='Spectral')
# Iterate over the patches (bars) in the Axes
for bar in ax.patches:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, yval, int(yval), ha='center',␣
 ↪va='bottom')

# Add labels and title
plt.xlabel("Number of Victims")
plt.ylabel("Number of Incidents")
plt.title("Number of Incidents by Number of Victims")

plt.show()
```

5.0.2 ANALYSIS BAES ON DATE-TIME DATA

[124]:
```
#crreating interactive stacked bar plot for number and type crimes committed␣
↪per hour using PLOTLY

#I AIM TO CONVERT ALL GRAPHS TO PLOTLY BECAUSE OF ITS INTERACTIVE CAPABILITIES

fig=df.crime_type.groupby(df.hour).value_counts().unstack().plot(kind='bar', ␣
↪barmode='stack')
fig.update_layout(width=1650, height=700)
fig.show()
```

[42]:
```
#using ploty's graph_objects to plot heatmap histogram for number of crimes␣
↪committed each hour each day of the week
x = df.day_of_week
y = df.hour
fig = go.Figure(go.Histogram2d(x=x,y=y))
fig.show()
```

Output hidden; open in https://colab.research.google.com to view.

[43]:
```
#Using Plotly to create donut graph of the percentage share of crimes per day␣
↪of the week
labels = df.day_of_week.unique()
values=[]
for each in labels:
    values.append(len(df[df.day_of_week==each]))

# Use `hole` to create a donut-like pie chart
fig = go.Figure(data=[go.Pie(labels=labels, values=values, hole=.3)])
fig.show()
```

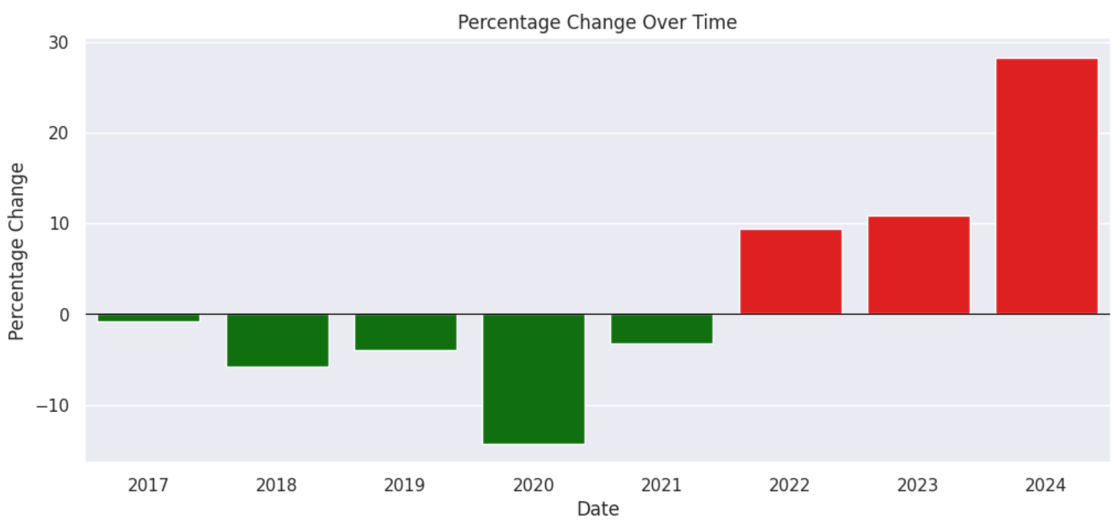
[122]:
```
#plotting bar chart for crimes committed each month using PLOTLY

fig=df.month.value_counts().sort_index().plot(x=df.month.value_counts().
↪sort_index().index,y=df.month.value_counts().sort_index().
↪values,kind='hist',nbins=30)
fig.update_layout(width=1500, height=700)
fig.show()
```

[37]:
```
#plotting line chart crimes committed each year
plt.figure(figsize=(15,6))

#create function to be called for each year
```

```
def year_plot(df,year):
  data = {'Month': list(df['month'][df.year==year].value_counts().index),
          'Count': list(df['month'][df.year==year].value_counts().values)}
  dff = pd.DataFrame(data)
  dff.sort_values(by='Month',inplace = True)

  x = dff['Month']
  y = dff['Count']
  plt.plot(x,y,label=year)
  plt.legend()


year_plot(df,2017)
year_plot(df,2018)
year_plot(df,2019)
year_plot(df,2020)
year_plot(df,2021)
year_plot(df,2022)
year_plot(df,2023)
plt.show()
```

[121]:
```
#plotting percentage change each year compared to previous year

#creating dataframe based on crimes count per year
data_index = df.year.value_counts().index
data_value = df.year.value_counts().values
change_df = pd.DataFrame({'year':data_index,'change': data_value})
change_df.sort_values(by='year', inplace=True)
change_df.reset_index(drop=True,inplace=True)
```

```
#adjusting crime count for years where all 12 months of crime data was not␣
↪recorded/entered
change_df.at[0, 'change'] = change_df.at[0, 'change'] * 2
change_df.at[8, 'change'] = change_df.at[8, 'change'] *12/8
change_df.change = change_df.change.pct_change()* 100
change_df.drop(change_df.index[0], inplace=True)
change_df.reset_index(drop=True,inplace=True)

plt.figure(figsize=(12, 5))
sns.barplot(x='year', y='change', data=change_df, palette=['green' if x < 0␣
↪else 'red' for x in change_df['change']])

# Adding a horizontal line at zero
plt.axhline(0, color='black', linewidth=0.8)

# Adding titles and labels
plt.title('Percentage Change Over Time')
plt.xlabel('Date')
plt.ylabel('Percentage Change')

plt.show()
```



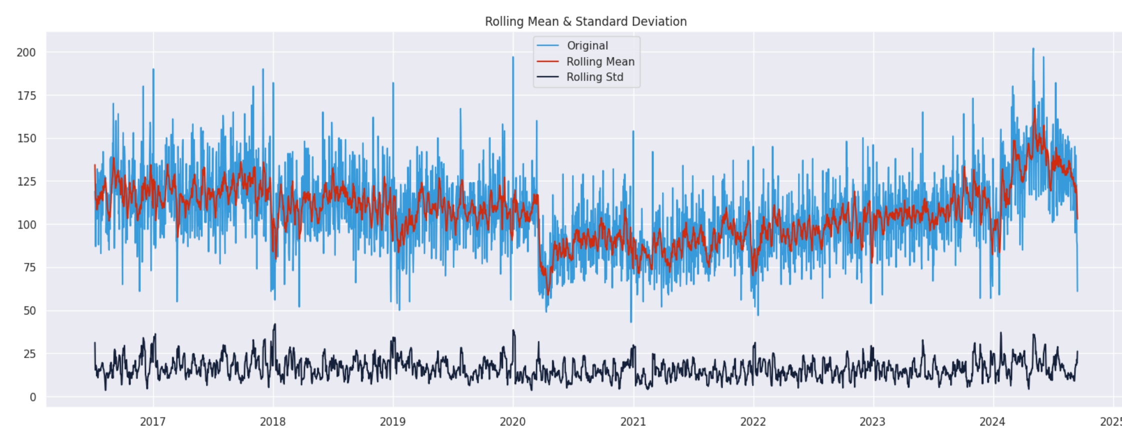
5.0.3 TIME SERIES PREDICTION USING ARIMA

[107]:
```
#copying data for time_series prediction
df_try=df.copy()
#df_try.drop(df_try.loc[df_try['year'] == 2016].index, inplace=True)
```

[108]:
```
#creating dataset with number of crimes per day for calculation of rolling mean␣
↪and prediction
data_ml = {'Date': list(df_try['date'].value_counts().index),
        'Count': list(df_try['date'].value_counts())}
df_ml = pd.DataFrame(data_ml)
df_ml.dropna(inplace=True)
df_ml['Date'] = pd.to_datetime(df_ml['Date'])
df_ml.sort_values(by='Date',inplace = True)
df_ml.reset_index(drop=True)
df_ml.drop(df_ml.index[-1],inplace=True)
df_ml=df_ml.set_axis(df_ml['Date'], axis=0)
df_ml.drop(columns=['Date'],inplace=True)
```

[109]:
```
#splitting dataset into train and test dataset
df_ml_train=df_ml.iloc[:-15,:]
df_ml_test =df_ml.iloc[-15:-1,:]
```

[110]:
```
#Determining rolling statistics
df_ml["rolling_avg"] = df_ml["Count"].rolling(window=7).mean() #window =7 means␣
↪7 day rolling average, aka one week rolling average
df_ml["rolling_std"] = df_ml["Count"].rolling(window=7).std()
df_ml.dropna(inplace=True)

#Plotting rolling statistics
plt.figure(figsize=(20,7))
plt.plot(df_ml["Count"], color='#379BDB', label='Original')
plt.plot(df_ml["rolling_avg"], color='#D22A0D', label='Rolling Mean')
plt.plot(df_ml["rolling_std"], color='#142039', label='Rolling Std')
plt.legend(loc='best')
plt.title('Rolling Mean & Standard Deviation')
plt.show(block=False)
```

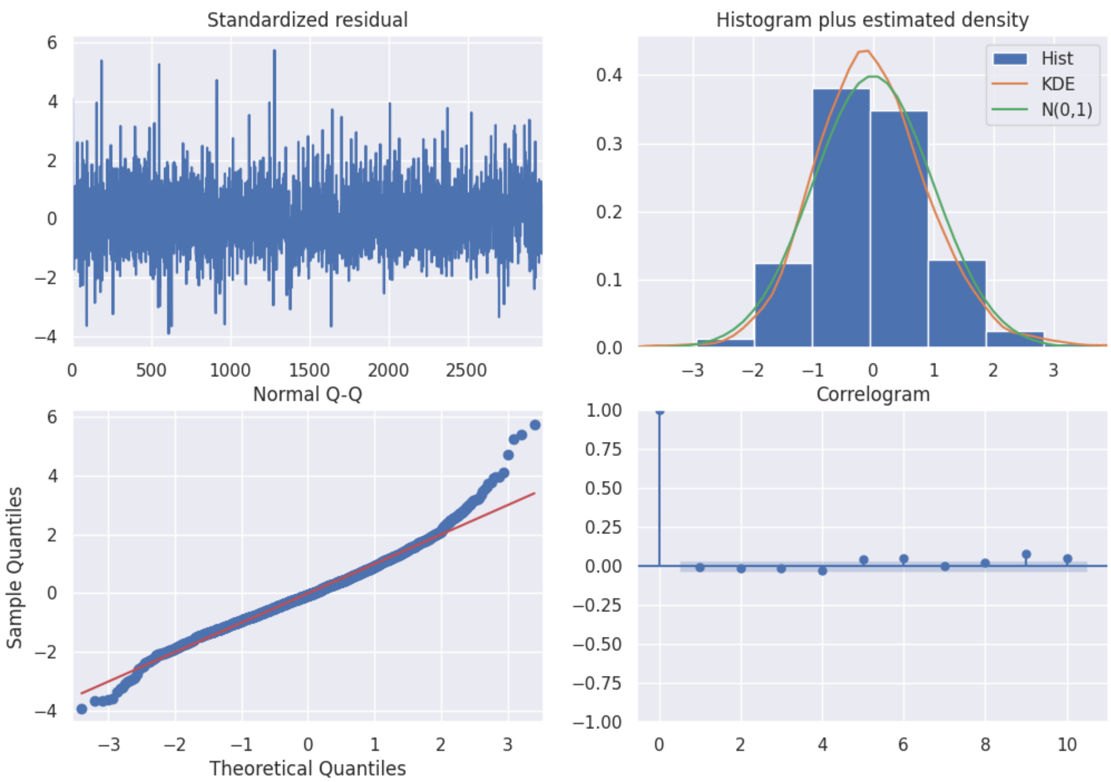


[111]:
```
#Augmented Dickey-Fuller test:
#a statistical test used to determine whether a time series is stationary or␣
↪not.
#required for proforming time_series analysis and prediction
print('Results of Dickey Fuller Test:')
dftest = adfuller(df_ml_train['Count'], autolag='AIC')

dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','#Lags␣
↪Used','Number of Observations Used'])
for key,value in dftest[4].items():
    dfoutput['Critical Value (%s)'%key] = value

print(dfoutput)
```

```
Results of Dickey Fuller Test:
Test Statistic                   -2.986409
p-value                           0.036183
#Lags Used                       27.000000
Number of Observations Used    2955.000000
Critical Value (1%)              -3.432565
Critical Value (5%)              -2.862519
Critical Value (10%)             -2.567291
dtype: float64
```

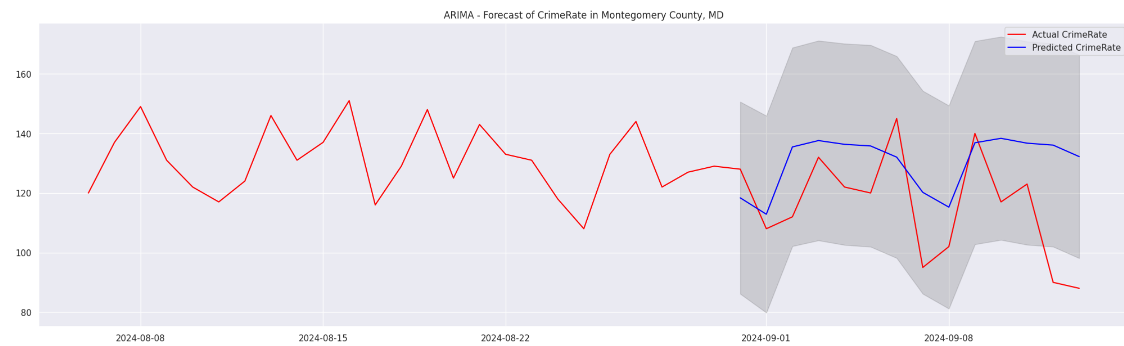
[112]:
```
#Standard ARIMA Model
#the below values have been tentatively alotted after few iterations
#need to finetune the model more
ARIMA_model = pm.auto_arima(list(df_ml_train['Count']),
                        start_p=20,
                        start_q=20,
                        test='adf', # use adftest to find optimal 'd'
                        max_p=35, max_q=35, # maximum p and q
                        m=7, # frequency of series (if m==1, seasonal is set to␣
↪FALSE automatically)
                        d=None,# let model determine 'd'
                        seasonal=True, # No Seasonality for standard ARIMA
                        trace=False, #logs
                        error_action='warn', #shows errors ('ignore' silences␣
↪these)
                        suppress_warnings=True,
                        stepwise=True)
```

[117]:
```
#statistical diagnostics from the trained ARIMA model
ARIMA_model.plot_diagnostics(figsize=(12,8))
plt.show()
```




```
[114]: #TIME-SERIES PREDICTION
def forecast(ARIMA_model, periods=14):
    # Forecast
    n_periods = periods
    fitted, confint = ARIMA_model.predict(n_periods=n_periods,␣
 ↪return_conf_int=True)
    index_of_fc = pd.date_range(df_ml_train.index[-1] + pd.DateOffset(days=1),␣
 ↪periods = n_periods, freq='D')

    # make series for plotting purpose
    fitted_series = pd.Series(fitted, index=index_of_fc)
    lower_series = pd.Series(confint[:, 0], index=index_of_fc)
    upper_series = pd.Series(confint[:, 1], index=index_of_fc)


    # Plot
    plt.figure(figsize=(25,7))
    plt.plot(df_ml.iloc[-40:-1,0], color='red',label='Actual CrimeRate')
    plt.plot(fitted_series, color='blue', label = 'Predicted CrimeRate')
    plt.fill_between(lower_series.index,
                    lower_series,
                    upper_series,
```

```
                    color='k', alpha=.15)

    plt.title("ARIMA - Forecast of CrimeRate in Montegomery County, MD")
    plt.legend()
    plt.show()
    return fitted_series


z=forecast(ARIMA_model)
```



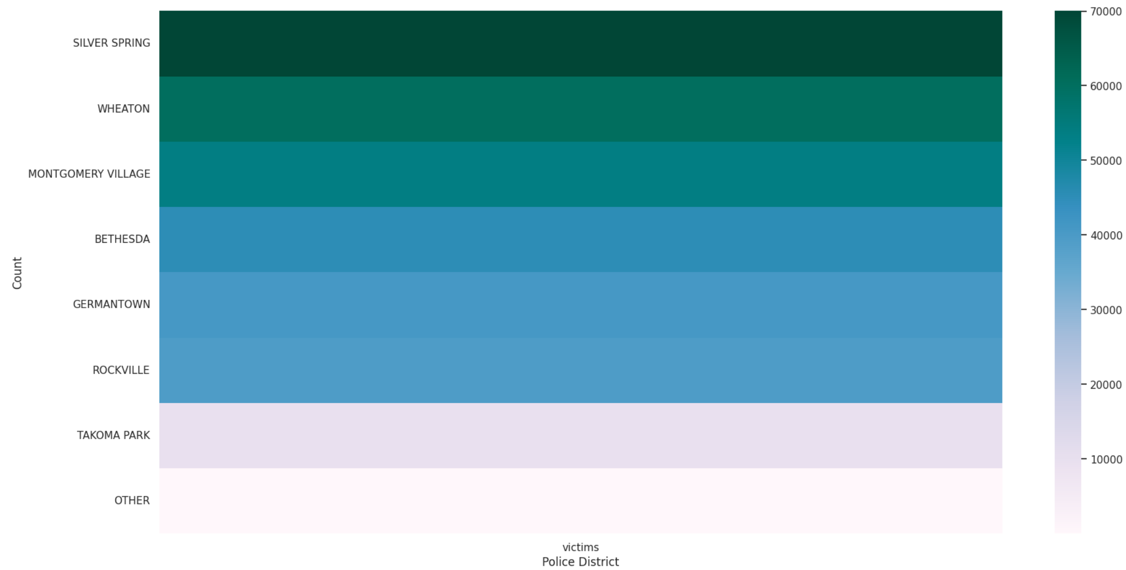
[115]:
```
#Calculating accuracy if the model using MAPE( Mean Absolute Percentage Error)
#It is decent since it is below 20 but need to bring it down, under 10
def MAPE(Y_actual,Y_Predicted):
    mape = np.mean(np.abs((Y_actual - Y_Predicted)/Y_actual))*100
    return mape
MAPE(df_ml.iloc[-15:-1,0],z)
```

[115]: 17.408015238646698

5.0.4 ANALYSIS BASED ON CITY AND OTHER LOCATION BASED DATA

[118]:
```
#plotting a pivot table from Police District using Seaborn
pivot_table = df.pivot_table(values='victims', index='district',␣
 ↪aggfunc='count').sort_values(by='victims', ascending=False)

plt.figure(figsize=(20, 10))
sns.heatmap(pivot_table, cmap='PuBuGn')
plt.xlabel("Police District")
plt.ylabel("Count")
```

[118]: Text(216.25, 0.5, 'Count')



[119]:
```
#Plotting bar graph using number of crimes per city using Plotly
fig=df.city.value_counts().plot(kind='bar')
fig.update_layout(width=2100, height=750)
fig.show()
```

[120]:
```
#Creating a Stacked bar plot for the number of crimes per district with stacks␣
↪showing the city

# Calculate total number of crimes per city per district
crime_totals = df.groupby('district')['city'].count()

# Sort distrcts by the total number of crimes
sorted_cities = crime_totals.sort_values(ascending=False).index

fig = df.city.groupby(df.district).value_counts().unstack().loc[sorted_cities].
↪plot(kind='bar', barmode='stack')
fig.update_layout(width=1800, height=800)
fig.show()
```

5.0.5 GEO-SPACIAL ANALYSIS

[ ]:
```
#using PLOTLY'S EXPRESS to plot interactive scatter plot on OPEN-STREET-MAP␣
↪based on location of crime
#colour coded on police district
fig = px.scatter_mapbox(df[(df.latitude!=0) & (df.longitude !=0)],␣
↪lat="latitude", lon="longitude", hover_name="incident_id",␣
↪hover_data=["Date","Hour",'crimename3'],
                        color='district', zoom=10, height=700,size_max=0.000001)
```

```
fig.update_layout(mapbox_style="open-street-map")
fig.update_layout(margin={"r":0,"t":0,"l":0,"b":0})
fig.show()
```

Output hidden; open in https://colab.research.google.com to view.

```
[ ]: #Plotting a interactive heatmap depicting death based on location using FOLIUM
vand=df[['latitude','longitude']][df['Intensity']=='Death']
vand.latitude.fillna(0, inplace = True)
vand.longitude.fillna(0, inplace = True)

CountyMap=folium.Map(location=[39.06,-77.09],zoom_start=10)
HeatMap(data=vand, radius=16).add_to(CountyMap)

CountyMap
```

```
[ ]: <folium.folium.Map at 0x7970e657ead0>
```

6 WILL DO MORE EDA ON LOCATION-BASED DATA AND MORE GEOSPATIAL ANALYSIS SOON

7 COLLECTING MORE KIND OF DATASETS FOR CORRELATIONS AND CAUSAL ANALYSIS