Name: Bryan Cheng Hengze

Data Exploration and Auditing

```
In [1]: import pandas as pd
```

```
In [2]: crime = pd.read_csv('Crime_Statistics_SA_2014_2019.csv')
```

Firstly, I import the pandas library with an alias pd which provides a data structure called a DataFrame. Then, I load the crime data (Crime_Statistics_SA_2014_2019.csv) into a pandas DataFrame called 'crime' using Pandas.

A1. Dataset size

```
In [3]: crime.shape
Out[3]: (385296, 7)
```

.shape returns a tuple which gives the axis dimensions of the table (Row, Column)

Row: 385296

Column: 7

A2. Null values in the dataset

```
In [4]: crime.isnull().values.any()
Out[4]: True
```

.isnull().values.any() will check whether any value is missing in the dataset and return True if there is and False if there isn't any.

Yes, there are null values in this dataset.

```
In [5]: crime.isnull().sum()

Out[5]: Reported Date                 0
        Suburb - Incident           159
        Postcode - Incident         403
        Offence Level 1 Description   0
        Offence Level 2 Description   0
        Offence Level 3 Description   0
        Offence Count                 0
        dtype: int64
```

```
In [6]: crime["Suburb - Incident"].fillna("No Location", inplace = True)
        crime["Postcode - Incident"].fillna("No Postcode", inplace = True)
```

```
In [7]: crime.isnull().values.any()

Out[7]: False
```

.isnull().sum checks the total value that is missing for each column in the dataset

Therefore, I have filled in the missing values (using .fillna) for now as we might be using it later on in the following question to ensure the best results is obtained.

A3. Data Types

```
In [8]: print(crime['Reported Date'].min())

        2014-01-01
```

```
In [9]: print(crime['Reported Date'].max())

        2019-12-03
```

Min: 2014-01-01

Max: 2019-12-03

```
In [10]: print(crime['Reported Date'].dtype)

         object
```

```
In [11]: crime['Reported Date'] = pd.to_datetime(crime['Reported Date'])
```

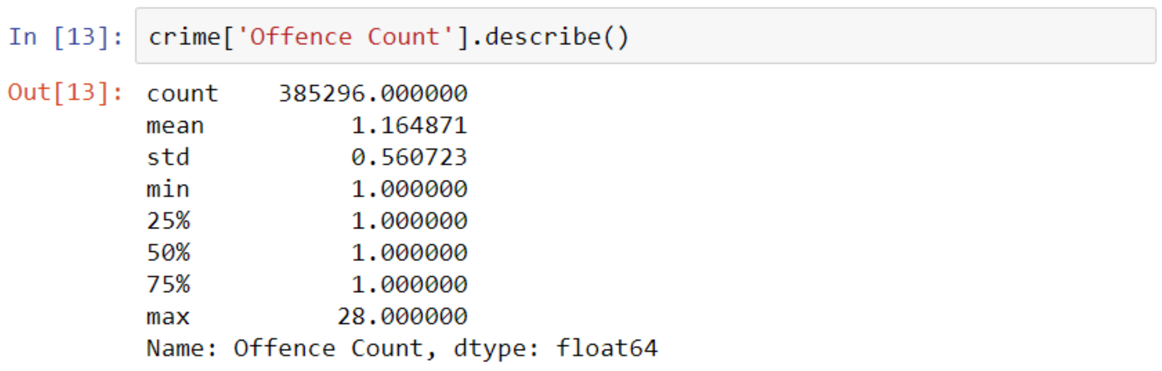
```
In [12]: print(crime['Reported Date'].dtype)

         datetime64[ns]
```

.dtype gives the data type of the column

The data type for the column ‘Reported Date’ was an object.

It did not have the correct data type. Therefore, I converted it to datetime data type using .to_datetime() method.

A4. Descriptive statistics



```
In [13]: crime['Offence Count'].describe()

Out[13]: count    385296.000000
         mean          1.164871
         std           0.560723
         min           1.000000
         25%           1.000000
         50%           1.000000
         75%           1.000000
         max          28.000000
         Name: Offence Count, dtype: float64
```

To find the statistics for the “Offence Count” column, I used .describe() which is a convenient function which computes a variety of summary statistics on that column.

Therefore, I can see that:

Count: 385296

Mean: 1.164871

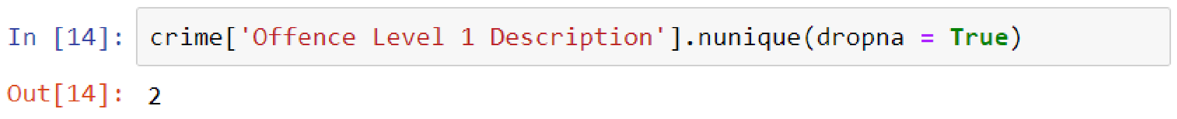
Standard Deviation: 0.560723

Minimum: 1.000000

Maximum: 28.000000

A5. Exploring Offence Level 1 Description

1)

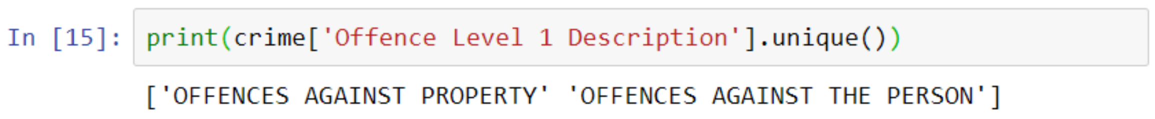


```
In [14]: crime['Offence Level 1 Description'].nunique(dropna = True)

Out[14]: 2
```

.nunique returns a number of distinct observations

Therefore, we can see that there are 2 unique values in the “Offence Level 1 Description” column.

2)

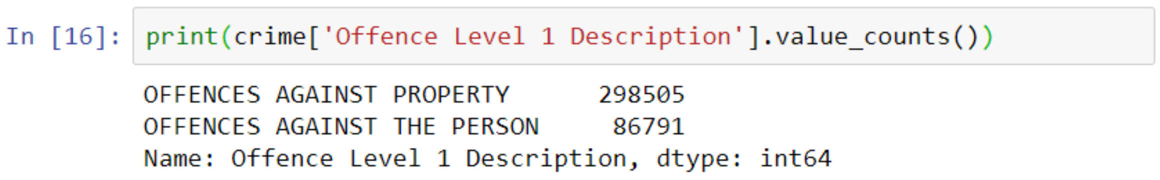


```
In [15]: print(crime['Offence Level 1 Description'].unique())

         ['OFFENCES AGAINST PROPERTY' 'OFFENCES AGAINST THE PERSON']
```

.unique displays the unique values into a list

The unique values of level 1 offences is OFFENCES AGAINST PROPERTY and OFFENCES AGAINST THE PERSON.

3)

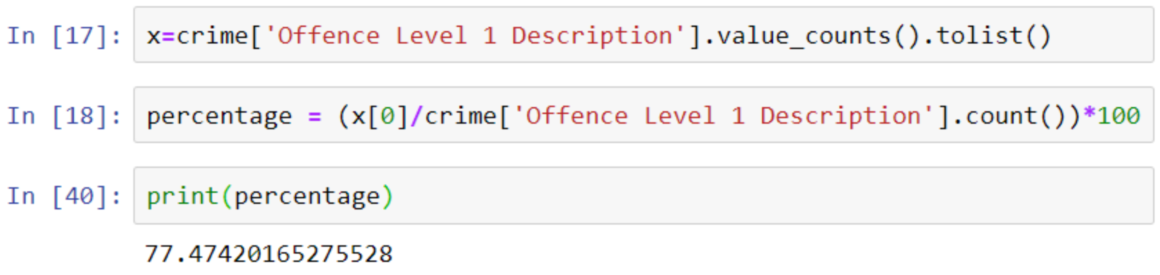


```
In [16]: print(crime['Offence Level 1 Description'].value_counts())

         OFFENCES AGAINST PROPERTY      298505
         OFFENCES AGAINST THE PERSON     86791
         Name: Offence Level 1 Description, dtype: int64
```

.value_counts() displays the count of all unique values

Therefore, I can see that “offences against the person” has 86791 records

4)



```
In [17]: x=crime['Offence Level 1 Description'].value_counts().tolist()

In [18]: percentage = (x[0]/crime['Offence Level 1 Description'].count())*100

In [40]: print(percentage)

         77.47420165275528
```

Firstly, I took the value_counts of the column ‘offence level 1 description’ and put them to a list using .tolist() then assign it to a variable x.

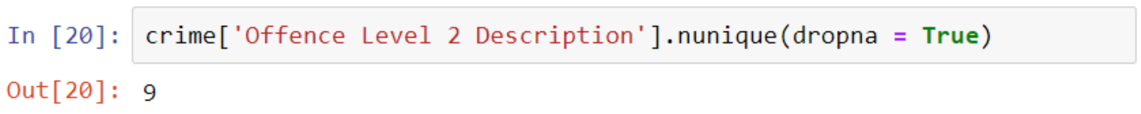
After that, I took the first index of x which is the value count of ‘offences against property’, divide it with the count of the ‘offence level 1 description’ column by using .count() and then multiplying it with 100 then assigning it to a variable called percentage.

Therefore, we can see that:

Percentage of records that are 'offences against the property': 77.47420165275528%

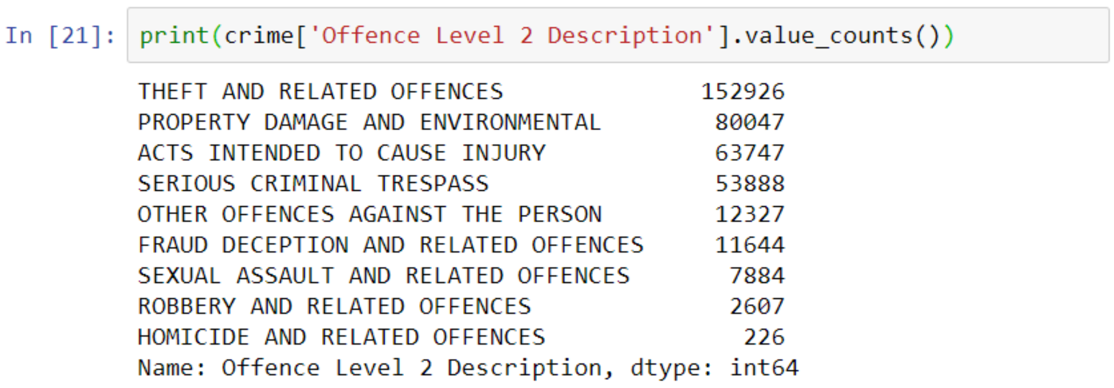
A6. Exploring Offence Level 2 Description

1)


```
In [20]: crime['Offence Level 2 Description'].nunique(dropna = True)
Out[20]: 9
```

.nunique returns a number of distinct observations

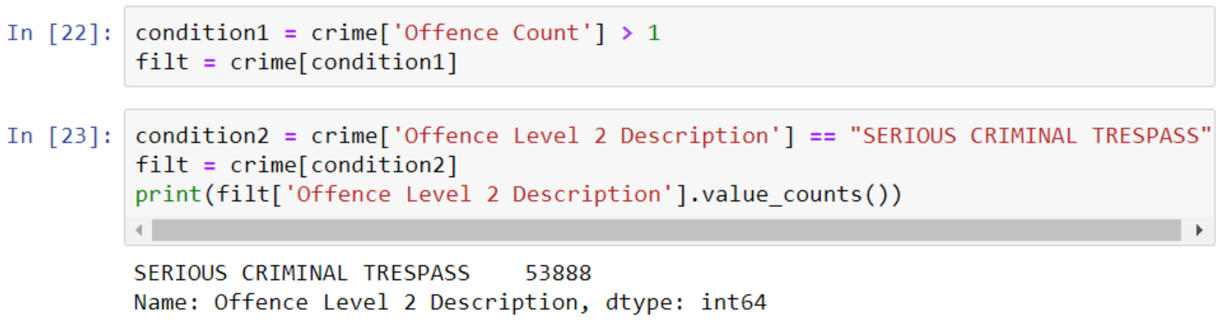
Therefore, we can see that there are 9 unique values in the “Offence Level 2 Description” column.


```
In [21]: print(crime['Offence Level 2 Description'].value_counts())

         THEFT AND RELATED OFFENCES              152926
         PROPERTY DAMAGE AND ENVIRONMENTAL        80047
         ACTS INTENDED TO CAUSE INJURY            63747
         SERIOUS CRIMINAL TRESPASS                53888
         OTHER OFFENCES AGAINST THE PERSON        12327
         FRAUD DECEPTION AND RELATED OFFENCES     11644
         SEXUAL ASSAULT AND RELATED OFFENCES       7884
         ROBBERY AND RELATED OFFENCES              2607
         HOMICIDE AND RELATED OFFENCES              226
         Name: Offence Level 2 Description, dtype: int64
```

.value_counts() displays the count of all unique values

Therefore, the code above prints out the unique values of level 2 offences together with their counts.

2)


```
In [22]: condition1 = crime['Offence Count'] > 1
         filt = crime[condition1]

In [23]: condition2 = crime['Offence Level 2 Description'] == "SERIOUS CRIMINAL TRESPASS"
         filt = crime[condition2]
         print(filt['Offence Level 2 Description'].value_counts())

         SERIOUS CRIMINAL TRESPASS    53888
         Name: Offence Level 2 Description, dtype: int64
```

First, I made a condition to take the ‘Offence Count’ that is more than one and assigned it to a variable called condition1.

Then, I implement condition1 into the crime dataset and assigned it to a variable called filt.

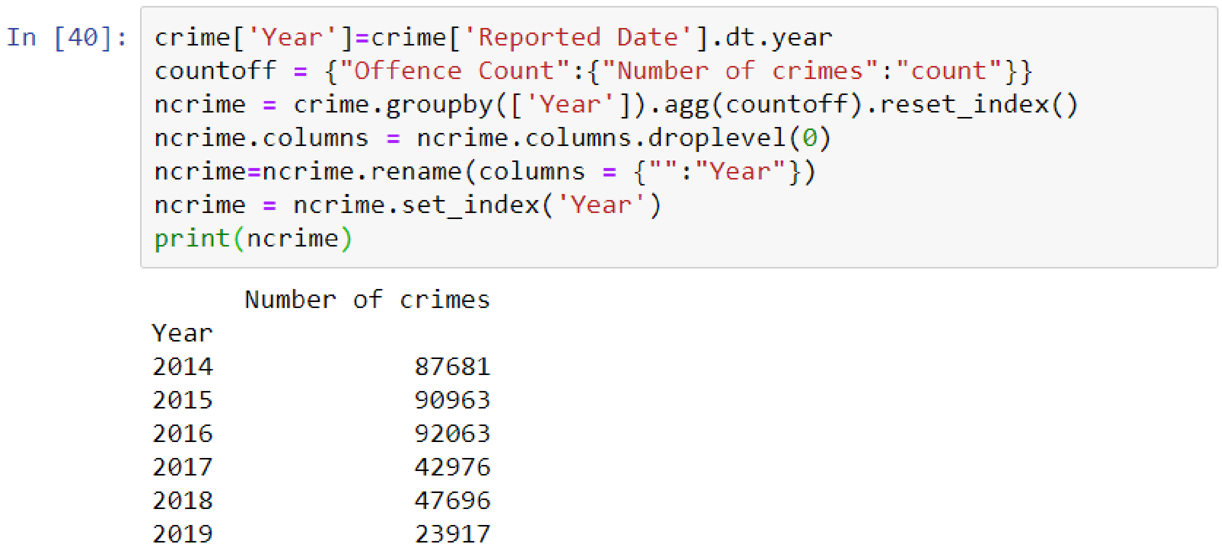
After that, I made another condition to take all ‘offence level 2 description’ that is equal to “SERIOUS CRIMINAL TRESPASS” and assigned it to a variable called filt.

Then I implement condition2 into the crime dataset and assigned it to the filt variable again.

Therefore, we can see that there are 53888 serious criminal trespasses that have occurred with more than 1 offence count.

Investigating Offence Count in different suburbs and different years

B1. Investigating the number of crimes per year



```
In [40]: crime['Year']=crime['Reported Date'].dt.year
         countoff = {"Offence Count":{"Number of crimes":"count"}}
         ncrime = crime.groupby(['Year']).agg(countoff).reset_index()
         ncrime.columns = ncrime.columns.droplevel(0)
         ncrime=ncrime.rename(columns = {"":"Year"})
         ncrime = ncrime.set_index('Year')
         print(ncrime)
```

```
      Number of crimes
Year
2014             87681
2015             90963
2016             92063
2017             42976
2018             47696
2019             23917
```

Firstly, I extracted the ‘year’ from column ‘Reported Date’ using method .dt.year, which returns the year of the datetime from ‘Reported Date’.

Then, I took the “Offence Count” column because it’s the column where I want to apply the aggregation to, followed by the name of the resulting new column which is “Number of crimes” and the aggregation operation which is “count” and assign it to a variable called countoff.

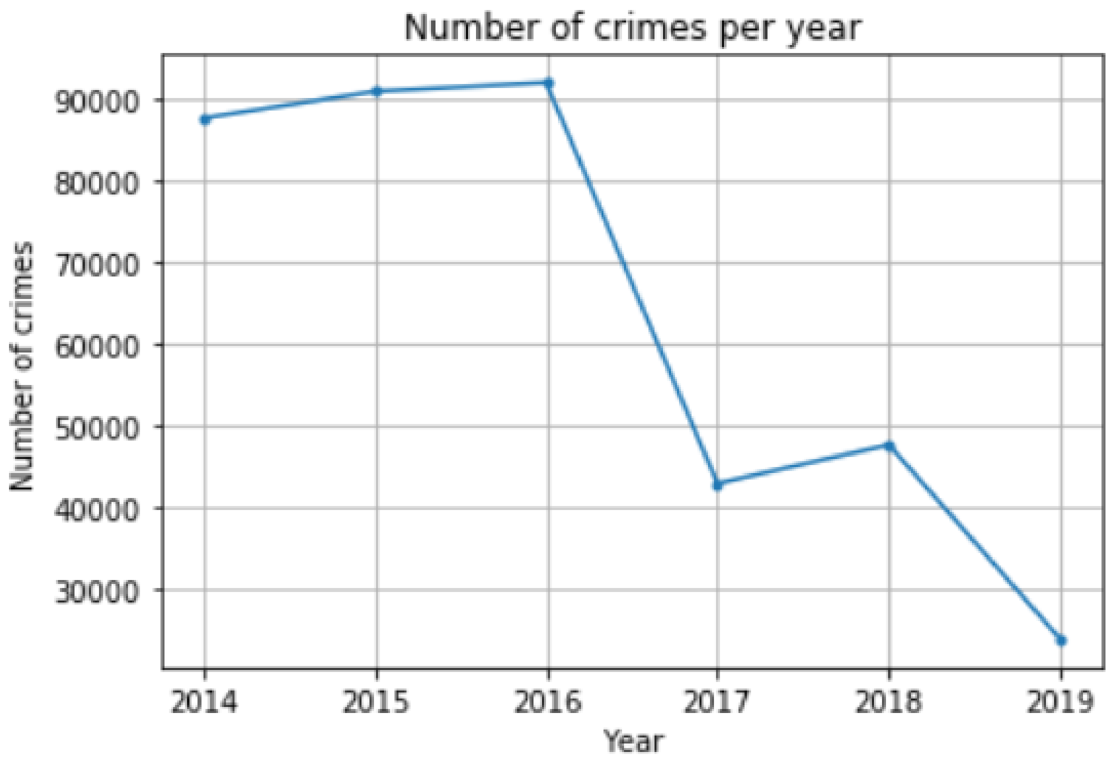
After that, I groupby year which combines the results of the each year and make it group into one row and applied the aggregation operation to it. I also "flatten it" using the 'reset_index()' and 'droplevel()' commands so that the output of the groupby operation can be further manipulated. I assigned these into a variable called ncrime.

Flattening will lose the column name for the 'Year' attribute. Therefore, I renamed the column by using .rename(columns = { ‘’ : ‘Year’})

Therefore, these codes will allow me to print out the number of crimes per year.

```
In [27]: import matplotlib.pyplot as plt
```

To plot the graph, I first need to import the matplotlib.pyplot library and with an alias plt.



plt.plot() is used to plot a line graph.

I chose to plot a line graph because it can clearly visualize the value of the number of crimes over the years. It also helps me to determine the relationship between the two variables.

The graph that I plot shows the number of crimes over a 5-year period from 2014 to 2019. The horizontal axis represents the years while the vertical axis shows the number of crimes.

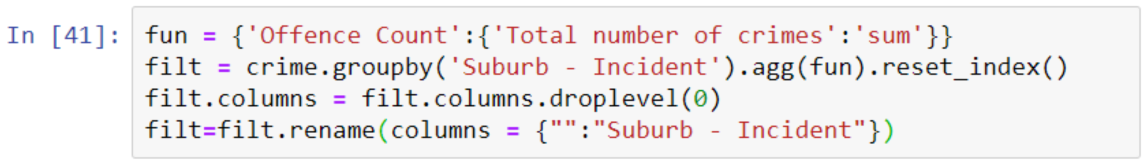
The graph indicates that the number of crimes each year have fluctuated. There was a increase in the number of crimes from 2014 to 2016, and 2017 to 2018 but generally, the number of crimes have significantly decrease over the period.

From the graph, it is clear to see that the highest number of crime happened in the year 2016, which is around 92,000 crimes, as the number of crimes slowly increase from 2014 to 2016. However, after 2016, we could see the number of crimes declined dramatically by around 49,000 crimes, before a slight increase of around 5000 crimes from 2017 to 2018. After that we see another decrease in the number of crimes by around 24,000 crimes from 2018 to 2019.

Overall, the number of crimes is seen to be generally decreasing over time.

B2. Investigating the total number of crimes in different suburbs

1)



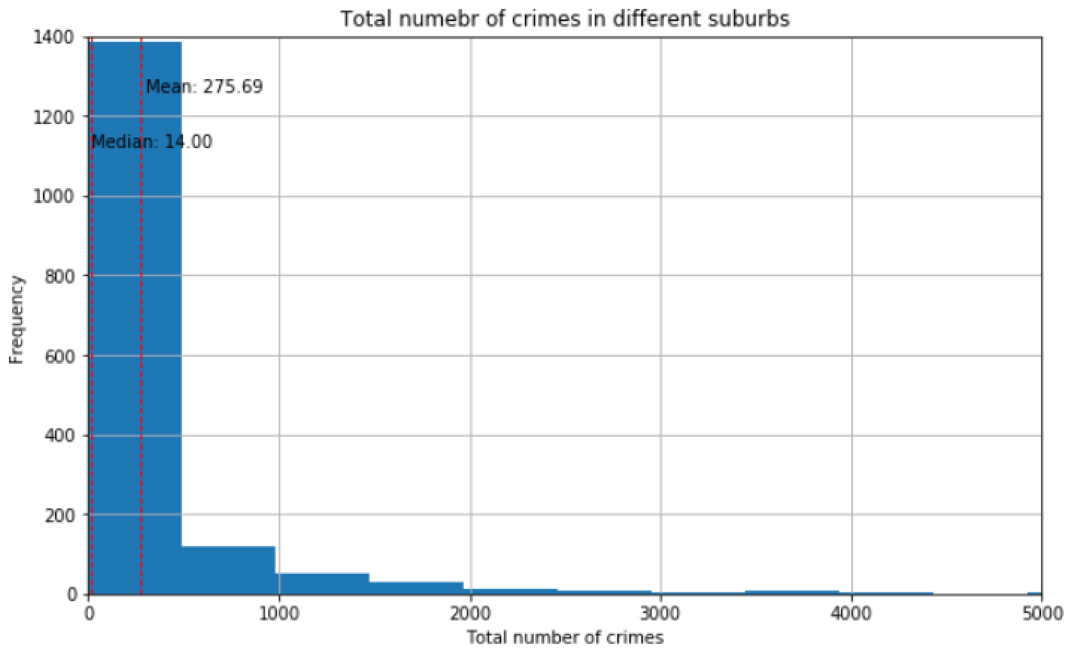
```
In [41]: fun = {'Offence Count':{'Total number of crimes':'sum'}}
         filt = crime.groupby('Suburb - Incident').agg(fun).reset_index()
         filt.columns = filt.columns.droplevel(0)
         filt=filt.rename(columns = {"":"Suburb - Incident"})
```

Firstly, I took the “Offence Count” column because it’s the column where I want to apply the aggregation to, followed by the name of the resulting new column which is “Total number of crimes” and the aggregation operation which is “sum” and assign it to a variable called fun.

After that, I groupby ‘Suburb – Incident’ which combines the each Suburb - Incident and make it group into one row and applied the aggregation operation to it. I also "flatten it" using the 'reset_index()' and 'droplevel()' commands so that the output of the groupby operation can be further manipulated. I assigned these into a variable called filt.

Flattening will lose the column name for the ‘Suburb – Incident’ attribute. Therefore, I renamed the column by using .rename(columns = { ‘’ : ‘Suburb – Incident’}).

Therefore, these codes will allow me to compute the total number of crimes in each suburb.



.hist () is used to plot a histogram.

The graph that I plot shows the total number of crimes in different suburbs. The horizontal axis represents the total number of crimes while the vertical axis shows the frequency.

The graph indicates that the total number of crimes between 0 to 500 occurs the most in different suburbs.

2)

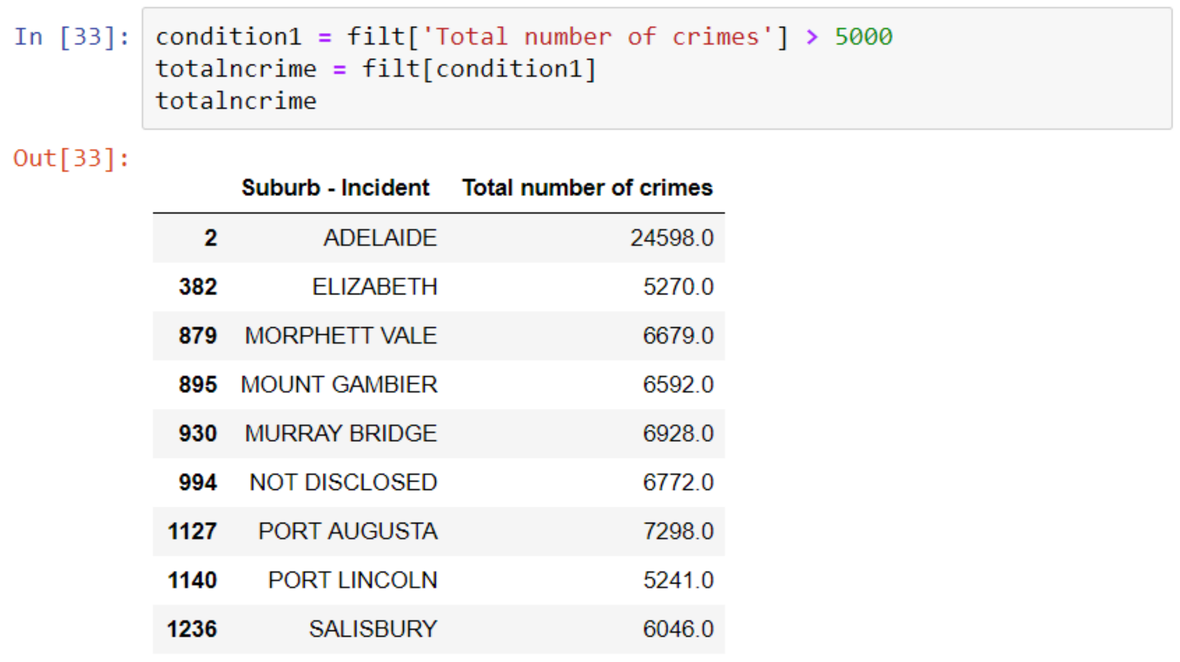
```
In [31]: x.mean()

Out[31]: 275.6879606879607

In [32]: x.median()

Out[32]: 14.0
```

Mean: 275. 6879606879607

Median: 14.0

The histogram is skewed right because the mean is greater than the median. This is because skewed-right data have a few large values that drive the mean upward but do not affect where the exact middle of the data is.

3)



```
In [33]: condition1 = filt['Total number of crimes'] > 5000
         totalncrime = filt[condition1]
         totalncrime
```

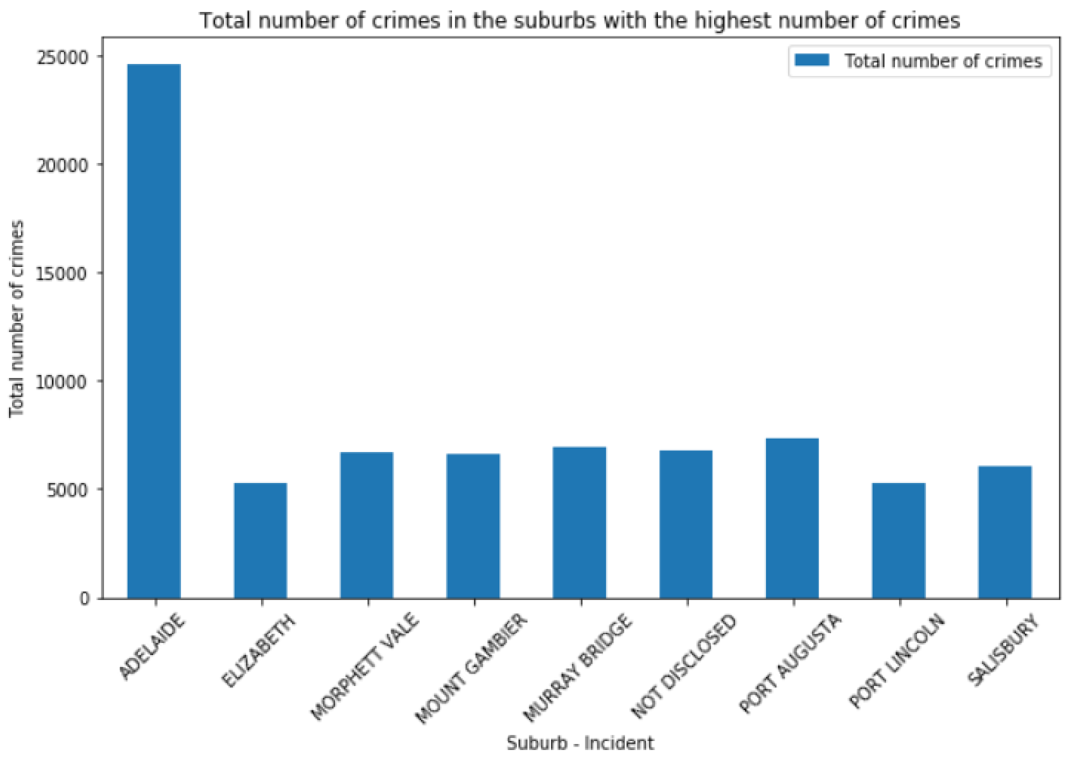
Out[33]:

	Suburb - Incident	Total number of crimes
2	ADELAIDE	24598.0
382	ELIZABETH	5270.0
879	MORPHETT VALE	6679.0
895	MOUNT GAMBIER	6592.0
930	MURRAY BRIDGE	6928.0
994	NOT DISCLOSED	6772.0
1127	PORT AUGUSTA	7298.0
1140	PORT LINCOLN	5241.0
1236	SALISBURY	6046.0

First, I made a condition to take the 'Total number of crimes' from filt that is more than 5000 and assigned it to a variable called condition1.

Then, I implement condition1 into the fillt dataset and assigned it to a variable called totalncrime.

Therefore, this shows me the suburbs which has the total number of crimes that are greater than 5000.



.plot.bar() is used to plot a bar graph.

The graph that I plot shows the total number of crimes in the suburbs with the highest number of crimes. The horizontal axis represents the Surburb - Incident while the vertical axis shows the total number of crimes.

The graph indicates that the highest number of crimes occur in Adelaide whereas the lowest number of crimes occur in Port Lincoln.

B3. Daily number of crimes

1)

```
In [43]: filt2 = crime.groupby(['Reported Date','Suburb - Incident'])['Offence Count'].sum().reset_index()
         fun = {'Offence Count':{" ": lambda x:x[x >=15].count()}}
         filt3 = filt2.groupby('Suburb - Incident').agg(fun).reset_index()
         filt3.head()
```

Out[43]:

	Suburb - Incident	Offence Count
0	ABERFOYLE PARK	0.0
1	ADDRESS UNKNOWN	0.0
2	ADELAIDE	877.0
3	ADELAIDE AIRPORT	0.0
4	AGERY	0.0

Firstly, I groupby ‘Reported Date’ and ‘Suburb – Incident’ which combines the each Suburb - Incident and Reported date with the offence count and sum them together using the .sum() method. I also "flatten it" using the 'reset_index()' so that the output of the groupby operation can be further manipulated. I assigned these into a variable called filt2

Then, I took the “Offence Count” column because it’s the column where I want to apply the aggregation to, follow by an anonymous function (called a lambda function) in Python which check for count that is more than equal to 15 and the aggregation operation which is “count” and assign it to a variable called fun.

After that, I groupby ‘Suburb – Incident’ which combines the each Suburb - Incident and make it group into one row and applied the aggregation operation to it. I also "flatten it" using the 'reset_index()' so that the output of the groupby operation can be further manipulated. I assigned these into a variable called filt3.

Therefore, these codes will allow me to calculate the number of days that at least 15 crimes have occurred per day for each suburb.

2)

```
In [36]: condition2 = filt2['Offence Count'] >= 15
         filt3 = filt2[condition2]
         filt3 = filt3.groupby('Suburb - Incident')['Offence Count'].count().reset_index(name ='Number of days')

         filt3
```

Out[36]:

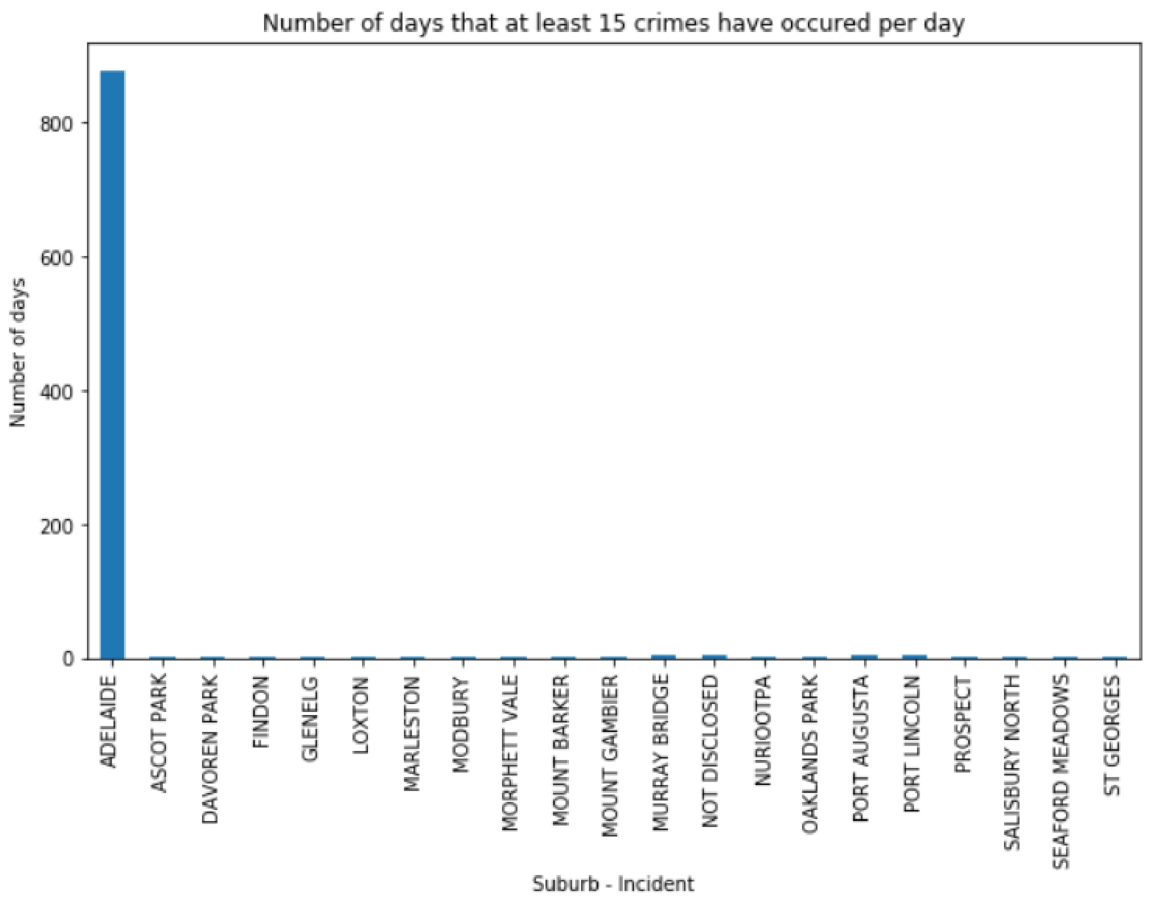
	Suburb - Incident	Number of days
0	ADELAIDE	877
1	ASCOT PARK	1
2	DAVOREN PARK	1
3	FINDON	1
4	GLENELG	1
5	LOXTON	1
6	MARLESTON	1
7	MODBURY	1
8	MORPHETT VALE	3
9	MOUNT BARKER	1
10	MOUNT GAMBIER	3
11	MURRAY BRIDGE	5
12	NOT DISCLOSED	5
13	NURIOOTPA	1
14	OAKLANDS PARK	3
15	PORT AUGUSTA	4
16	PORT LINCOLN	5
17	PROSPECT	2
18	SALISBURY NORTH	1
19	SEAFORD MEADOWS	1
20	ST GEORGES	1

First, I made a condition to take the 'Offence Count' from filt2 that is at least 15 and assigned it to a variable called condition2.

Then, I implement condition2 into the fillt2 dataset and assigned it to a variable called filt3.

After that, I groupby 'Suburb – Incident' which combines the each Suburb - Incident and make it group into one row with the offence count and count them together using the .count() method. I also "flatten it" using the 'reset_index()' so that the output of the groupby operation can be further manipulated and named it number of days. I assigned these into a variable called filt3.

Therefore, this shows me the suburbs which has at least one day where the daily number of crimes that is at least 15 crimes.

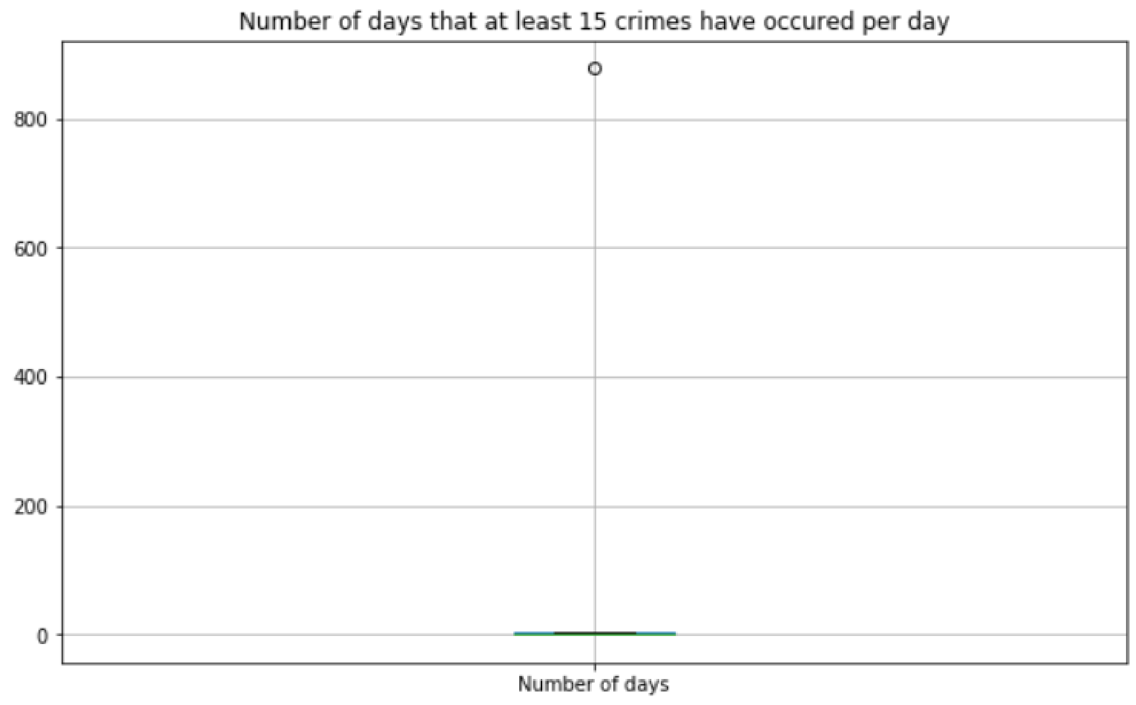


.plot.bar() is used to plot a bar graph.

The graph that I plot shows the number of days that at least 15 crimes have occurred per day. The horizontal axis represents the Surburb - Incident while the vertical axis shows the number of days.

The graph indicates that the highest number of days that at least 15 crimes have occurred per day occur in Adelaide and is the number of days is very high compared to the other suburbs until the number of days of other suburbs is not that clear to interpret.

3)



.boxplot is used to plot the boxplot

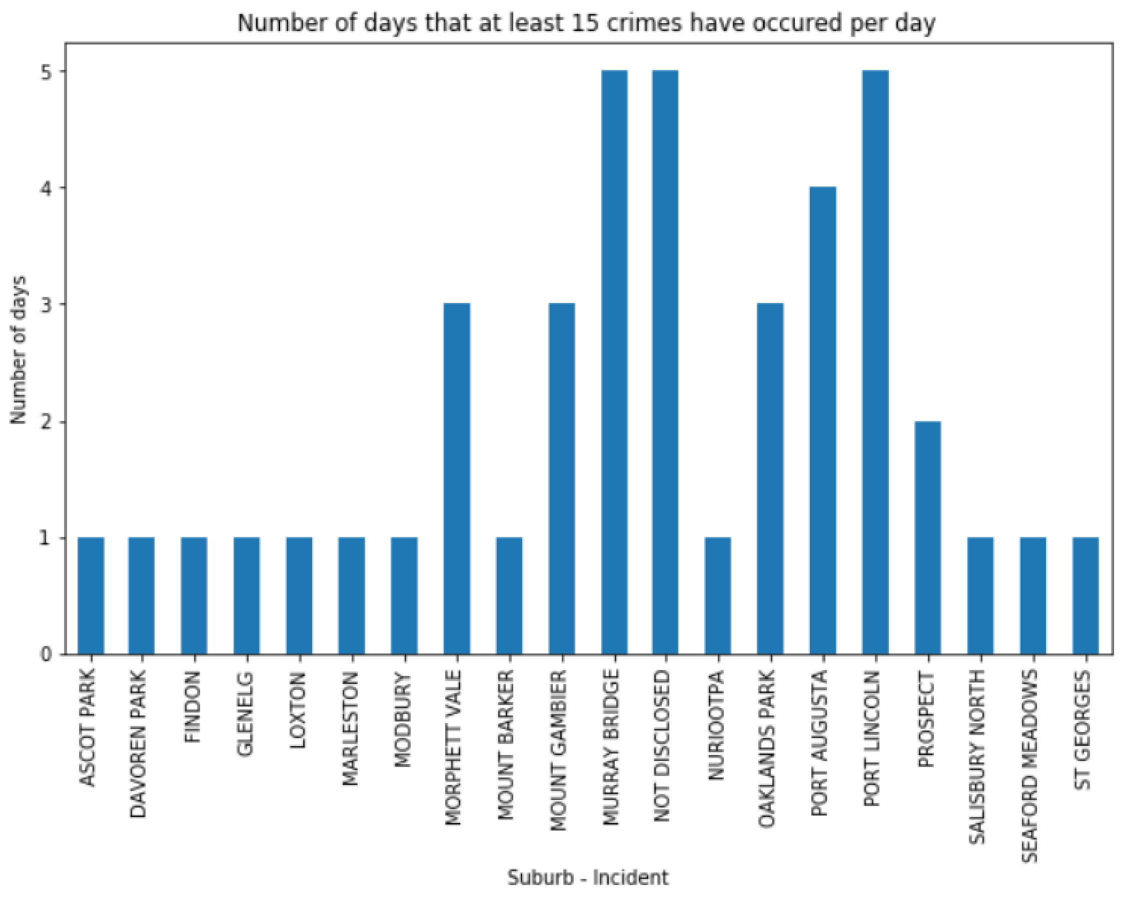
From the boxplot above, we can clearly see the outlier which is the number of days that is more than 800.

```
In [37]: outlier = ~(filt3['Number of days'] > 160)
         filt3 = filt3[outlier]
```

First, I made a condition to **take out** the 'Number of days' from filt3 that is more than 160 and assigned it to a variable called outlier.

Then, I implement the outlier into the fillt3 dataset and assigned it to filt3.

Therefore, this removes the outlier on filt3.



.plot.bar() is used to plot a bar graph.

The graph that I plot shows the number of days that at least 15 crimes have occurred per day. The horizontal axis represents the Surburb - Incident while the vertical axis shows the number of days.

The graph indicates that the highest number of days that at least 15 crimes have occurred per day after taking out the outlier occur in Murray Bridge, Not Disclosed and Port Lincoln which is 5 days while the lowest occur in Ascot Park, Davoren Park, Findon, Glenelg, Loxton, Marleston, Modbury, Mount Gambier, Nuriootpa, Salisbury North, Seaford Meadows and St Georges which is 1 day only.

4)

The bar graph in step 3 is easier to interpret compared to the bar graph in step 2. This is because there is an outlier in the bar graph of step 2, which is Adelaide's number of days. As a result the vertical axis of the bar graph would go up to 877 which causes the other values to not be clearly visible as the other values are way smaller than 877. In bar graph step 3, I have removed the outlier which causes the limit of the vertical axis to decrease significantly. Thus, this has enabled us to interpret the number of days more clearly as we could see the number of days that at least 15 crimes have occurred per day.