IST 652 - SCRIPTING FOR DATA ANALYSIS

DATA ANALYSIS ON CRIMES IN BOSTON

FINAL PROJECT REPORT

Submitted By

Vedant Devidas Patil

1. Project Purpose

The purpose is to exhibit the capacity to create python program code that access and collect data from fields in one or more of the three categories of data examined in the course and then prepare the information to generate reports, lists, and other structures for analysis.

2. Deliverables

- Choose an investigational subject which we can used to perform meaningful Data Analysis.
- Use Python code for analyzing data.
- Gather data from one or multiple resources having individual or multiple datasets.
- Use multiple data acquisition and analysis techniques analyze the data.
- Generate actionable insights from the analyzed data.

3. Dataset Description

The Boston Police Department (BPD) provides crime event reports to record the preliminary information around an occurrence to which BPD officers react. This dataset contains data from the new criminal event report system, which has fewer fields designed to capture the incident's kind as well as its timing and location. The project utilizes four datasets from 2015 to 2018.

Dataset Source:
https://data.boston.gov/dataset/crime-incident-reports-august-2015-to-date-source-new-system

The dataset combined from 2015 to 2018 consists of 353253 rows and 17 columns. It contains following attributes:

- INCIDENT_NUMBER - A uniques identification number for every new crime
- OFFENSE_CODE - A unique code for every group of offense.
- OFFENSE_CODE_GROUP - A description where each offense code belong to.

- OFFENSE_DESCRIPTION - Description of crime commited.
- DISTRICT - The district code where crime has been committed.
- REPORTING_AREA - The area code where crime has been reported.
- SHOOTING - Specifies if shooting is involved in that crime.
- OCCURRED_ON_DATE - The date timestamp when the crime occured.
- YEAR - The year in which the crime was committed.
- MONTH - The month in which the crime was committed.
- DAY_OF_WEEK - The day of week in which the crime was committed.
- HOUR - States at what hour of the day the crime was committed.
- UCR_PART - States the seriousness of the crime with Part One being the least serious crime.
- STREET - Describes the street at which crime was committed.
- Lat - The latitude of the location where crime was committed.
- Long - The longitude of the location where crime was committed.
- Location - Describes the latitude and longitude of an exact place where crime took place.

4. Data Acquisition

The initial stage in exploratory data analysis is data acquisition, which entails importing the dataset from the necessary resources into our working Python environment. In this project, I had downloaded the csv files from the above dataset source mentioned and read it into my python notebook. Below python code shows the procedure that I followed to import the csv files into my working environment.

```
cwd = os.path.abspath('H:/IST 652 Scripting for DA/Assignments/Final Project/Dataset')
```

```
#The method listdir() returns a list containing the names of the entries in the directory given by path.
file_list = os.listdir(cwd)

#accessing the name of files in a folder through list.
file_list
```

```
['df_2015.csv', 'df_2016.csv', 'df_2017.csv', 'df_2018.csv']
```

```
df_append = pd.DataFrame()

#append all files together
for file in file_list:
            doc = 'H:/IST 652 Scripting for DA/Assignments/Final Project/Dataset/' + file
            df_temp = pd.read_csv(doc)
            df_append = df_append.append(df_temp, ignore_index=True)
df_append
```

Output:

	INCIDENT_NUMBER	OFFENSE_CODE	OFFENSE_CODE_GROUP	OFFENSE_DESCRIPTION	DISTRICT	REPORTING_AREA	SHOOTING	OCCURRED_ON_D
0	I192068249	2647	Other	THREATS TO DO BODILY HARM	B2	280	NaN	2015-08-28 10:2
1	I192061894	1106	Confidence Games	FRAUD - CREDIT CARD / ATM FRAUD	C11	356	NaN	2015-08-20 00:0
2	I192038828	1107	Fraud	FRAUD - IMPERSONATION	A1	172	NaN	2015-11-02 12:2
3	I192008877	1107	Fraud	FRAUD - IMPERSONATION	E18	525	NaN	2015-07-31 10:0
4	I182090828	1102	Fraud	FRAUD - FALSE PRETENSE / SCHEME	D4	159	NaN	2015-12-01 12:0
...	...	...	...	...	...	...	...	
353248	I070720870-00	802	Simple Assault	ASSAULT & BATTERY	B2	318	NaN	2018-12-13 00:0
353249	I070720870-00	3125	Warrant Arrests	WARRANT ARREST	B2	318	NaN	2018-12-13 00:0

REPORTING_AREA	SHOOTING	OCCURRED_ON_DATE	YEAR	MONTH	DAY_OF_WEEK	HOUR	UCR_PART	STREET	Lat	Long	Location
280	NaN	2015-08-28 10:20:00	2015	8	Friday	10	Part Two	WASHINGTON ST	42.330119	-71.084251	(42.33011862, -71.08425106)
356	NaN	2015-08-20 00:00:00	2015	8	Thursday	0	Part Two	CHARLES ST	42.300605	-71.061268	(42.30060543, -71.06126785)
172	NaN	2015-11-02 12:24:00	2015	11	Monday	12	Part Two	ALBANY ST	42.334288	-71.072395	(42.33428841, -71.07239518)
525	NaN	2015-07-31 10:00:00	2015	7	Friday	10	Part Two	WINGATE RD	42.237009	-71.129566	(42.23700950, -71.12956606)
159	NaN	2015-12-01 12:00:00	2015	12	Tuesday	12	Part Two	UPTON ST	42.342432	-71.072258	(42.34243222, -71.07225766)
...	...	...	...	...	...	...	...	...	...	...	...
318	NaN	2018-12-13 00:00:00	2018	12	Thursday	0	Part Two	BROOKLEDGE ST	42.309563	-71.089902	(42.30956305, -71.08990197)

The above output shows the structure of the dataset that I have imported. It includes all the data from 2015 to 2018 combined.

5. Data Preprocessing

Data preprocessing is data alteration or deletion before usage, which is done to assure or improve performance. After importing the dataset in our environment, cleaning the data is very important step. Data cleaning helps one to identify all the noise from data and help to remove any redundant data that can lead to false results.

To begin with, I wanted to eradicate all the duplicate data that might exist into my dataset as this data can lead to deviation in the actual expected analysis as same records serves no meaningful purpose. Below code shows the procedure to access all the duplicate rows from the dataset including the first occurrence that has same information in all the attributes.

Code

```
#selecting all the duplicate rows in the dataframe including the first occurance.
duplicate = main_df[main_df.duplicated(keep=False)]
duplicate
```

Output

	INCIDENT_NUMBER	OFFENSE_CODE	OFFENSE_CODE_GROUP	OFFENSE_DESCRIPTION	DISTRICT	REPORTING_AREA	SHOOTING	OCCURRED_ON
1249	I152107190	413	Aggravated Assault	ASSAULT - AGGRAVATED - BATTERY	B3	427	Y	2015-12-29 0
1250	I152107190	413	Aggravated Assault	ASSAULT - AGGRAVATED - BATTERY	B3	427	Y	2015-12-29 0
1251	I152107190	111	Homicide	MURDER, NON-NEGLIGIENT MANSLAUGHTER	B3	427	Y	2015-12-29 0
1252	I152107190	111	Homicide	MURDER, NON-NEGLIGIENT MANSLAUGHTER	B3	427	Y	2015-12-29 0
1253	I152107190	3125	Warrant Arrests	WARRANT ARREST	B3	427	Y	2015-12-29 0

Above image shows some of the sample output rows having duplicated data from 1137 rows.

Now, the duplicated data needs to be dropped from the dataset by keeping only the first occurrence in the dataset, so that we can have one unique entry for all duplicated data. Below code shows the same

```
#Dropping the duplicates from the dataset keeping the first occurance.
main_df = main_df.drop_duplicates()
```

Info() method

The below screenshot shows the info() method that gives information about all the Columns and datatypes, from which we can infer which column data types should be changed according to our requirements. Also, it shows us how many non-null values are present in each column.

```
#The info() contains the number of columns, column labels, column data types, memory usage,
#range index, and the number of cells in each column

main_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 352610 entries, 0 to 353252
Data columns (total 17 columns):
 #   Column               Non-Null Count   Dtype  
---  ------               --------------   -----  
 0   INCIDENT_NUMBER      352610 non-null  object 
 1   OFFENSE_CODE         352610 non-null  int64  
 2   OFFENSE_CODE_GROUP   352610 non-null  object 
 3   OFFENSE_DESCRIPTION  352610 non-null  object 
 4   DISTRICT             350783 non-null  object 
 5   REPORTING_AREA       352610 non-null  object 
 6   SHOOTING             1153 non-null    object 
 7   OCCURRED_ON_DATE     352610 non-null  object 
 8   YEAR                 352610 non-null  int64  
 9   MONTH                352610 non-null  int64  
 10  DAY_OF_WEEK          352610 non-null  object 
 11  HOUR                 352610 non-null  int64  
 12  UCR_PART             352516 non-null  object 
 13  STREET               341417 non-null  object 
 14  Lat                  330106 non-null  float64
 15  Long                 330106 non-null  float64
 16  Location             352610 non-null  object 
dtypes: float64(2), int64(4), object(11)
memory usage: 48.4+ MB
```

From the above data, I can infer that most of my dataset is in expected form.

Removal of Location, Incident number and offense code attribute

The dataset consists of a Location attribute which is a result of concatenation of Lat and Long attributes. Hence, I dropped the location column as it was not much helpful for my analysis and created a complex dataset structure. Also, INCIDENT_NUMBER and OFFENSE_CODE served no purpose for my analysis and thus dropped the same.

The screenshot below shows that there is no location column after Lat and Long.

```
#By observing the variables, it can be easily deduced that the Location variable
#is just storing the concatenation of the Lat and Long,
#so let's get rid of this.
df = main_df.drop("Location", axis = 1)
df.head()
```

ON	DISTRICT	REPORTING_AREA	SHOOTING	OCCURRED_ON_DATE	YEAR	MONTH	DAY_OF_WEEK	HOUR	UCR_PART	STREET	Lat	Long
LY RM	B2	280	NaN	2015-08-28 10:20:00	2015	8	Friday	10	Part Two	WASHINGTON ST	42.330119	-71.084251
D / JD	C11	356	NaN	2015-08-20 00:00:00	2015	8	Thursday	0	Part Two	CHARLES ST	42.300605	-71.061268
D - ON	A1	172	NaN	2015-11-02 12:24:00	2015	11	Monday	12	Part Two	ALBANY ST	42.334288	-71.072395
D - ON	E18	525	NaN	2015-07-31 10:00:00	2015	7	Friday	10	Part Two	WINGATE RD	42.237009	-71.129566
SE ME	D4	159	NaN	2015-12-01 12:00:00	2015	12	Tuesday	12	Part Two	UPTON ST	42.342432	-71.072258

The screenshot below shows that there is no INCIDENT_NUMBER and OFFENSE_CODE column too before the offense description.

```
#next we have variables like INCIDENT_NUMBER and OFFENSE_CODE,
#which serve no purpose in answering our questions. So, let's get rid of them too.

df = df.drop(["INCIDENT_NUMBER", "OFFENSE_CODE"], axis = 1)
df.head(5)
```

OFFENSE_DESCRIPTION	DISTRICT	REPORTING_AREA	SHOOTING	OCCURRED_ON_DATE	YEAR	MONTH	DAY_OF_WEEK	HOUR	UCR_PART	STREET
THREATS TO DO BODILY HARM	B2	280	NaN	2015-08-28 10:20:00	2015	8	Friday	10	Part Two	WASHINGTON ST
FRAUD - CREDIT CARD / ATM FRAUD	C11	356	NaN	2015-08-20 00:00:00	2015	8	Thursday	0	Part Two	CHARLES ST
FRAUD - IMPERSONATION	A1	172	NaN	2015-11-02 12:24:00	2015	11	Monday	12	Part Two	ALBANY ST
FRAUD - IMPERSONATION	E18	525	NaN	2015-07-31 10:00:00	2015	7	Friday	10	Part Two	WINGATE RD
FRAUD - FALSE PRETENSE / SCHEME	D4	159	NaN	2015-12-01 12:00:00	2015	12	Tuesday	12	Part Two	UPTON ST

Analyzing Shooting attribute

After having a closer look at the dataset, I found that shooting had just the NAN values in it, however, I wanted to confirm whether it had any meaningful data or not. Hence, checked that using the describe method.

```
#The column SHOOTING appears to have nothing but NaN values. Let's confirm that.
df['SHOOTING'].describe()
```

```
count     1153
unique       1
top          Y
freq      1153
Name: SHOOTING, dtype: object
```

The above screenshot shows that it contains just 1153 rows with a shooting record as ‘Y’. Hence, we need to replace remaining NAN values with N which represents NO as there is either no data available for those records or there was no shooting for those records.

Below screenshot shows that NAN has been replaced with N.

```
#Counting the number of null values in the dataframe
df["SHOOTING"].isna().sum()
```

```
351457
```

```
#Replacing the NAN values in shooting column using N values
df["SHOOTING"].fillna('N', inplace=True)
df.head(5)
```

	OFFENSE_CODE_GROUP	OFFENSE_DESCRIPTION	DISTRICT	REPORTING_AREA	SHOOTING	OCCURRED_ON_DATE	YEAR	MONTH	DAY_OF_WEEK	HOUF
0	Other	THREATS TO DO BODILY HARM	B2	280	N	2015-08-28 10:20:00	2015	8	Friday	1(
1	Confidence Games	FRAUD - CREDIT CARD / ATM FRAUD	C11	356	N	2015-08-20 00:00:00	2015	8	Thursday	(
2	Fraud	FRAUD - IMPERSONATION	A1	172	N	2015-11-02 12:24:00	2015	11	Monday	1:
3	Fraud	FRAUD - IMPERSONATION	E18	525	N	2015-07-31 10:00:00	2015	7	Friday	1(
4	Fraud	FRAUD - FALSE PRETENSE / SCHEME	D4	159	N	2015-12-01 12:00:00	2015	12	Tuesday	1:

Null Values

Checking null for other remaining columns as below.

```
#Checking missing/null values in all the below columns

time_list = ["HOUR", "DAY_OF_WEEK", "MONTH", "OCCURRED_ON_DATE", "DISTRICT", "REPORTING_AREA", "STREET", "Lat", "Long"]
for item in time_list:
    print("No. of NaNs in {} = {}".format(item, df[item].isna().sum()))

#DISTRICT and STREET have missing values. REPORTING_AREA looks like a good variable to consider,
#since it has no missing values.
```

```
No. of NaNs in HOUR = 0
No. of NaNs in DAY_OF_WEEK = 0
No. of NaNs in MONTH = 0
No. of NaNs in OCCURRED_ON_DATE = 0
No. of NaNs in DISTRICT = 1827
No. of NaNs in REPORTING_AREA = 0
No. of NaNs in STREET = 11193
No. of NaNs in Lat = 22504
No. of NaNs in Long = 22504
```

The above screenshot displays that there are no null values in any of the attributes except Lat and Long.

After analyzing further, I realized that Lat and Long columns has been entered with -1 to indicate absence. However, this is not the correct way to represent Null values as -1 value could represent any of the location or might give error while performing analysis. Hence, replaced -1 with NaN in both columns as below:

```
df["Lat"].replace(-1, None, inplace=True)
df["Long"].replace(-1, None, inplace=True)
(df["Long"].isna()).sum()
```

```
22557
```

Analyzing the months

Identifying the completeness of the data also plays a very important role because it gives an idea about how we can proceed with the data analysis and what particular data needs to be added or subtracted for our analysis to be perfect.

Hence, I analyzed whether I have data for all the months of all 4 years using below code.

Code and Output

```
#Checking which months are included for which particular year.

years = [2015, 2016, 2017, 2018]
for year in years:
    print(df[df["YEAR"] == year]["MONTH"].unique())

#Here, we can see that 2015 has data from June to Dec whereas rest all year have all 12 months data.
```

```
[ 8 11  7 12  9  6 10]
[ 9  8  6  4  5 10 12  1  7  3  2 11]
[ 8  2  7  9 11 12  3  5  1  6  4 10]
[ 4  3 10  1 11  5  8  9  2 12  7  6]
```

From the above output, we can see that 2015 misses data for January to May month. This was very important to know as we can select the scope for our data analysis.

6. Exploratory Data Analysis

Now that our dataset is cleaned and is in expected shape, we can proceed with the analysis.

Q1 Which are the different types of crimes that are most and least common in Boston?

Firstly, I intended to find the crimes that occur the maximum times and minimum times and each row represents one crime. Hence, taking the count of the OFFENCE_CODE_GROUP by performing group by would give the count of each unique crime as follows:

```python
#First, we perform group-by on the offence code group column as the crimes description/names are mentioned in that column
#size() function return the number of rows if Series.
#Otherwise it returns the number of rows times number of columns if DataFrame

#The sort_values() function sorts the dataframe column in ascending or descending order.
crime_count = pd.DataFrame(df.groupby('OFFENSE_CODE_GROUP').size().sort_values(ascending=False).rename('Count').reset_index())


crime_count
#Displaying the dataframe.
```

	OFFENSE_CODE_GROUP	Count
0	Motor Vehicle Accident Response	41062
1	Larceny	28857
2	Medical Assistance	26195
3	Investigate Person	20410
4	Other	19842
...	...	...
62	HUMAN TRAFFICKING	8
63	INVESTIGATE PERSON	4
64	Biological Threat	2
65	Burglary - No Property Taken	2
66	HUMAN TRAFFICKING - INVOLUNTARY SERVITUDE	2

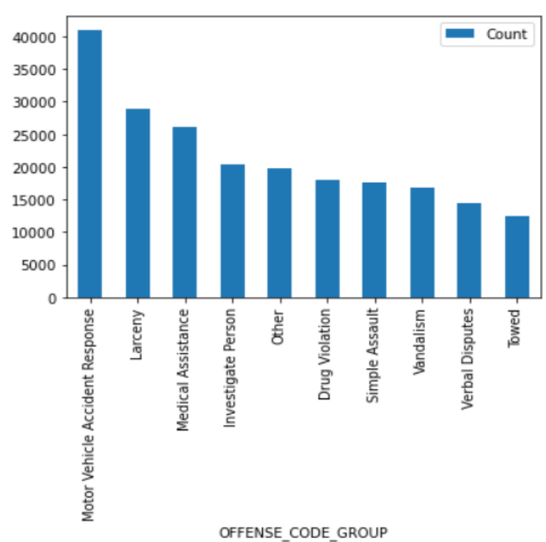
Further, to better understand the top 10 crimes that occur the most, I performed a visualization using the Advanced python package 'Matplotlib'.

```python
#importing the matplotlib package for displaying bar plot.
import matplotlib.pyplot as plt
```

```python
#Since, the dataframe contains 67 columns, I selected just top 10 crimes that happens the most.

crime_count[:10].plot(x='OFFENSE_CODE_GROUP',y='Count',kind='bar')
```

```
<AxesSubplot:xlabel='OFFENSE_CODE_GROUP'>
```



Conclusion:

The above screenshots shows the 5 crimes that are most common and least common with Motor Vehichle Accident Response had highest crime record of 41062 in 4 years whereas Human trafficking had crime record of 2 crimes in 4 years. This data can be constructively used by the Police department of the Boston City to appropriately focus on specific activities and take strict actions.

Q2 Which are the most and least unsafe areas in Boston?

Here, The unsafe areas could be identified by the streets column. Alternatively, we could also use Lat and long attributes to find the areas, however, street column makes more sense as it provides us with data that can be understood without much effort as it is easy to read and understand.

Here, I Grouped the data using the street column and arranged them in ascending order.

```
#Grouping the data using the street column and arranging them in ascending order.
#Renaming the newly created column as Count, which displays the count of crimes happening on those particular streets
crime_location = pd.DataFrame(df.groupby('STREET').size().sort_values(ascending=False).rename('Count').reset_index())

#Dsiplaying the top 5 streets that have the most cost.
crime_location.head()
```

	STREET	Count
0	WASHINGTON ST	15774
1	BLUE HILL AVE	8625
2	BOYLSTON ST	8021
3	DORCHESTER AVE	5687
4	TREMONT ST	5307

```
crime_location = pd.DataFrame(df.groupby('STREET').size().sort_values(ascending=True).rename('Count').reset_index())

#Displaying the top 5 streets with least crimes.
crime_location.head()
```

	STREET	Count
0	SCHORTMANNS TER	1
1	CAROL	1
2	RONALD ST	1
3	ROMAR TER	1
4	ROLLINS PL	1

The above screenshot shows that Washington St and Blue Hillave streets are the worst streets in Boston city to live in, whereas, the Schortmanns ter and Carol are one of the best streets to live and have fun around.

To analyze these outcome further, I wanted to confirm whether Washington Street is actually the worst street to live by having high number of heinous crimes or it is a street which has least number of heinous crimes but high number of non-serious, minor crimes.

In order to analyze this, I used shooting as a criteria as it was the most heinous crime present in the data. I found out the shooting data for each street by performing group by on Street column on the dataset that has only shooting records.

```
#It is important to have an information about which streets have the most heinous crimes
#After analysis, I fighured out that Shooting was the most heinous crimes amongst all the crimes that have been happening in
#Boston.
```

```
#Getting all the shooting data from the dataframe.
shootings = main_df[~main_df['SHOOTING'].isna()]
shootings.head()
```

	INCIDENT_NUMBER	OFFENSE_CODE	OFFENSE_CODE_GROUP	OFFENSE_DESCRIPTION	DISTRICT	REPORTING_AREA	SHOOTING	OCCURRED_ON_DAT
860	I152107623	413	Aggravated Assault	ASSAULT - AGGRAVATED - BATTERY	A7	26	Y	2015-12-30 22:55:
865	I152107617	413	Aggravated Assault	ASSAULT - AGGRAVATED - BATTERY	B3	444	Y	2015-12-30 21:53:
1249	I152107190	413	Aggravated Assault	ASSAULT - AGGRAVATED - BATTERY	B3	427	Y	2015-12-29 08:47:
1251	I152107190	111	Homicide	MURDER, NON-NEGLIGIENT MANSLAUGHTER	B3	427	Y	2015-12-29 08:47:
1253	I152107190	3125	Warrant Arrests	WARRANT ARREST	B3	427	Y	2015-12-29 08:47:

The above accesses the data where shooting has been taken place from the entire dataset. Further, I perfomed group by on the street column to get the shooting details.

```
#Counting the nnumber of shooting in each top 5 streets.
shooting_location = pd.DataFrame(shootings.groupby('STREET').size().sort_values(ascending=False).rename('Count').reset_index())
shooting_location.head()
```

	STREET	Count
0	WASHINGTON ST	53
1	BLUE HILL AVE	41
2	DUDLEY ST	29
3	COLUMBIA RD	24
4	CENTRE ST	16

Conclusion:

From the above image, it is clear that Washington street and Blue Hill Ave had 53 and 41 occurrences of shooting and consequently are the worst streets to live in, considering all types of crimes. This data can be used by the government and police department to provide extra security, impose stricter laws in those areas. Also, tourists and local people could use this data to avoid those areas.

Q3 Monthly Crime Activity from 2015 - 2018 / Does the frequency of crimes change over the Year?

The monthly crime data will enable us to identify the crime trend in the city and also help us to identify which year the city had recorded the most incidents of crimes. To approach this problem, we could group the data using 'Year' and 'Month' columns and sort the data in ascending order. Following, I used pivot table to create a spreadsheet-style pivot table as a DataFrame as follows:

```
#A pivot table usually consists of row, column, and data fields and arranges them accordingly
crime_activity_plot = pd.DataFrame(df.groupby(['MONTH','YEAR']).size().sort_values(ascending=False).rename('Count').reset_index(

crime_activity_2015_2018 = crime_activity_plot.pivot_table(values='Count',index='MONTH',columns='YEAR')
```

```
crime_activity_2015_2018
```

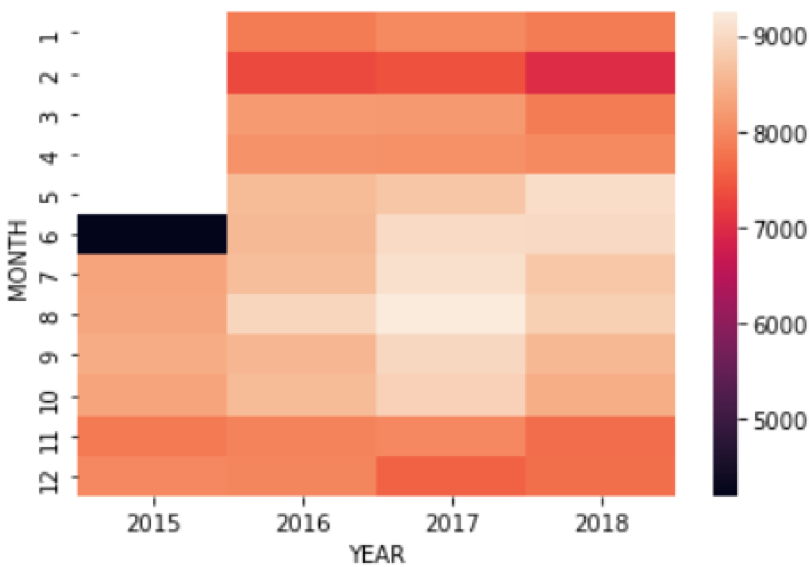
YEAR	2015	2016	2017	2018
MONTH				
1	NaN	7848.0	8008.0	7846.0
2	NaN	7321.0	7423.0	6992.0
3	NaN	8206.0	8188.0	7842.0
4	NaN	8114.0	8086.0	7996.0
5	NaN	8590.0	8736.0	9059.0
6	4188.0	8574.0	9007.0	8995.0
7	8328.0	8635.0	9098.0	8762.0
8	8341.0	8948.0	9239.0	8862.0
9	8415.0	8540.0	8974.0	8553.0
10	8314.0	8599.0	8889.0	8434.0
11	7823.0	7936.0	7978.0	7682.0
12	7995.0	7970.0	7572.0	7704.0

The above image shows the crime count for every month of every year. However, it is very time consuming to grasp the data from the above table and understand the changes monthly or yearly. Hence, I used an advanced python library seaborn to generate a heatmap so that we could understand the chart just by looking at it in an interactive way.

```
import seaborn as sns

#Generating heatmap using seaborn.
sns.heatmap(crime_activity_2015_2018)
```

```
<AxesSubplot:xlabel='YEAR', ylabel='MONTH'>
```



Conclusion:

We may infer from the heatmap and table above that from 2015 through 2017, crime climbed, and then it somewhat decreased again in 2018. The year 2017 and the months of May to October saw the most crimes in Boston across these four years.

Q4 Check the trend of crimes across all 4 seasons for all years.

Understanding the trend of crime during different times of the year can be very useful as it can help tourists to understand which time of the year would be best for them to visit the city. To solve this problem, I accessed the Months column from the dataframe and grouped all the months to their respective seasons and created a dataframe as shown in the screenshot below. Further, I assigned these values and created a combined dataset to find the count of crimes in each season.

```
#accessing the data from 12,1,2 months and storing it in winter variable
winter = pd.DataFrame(df[(main_df.MONTH.isin([12,1,2]))])
winter_data = len(winter.index)

#accessing the data from 3,4,5 months and storing it in spring variable
spring = pd.DataFrame(df[(main_df.MONTH.isin([3,4,5]))])
spring_data = len(spring.index)

summer = pd.DataFrame(df[(main_df.MONTH.isin([6,7,8]))])
summer_data = len(summer.index)

fall = pd.DataFrame(df[(main_df.MONTH.isin([9,10,11]))])
fall_data = len(fall.index)
```

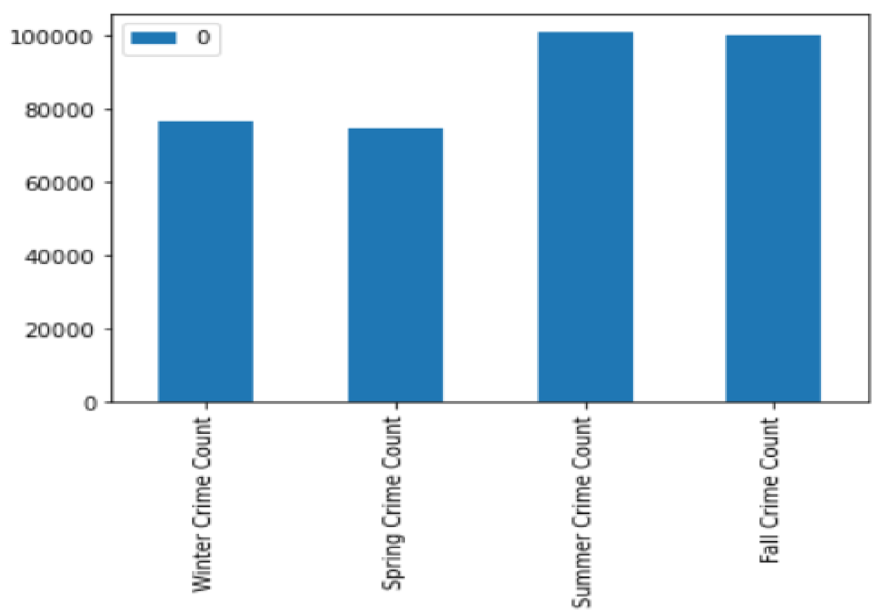
```
season_df = pd.DataFrame({ 'Winter Crime Count': [winter_data], 'Spring Crime Count': [spring_data], 'Summer Crime Count': [summe
              'Fall Crime Count': [fall_data]
})
#assigning the data from above varibale in a dataframe

#printing the dataframe
season_df
```

	Winter Crime Count	Spring Crime Count	Summer Crime Count	Fall Crime Count
0	76679	74817	100977	100137

Conclusion:

From the above output, we can see that Summer recorded the highest amount of crimes in the city. To further understand the differences and compare it with other season I created a matplotlib visualization as below.



Q5 Analyze the crimes for each day of the week for all categories of crime.

The data frame consists of UCR_PART column that divides each crime into one of the tree types which is Part one, part two and part three, wherein, Part one represents the least least serious crime and part three contains the most serious crime. Using this data we can identify which days of the week have the highest or lowest number of crimes along with the total number of Part one, part two and part three level crimes.

The screenshot below explains the code and approach through comments.

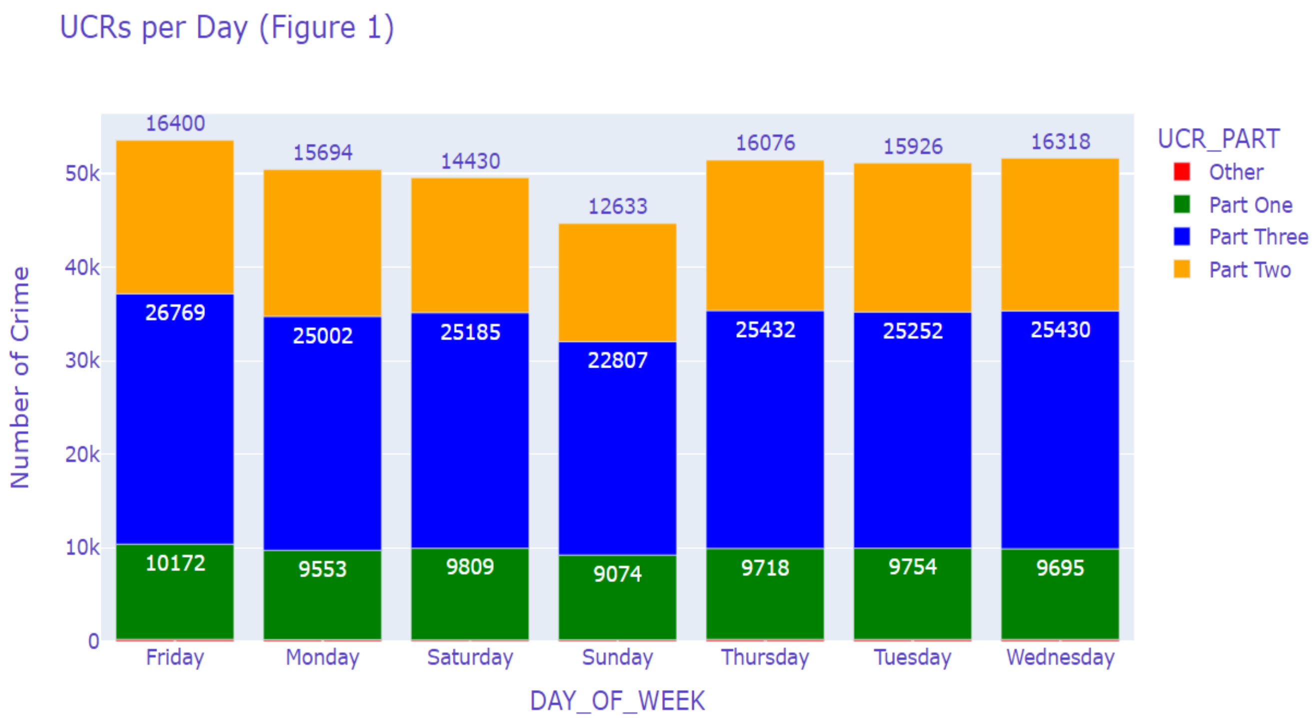
```
#importing required library
import plotly.express as px

#performed group-by on day_of_week, Ucr_part column as we want to acess data on each of the week and each ucr part accordingly.
ucr_day = pd.DataFrame(data = (df.groupby(["DAY_OF_WEEK", "UCR_PART"]).count()["OFFENSE_CODE_GROUP"]).reset_index().values,
                       columns = ["DAY_OF_WEEK", "UCR_PART", "noc"]).sort_values("DAY_OF_WEEK").reset_index(drop = True)
```

```
#setting the x-axis and y-axis along with title and color of the plot
fig3 = px.bar(ucr_day, x = "DAY_OF_WEEK", y = "noc",
              color = "UCR_PART", title = "UCRs per Day",
              labels = {"day" : "DAY_OF_WEEK", "noc" : "Number of Crime", "ucr_part" : "UCR_PART"}, text = "noc",
                color_discrete_sequence=["red", "green", "blue", "orange"])
```

```
fig3.update_traces(textposition = "outside")
#assigning the UI for the bar plot
fig3.update_layout(
    font_color="#5642C5",
    title_font_color="#5642C5",
    legend_title_font_color="#5642C5",
    font_size = 14)
```

Output:



Conclusion:

From the above screenshot, we could see that Weekdays had a comparatively high amount of crimes, with Friday being the day having the highest crimes. Also, The crimes are further reduced towards weekends, that is Saturday and Sunday in that order. Additionally, Friday again tops for the days having the highest number of heinous crimes, ie Part One crimes. Lastly, we could see how UCR_PART can be compared for all days for all three parts.