Copy of CRIME_DATA_ANALYSIS.ipynb

File Edit View Insert Runtime Tools Help All changes saved

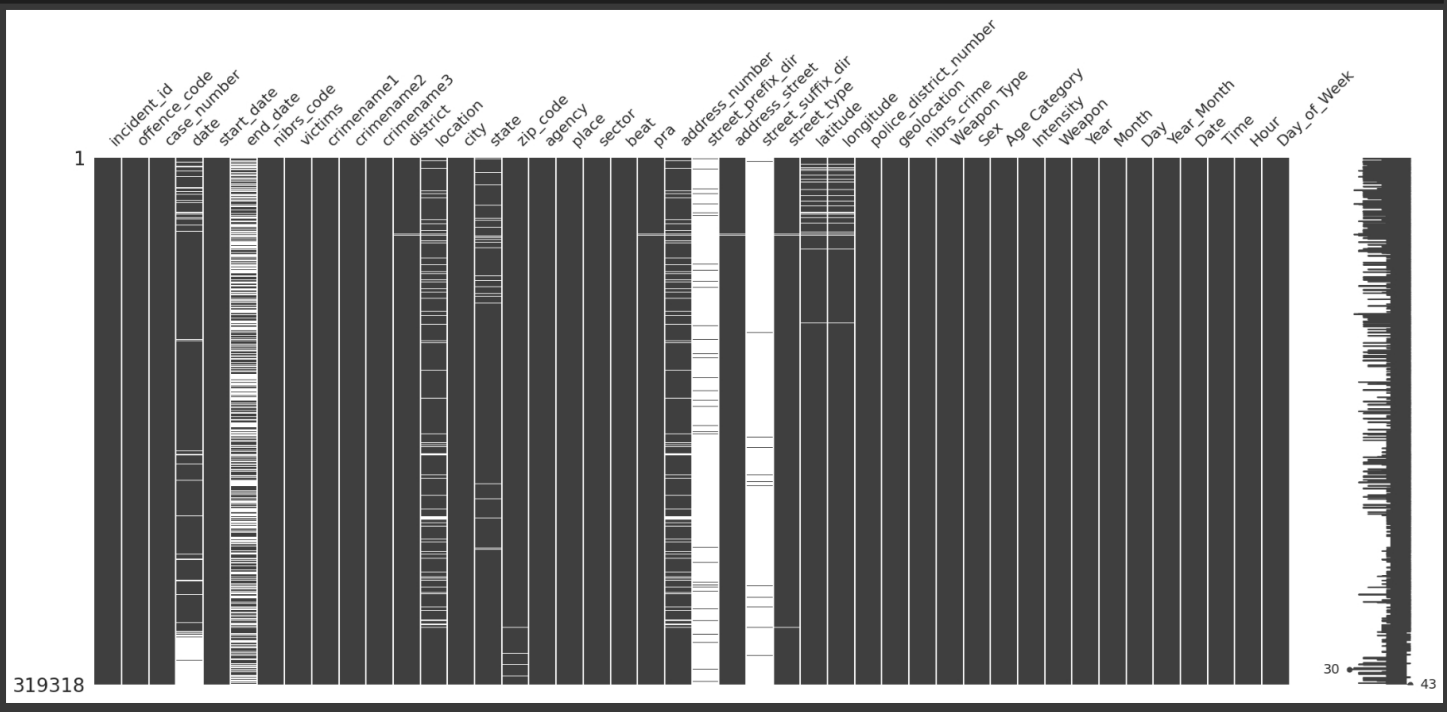
Comment Share

+ Code + Text

Connect GPU High-RAM Gemini

DEALING WITH MISSING VALUES AND UNWANTED COLUMNS

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



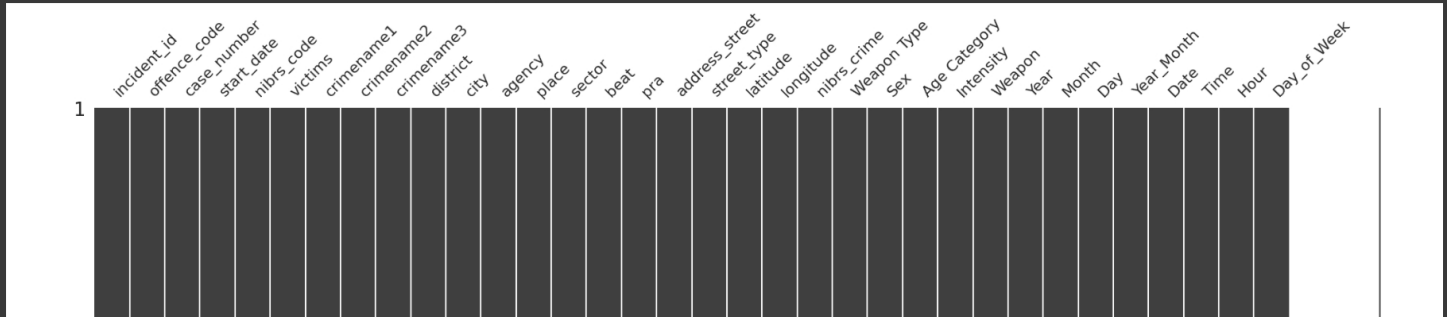

```
[ ] #Did backgroud search about each columns, decided which ones to drop
    df.drop(columns=['end_date','date','location','state', 'address_number','street_prefix_dir', 'street_suffix_dir', 'geolocation','police_district_number','zip_code'],inplace=True)

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

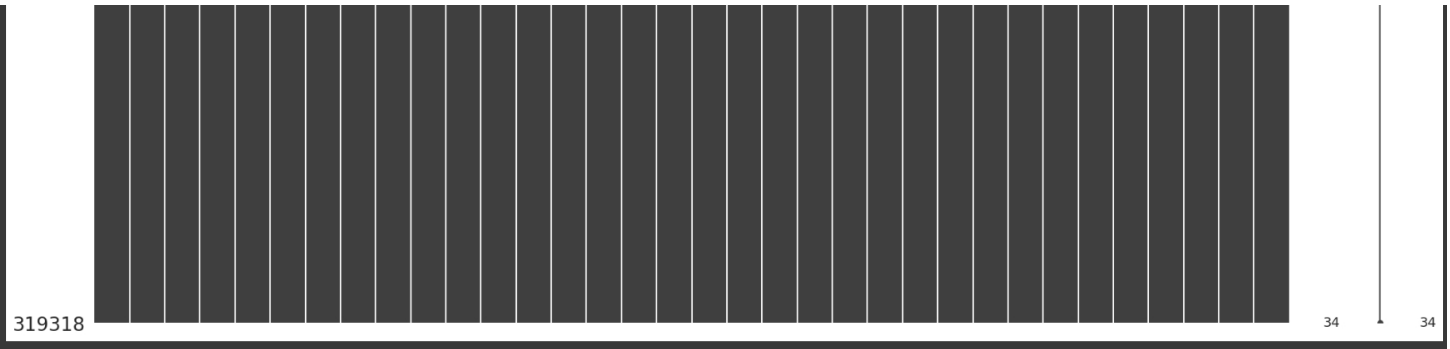
```
[ ] #Create a function to impute categorical values
    #Can also use KNN Imputation, I might use it later while updating the project
    def col_imputer(col1, col2, b=True):
      if b:
        miss_index= df[df[col1].isin(df[col1][df[col2].isna()].unique())].groupby(df[col1])[col2].agg(pd.Series.mode).index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

```
[ ] #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)'''
```

```
'zip_miss_city= df[df.city.isin(df.city[df.zip_code.isna()].unique())].groupby(df.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.mode).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)'
```

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





FINALIZING DATASET

```
[ ] #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)
```

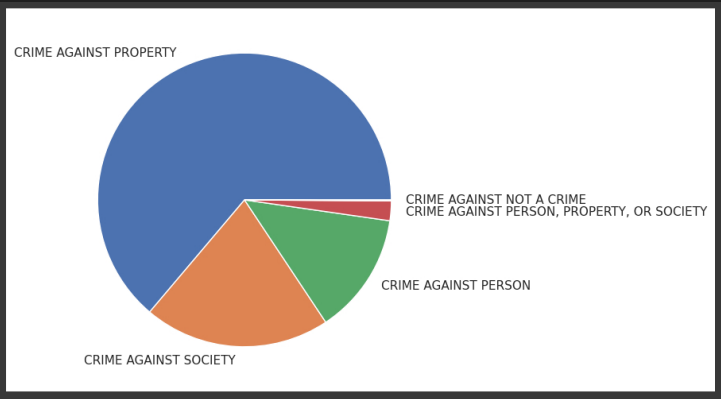
	id	offence_code	case_number	start_date	nibrs_code	victims	crime_category	crime_type	crime_detail	district	...	intensity	weapon	year	month	day	year_month	date	time	hour	day_of_week
0	201222351	1199	180049421	2016-07-01	11D	1	CRIME AGAINST PERSON	SEX OFFENCE	FONDLING	MONTGOMERY VILLAGE	...	No Entry	False	2016	7	1	2016-07	2016-07-01	12:00:00 AM	0	Friday
1	201130906	2308	170503555	2016-07-01	23D	1	CRIME AGAINST PROPERTY	LARCENY	THEFT FROM BUILDING	SILVER SPRING	...	No Entry	False	2016	7	1	2016-07	2016-07-01	12:00:00 AM	0	Friday
2	201089700	2308	16036427	2016-07-01	23D	1	CRIME AGAINST PROPERTY	LARCENY	THEFT FROM BUILDING	BETHESDA	...	No Entry	False	2016	7	1	2016-07	2016-07-01	12:00:00 AM	0	Friday

3 rows × 34 columns

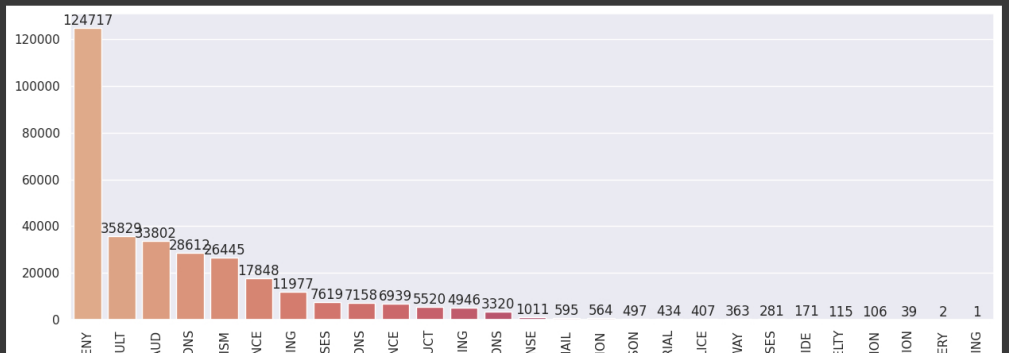
EXPLORATORY DATA ANALYSIS

ANALYSIS BASED ON CRIME CATEGORIES

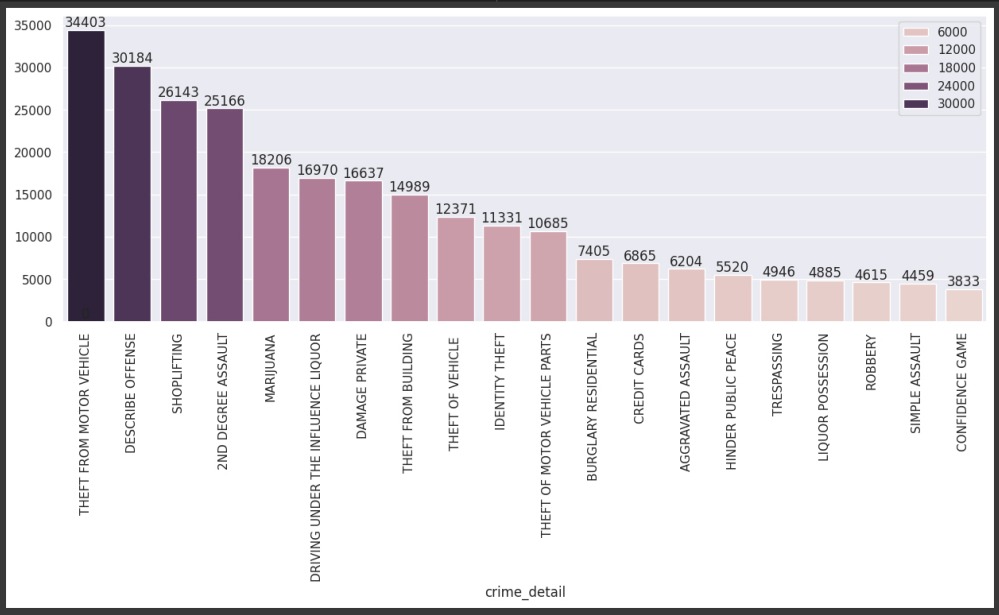
```
[ ] #Using Matplotlib for plotting pie chart for crime_category(crime_name1)
    x=df.crime_category.value_counts().index
    y=df.crime_category.value_counts().values
    plt.figure(figsize=(8,6))
    plt.pie(y,labels=x)
    plt.xticks(rotation=90)
    plt.show()
```



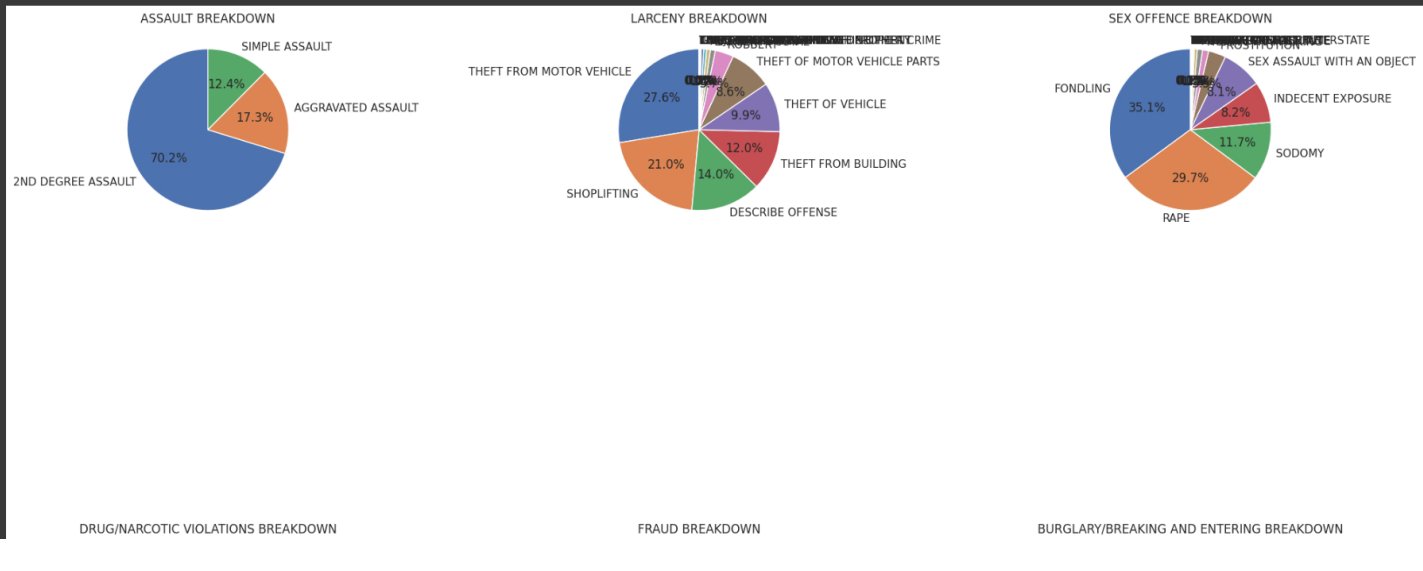

```
[ ] #Using Seaborn for plotting bar chart for crime_type(crime_name2)
    x=df.crime_type.value_counts().index
    y=df.crime_type.value_counts().values

    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()
```

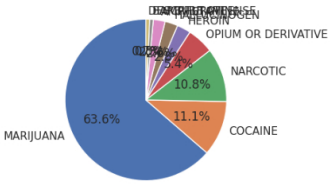


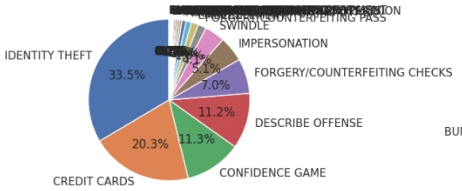

```
[ ] #Using Seaborn for plotting bar chart for crime_details(crime_name3)
    x=df.crime_detail.value_counts().head(20).index
    y=df.crime_detail.value_counts().head(20).values

    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()
```

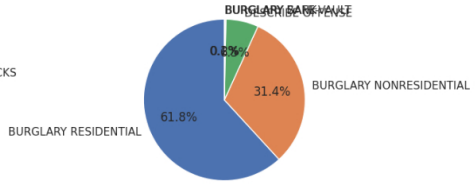



```
[ ] # Comparing the major crime_types(crime_name2) commited using matplotlib
    crime_types = ['ASSAULT', 'LARCENY', 'SEX OFFENCE', 'DRUG/NARCOTIC VIOLATIONS', 'FRAUD', 'BURGLARY/BREAKING AND ENTERING']


    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()
```



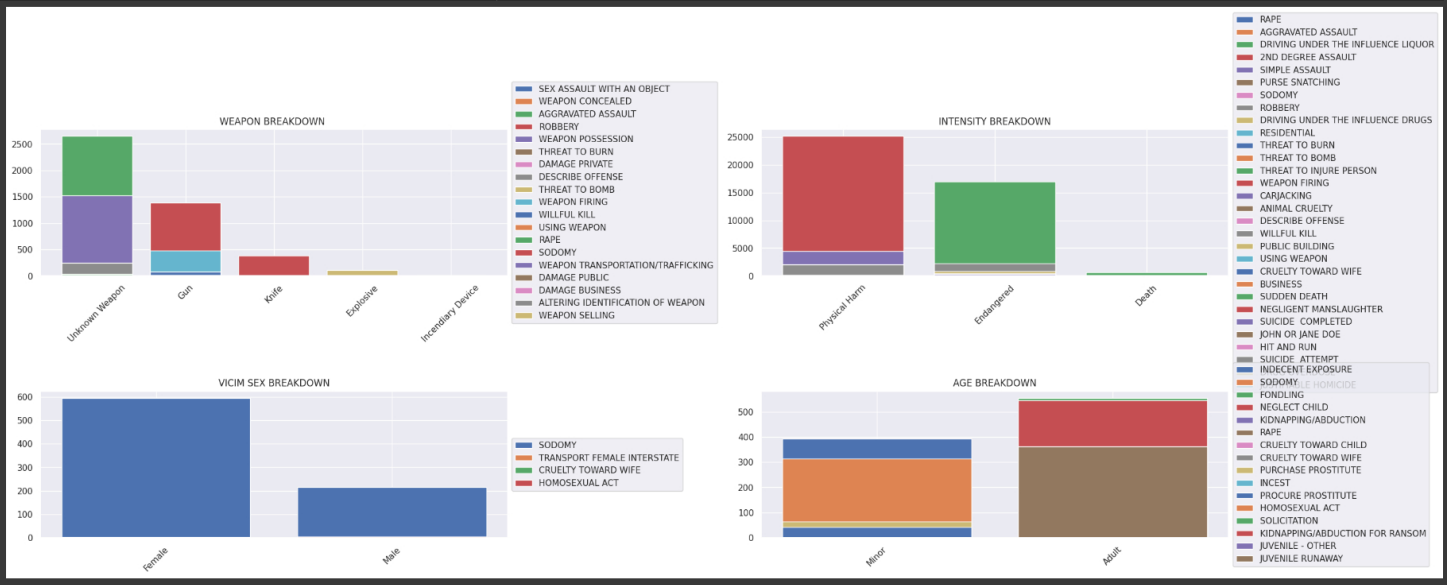






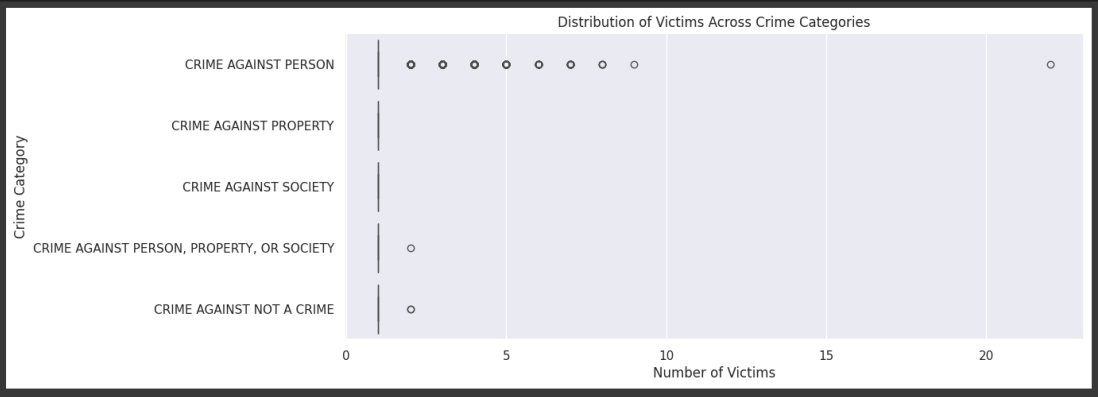

```
[ ] #Creating 4 different stacked bar plot for:-
    # 1. weapon_type,
    #2. weapon used or not,
    #3. intensity and
    #4. victim age
    #the original  data does not have details for these columns but this is my attempt to paint a vague picture
    fig, axes = plt.subplots(2,2, figsize=(26,10))#2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

[4]

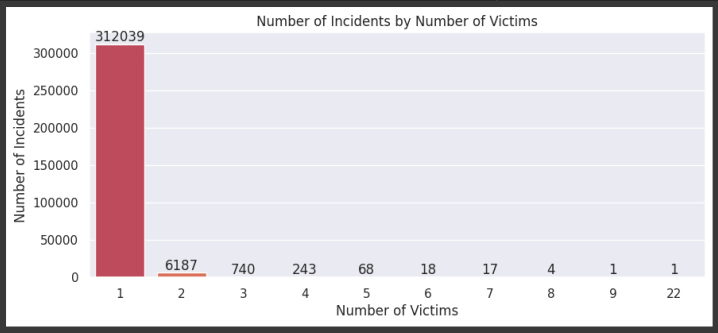



```
[ ] #creating a box plot for number of victims and crime_category

    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()
```

[4]

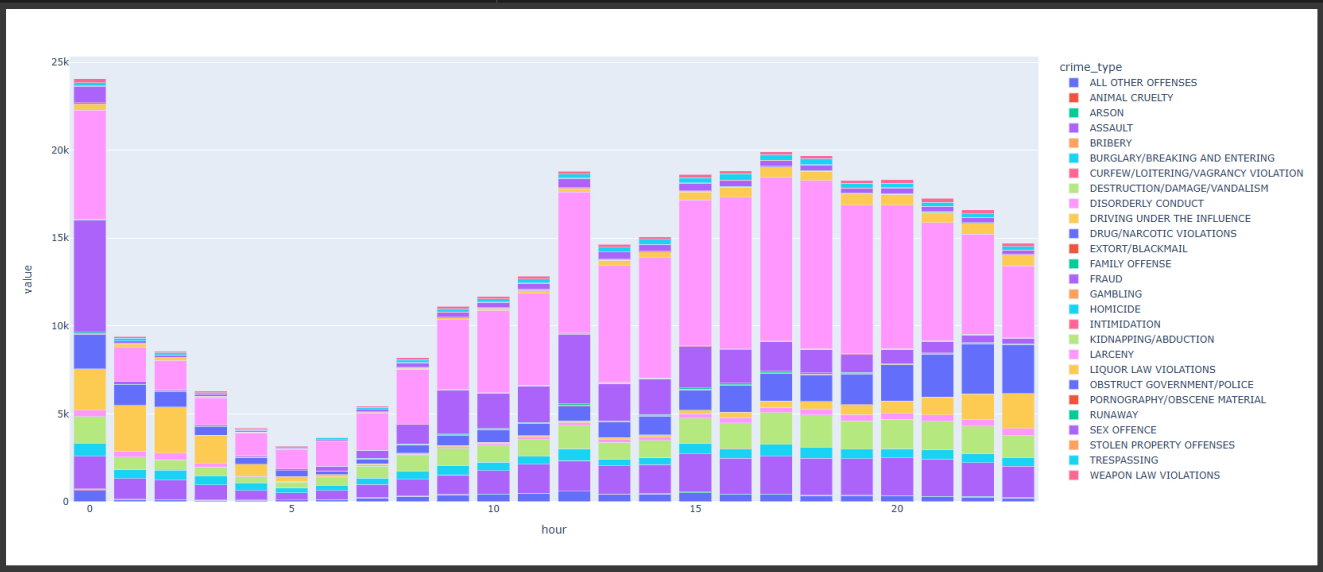


```
[ ] #Using seaborn to map number of victims
    x=df.victims.value_counts().index
    y=df.victims.value_counts().values

    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()
```

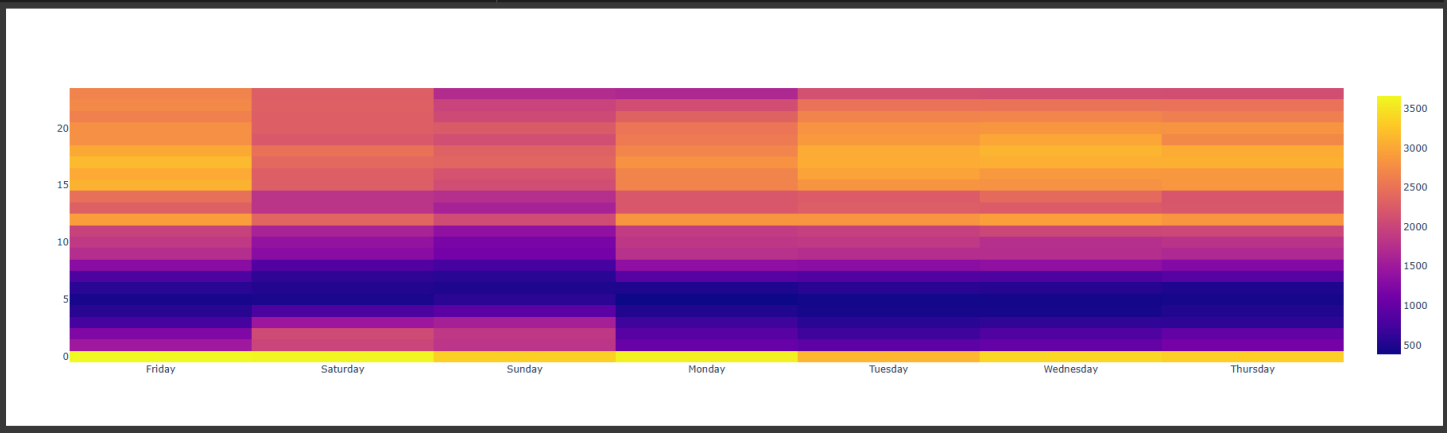


ANALYSIS BAES ON DATE-TIME DATA

```
[ ] #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()
```

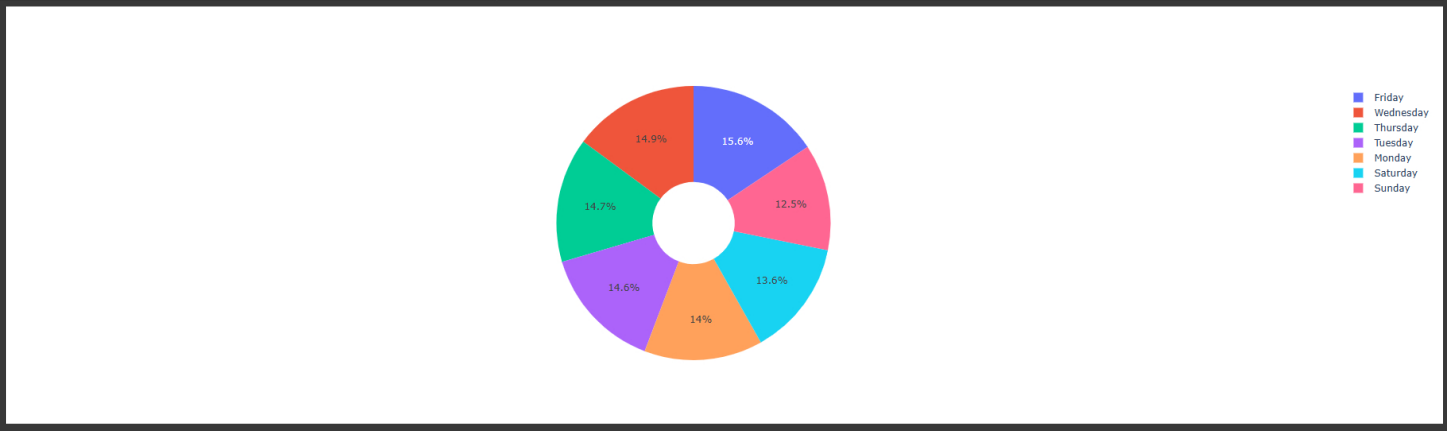



```
[ ] #using ploty's graph_objects to plot heatmap histogram for number of crimes committed each hour each day of the week
    x = df.day_of_week
    y = df.hour
    fig = go.Figure(go.Histogram2d(x=x,y=y))
    fig.show()
```

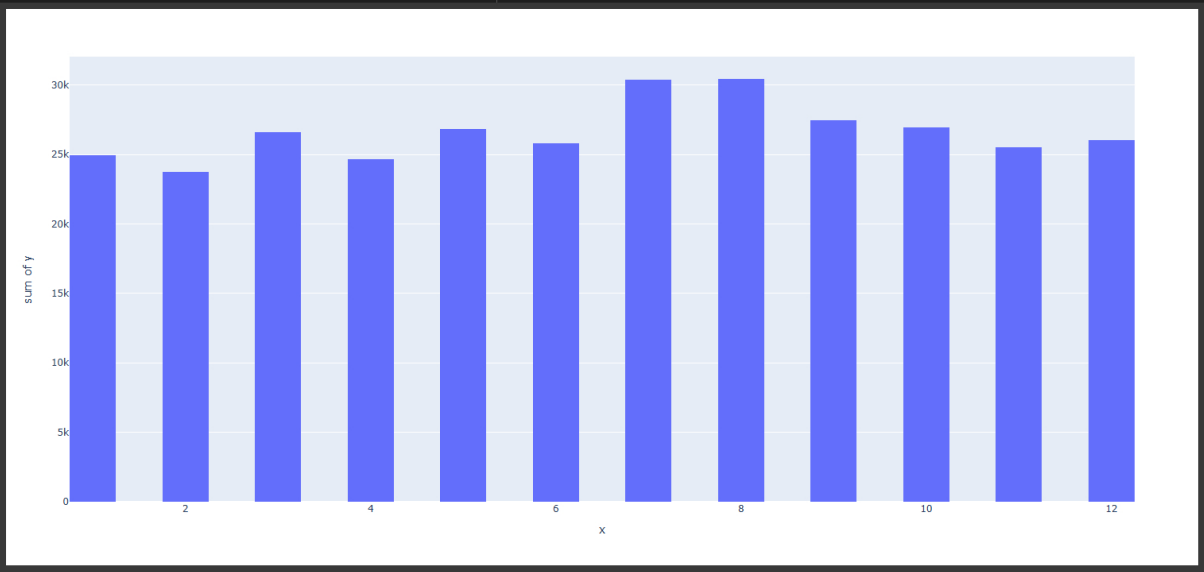



```
[ ] #Using Plotly to create donut graph of the percentage share of crimes per day of the week
    labels = df.day_of_week.unique()
    values=[]
    for each in labels:
        values.append(len(df[df.day_of_week==each]))

    # Use `hole` to create a donut-like pie chart
```

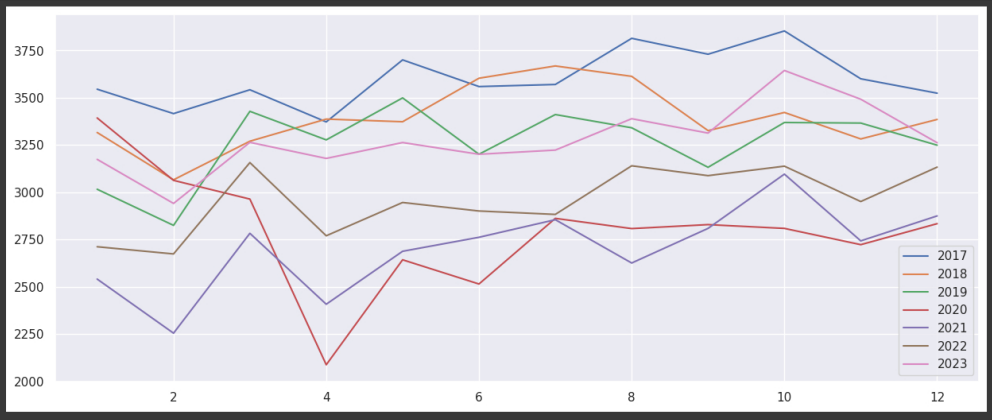
```
fig = go.Figure(data=[go.Pie(labels=labels, values=values, hole=.3)])
fig.show()
```




```
[ ] #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()
```

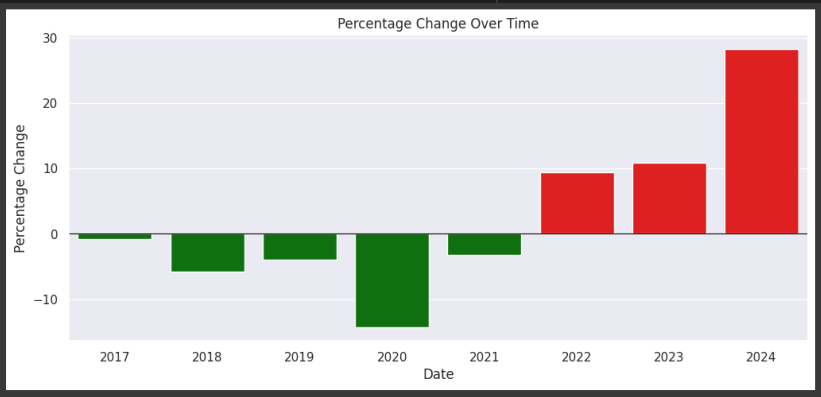



```
[ ] #plotting line chart crimes committed each year
plt.figure(figsize=(15,6))

#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()
```




```
[ ] #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()
```



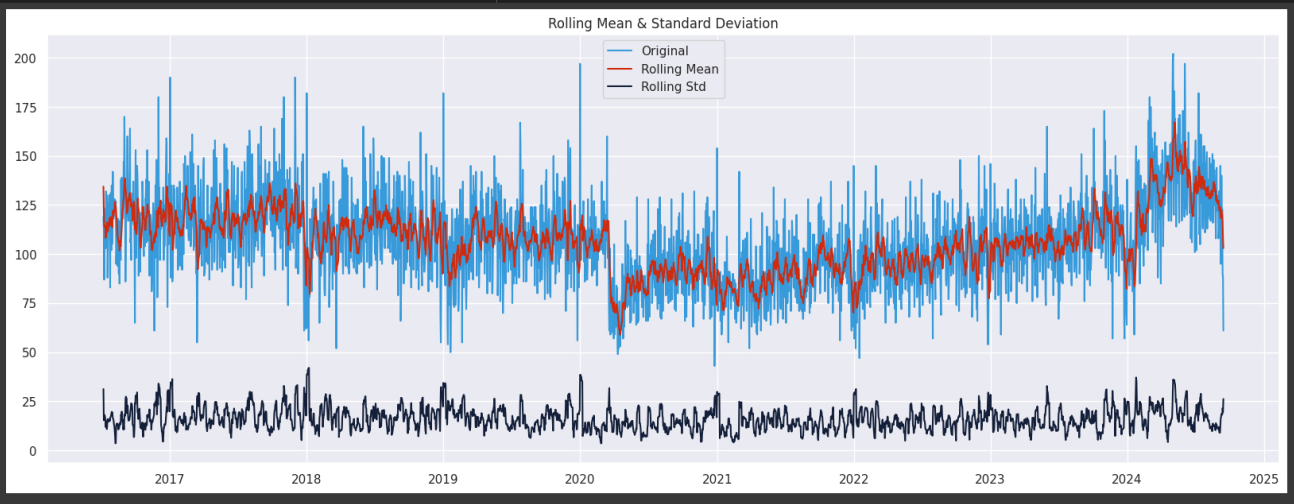
TIME SERIES PREDICTION USING ARIMA

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

```
[ ] #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)
```

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

```
[ ] #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)
```



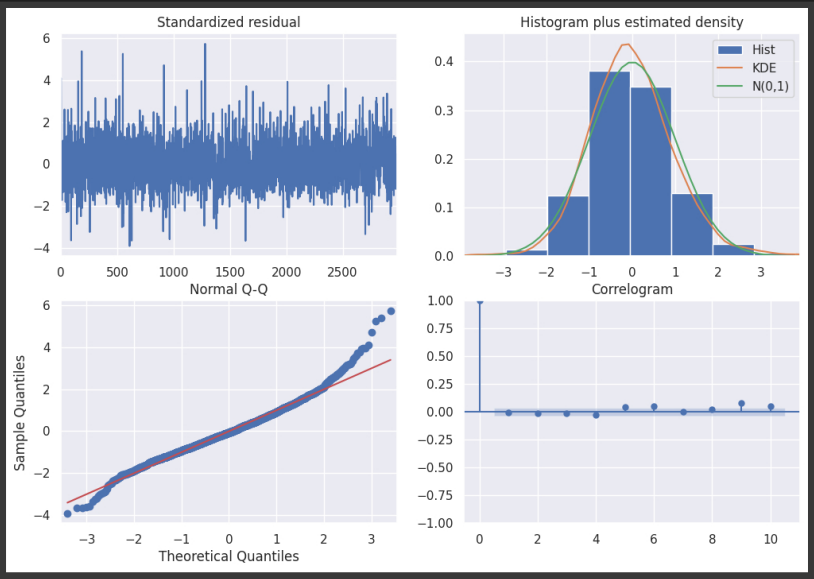

```
[ ] #Augmented Dickey-Fuller test:
#a statistical test used to determine whether a time series is stationary or not.
#required for proforming time_series analysis and prediction
print('Results of Dickey Fuller Test:')
dftest = adfuller(df_ml_train['Count'], autolag='AIC')
```

```
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)
```

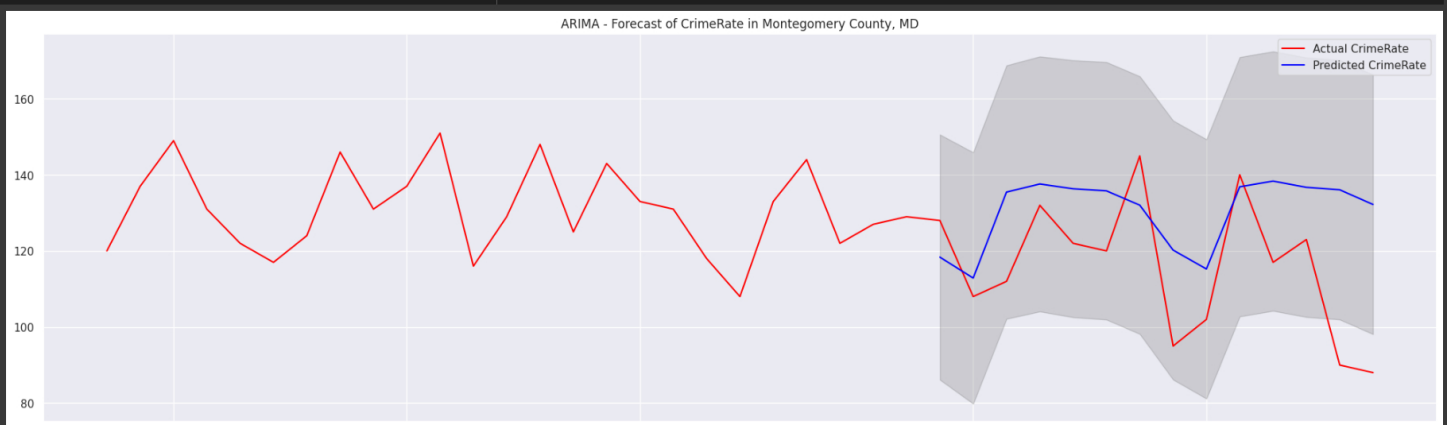
```
Results of Dickey Fuller Test:
Test Statistic                  -2.986409
p-value                          0.036183
#Lags Used                      27.000000
Number of Observations Used    2955.000000
Critical Value (1%)              -3.432565
Critical Value (5%)              -2.862519
Critical Value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

```
#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)
```

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




```
#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)
```



2024-08-08 2024-08-15 2024-08-22 2024-09-01 2024-09-08

```
#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)
```

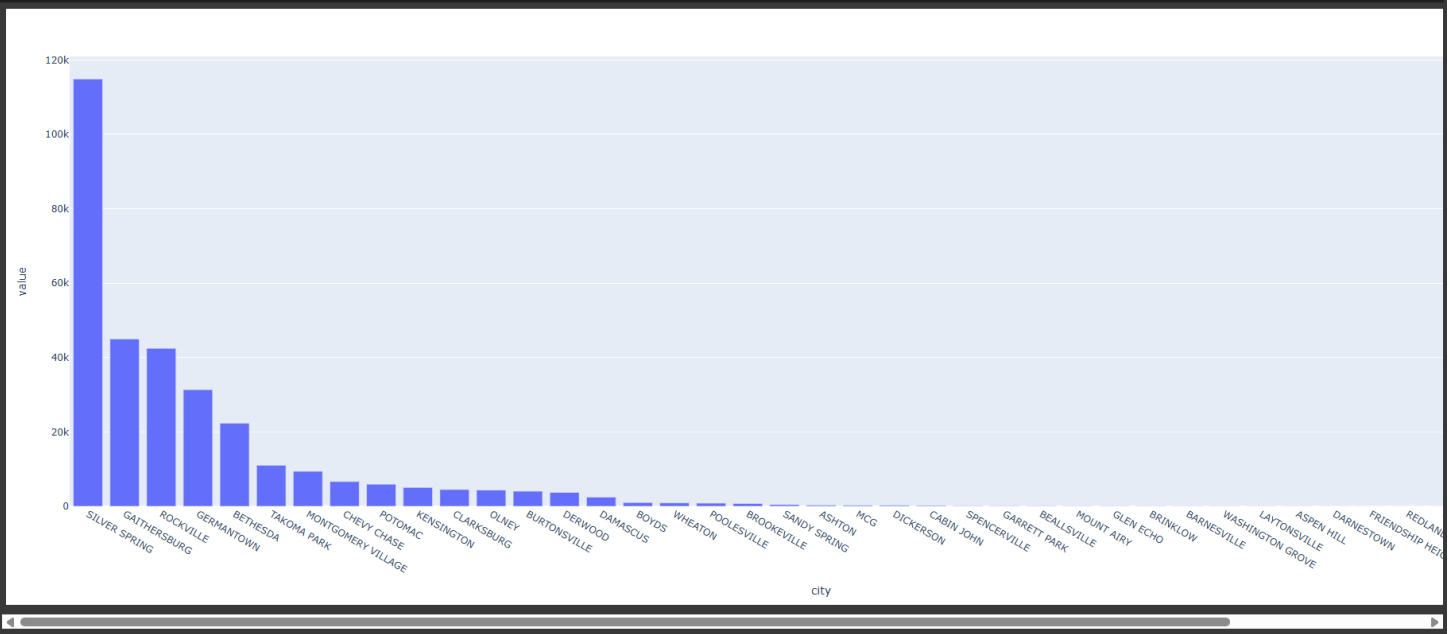
```
17.408015238646698
```

ANALYSIS BASED ON CITY AND OTHER LOCATION BASED DATA

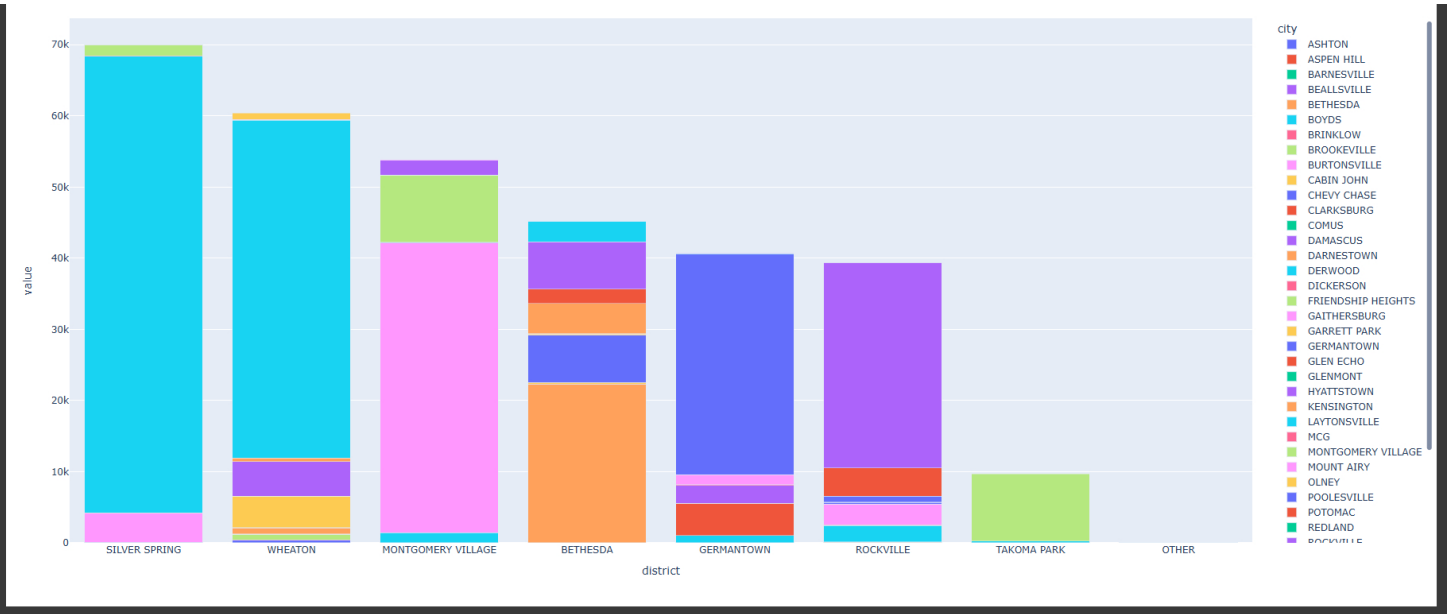
```
#plotting a pivot table from Police District using Seaborn
pivot_table = df.pivot_table(values='victims', index='district', aggfunc='count').sort_values(by='victims', ascending=False)

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

```
Text(216.25, 0.5, 'Count')
```


```
#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()
```

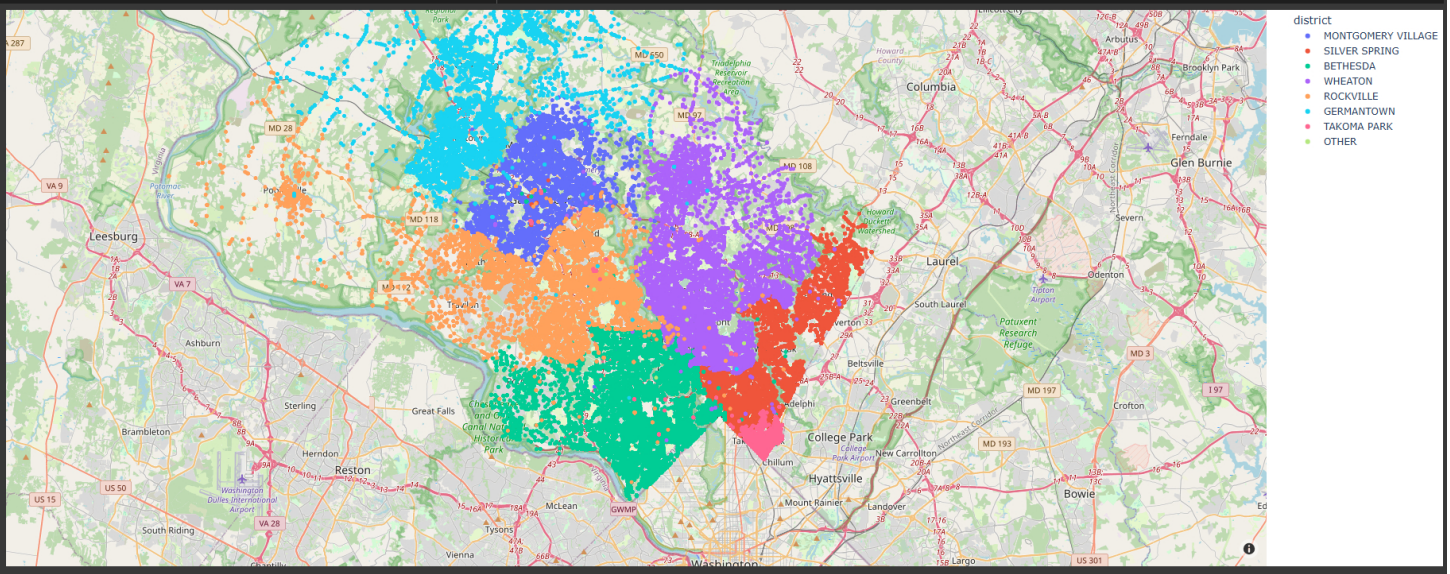



```
#Creating a Stacked bar plot for the number of crimes per district with stacks showing the city

# 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()
```

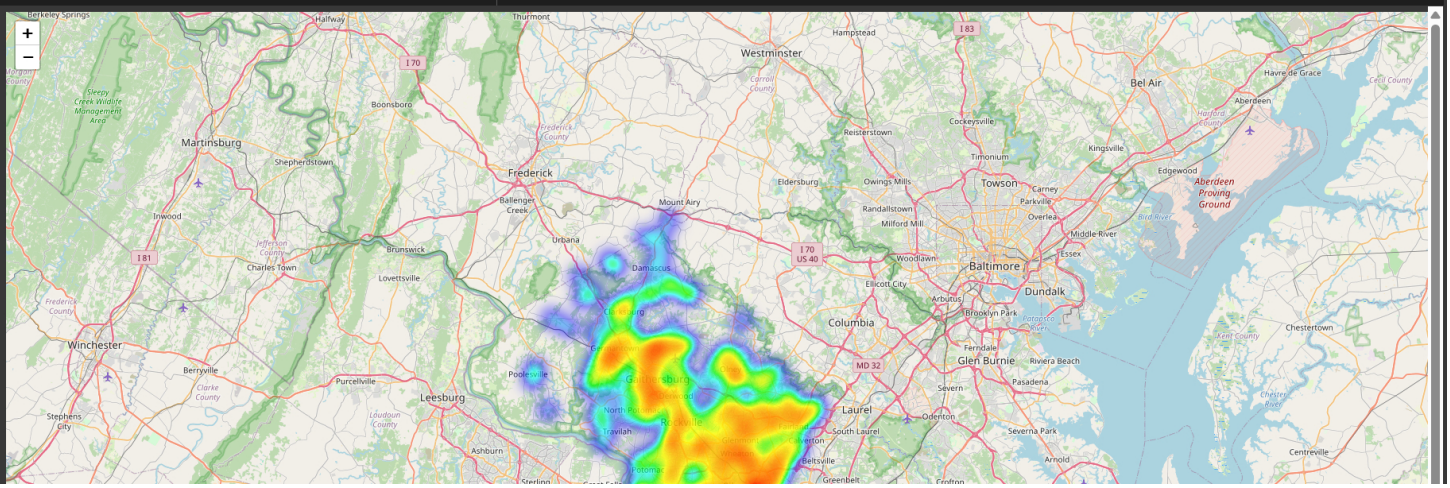


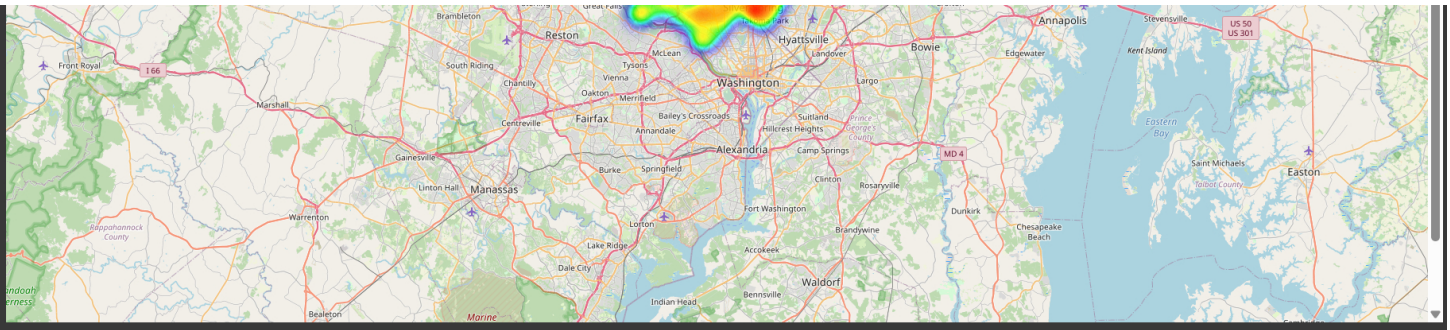
GEO-SPACIAL ANALYSIS

```
[ ] #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="id", hover_data=["date","hour",'crime_detail'],
                            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()
```




```
[ ] #Plotting an 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
```





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

COLLECTING MORE KIND OF DATASETS FOR CORRELATIONS AND CAUSAL ANALYSIS