GEOSPATIAL THEFT POINT PATTARN ANALYSIS AND PREDICTION IN VANCOUVER

- # Importing the need libraries

In [1]:
```
import numpy as np
import pandas as pd
import geopandas as gpd
import pysal
import seaborn as sns
import contextily as ctx
import calendar
from pointpats import centrography
import matplotlib.pyplot as plt
from sklearn.cluster import DBSCAN
from scipy import stats
import statsmodels.api as sm
from statsmodels.formula.api import ols
from statsmodels.stats.multicomp import pairwise_tukeyhsd
import warnings
warnings.filterwarnings("ignore")
```

```
C:\Users\ababi\anaconda3\envs\geo_env\Lib\site-packages\pysal\explore\segregation\ne
twork\network.py:15: UserWarning: You need pandana and urbanaccess to work with segr
egation's network module
You can install them with  `pip install urbanaccess pandana` or `conda install -c ud
st pandana urbanaccess`
  warn(
C:\Users\ababi\anaconda3\envs\geo_env\Lib\site-packages\pysal\model\spvcm\abstracts.
py:10: UserWarning: The `dill` module is required to use the sqlite backend fully.
  from .sqlite import head_to_sql, start_sql
```

In [2]:
```
#reading in the vancouver crime data obtain from kaggle between 2003-2017
df = pd.read_csv("crime.csv (1).zip")
```

In [3]:
```
#checking the first five rows/columns
df.head()
```

Out[3]:

	TYPE	YEAR	MONTH	DAY	HOUR	MINUTE	HUNDRED_BLOCK	NEIGHBOURHOOD	
0	Other Theft	2003	5	12	16.0	15.0	9XX TERMINAL AVE	Strathcona	49
1	Other Theft	2003	5	7	15.0	20.0	9XX TERMINAL AVE	Strathcona	49
2	Other Theft	2003	4	23	16.0	40.0	9XX TERMINAL AVE	Strathcona	49
3	Other Theft	2003	4	20	11.0	15.0	9XX TERMINAL AVE	Strathcona	49
4	Other Theft	2003	4	12	17.0	45.0	9XX TERMINAL AVE	Strathcona	49

- # Exploratory Data Analysis (EDA)

In [4]:

```
#checking the crime data information especially it types
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 530652 entries, 0 to 530651
Data columns (total 12 columns):
 #   Column         Non-Null Count   Dtype  
---  ------         --------------   -----  
 0   TYPE           530652 non-null  object 
 1   YEAR           530652 non-null  int64  
 2   MONTH          530652 non-null  int64  
 3   DAY            530652 non-null  int64  
 4   HOUR           476290 non-null  float64
 5   MINUTE         476290 non-null  float64
 6   HUNDRED_BLOCK  530639 non-null  object 
 7   NEIGHBOURHOOD  474028 non-null  object 
 8   X              530652 non-null  float64
 9   Y              530652 non-null  float64
 10  Latitude       530652 non-null  float64
 11  Longitude      530652 non-null  float64
dtypes: float64(6), int64(3), object(3)
memory usage: 48.6+ MB
```

In [5]:

```
#checking the number of null/empty values in various columns
df.isnull().sum() /len(df) * 100
```

Out[5]:
```
TYPE              0.000000
YEAR              0.000000
MONTH             0.000000
DAY               0.000000
HOUR             10.244379
MINUTE           10.244379
HUNDRED_BLOCK     0.002450
NEIGHBOURHOOD    10.670647
X                 0.000000
Y                 0.000000
Latitude          0.000000
Longitude         0.000000
dtype: float64
```

In [6]:
```
#checking number of duplicates values in the entire dataset
df.duplicated().sum()
```

Out[6]:
```
48838
```

In [7]:
```
#most columns has null values less than 15% so we decided to drop all null values
df = df.dropna()
```

In [8]:
```
#duplicates values we drop to ensure quality data for future  analysis
df.drop_duplicates(inplace = True)
```

In [9]:
```
#lets re-check the duplicates
df.duplicated().sum()
```

Out[9]:
```
0
```

In [10]:
```
# we load in demographic data containing all population in each cities in Vancouver
data = pd.read_csv('CensusLocalAreaProfiles2016.csv',  encoding='latin-1', skiprows
```

In [11]:
```
#checking the first five rows and columns in the population data
data.head()
```

Out[11]:

	ID	Variable	Arbutus-Ridge	Downtown	Dunbar-Southlands	Fairview	Grandview-Woodland	Hastings-Sunrise	Kensi C
0	1	Total - Age groups and average age of the pop...	15,295	62,030	21,425	33,620	29,175	34,575	
1	2	0 to 14 years	2015	4000	3545	2580	3210	4595	
2	3	0 to 4 years	455	2080	675	1240	1320	1510	
3	4	5 to 9 years	685	1105	1225	760	1025	1560	
4	5	10 to 14 years	880	810	1650	580	865	1525	

5 rows × 26 columns

In [12]:

```
#lets check the total number of rows and columns in the population dataset
data.shape
```

Out[12]: (5589, 26)

In [ ]:

```
data.head()
```

Out[ ]:

	ID	Variable	Arbutus-Ridge	Downtown	Dunbar-Southlands	Fairview	Grandview-Woodland	Hastings-Sunrise	Kensi... C
0	1	Total - Age groups and average age of the pop...	15,295	62,030	21,425	33,620	29,175	34,575	
1	2	0 to 14 years	2015	4000	3545	2580	3210	4595	
2	3	0 to 4 years	455	2080	675	1240	1320	1510	
3	4	5 to 9 years	685	1105	1225	760	1025	1560	
4	5	10 to 14 years	880	810	1650	580	865	1525	

5 rows × 26 columns

- **from the population data we could see that the first row contains the sum of all the population in each municipalities, so we proceed to filter the first row to further our analysis.**

In [14]:
```
# we use the iloc to filter only the first row
data = data.iloc[[0]]
```

In [15]:
```
# The code transpose the filtered row and columns names were rename
data_1 = data.T
data_1.reset_index(inplace = True)
data_1.column = ['Municipalities', 'Population']
```

In [16]:
```
#This transposed data were save to csv for future analysis
town = pd.read_csv("town.csv", skiprows = 2)
```

In [17]:
```
town.head()
```

Out[17]:

	1	Variable	Total - Age groups and average age of the population - 100% data
0	2	Arbutus-Ridge	15,295
1	3	Downtown	62,030
2	4	Dunbar-Southlands	21,425
3	5	Fairview	33,620
4	6	Grandview-Woodland	29,175

In [18]:
```
# so the columns for the town dataframe was rename to our desire names
dataframe = town.rename(columns={
    '1': 'index',
    'Variable': 'NEIGHBOURHOOD',
    ' Total - Age groups and average age of the population - 100% data ': 'populati
})
```

In [19]:
```
#checkout our fresh municipal and population dataframe
dataframe.head()
```

Out[19]:

	index	NEIGHBOURHOOD	population
0	2	Arbutus-Ridge	15,295
1	3	Downtown	62,030
2	4	Dunbar-Southlands	21,425
3	5	Fairview	33,620
4	6	Grandview-Woodland	29,175

In [20]:
```
#check the tail to ensure there is no aggragation at the tail
dataframe.tail()
```

Out[20]:

	index	NEIGHBOURHOOD	population
19	21	Victoria-Fraserview	31,065
20	22	West End	47,200
21	23	West Point Grey	13,065
22	24	Vancouver CSD	631,485
23	25	Vancouver CMA	2,463,430

- **row 22 and 23 is an aggragated rows so the proceeding code will drop it for further analysis**

In [21]:
```
town_m = dataframe.drop([22, 23])
```

In [22]:
```
town_m.tail()
```

Out[22]:

	index	NEIGHBOURHOOD	population
17	19	Strathcona	12,585
18	20	Sunset	36,500
19	21	Victoria-Fraserview	31,065
20	22	West End	47,200
21	23	West Point Grey	13,065

In [23]:
```
# a function was design to strip all spaces from the columns
def strip_spaces(df, column_name):
    df[column_name] = df[column_name].apply(lambda x: str(x).replace(" ", ""))
    return df
```

In [24]:
```
df_stripped = strip_spaces(df, 'NEIGHBOURHOOD')
```

In [25]:
```
town_df = strip_spaces(town_m, 'NEIGHBOURHOOD')
```

In [26]:
```
# the crime and population dataframe were merge together using the neighbourhood
data_1 = pd.merge(df_stripped, town_df, left_on='NEIGHBOURHOOD', right_on='NEIGHBOU
```

In [27]:
```
data_1.head()
```

Out[27]:

	TYPE	YEAR	MONTH	DAY	HOUR	MINUTE	HUNDRED_BLOCK	NEIGHBOURHOOD	
0	Other Theft	2003	5	12	16.0	15.0	9XX TERMINAL AVE	Strathcona	4(
1	Other Theft	2003	5	7	15.0	20.0	9XX TERMINAL AVE	Strathcona	4(
2	Other Theft	2003	4	23	16.0	40.0	9XX TERMINAL AVE	Strathcona	4(
3	Other Theft	2003	4	20	11.0	15.0	9XX TERMINAL AVE	Strathcona	4(
4	Other Theft	2003	4	12	17.0	45.0	9XX TERMINAL AVE	Strathcona	4(

In [28]:
```
data_1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 352697 entries, 0 to 352696
Data columns (total 14 columns):
 #   Column          Non-Null Count   Dtype  
---  ------          --------------   -----  
 0   TYPE            352697 non-null  object 
 1   YEAR            352697 non-null  int64  
 2   MONTH           352697 non-null  int64  
 3   DAY             352697 non-null  int64  
 4   HOUR            352697 non-null  float64
 5   MINUTE          352697 non-null  float64
 6   HUNDRED_BLOCK   352697 non-null  object 
 7   NEIGHBOURHOOD   352697 non-null  object 
 8   X               352697 non-null  float64
 9   Y               352697 non-null  float64
 10  Latitude        352697 non-null  float64
 11  Longitude       352697 non-null  float64
 12  index           352697 non-null  int64  
 13  population      352697 non-null  object 
dtypes: float64(6), int64(4), object(4)
memory usage: 37.7+ MB
```

In [29]:
```
data_2 = strip_spaces(data_1, 'population')
```

In [30]:
```
# Clean and convert the "population" column to numeric
data_2['population'] = data_2['population'].str.replace(',', '').astype(float)
```

In [31]:
```
# we use the hundred_brlock as a unique column to count the crime in each municipal
crime_df = data_2["HUNDRED_BLOCK"].value_counts().rename_axis('HUNDRED_BLOCK').rese
crime_df.head()
```

Out[31]:

	HUNDRED_BLOCK	total_crime
0	6XX W 41ST AVE	1900
1	31XX GRANDVIEW HWY	1784
2	11XX ROBSON ST	1758
3	17XX E BROADWAY AVE	1718
4	3XX E BROADWAY AVE	1437

In [32]:
```
# this total crime is merge to the main that
new_geo=data_2.merge(crime_df,on='HUNDRED_BLOCK')
```

In [33]:
```
# a column of crime per 1000 population
new_geo['crime_per_pop_1000'] = (new_geo['total_crime'] / new_geo['population']) *
```

In [34]:
```
# month was modify to get the calender name of each month
new_geo['month_name'] = new_geo['MONTH'].apply(lambda x: calendar.month_name[x])
```

In [35]:
```
# convert pandas dataframe to geodataframe
geo_data = gpd.GeoDataFrame(new_geo,
```

```
                         crs='EPSG:4326',
                         geometry=gpd.points_from_xy(new_geo.Longitude, new_geo.Lat
```

In [36]:
```
geo_data.columns = geo_data.columns.str.lower()
```

In [37]:
```
geo_data.head()
```

Out[37]:

	type	year	month	day	hour	minute	hundred_block	neighbourhood	x	
0	Other Theft	2003	5	12	16.0	15.0	9XX TERMINAL AVE	Strathcona	493906.5	545
1	Other Theft	2003	5	7	15.0	20.0	9XX TERMINAL AVE	Strathcona	493906.5	545
2	Other Theft	2003	4	23	16.0	40.0	9XX TERMINAL AVE	Strathcona	493906.5	545
3	Other Theft	2003	4	20	11.0	15.0	9XX TERMINAL AVE	Strathcona	493906.5	545
4	Other Theft	2003	4	12	17.0	45.0	9XX TERMINAL AVE	Strathcona	493906.5	545

In [ ]:
```

```

In [38]:
```
# Replacing Male/MF  with Male and Female/F  with Female
geo_data["type"].replace(["Theft from Vehicle"],value = 'Theft from Vehicle' , inpl
geo_data["type"].replace(["Break and Enter Residential/Other"],value = "Residential
geo_data["type"].replace(["Mischief"],value = 'Mischief',inplace=True)
geo_data["type"].replace(["Theft of Vehicle"],value = 'Theft of Vehicle' , inplace=
geo_data["type"].replace(["Break and Enter Commercial"],value = 'Commecial Theft' ,
geo_data["type"].replace(["Theft of Bicycle"],value = 'Bicycle Theft' , inplace=Tru
geo_data["type"].replace(["Vehicle Collision or Pedestrian Struck (with Injury)"],v
geo_data["type"].replace(["Break and Enter Commercial"],value = 'Commecial Theft' ,
geo_data["type"].replace(["Vehicle Collision or Pedestrian Struck (with Fatality)"]
```

- # Quick Statistics

In [39]:
```
# Top 5 municipalities was filter for ANOVA analysis to check it statistical signif
names_to_filter = ['WestEnd', 'Fairview', 'MountPleasant','MountPleasant', 'Grandvi

# Use isin to filter the DataFrame
filtered_df = geo_data[geo_data['neighbourhood'].isin(names_to_filter)]
```

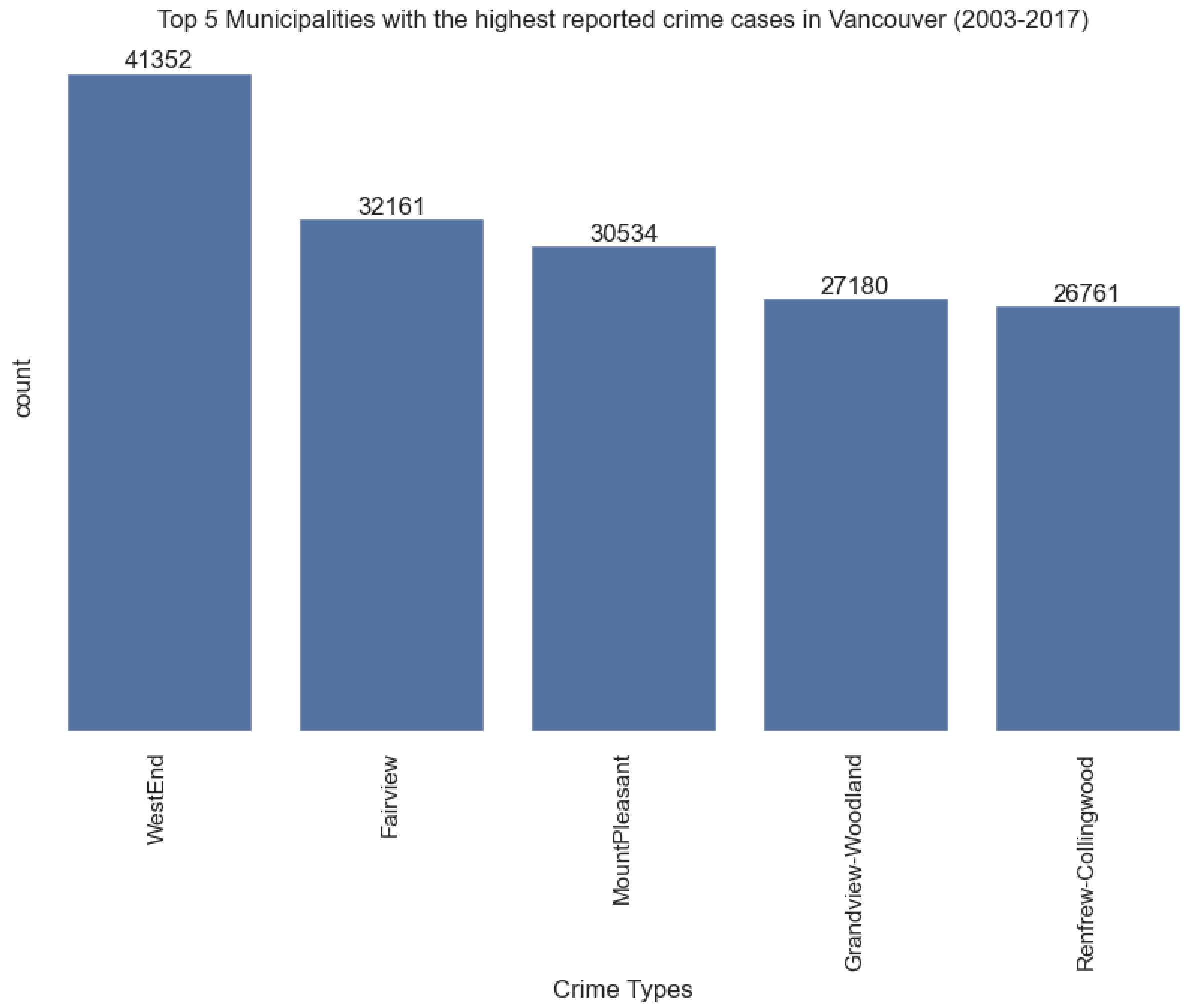
In [40]:
```
# Perform one-way ANOVA using crime_per_pop_1000 and the top 5 municipalities
model = ols('crime_per_pop_1000 ~ neighbourhood', data= filtered_df).fit()
anova_table = sm.stats.anova_lm(model, typ=2)

# Print ANOVA table
print(anova_table)

# Perform Tukey's HSD test for multiple comparisons
tukey_result = pairwise_tukeyhsd(endog=filtered_df['crime_per_pop_1000'], groups=fi

# Print the results
print(tukey_result)
```

```
                     sum_sq        df           F  PR(>F)
neighbourhood  1.858478e+05       3.0  865.285365     0.0
Residual       9.394783e+06  131223.0         NaN     NaN
            Multiple Comparison of Means - Tukey HSD, FWER=0.05
===========================================================================
      group1             group2        meandiff p-adj   lower  upper  reject
---------------------------------------------------------------------------
          Fairview Grandview-Woodland   -1.2691    0.0 -1.4482  -1.09   True
          Fairview      MountPleasant     0.116 0.3157 -0.0577 0.2896  False
          Fairview            WestEnd    1.9747    0.0  1.8131 2.1363   True
Grandview-Woodland      MountPleasant     1.385    0.0  1.2038 1.5663   True
Grandview-Woodland            WestEnd    3.2438    0.0  3.0741 3.4136   True
     MountPleasant            WestEnd    1.8588    0.0  1.6948 2.0228   True
---------------------------------------------------------------------------
```

- **The ANOVA is use to check the statistical significant between multiple values.**
- **The null hypothesis says that there is no statistical significance in theft in the top five municipalities in Vancouver.**
- **but according to the ANOVA report we reject the null hypothesis in five instance except on case between Fairview and MountPleasant were we fail to reject the null hypothesis, because there is no statistical significance in reported theft cases between these municipalities.**

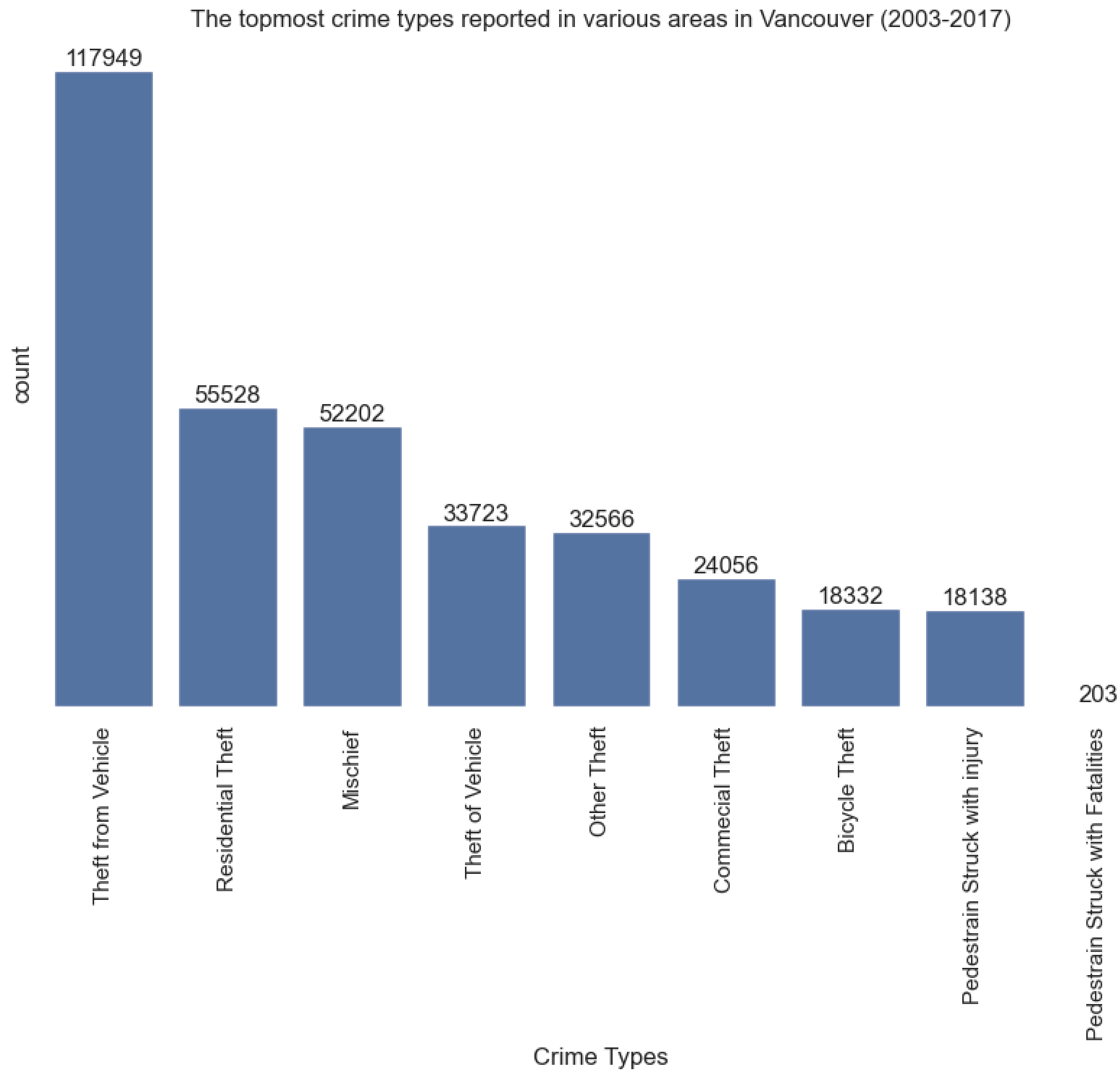
• Data Visualization

In [41]:
```
plt.figure(figsize=(10, 6))
sns.set(style='darkgrid')
ax = sns.countplot(x = 'neighbourhood',
              data = geo_data,
              order = geo_data['neighbourhood'].value_counts().nlargest().index)

ax.set(xlabel='Crime Types', yticks=[], title='Top 5 Municipalities with the highes
ax.bar_label(container=ax.containers[0])
plt.xticks(rotation=90)
plt.show()
```



- **From the above graph, we can observed that WestEnd, Fairview, MountPleasant, Grandview_Woodland and Renfrew-Collingwood are the dangrious municipalities inVancouver. These Municipalities attracts all sort of theft.**

In [42]:
```
plt.figure(figsize=(10, 6))
sns.set(style='darkgrid')
ax = sns.countplot(x = 'type',
              data = geo_data,
              order = geo_data['type'].value_counts().index)

ax.set(xlabel='Crime Types', yticks=[], title='The topmost crime types reported in
ax.bar_label(container=ax.containers[0])
plt.xticks(rotation=90)
plt.show()
```

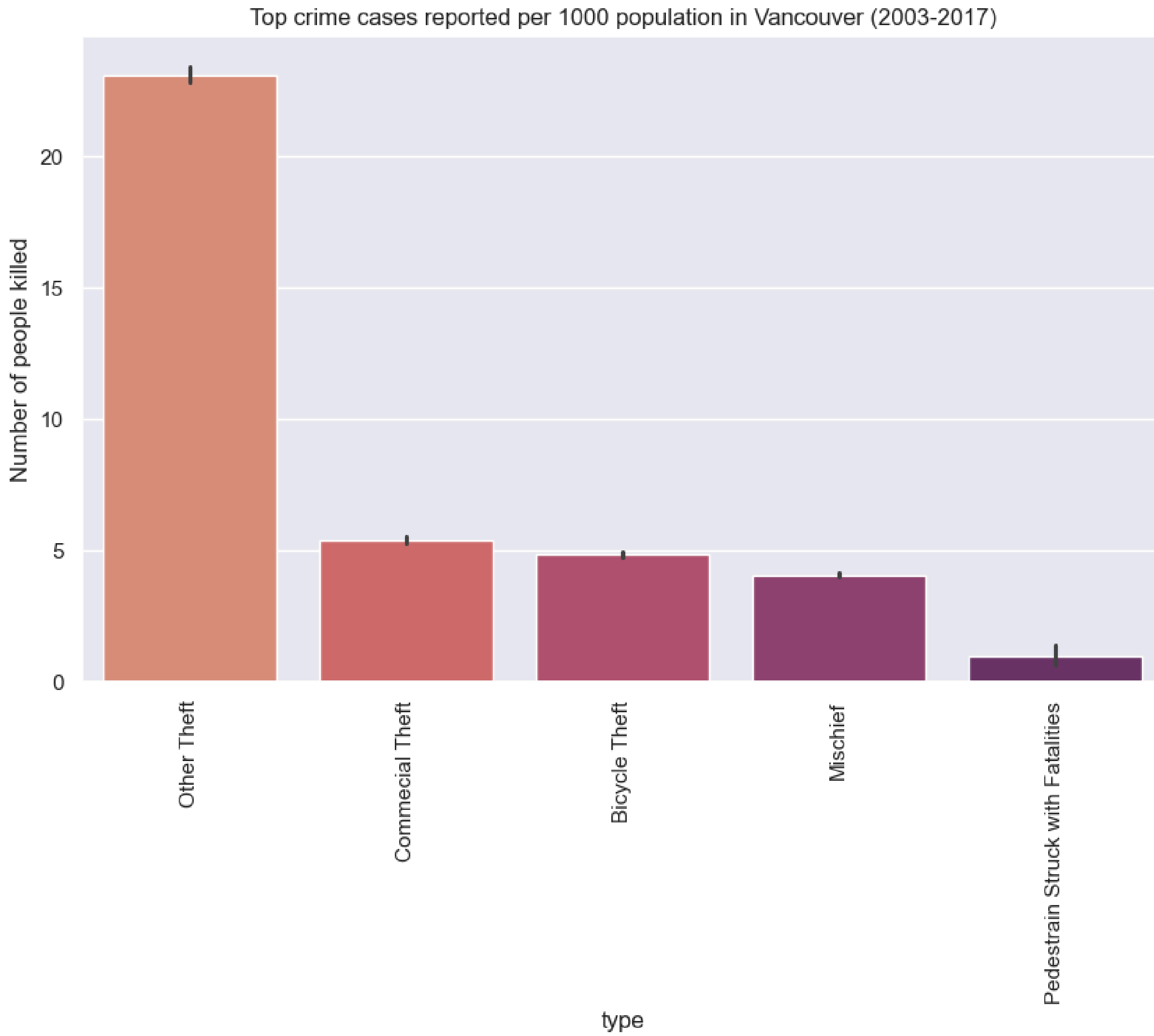


- **From the chart above we can observed that theft from vehicle, residential theft, mischief, theft of vehicle and other thefty are the most rampant theft cases reported in Vancouver between 2003 to 2017**

In [43]:

```
plt.figure(figsize=(10, 6))
# Create a Seaborn bar plot with hue
ax =sns.barplot(x="type",
            y="crime_per_pop_1000",
            data=geo_data,
            palette ="flare",
            order=geo_data.groupby("type")["crime_per_pop_1000"].mean().head(5).sor

# Set labels and title
plt.xlabel('type')
plt.ylabel('Number of people killed')
plt.title('Top crime cases reported per 1000 population in Vancouver (2003-2017)')

# Show the plot
```

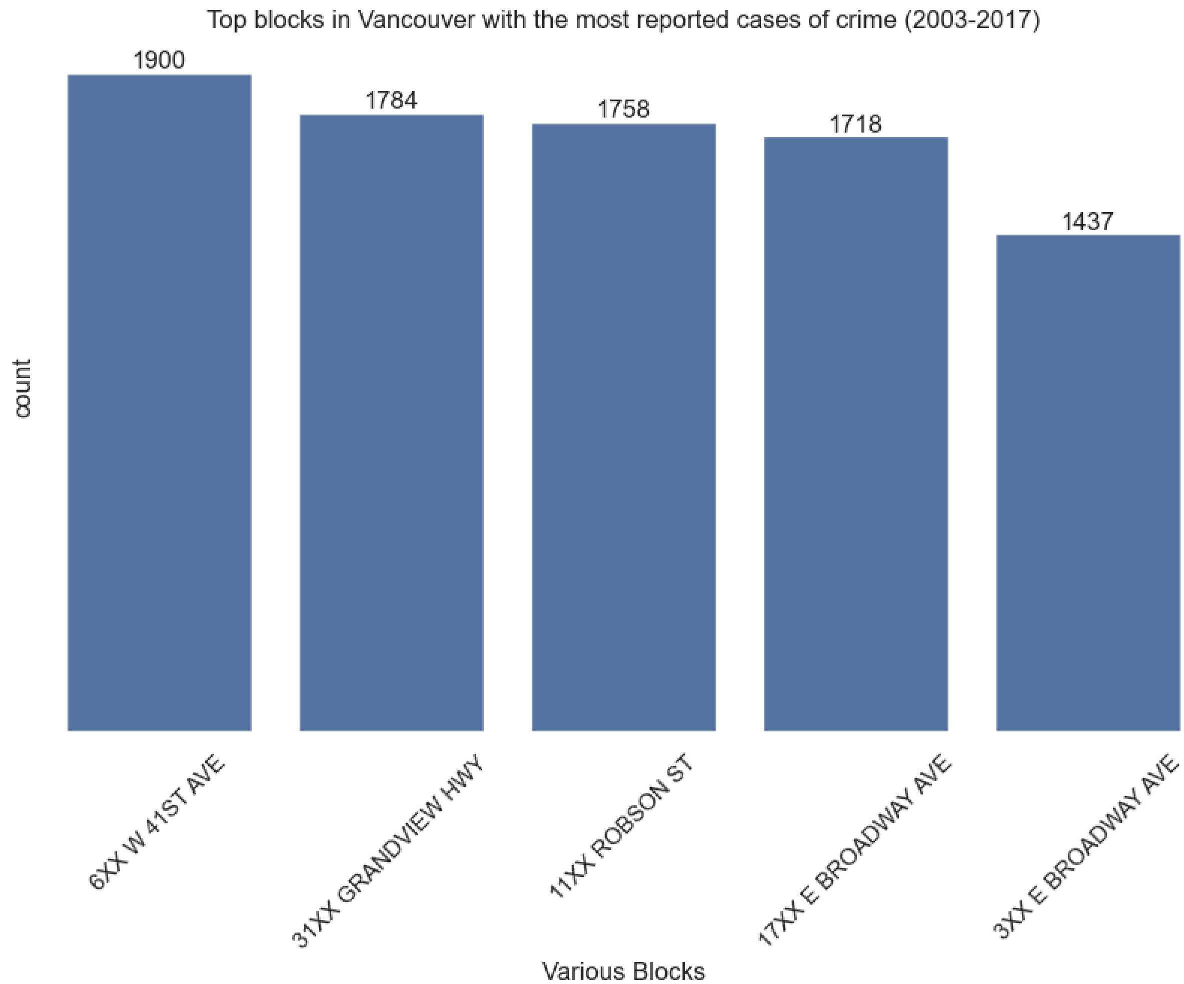
```python
plt.xticks(rotation=90)
plt.show()
```



- **Theft from Vehicle is the most reported theft case in Vancouver though, but from the graph above we can observed that per every 1000 population other theft cases are mostly reported in various areas in Vancouver, followed by commercial theft, bicycle theft and mischief respectively.**

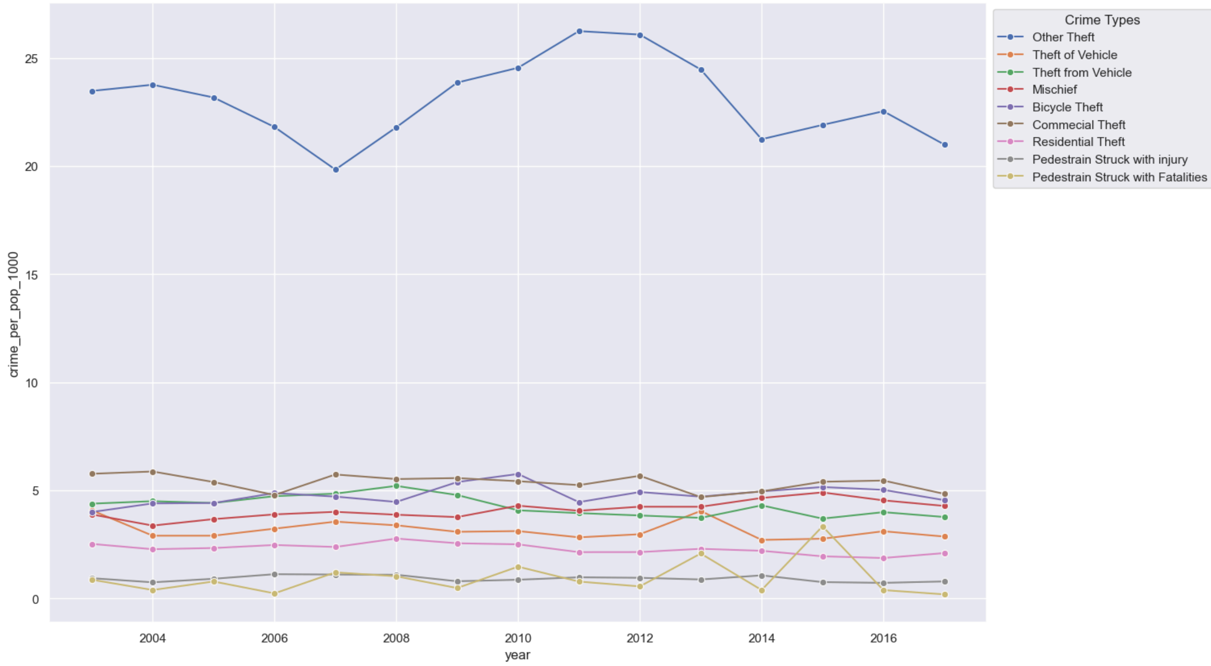
In [44]:

```python
plt.figure(figsize=(10, 6))
sns.set(style='darkgrid')
ax = sns.countplot(x = 'hundred_block',
              data = geo_data,
              order = geo_data["hundred_block"].value_counts().nlargest().index)

ax.set(xlabel='Various Blocks', yticks=[], title='Top blocks in Vancouver with the
ax.bar_label(container=ax.containers[0])
plt.xticks(rotation=45)
plt.show()
```



- **6XX W 41ST AVE, 31XX GRANDVIEW AND 11XX ROBSON ST are the top 3 areas with the most theft cases reported to authorities between 2003 to 2017. Its a very wild area to find yourself especially at night.**

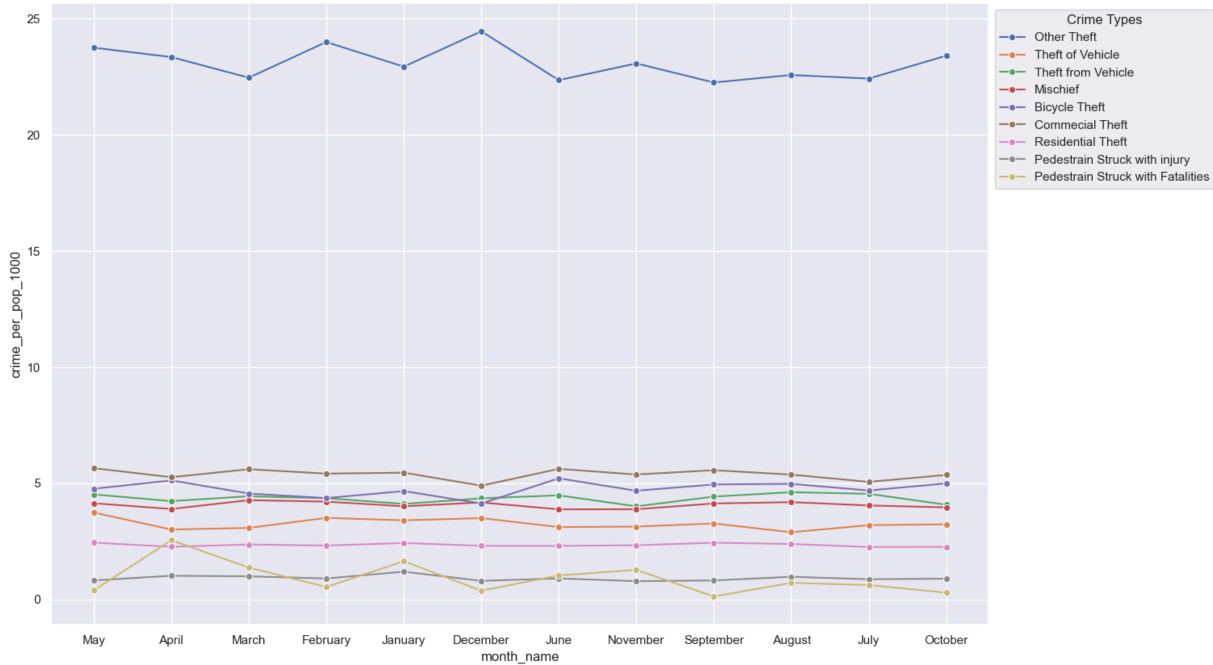
In [45]:
```
# Plot a lineplot with hue and adjust legend
plt.figure(figsize=(15, 10))
sns.lineplot(x='year',
             y='crime_per_pop_1000',
             hue='type', data=geo_data,
             marker='o',
             ci =None
            )

# Adjust the legend
plt.legend(title='Crime Types', bbox_to_anchor=(1, 1), loc='upper left')

# Show the plot
plt.show()
```



- **From the above plot, we can see that OTHER THEFT per 1000 population constitute the highest which is above known theft types average in the area. One critical observation shows that most theft cases are showing a sign of decline in 2017, except residential theft and pedestrain struck with injury.**

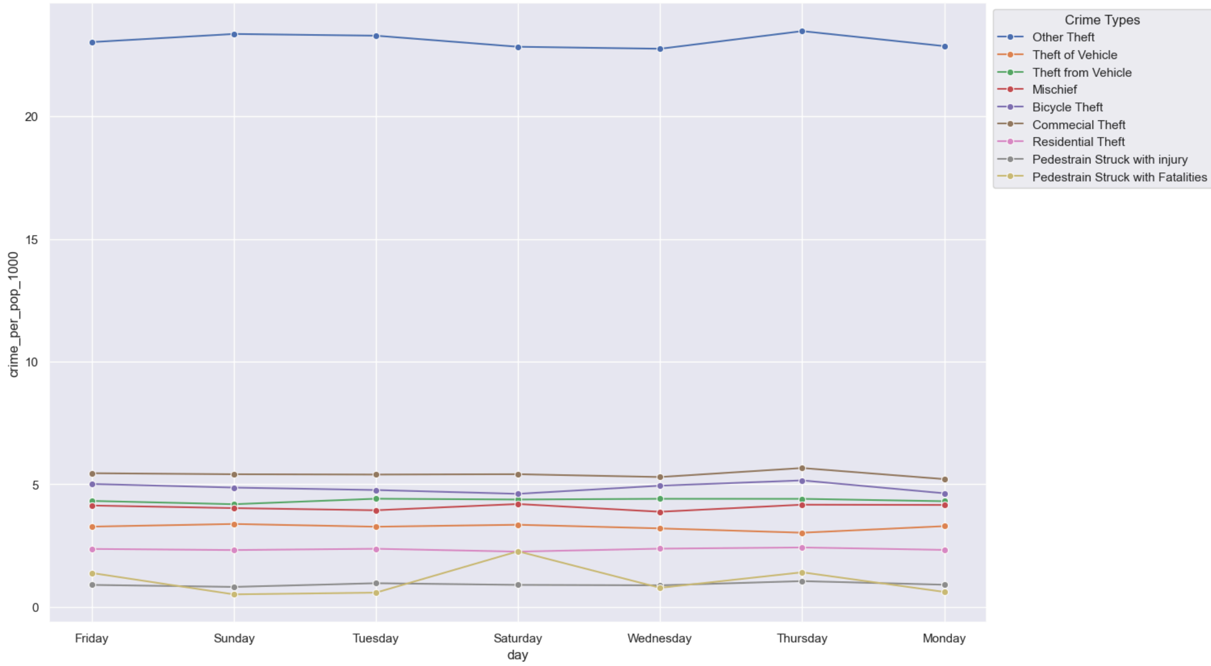
In [46]:

```
# Plot a lineplot with hue and adjust legend
plt.figure(figsize=(15, 10))
sns.lineplot(x='month_name',
             y='crime_per_pop_1000',
             hue='type', data=geo_data,
             marker='o',
             ci=None
            )

# Adjust the legend
plt.legend(title='Crime Types', bbox_to_anchor=(1, 1), loc='upper left')

# Show the plot
plt.show()
```



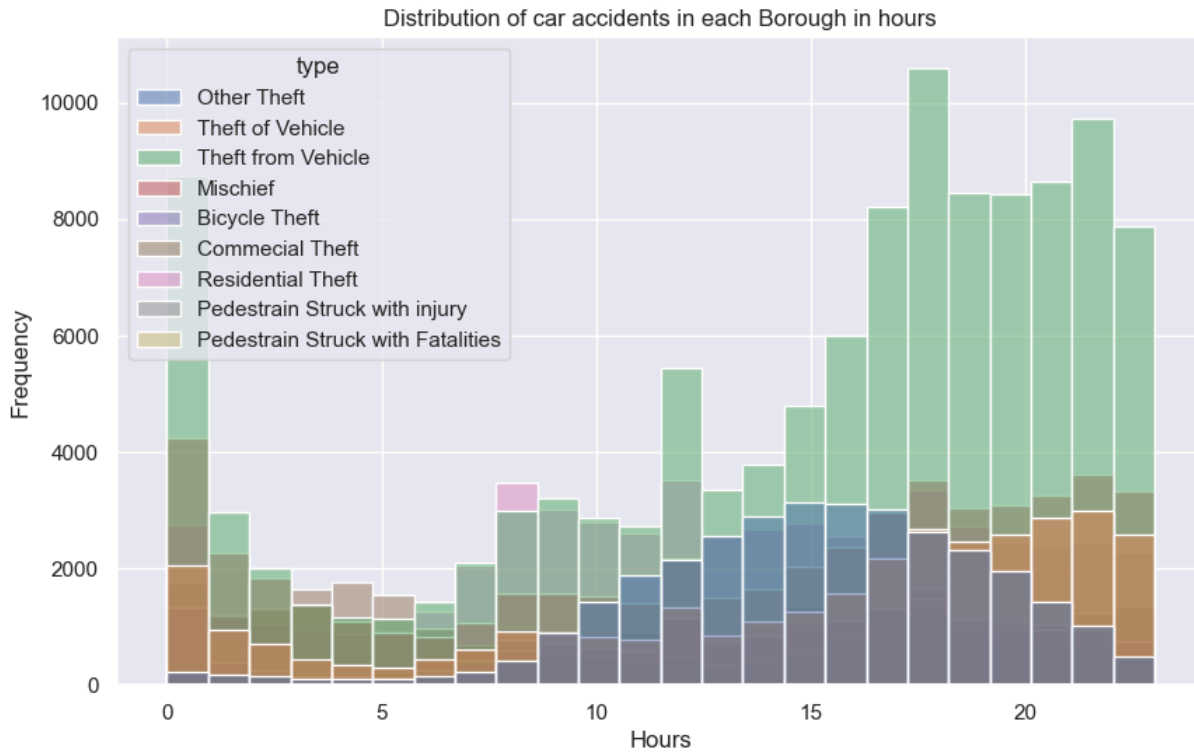
- **Other theft cases reported per 1000 population continuous to the highest across all month between 2003 to 2017, with no doubt it reaches it peak in December and shows a sign of increase in October. Theft cases like commercial theft, bicycle theft show a similar rise in October.**

In [53]:
```
# Plot a lineplot with hue and adjust legend
plt.figure(figsize=(15, 10))
sns.lineplot(x='day',
             y='crime_per_pop_1000',
             hue='type', data=geo_data,
             marker='o',
             ci=None
            )

# Adjust the legend
plt.legend(title='Crime Types', bbox_to_anchor=(1, 1), loc='upper left')

# Show the plot
plt.show()
```



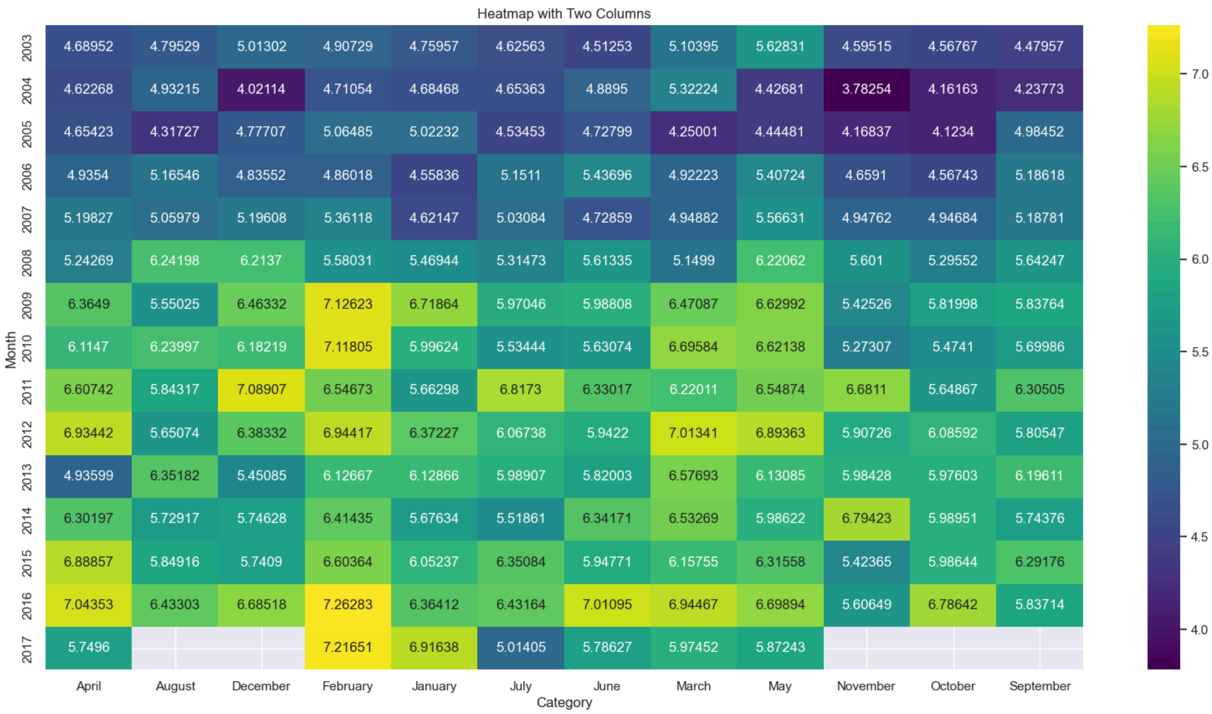
In [65]:

```
plt.figure(figsize=(10, 6))
# Create a Seaborn histogram with KDE
sns.histplot(data= geo_data,
             x = "hour",
             hue = "type",
             bins=24,
             kde=False)

# Set labels and title
plt.xlabel('Hours')
plt.ylabel('Frequency')
plt.title('Distribution of car accidents in each Borough in hours')
# Show the plot
plt.show()
```



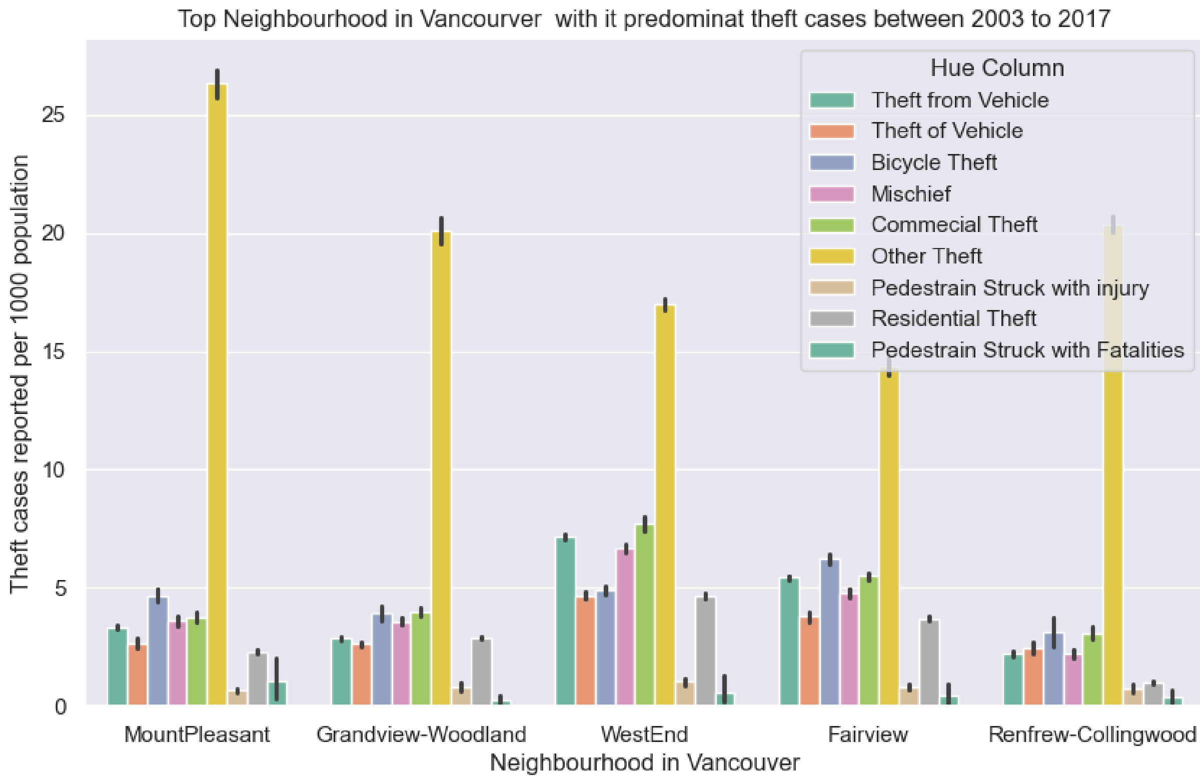
- **The observation from the above graph shows that, the peak of most theft cases in Vancouver are around 6pm to 12 midnight**

In [84]:
```
# Pivot the DataFrame to create a matrix suitable for heatmap
heatmap_data = geo_data.pivot_table(index='year', columns='month_name', values='cri

# Plot heatmap using Seaborn
plt.figure(figsize=(20, 10))
sns.heatmap(heatmap_data, annot=True, cmap='viridis', fmt='g', cbar=True)

# Customize the plot
plt.title('Heatmap with Two Columns')
plt.xlabel('Category')
plt.ylabel('Month')

# Show the plot
plt.show()
```



- **From the heatmap above we can identify that from 2009 to 2017 the month of February has had almost consistance higher average of theft cases per 1000 population in various neighbourhood in Vancouver. Other months like April , December and March had some higher average peaks in 2016, 2011 and 2012 respectively, even though these months are witnessing a declining rate in theft cases reported in the current years.**

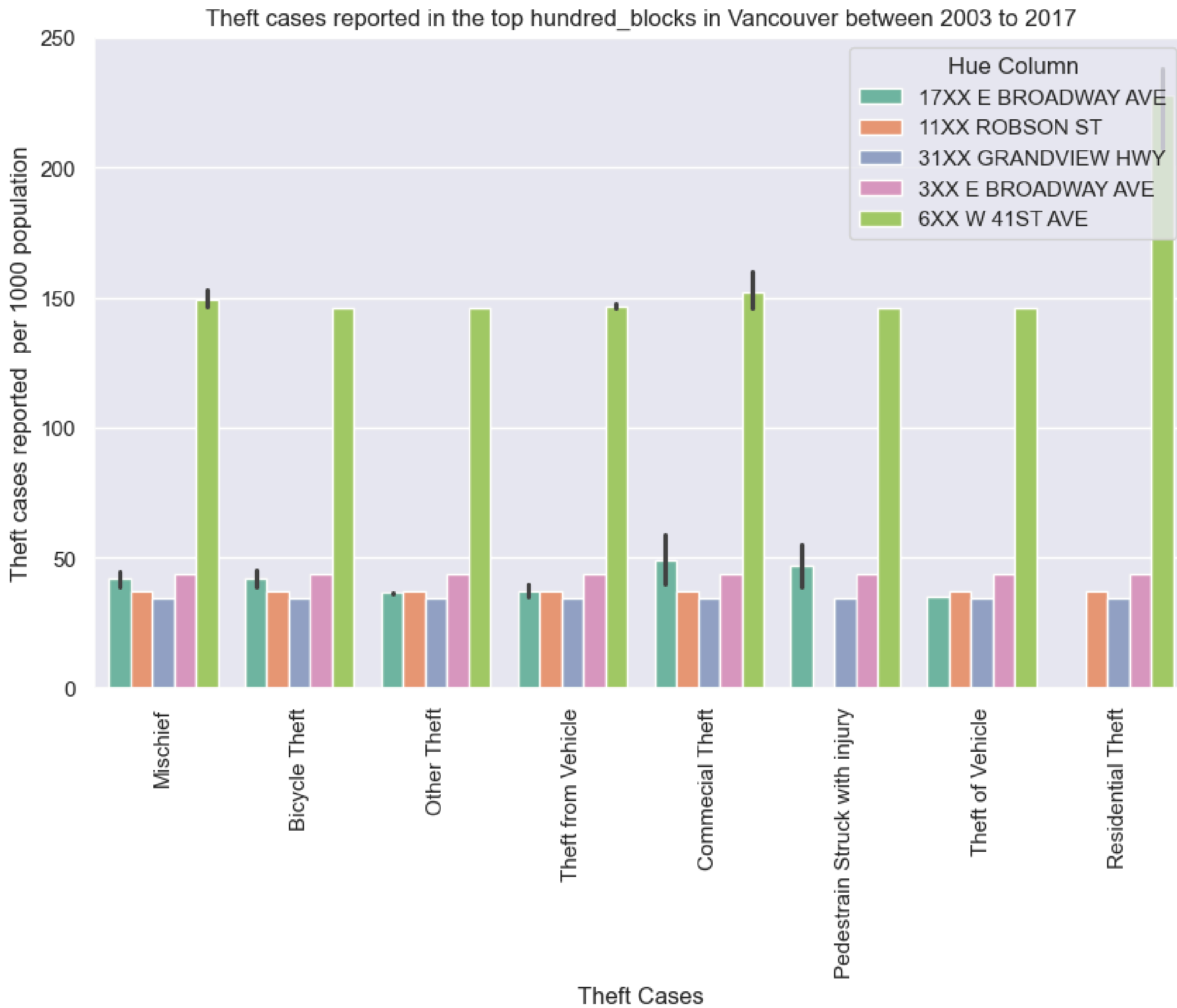
In [81]:
```
# Assuming 'category_column' is the categorical column to be sorted
sorted_categories = geo_data['neighbourhood'].value_counts().index[:5]

# Filter the DataFrame to include only the top five sorted categories
df_top5 = geo_data[geo_data['neighbourhood'].isin(sorted_categories)]

# Plot barplot using Seaborn with hue
plt.figure(figsize=(10, 6))
sns.barplot(x='neighbourhood', y='crime_per_pop_1000', hue='type', data=df_top5, pa

# Customize the plot
plt.title('Top Neighbourhood in Vancourver  with it predominat theft cases between
plt.xlabel('Neighbourhood in Vancouver')
plt.ylabel('Theft cases reported per 1000 population')
plt.legend(title='Hue Column',bbox_to_anchor=(1, 1), loc='upper right')

# Show the plot
plt.show()
```



- **From the diagram above we can observe that Other theft cases are dominant theft in all the top neighborhood in Vancouver, followed by some major types like Pedestrian Struck with fatalities, Theft from Vehicle and Mischief.**

In [ ]:

```
# Assuming 'category_column' is the categorical column to be sorted
sorted_categories = geo_data['hundred_block'].value_counts().index[:5]

# Filter the DataFrame to include only the top five sorted categories
df_top5 = geo_data[geo_data['hundred_block'].isin(sorted_categories)]

# Plot barplot using Seaborn with hue
plt.figure(figsize=(10, 6))
sns.barplot(x='type', y='crime_per_pop_1000', hue='hundred_block', data=df_top5, pa

# Customize the plot
plt.title('Theft cases reported in the top hundred_blocks in Vancouver between 2003
plt.xlabel('Theft Cases')
plt.ylabel('Theft cases reported  per 1000 population')
plt.legend(title='Hue Column',bbox_to_anchor=(1, 1), loc='upper right')

# Show the plot
plt.xticks(rotation=90)
plt.show()
```



- **From the graph above we can observe that 6XXX W 41ST AVENUE dominants in all the top theft cases in Vancouver, followed, other major areas like 17XX E Broadway, 11XX Robson Street and 31XX Grandview highway.**

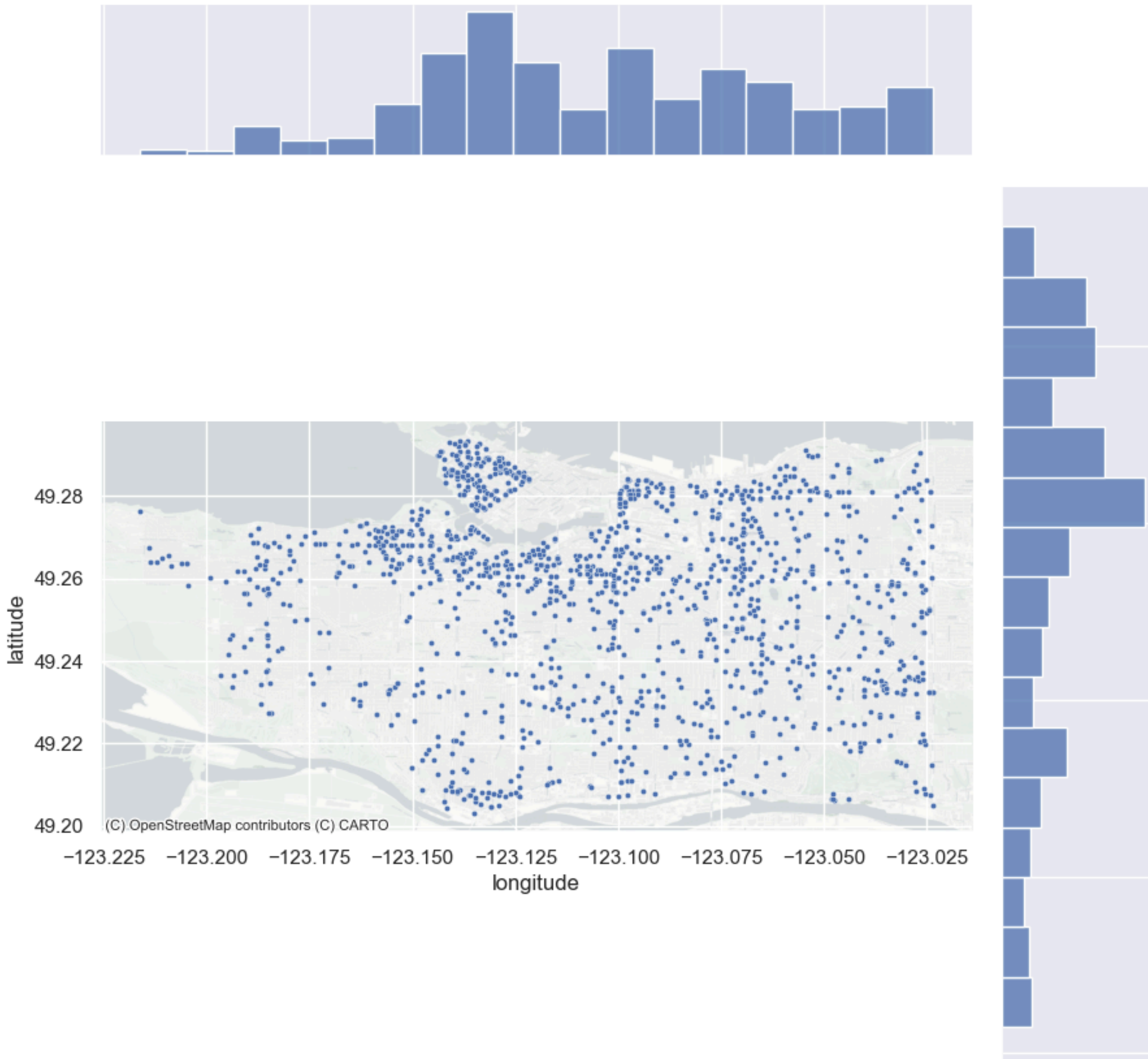
- # Spatial Point Pattern Analysis

In [69]:
```
# a random sample of 1500 theft cases in Vancauver wassubset for geospatial analysi
sample_geo = geo_data.sample(n=1500, random_state = 13490, replace = True)
```

In [70]:
```
# Generate scatter plot
plt.figure(figsize=(15, 10))
joint_axes = sns.jointplot(
    x="longitude", y="latitude", data=sample_geo, s=10,
    height=9
)
ctx.add_basemap(
    joint_axes.ax_joint,
    #crs="EPSG:4326",
    crs=sample_geo.crs,source=ctx.providers.CartoDB.Positron, zoom = 15
```

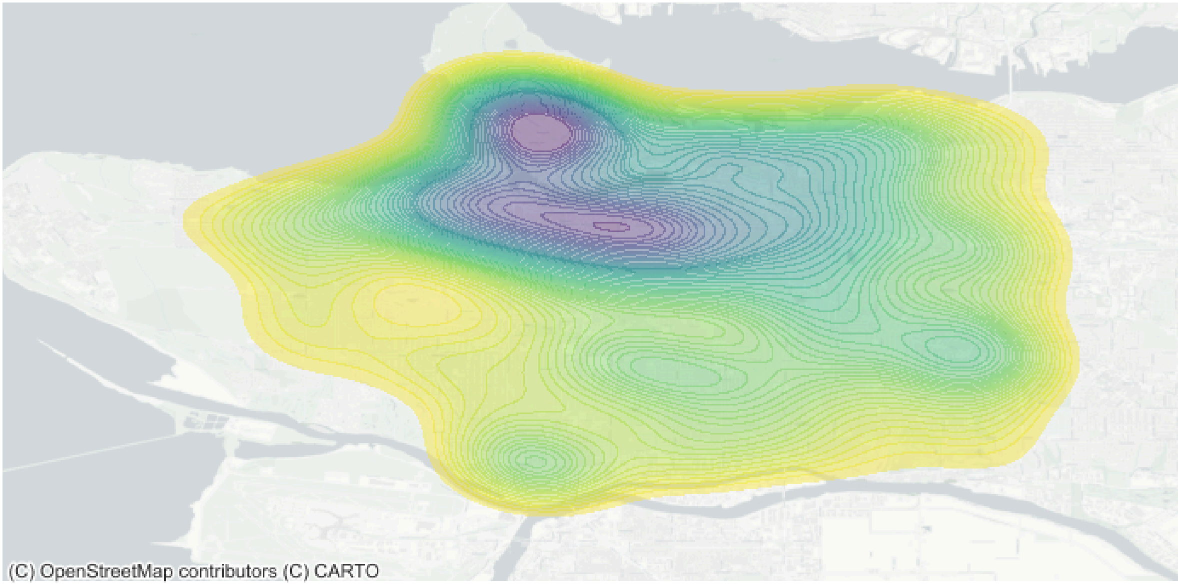
```
)
plt.show()
```

<Figure size 1500x1000 with 0 Axes>



In [71]:
```
# Set up figure and axis
f, ax = plt.subplots(1, figsize=(9, 9))
# Generate and add KDE with a shading of 50 gradients
# coloured contours, 75% of transparency,
# and the reverse viridis colormap
sns.kdeplot(
    x="longitude",
    y="latitude",
    data=sample_geo,
    n_levels=50,
    shade=True,
    alpha=0.4,
    cmap="viridis_r",
)
# Add basemap
ctx.add_basemap(
    ax, crs=geo_data.crs,source=ctx.providers.CartoDB.Positron, zoom = 15
)
```

```python
# Remove axes
ax.set_axis_off()
```



- **Though the kernel density shows a higher cluster of crime cases at the north.**

In [64]:
```python
mean_center = centrography.mean_center(sample_geo[["longitude", "latitude"]])
med_center = centrography.euclidean_median(sample_geo[["longitude", "latitude"]])
```

In [72]:
```python
# Generate scatterplot
joint_axes = sns.jointplot(
    x="longitude", y="latitude", data=sample_geo, s=5, height=9
)
# Add mean point and marginal lines
joint_axes.ax_joint.scatter(
    *mean_center, color="red", marker="x", s=50, label="Mean Center"
)
joint_axes.ax_marg_x.axvline(mean_center[0], color="red")
joint_axes.ax_marg_y.axhline(mean_center[1], color="red")
# Add median point and marginal lines
joint_axes.ax_joint.scatter(
    *med_center,
    color="limegreen",
    marker="o",
    s=50,
    label="Median Center"
)
joint_axes.ax_marg_x.axvline(med_center[0], color="limegreen")
joint_axes.ax_marg_y.axhline(med_center[1], color="limegreen")
# Legend
joint_axes.ax_joint.legend()
# Add basemap
ctx.add_basemap(
    joint_axes.ax_joint, crs=geo_data.crs,source=ctx.providers.CartoDB.Positron, zo
)
# Clean axes
```

```
joint_axes.ax_joint.set_axis_off()
# Display
plt.show()
```

- **The mean and median of reported crime cases in the geographic area exhibit a notable overlap, particularly in the northern part on the map. This convergence suggests that a substantial portion of the recorded crime cases is concentrated in the northern part. The central tendency of the distribution, characterized by both the mean and median, may be significantly influenced by outliers in the northern region, further highlighting the prevalence of crime in that specific geographic area.**

In [73]:
```
coordinates = sample_geo[["longitude", "latitude"]].values
```

In [48]:
```
from pointpats import (
    distance_statistics,
    QStatistic,
    random,
```

```
    PointPattern,
)
```

In [74]:

```
import libpysal

alpha_shape, alpha, circs = libpysal.cg.alpha_shape_auto(
    coordinates, return_circles=True
)
```

In [55]:

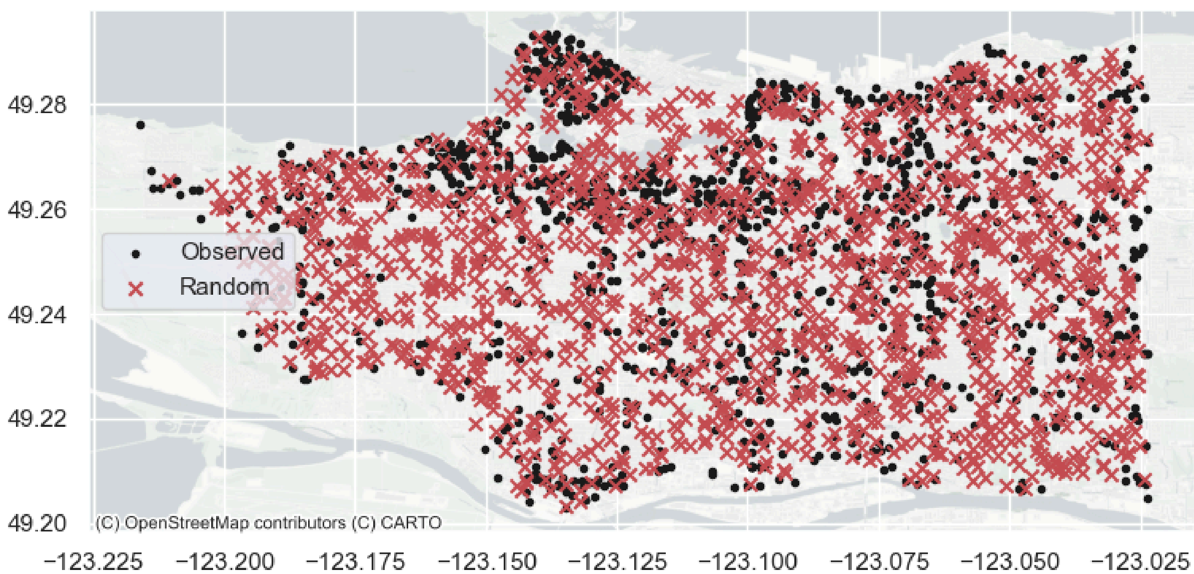
```
random_pattern = random.poisson(coordinates, size=len(coordinates))
```

In [75]:

```
random_pattern_ashape = random.poisson(
    alpha_shape, size=len(coordinates)
)
```

In [76]:

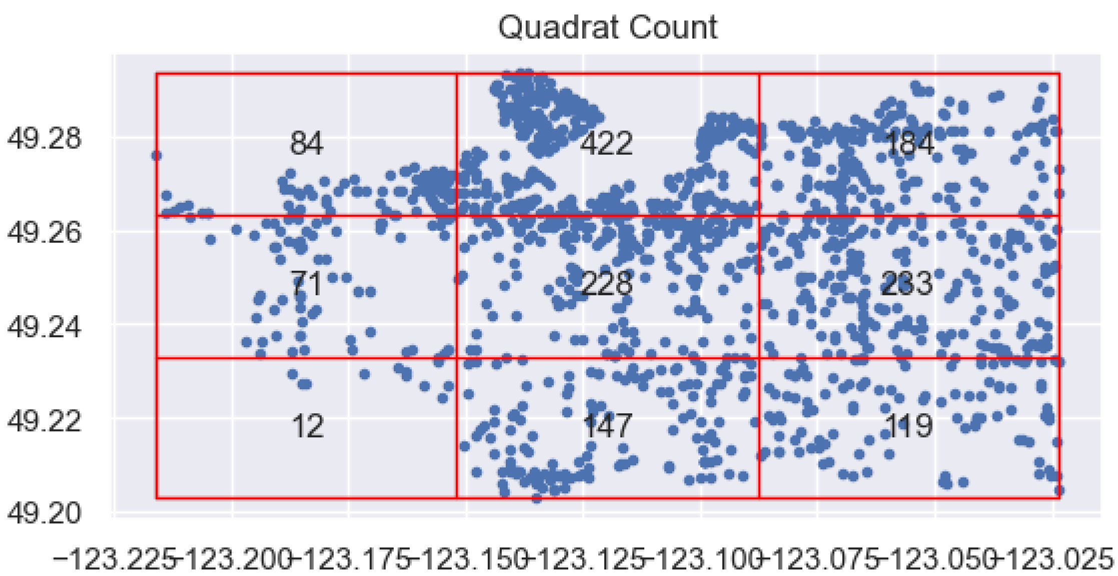
```
f, ax = plt.subplots(1, figsize=(9, 9))
plt.scatter(*coordinates.T, color="k", marker=".", label="Observed")
plt.scatter(
    *random_pattern_ashape.T, color="r", marker="x", label="Random"
)
ctx.add_basemap(
    ax, crs=geo_data.crs,source=ctx.providers.CartoDB.Positron, zoom = 15
)
ax.legend(ncol=1, loc="center left")
plt.show()
```



- The map above provide a pattern derived from a known completely spatially random process,random data was generated using the Poisson point process concept to anlysis a point patterns in our crime data. We can observed that, there more are clusters of crime reported between 2003 to 2017 at the north compare to the south of Vancouver, These clusters are all statistical significant per our chi-square results. So we can state that Northern Vancouver is more risky than the Southern areas.

In [77]:
```python
qstat = QStatistic(coordinates)
qstat.plot()
```

Out[77]: <Axes: title={'center': 'Quadrat Count'}>



- **Quadrat statistics examine the spatial distribution of points in an area in terms of the count of observations that fall within a given cell. By examining whether observations are spread evenly over cells. In this studies, for the default of a three-by-three grid spanning the point pattern, we see that the second squre at the north has over 422 observations, but the surrounding cells have many less theft cases reported**

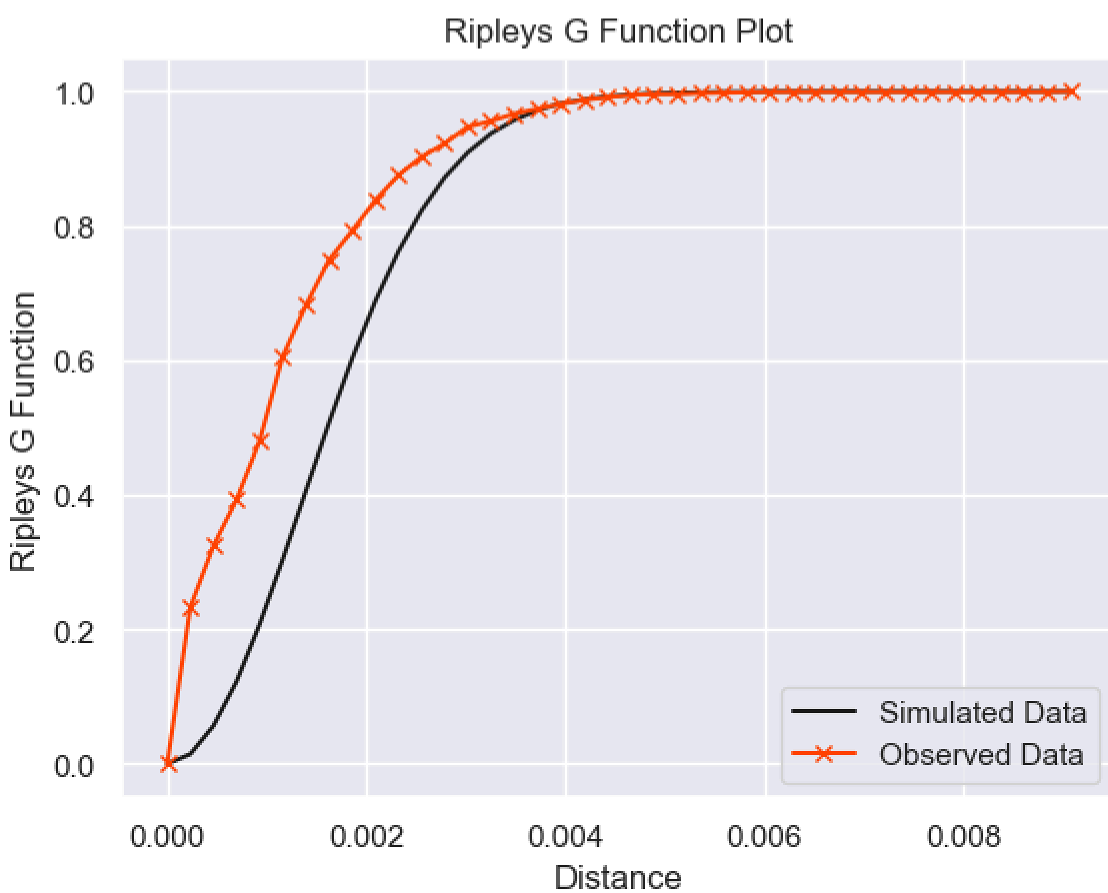
In [79]:
```python
# Assuming you have calculated the chi-squared p-value
chi2_pvalue = qstat.chi2_pvalue

# Set your significance level (alpha)
alpha = 0.05

# Compare the p-value with the significance level
if chi2_pvalue < alpha:
    print(f"The chi-squared test is statistically significant at the {alpha} level.
else:
    print(f"The chi-squared test is not statistically significant at the {alpha} le
```

```
The chi-squared test is statistically significant at the 0.05 level. Reject the null
hypothesis.
```

- **So per the chi-square statistics we reject the null hypothesis on the assumption that these thft cases are not distributed on complete spatial randomness, these theft clusters in Vancouver has a significant spatial relationship with it geography**

In [72]:
```
g_test = distance_statistics.g_test(
sample_geo[['longitude','latitude']].values, support=40, keep_simulations=True
)
```

In [73]:
```
plt.plot(g_test.support, np.median(g_test.simulations, axis=0),
color='k', label='Simulated Data')
plt.plot(g_test.support, g_test.statistic,
marker='x', color='orangered', label='Observed Data')


plt.legend()
plt.xlabel('Distance')
plt.ylabel('Ripleys G Function')
plt.title('Ripleys G Function Plot')
plt.show()
```

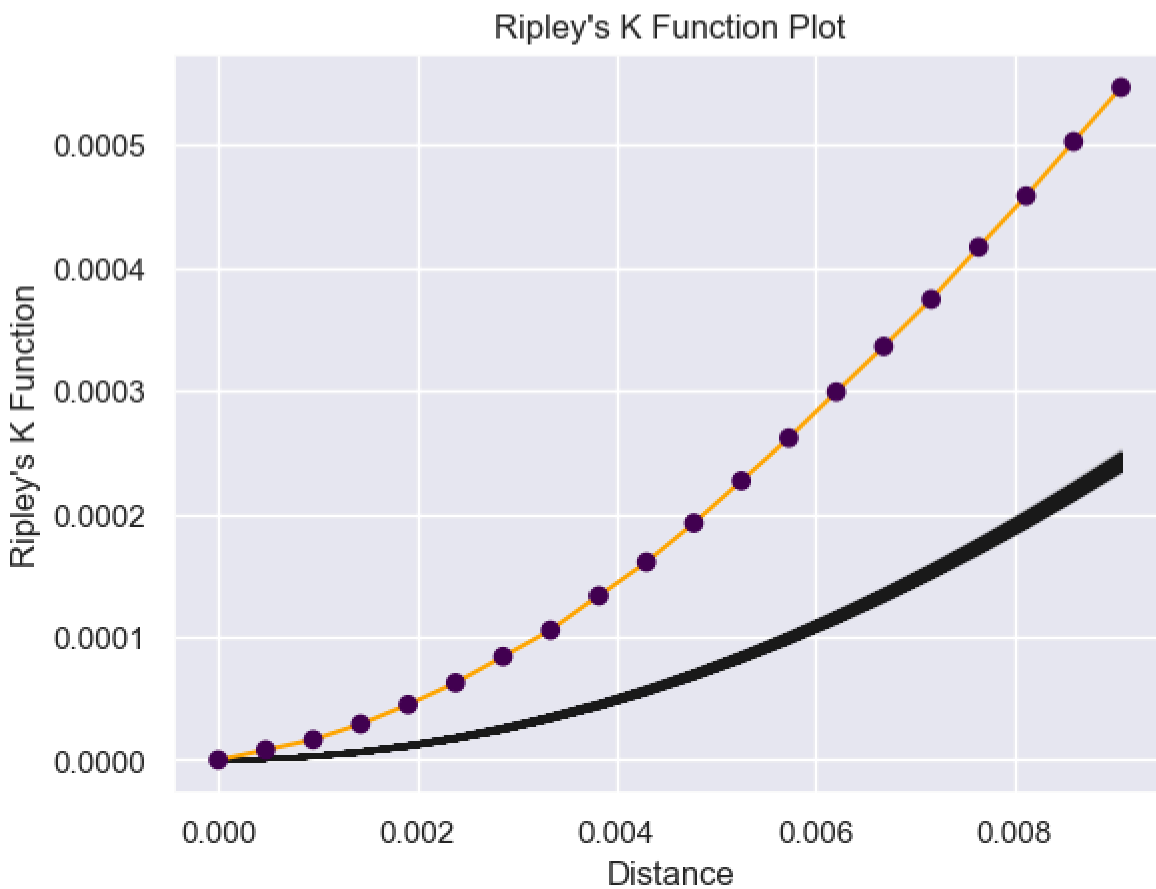


- the graph above shows the results of Ripley's G plot. The orange line represents the cumulative distancefunction from the theft locations dataset. The black line represents a simulated CSR distribution.The observed data rises much more rapidly than the simulated datafrom a CSR process. From this, we can deduce that the store location data has a significant spatialpattern.

In [74]:
```
k_test = distance_statistics.k_test(sample_geo[['longitude','latitude']].values, ke
```

In [75]:

```
# Assuming you have a point pattern called 'pp'
plt.plot(k_test.support, k_test.simulations.T, color='k', alpha=.01)
plt.plot(k_test.support, k_test.statistic, color='orange')
plt.scatter(
    k_test.support,
    k_test.statistic,
    cmap='viridis',
    c=k_test.pvalue < .05,
    zorder=4
)
plt.xlabel('Distance')
plt.ylabel('Ripley\'s K Function')
plt.title('Ripley\'s K Function Plot')
plt.show()

# Access the p-value
p_value = k_test.pvalue
print(f"P-value: {p_value}")
```




```
P-value: [0.     0.0001 0.0001 0.0001 0.0001 0.0001 0.0001 0.0001 0.0001 0.0001
 0.0001 0.0001 0.0001 0.0001 0.0001 0.0001 0.0001 0.0001 0.0001 0.0001]
```

- **With the Ripley's K plot too, we can see that the observed data is well above that of the simulated data, which confirms again that theft cases in Vanc0uver is from a process that is not spatially random.**

- # AREA CLASSIFICATION BETWEEN SAFE AND NON SAFE AREAS IN VANCOUVER USING LOGISTIC REGRESSION

In [76]:
```
geo_data['neighbourhood'] = geo_data['neighbourhood'].str.strip()
```

In [77]:
```
sub_2 = ['6XX W 41ST AVE', '31XX GRANDVIEW HWY', '11XX ROBSON ST','17XX E BROADWAY
data_1= geo_data[geo_data['hundred_block'].isin(sub_2)]
```

In [79]:
```
selected_df =data_1[['year','day','hour',
                    'hundred_block','neighbourhood',
                    'latitude', 'longitude','population','total_crime',
                    'crime_per_pop_1000','type']]
```

In [80]:
```
selected_df.reset_index(inplace = True)
```

In [85]:
```
from sklearn.preprocessing import OneHotEncoder
```

In [86]:
```
def one_hot_encode(df, column_name):
    """
    One-hot encodes a categorical column in a DataFrame.

    Parameters:
    - df: DataFrame
        The input DataFrame.
    - column_name: str
        The name of the categorical column to be one-hot encoded.

    Returns:
    - DataFrame
        The DataFrame with the one-hot encoded column.
    """

    # Copy the DataFrame to avoid modifying the original
    df_encoded = df.copy()

    # Extract the specified column for one-hot encoding
    column_data = df_encoded[[column_name]]

    # Initialize the OneHotEncoder
    encoder = OneHotEncoder(sparse=False, drop='first')

    # Fit and transform the data
    encoded_data = encoder.fit_transform(column_data)

    # Create new column names based on the original category names
    new_columns = [f"{column_name}_{category}" for category in encoder.get_feature_

    # Create a DataFrame with the encoded data and new column names
    df_encoded[new_columns] = encoded_data
```

```python
    # Drop the original categorical column
    df_encoded.drop(column_name, axis=1, inplace=True)

    return df_encoded
```

In [87]:
```python
# One-hot encode the 'Category' column
df_encoded_1 = one_hot_encode(selected_df, 'hundred_block')
```

In [88]:
```python
df_encoded_2 = one_hot_encode(df_encoded_1, 'neighbourhood')
```

In [89]:
```python
df_encoded_2 = one_hot_encode(df_encoded_2, 'type')
```

In [90]:
```python
df_encoded_2.head()
```

Out[90]:

	index	year	day	hour	latitude	longitude	population	total_crime	crime_per
0	60135	2003	Monday	4.0	49.262362	-123.069215	29175.0	1718	
1	60136	2003	Friday	20.0	49.262357	-123.068675	29175.0	1718	
2	60137	2003	Monday	13.0	49.262357	-123.068675	29175.0	1718	
3	60138	2003	Saturday	11.0	49.262357	-123.068675	29175.0	1718	
4	60139	2003	Thursday	18.0	49.262357	-123.068675	29175.0	1718	

5 rows × 26 columns

In [91]:
```python
def categorize_safety(df, column_name):
    """
    Group a column into 'safe' or 'not safe' based on the mean value.

    Parameters:
    - df: DataFrame
        The DataFrame containing the data.
    - column_name: str
        The name of the column to be categorized.

    Returns:
    - DataFrame
        A new DataFrame with an additional column 'Safety' indicating 'safe' or 'no
    """
    mean_value = df[column_name].mean()

    # Create a new column 'Safety' based on the mean value
    df['Safety'] = df[column_name].apply(lambda x: 'SAFE' if x >= mean_value else '

    return df
```

In [92]:
```python
new_2= categorize_safety(df_encoded_2, 'crime_per_pop_1000')
```

In [93]:
```
new_2["Safety"].value_counts()
```

Out[93]:
```
Safety
NOT SAFE    6697
SAFE        1900
Name: count, dtype: int64
```

In [94]:
```
from sklearn.preprocessing import LabelEncoder
```

In [95]:
```
def encode_target_binary(df, target_column):
    """
    Encodes the target column to binary values (0 and 1).

    Parameters:
    - df: DataFrame
        The input DataFrame.
    - target_column: str
        The name of the target column to be encoded.

    Returns:
    - DataFrame
        The DataFrame with the target column encoded to binary values.
    """

    # Copy the DataFrame to avoid modifying the original
    df_encoded = df.copy()

    # Extract the specified target column
    target_data = df_encoded[[target_column]]

    # Initialize the LabelEncoder
    encoder = LabelEncoder()

    # Fit and transform the target data
    encoded_data = encoder.fit_transform(target_data)

    # Replace the original target column with the encoded values
    df_encoded[target_column] = encoded_data

    return df_encoded
```

In [96]:
```
# Encode the 'Target' column to binary values
df_new_1 = encode_target_binary(new_2, 'Safety')
```

In [97]:
```
final_df_1 = df_new_1.drop(columns = ["day", "index"], axis =1)
```

In [99]:
```
final_df_1.corr()['Safety']\
.abs().sort_values(ascending=False).head(30)\
.to_frame().style.background_gradient()
```

Out[99]:

	Safety
Safety	1.000000
hundred_block_hundred_block_6XX W 41ST AVE	1.000000
neighbourhood_neighbourhood_Oakridge	0.993241
crime_per_pop_1000	0.991236
population	0.902066
latitude	0.824169
total_crime	0.605874
longitude	0.466379
hundred_block_hundred_block_31XX GRANDVIEW HWY	0.272562
neighbourhood_neighbourhood_Renfrew-Collingwood	0.272562
neighbourhood_neighbourhood_WestEnd	0.270053
hundred_block_hundred_block_17XX E BROADWAY AVE	0.266186
neighbourhood_neighbourhood_Kensington-CedarCottage	0.252651
neighbourhood_neighbourhood_MountPleasant	0.238621
hundred_block_hundred_block_3XX E BROADWAY AVE	0.238621
type_type_Other Theft	0.155165
type_type_Theft of Vehicle	0.122988
type_type_Theft from Vehicle	0.107788
neighbourhood_neighbourhood_SouthCambie	0.090659
hour	0.050974
type_type_Pedestrain Struck with injury	0.030226
year	0.027303
type_type_Mischief	0.024680
type_type_Residential Theft	0.015195
type_type_Commecial Theft	0.013336

In [101…

```
# Calculate correlation coefficients
correlation_matrix = final_df_1.corr()
correlation_with_target = correlation_matrix["Safety"].abs()

# Set a correlation threshold (adjust as needed)
correlation_threshold = 0.1

# Select features with correlation above the threshold
selected_features = correlation_with_target[correlation_with_target >= correlation_
```

```
# Drop unselected features from the DataFrame
df_filtered = final_df_1[selected_features]

# Display the DataFrame with selected features
df_filtered.head()
```

Out[101…

	latitude	longitude	population	total_crime	crime_per_pop_1000	hundred_block_hur E
0	49.262362	-123.069215	29175.0	1718	58.886033	
1	49.262357	-123.068675	29175.0	1718	58.886033	
2	49.262357	-123.068675	29175.0	1718	58.886033	
3	49.262357	-123.068675	29175.0	1718	58.886033	
4	49.262357	-123.068675	29175.0	1718	58.886033	

In [144…

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.impute import SimpleImputer
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.feature_selection import SelectFromModel
from imblearn.over_sampling import SMOTE
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

In [211…

```
df_filtered.columns
```

Out[211…

```
Index(['latitude', 'longitude', 'population', 'total_crime',
       'crime_per_pop_1000', 'hundred_block_hundred_block_17XX E BROADWAY AVE',
       'hundred_block_hundred_block_31XX GRANDVIEW HWY',
       'hundred_block_hundred_block_3XX E BROADWAY AVE',
       'hundred_block_hundred_block_6XX W 41ST AVE',
       'neighbourhood_neighbourhood_Kensington-CedarCottage',
       'neighbourhood_neighbourhood_MountPleasant',
       'neighbourhood_neighbourhood_Oakridge',
       'neighbourhood_neighbourhood_Renfrew-Collingwood',
       'neighbourhood_neighbourhood_WestEnd', 'type_type_Other Theft',
       'type_type_Theft from Vehicle', 'type_type_Theft of Vehicle', 'Safety'],
      dtype='object')
```

In [103…

```
df_filtered["Safety"].value_counts()
```

Out[103…

```
Safety
0    6697
1    1900
Name: count, dtype: int64
```

In [104…
```
non = df_filtered[df_filtered["Safety"] == 0]
safe = df_filtered[df_filtered["Safety"]== 1]
print(non.shape)
print(safe.shape)
```

```
(6697, 18)
(1900, 18)
```

In [109…
```
com_sample = non.sample(n=1900)
print(com_sample.shape)
```

```
(1900, 18)
```

In [110…
```
new= pd.concat([com_sample, safe], axis = 0)
```

In [111…
```
new.to_csv('new_vanco_1.csv', index=False)
```

In [112…
```
new["Safety"].value_counts()
```

Out[112…
```
Safety
0    1900
1    1900
Name: count, dtype: int64
```

In [193…
```
# separating the data and labels
X = new.drop(columns = "Safety", axis=1)
Y = new["Safety"]
```

In [206…
```
X
```

Out[206…
```
array([[ 1.84197944, -0.8152879 ,  1.04014845, ...,  0.69597055,
        -0.45888628, -0.22750284],
       [ 1.8837943 , -0.85400723,  1.04014845, ...,  0.69597055,
        -0.45888628, -0.22750284],
       [ 0.4074832 ,  2.03381959,  1.29267162, ...,  0.69597055,
        -0.45888628, -0.22750284],
       ...,
       [-0.90631249, -0.64684976, -0.95262678, ..., -1.43684242,
        -0.45888628, -0.22750284],
       [-0.90631249, -0.64684976, -0.95262678, ..., -1.43684242,
        -0.45888628, -0.22750284],
       [-0.90451644, -0.6758538 , -0.95262678, ...,  0.69597055,
        -0.45888628, -0.22750284]])
```

In [133…
```
print(X.shape, Y.shape)
```

```
(3800, 17) (3800,)
```

In [194…
```
scaler = StandardScaler()
scaler.fit(X)
standardized_data = scaler.transform(X)
```

In [195…
```
X = standardized_data
Y = new["Safety"]
```

In [196…

```
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size = 0.3, random_st
```

In [197…

```
print(X.shape, X_train.shape, X_test.shape)
```

```
(3800, 17) (2660, 17) (1140, 17)
```

In [198…

```
# Initialize the Logistic Regression model
model = LogisticRegression()

#Fit the model on the training data
model.fit(X_train , Y_train)

# Make predictions on the test data
y_pred = model.predict(X_test)

# Evaluate the model performance
accuracy = accuracy_score(Y_test, y_pred)
conf_matrix = confusion_matrix(Y_test, y_pred)
class_report = classification_report(Y_test, y_pred)

# Print the results
print(f"Accuracy: {accuracy:.4f}")
print("Confusion Matrix:")
print(conf_matrix)
print("Classification Report:")
print(class_report)
```

```
Accuracy: 1.0000
Confusion Matrix:
[[592   0]
 [  0 548]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       592
           1       1.00      1.00      1.00       548

    accuracy                           1.00      1140
   macro avg       1.00      1.00      1.00      1140
weighted avg       1.00      1.00      1.00      1140
```

In [199…

```
logreg = LogisticRegression(max_iter=1000,tol=1e-5,random_state = 50)
param_grid = {
    "C": [0.001, 0.01, 1, 100, 1000, 10000],
    "penalty" : ["l1", "l2"],
    "solver"  : ["liblinear", "saga"]
}


grid_search = GridSearchCV(logreg, param_grid, cv=5)
grid_search.fit(X_train, Y_train)

print(grid_search.best_params_)
```

```
{'C': 0.001, 'penalty': 'l1', 'solver': 'liblinear'}
```

In [200…

```python
# Create logistic regression model with best hyperparameters
optimized_logreg = LogisticRegression(max_iter = 1000, random_state = 50, C = 0.001

# Train the model
optimized_logreg.fit(X_train, Y_train)

# Make predictions on the test set
y_pred = optimized_logreg.predict(X_test)

# Evaluate the model
print("Confusion Matrix:\n", confusion_matrix(Y_test, y_pred))
print("\nClassification Report:\n", classification_report(Y_test, y_pred))
print("\nAccuracy:", accuracy_score(Y_test, y_pred))
```

```
Confusion Matrix:
 [[592   0]
 [  0 548]]

Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00       592
           1       1.00      1.00      1.00       548

    accuracy                           1.00      1140
   macro avg       1.00      1.00      1.00      1140
weighted avg       1.00      1.00      1.00      1140


Accuracy: 1.0
```

In [265…

```python
input_data = (49.23352408,-123.1185036,13030.0,1900,145.81734458940906,0.0,0.0,0.0,

# changing the input_data to numpy array
input_data_as_numpy_array = np.asarray(input_data)

# reshape the array as we are predicting for one instance
input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)

# standardize the input data
std_data = scaler.transform(input_data_reshaped)
print(std_data)

prediction = optimized_logreg.predict(std_data)
print(prediction)

if (prediction[0] == 0):
  print('!!! NOT A SAFE PLACE TO BE NOW')
else:
  print('DONT WORRY ITS SAFE TO WALK AROUND AT ANY TIME')
```

```
[[-0.90960205 -0.59381213 -0.95262678  0.75361021  0.97313086 -0.38159338
  -0.40347329 -0.32646783  1.         -0.36187343 -0.32646783  1.01058231
  -0.40347329 -0.39772287  0.69597055 -0.45888628 -0.22750284]]
[1]
DONT WORRY ITS SAFE TO WALK AROUND AT ANY TIME
```

- # Conclusion

- In conclusion, the comprehensive analysis of crime data in Vancouver from 2003 to 2017 reveals significant patterns and trends that contribute to a nuanced understanding of criminal activities in the region. The application of statistical methods such as ANOVA, chi-square, and spatial analysis techniques provided valuable insights.The rejection of the null hypothesis in ANOVA tests suggests that there is statistical significance in theft distribution among the top five municipalities, with West End, Fairview, Mount Pleasant, Grandview-Woodland, and Renfrew-Collingwood identified as high-risk areas for various crimes.The predominant theft types, including theft from vehicles, residential theft, mischief, theft of vehicles, and other theft, were highlighted through graphical representations. Specific high-crime areas, such as 6XX W 41ST AVE, 31XX GRANDVIEW, and 11XX ROBSON ST, were identified.Temporal analysis showcased monthly and hourly patterns, emphasizing the importance of time in crime occurrence. The spatial analysis using kernel density, Poisson point process, chi-square, and quadrat statistics confirmed the concentration of crime in the northern part of Vancouver, indicating its higher risk compared to the south.The convergence of mean and median in the northern region further underscores the prevalence of crime, potentially influenced by outliers. The rejection of the null hypothesis in chi-square statistics supports the spatial clustering of crime in Vancouver, indicating a significant spatial relationship with its geography. Ripley's G and K plots further validated the non-random spatial pattern of theft cases, emphasizing the need for targeted interventions and strategic policing in specific areas. These findings contribute to informed decision-making for law enforcement and policymakers, aiming to enhance public safety in Vancouver.

In [ ]:
```

```