PERFORMING SPATIAL AUTOCORRELATION ON THE RECORDED ACCIDENT CASES IN NEW YORK FOR 2020

- ## IMPORTING THE NEEDED LIBRARIES

In [52]:
```
# to read and wrangle data
import pandas as pd
import numpy as np
import seaborn as sns
from pointpats import centrography


# to create spatial data
import geopandas as gpd
from shapely.geometry import MultiPolygon


# for basemaps
import contextily as ctx

# For spatial statistics
import esda
from esda.moran import Moran, Moran_Local
from pysal.lib import weights
from splot import esda as esdaplot

import splot
from splot.esda import moran_scatterplot, plot_moran, lisa_cluster,plot_moran_simulation
from IPython.display import display, Markdown, display_latex, display_markdown, display_html

import libpysal as lps

# Graphics
import matplotlib.pyplot as plt
import plotly.express as px
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
#impoting the dataset for the accident cases
car = pd.read_csv("NYC Accidents 2020.csv")
```

In [3]:
```
car.columns = car.columns.str.lower()
```

In [4]:
```
car.head()
```

Out[4]:

	crash date	crash time	borough	zip code	latitude	longitude	location	on street name	cross street name	s n
0	2020-08-29	15:40:00	BRONX	10466.0	40.89210	-73.833760	POINT (-73.83376 40.8921)	PRATT AVENUE	STRANG AVENUE	
1	2020-08-29	21:00:00	BROOKLYN	11221.0	40.69050	-73.919914	POINT (-73.919914 40.6905)	BUSHWICK AVENUE	PALMETTO STREET	
2	2020-08-29	18:20:00	NaN	NaN	40.81650	-73.946556	POINT (-73.946556 40.8165)	8 AVENUE	NaN	
3	2020-08-29	00:00:00	BRONX	10459.0	40.82472	-73.892960	POINT (-73.89296 40.82472)	NaN	NaN	SIMP ST
4	2020-08-29	17:10:00	BROOKLYN	11203.0	40.64989	-73.933890	POINT (-73.93389 40.64989)	NaN	NaN	SNY AVE

5 rows × 29 columns

- ## Exploratory Data Analysis (EDA)

In [5]:
```
car.isna().sum()
```

Out[5]:
```
crash date                          0
crash time                          0
borough                         25741
zip code                        25747
latitude                         5946
longitude                        5946
location                         5946
on street name                  19437
cross street name               39200
off street name                 55444
number of persons injured           0
number of persons killed            0
number of pedestrians injured       0
number of pedestrians killed        0
number of cyclist injured           0
number of cyclist killed            0
number of motorist injured          0
number of motorist killed           0
contributing factor vehicle 1     304
contributing factor vehicle 2   15596
contributing factor vehicle 3   68116
contributing factor vehicle 4   73030
contributing factor vehicle 5   74358
collision_id                        0
vehicle type code 1               635
vehicle type code 2             21243
vehicle type code 3             68457
vehicle type code 4             73110
vehicle type code 5             74378
dtype: int64
```

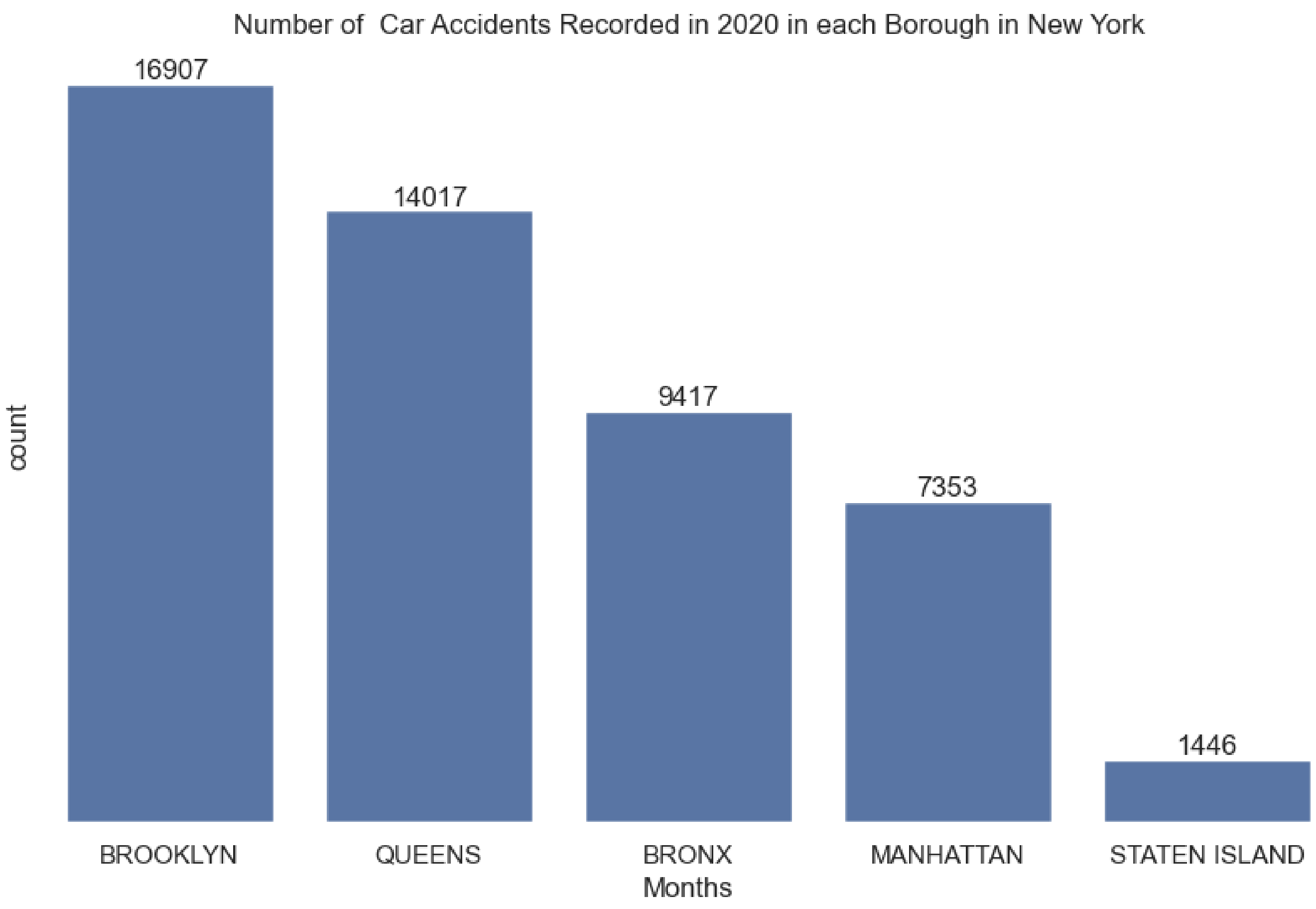
In [6]:
```
car.duplicated().sum()
```

Out[6]:
```
0
```

In [7]:
```
car["borough"].value_counts()
```

Out[7]:
```
borough
BROOKLYN         16907
QUEENS           14017
BRONX             9417
MANHATTAN         7353
STATEN ISLAND     1446
Name: count, dtype: int64
```

In [128…
```
plt.figure(figsize=(10, 6))
sns.set(style='darkgrid')
ax = sns.countplot(x = 'borough',
              data = car,
              order = car['borough'].value_counts().index)

ax.set(xlabel='Months', yticks=[], title='Number of  Car Accidents Recorded in 2020 in each B
ax.bar_label(container=ax.containers[0])
plt.show()
```

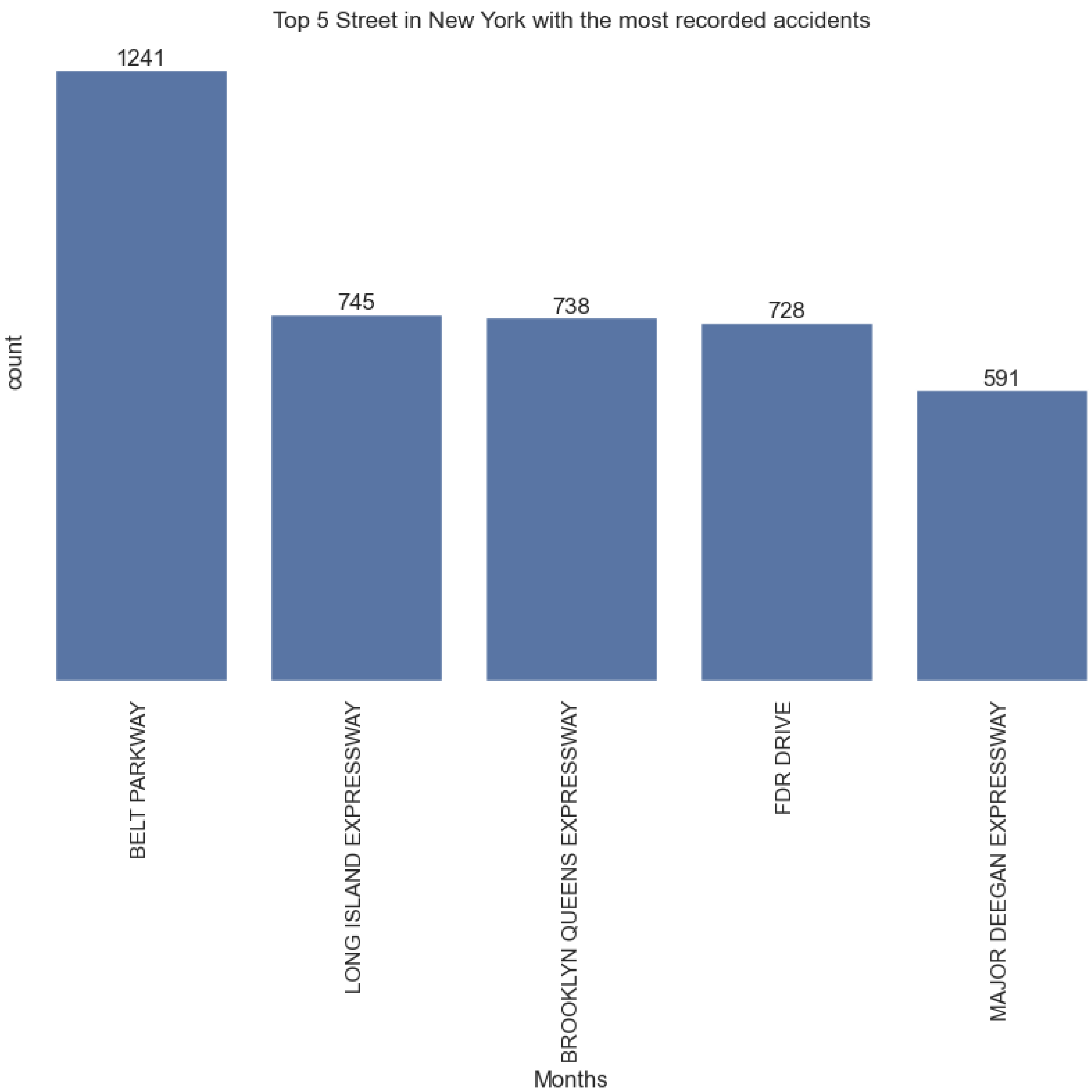


- **From the graph above we can observed that Brooklyn and Queens are the top two Borough with the most recorded cases of car accidents followed by Bronx and the rest.**

In [8]:
```
car["on street name"].value_counts().nlargest()
```

Out[8]:
```
on street name
BELT PARKWAY                 1241
LONG ISLAND EXPRESSWAY        745
BROOKLYN QUEENS EXPRESSWAY    738
FDR DRIVE                     728
MAJOR DEEGAN EXPRESSWAY       591
Name: count, dtype: int64
```

In [104…
```
plt.figure(figsize=(10, 6))
sns.set(style='darkgrid')
ax = sns.countplot(x = 'on street name',
              data = car,
              order = car['on street name'].value_counts().nlargest().index)

ax.set(xlabel='Months', yticks=[], title='Top 5 Street in New York with the most recorded acc
ax.bar_label(container=ax.containers[0])
plt.xticks(rotation=90)
plt.show()
```



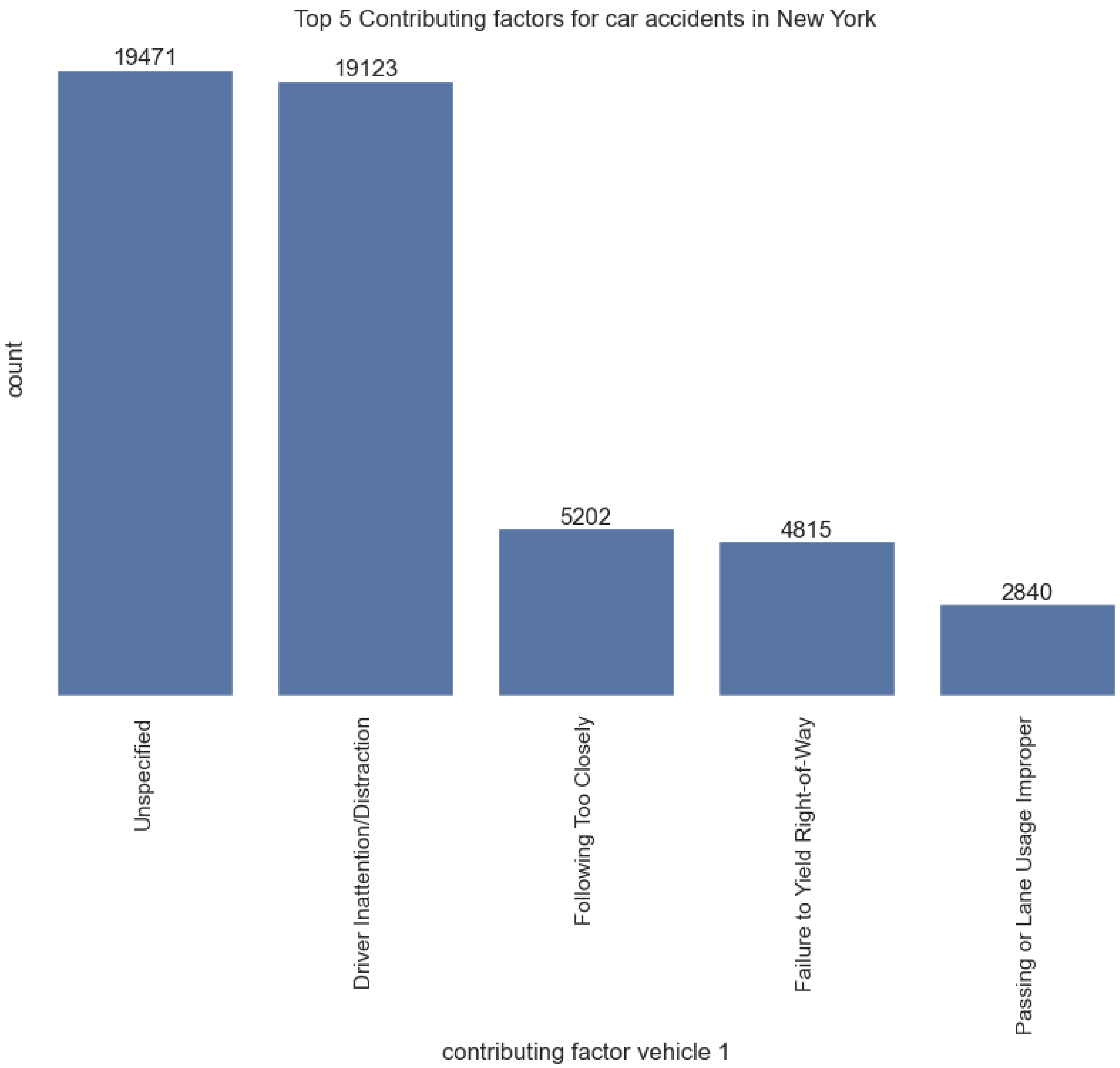
- **The top five street with the most recorded cases of car accidents in New York are, Belt Parkway, Long Island Expressway, Brooklyn Queens Expressway, FDR Drive and Major Deegan Expressway.**

In [9]:
```
car["contributing factor vehicle 1"].value_counts().nlargest()
```

Out[9]:
```
contributing factor vehicle 1
Unspecified                       19471
Driver Inattention/Distraction    19123
Following Too Closely              5202
Failure to Yield Right-of-Way      4815
Passing or Lane Usage Improper     2840
Name: count, dtype: int64
```

In [105…
```
plt.figure(figsize=(10, 6))
sns.set(style='darkgrid')
ax = sns.countplot(x = 'contributing factor vehicle 1',
              data = car,
              order = car['contributing factor vehicle 1'].value_counts().nlargest().index)

ax.set(xlabel='contributing factor vehicle 1', yticks=[], title='Top 5 Contributing factors fo
ax.bar_label(container=ax.containers[0])
```

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



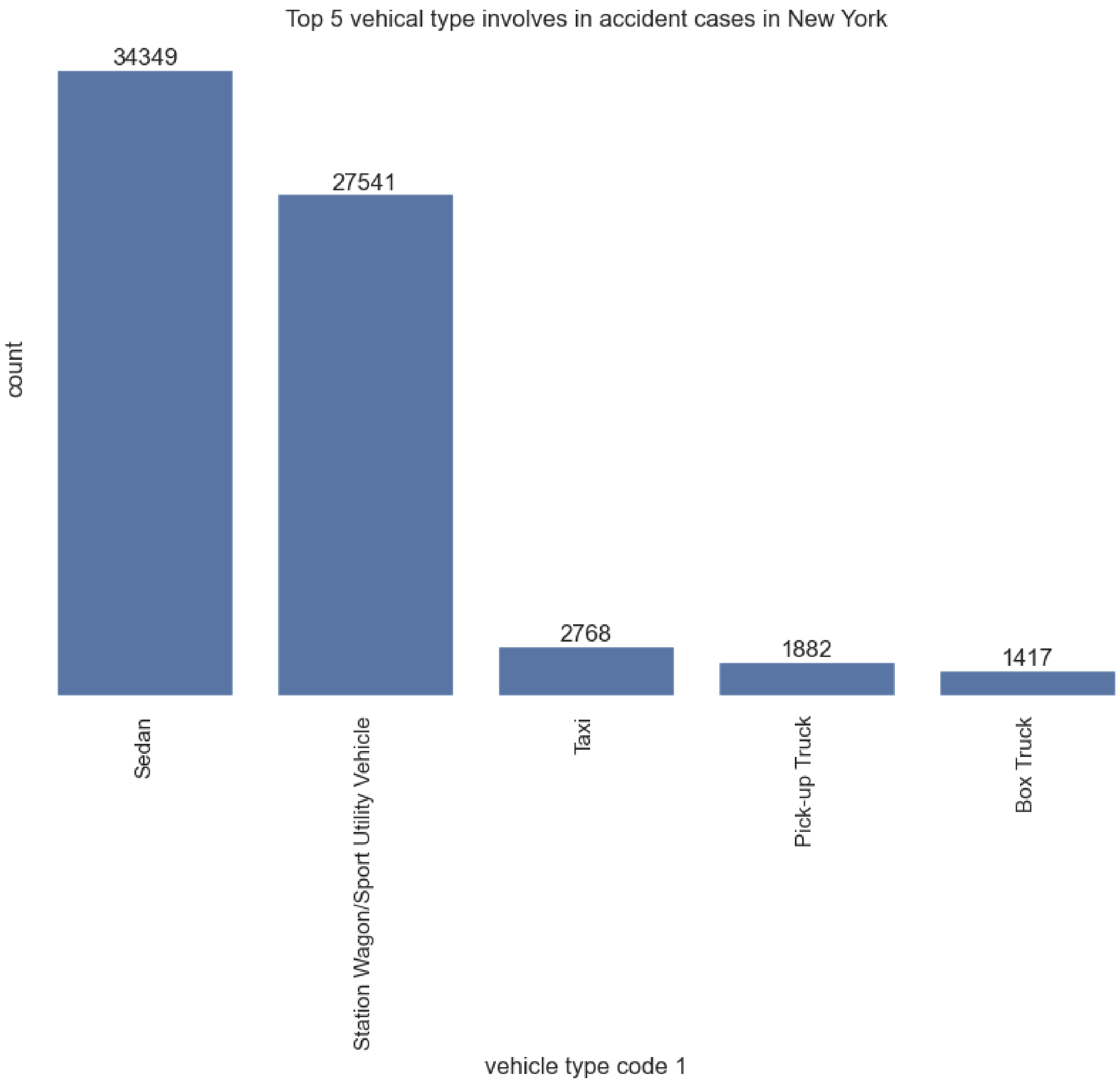
- **The most contributing factors leading to most car accidents in New York are unspecified reasons, Driver Inattention, Too closely following and Passing Lane Usage Improper.**

In [10]:
```
car["vehicle type code 1"].value_counts().nlargest()
```

Out[10]:
```
vehicle type code 1
Sedan                                  34349
Station Wagon/Sport Utility Vehicle    27541
Taxi                                    2768
Pick-up Truck                           1882
Box Truck                               1417
Name: count, dtype: int64
```

In [129…
```
plt.figure(figsize=(10, 6))
sns.set(style='darkgrid')
ax = sns.countplot(x = 'vehicle type code 1',
              data = car,
              order = car['vehicle type code 1'].value_counts().nlargest().index)

ax.set(xlabel='vehicle type code 1', yticks=[], title='Top 5 vehical type involves in acciden
```

```
ax.bar_label(container=ax.containers[0])
plt.xticks(rotation=90)
plt.show()
```



- **The top cars with frequent accident cases in New York are Sedan, Station Wagon, Taxi, Pick-up Truck and Box Truck.**

In [54]:
```
# lets convert all date to datatime
car["crash date"] = pd.to_datetime(car["crash date"], format = "mixed")
```

In [57]:
```
# Extract year, month, and day from the 'date' column
car['year'] = car['crash date'].dt.year
car['month'] = car['crash date'].dt.strftime('%B')
car['day'] = car['crash date'].dt.strftime('%A')
```

In [58]:
```
car.head()
```

Out[58]:

	crash date	crash time	borough	zip code	latitude	longitude	location	on street name	cross street name	s n
0	2020-08-29	15:40:00	BRONX	10466.0	40.89210	-73.833760	POINT (-73.83376 40.8921)	PRATT AVENUE	STRANG AVENUE	
1	2020-08-29	21:00:00	BROOKLYN	11221.0	40.69050	-73.919914	POINT (-73.919914 40.6905)	BUSHWICK AVENUE	PALMETTO STREET	
2	2020-08-29	18:20:00	NaN	NaN	40.81650	-73.946556	POINT (-73.946556 40.8165)	8 AVENUE	NaN	
3	2020-08-29	00:00:00	BRONX	10459.0	40.82472	-73.892960	POINT (-73.89296 40.82472)	NaN	NaN	SIMP ST
4	2020-08-29	17:10:00	BROOKLYN	11203.0	40.64989	-73.933890	POINT (-73.93389 40.64989)	NaN	NaN	SNY AVE

5 rows × 32 columns

In [87]:

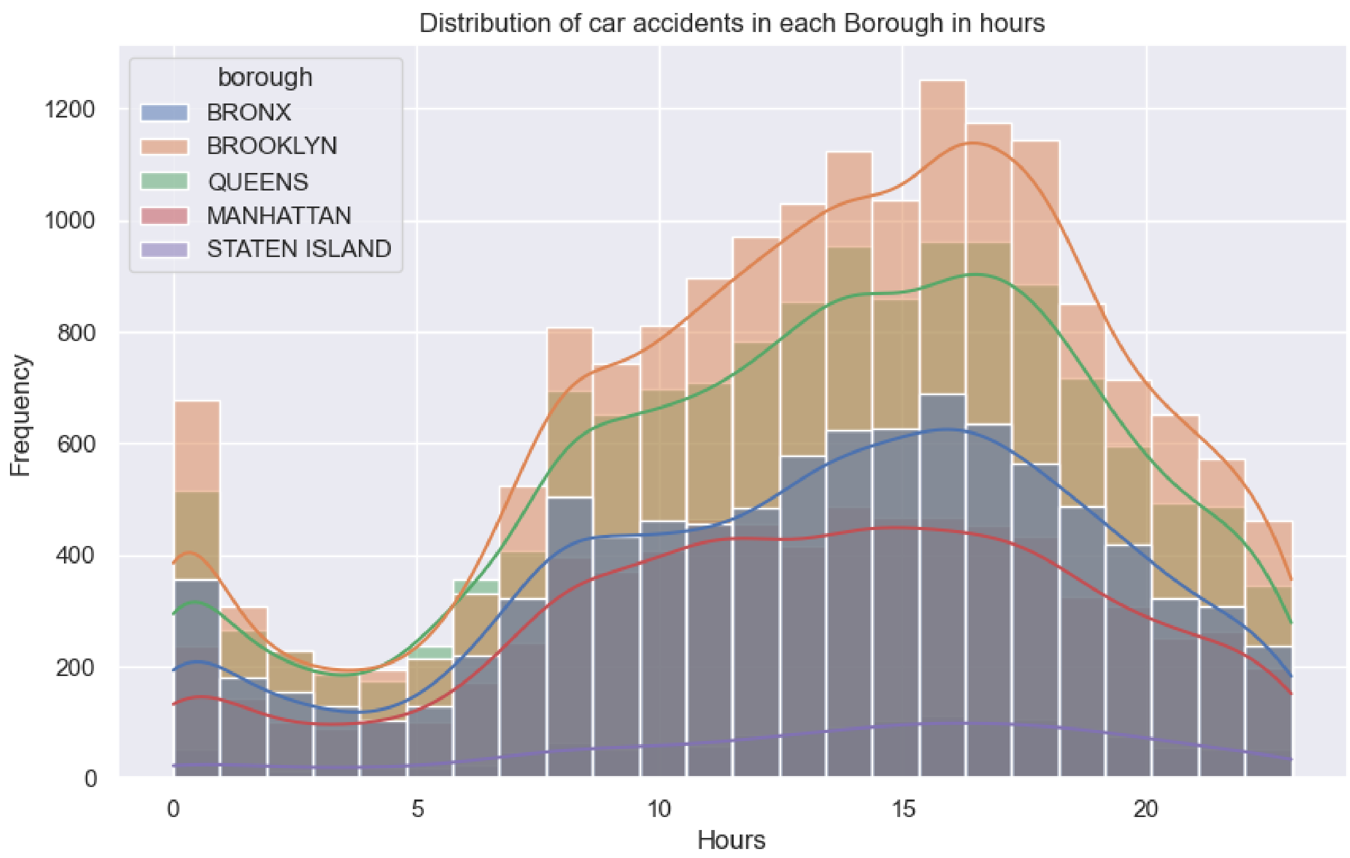
```
# lets convert all date to datatime
car["crash time"] = pd.to_datetime(car["crash time"], format = "mixed")
```

In [89]:

```
car['hours'] = car['crash time'].dt.hour
```

In [130…

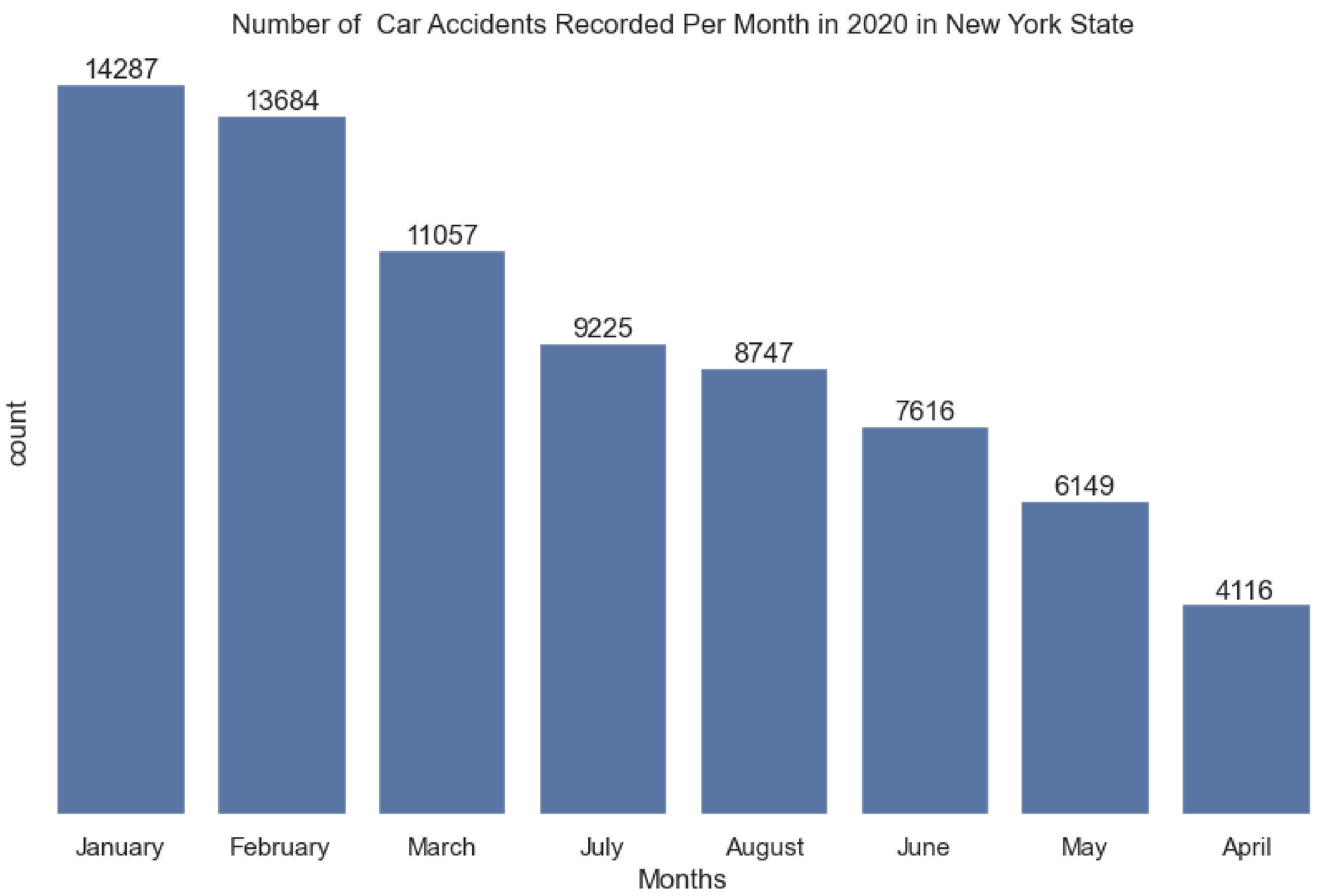
```
plt.figure(figsize=(10, 6))
# Create a Seaborn histogram with KDE
sns.histplot(data= car,
             x = "hours",
             hue = "borough",
             bins=24,
             kde=True)

# 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()
```



- **In all the Borough car accidents cases pike at the same time which is roughly around 14:00pm to 18:00pm, which implies rush hour from work.**

In [126…

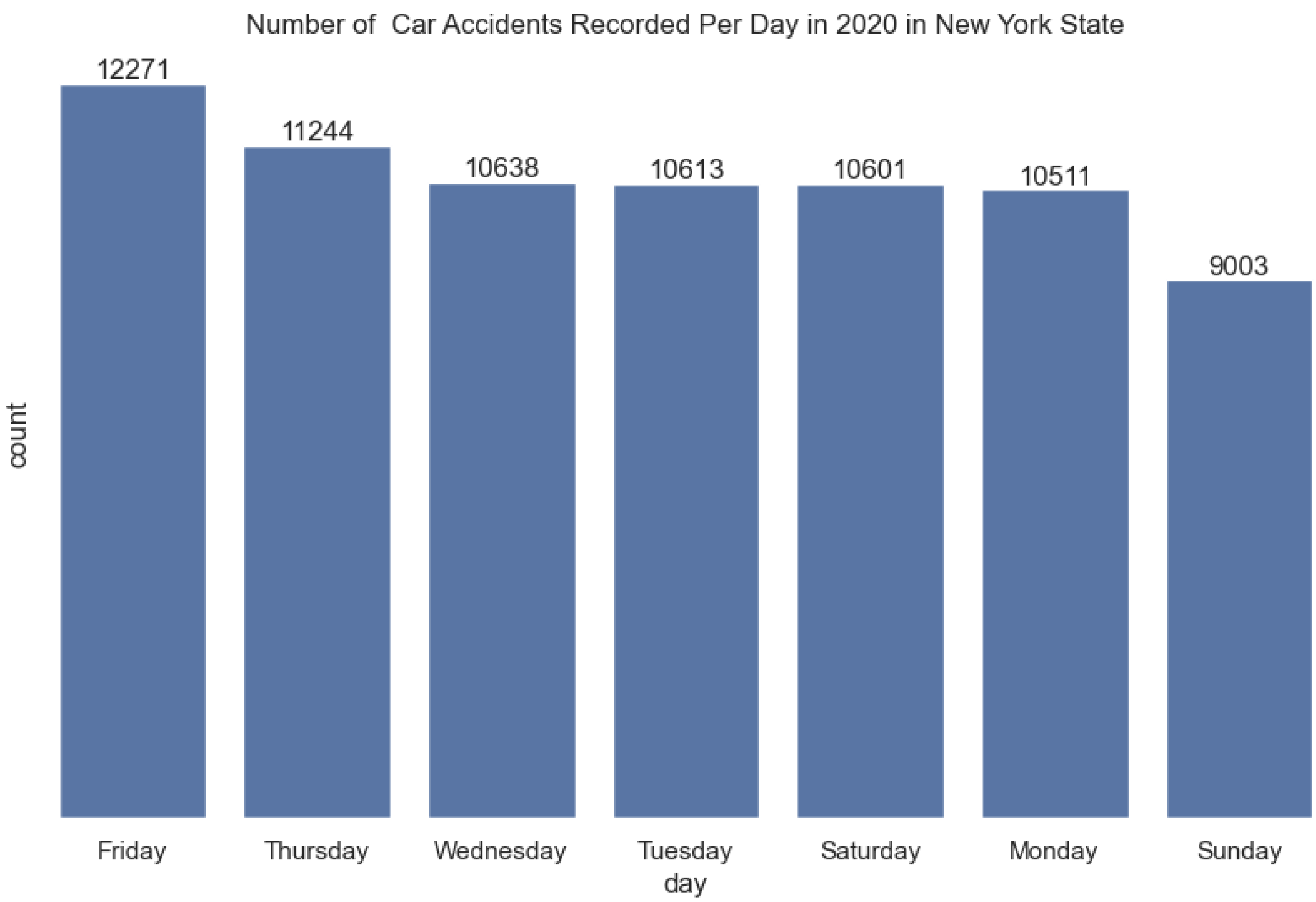
```
plt.figure(figsize=(10, 6))
sns.set(style='darkgrid')
ax = sns.countplot(x = 'month',
              data = car,
              order = car['month'].value_counts().index)

ax.set(xlabel='Months', yticks=[], title='Number of  Car Accidents Recorded Per Month in 2020
ax.bar_label(container=ax.containers[0])
plt.show()
```



- **January February and March recorded the highest cases of car accidents in New York in 2020.**

In [127…

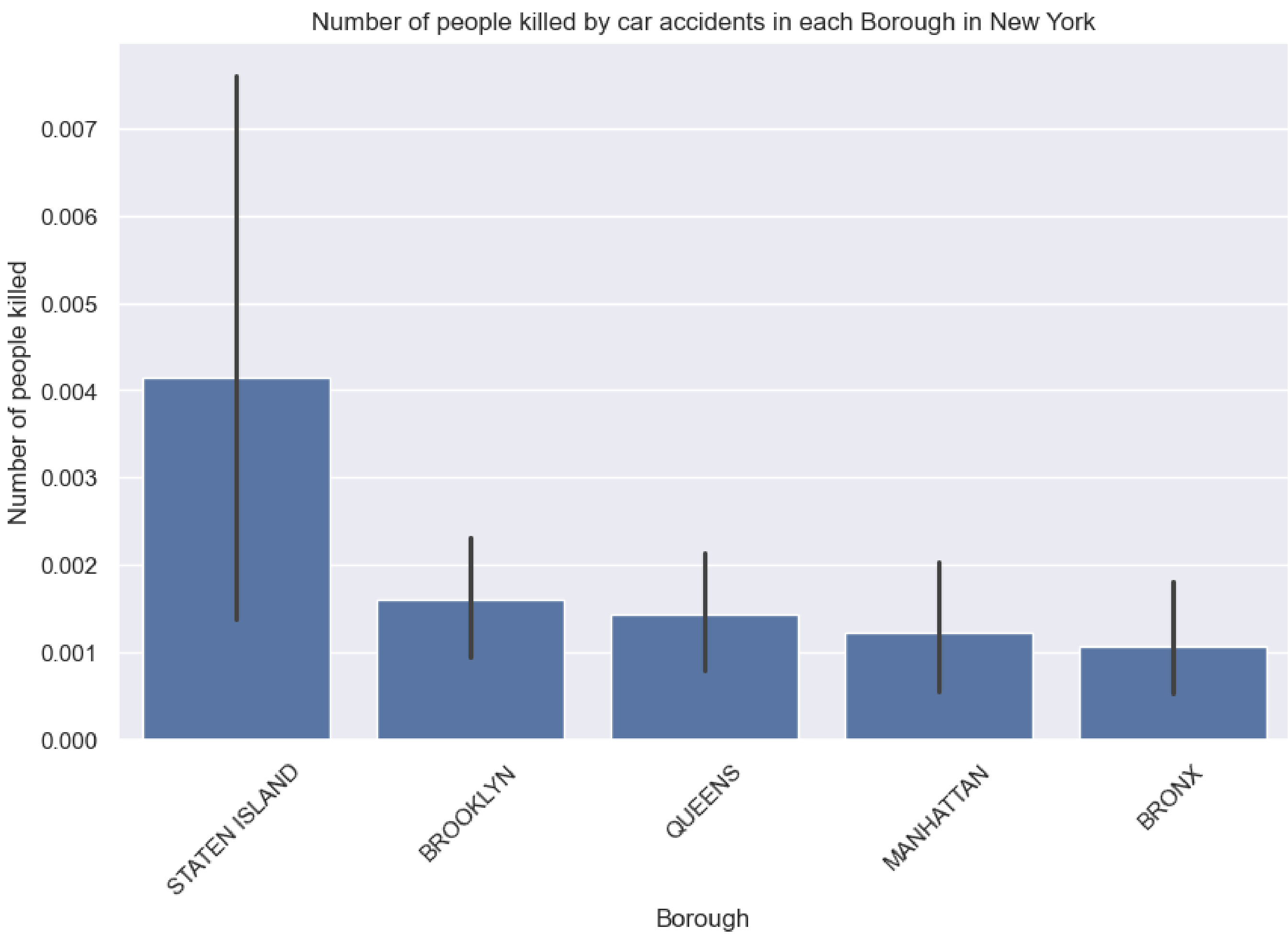
```
plt.figure(figsize=(10, 6))
sns.set(style='darkgrid')
ax = sns.countplot(x = 'day',
              data = car,
             order = car['day'].value_counts().index)
ax.set(xlabel='day', yticks=[], title='Number of  Car Accidents Recorded Per Day in 2020 in Ne
ax.bar_label(ax.containers[0]);
plt.show()
```



- **Most car accidents in New York in 2020 occurs on Friday and Thursday.**

In [116…

```
plt.figure(figsize=(10, 6))
# Create a Seaborn bar plot with hue
ax =sns.barplot(x="borough",
            y="number of persons killed",
            data=car,
            order=car.groupby("borough")["number of persons killed"].mean().sort_values(ascen

# Set labels and title
plt.xlabel('Borough')
plt.ylabel('Number of people killed')
plt.title('Number of people killed by car accidents in each Borough in New York')

# Show the plot
plt.xticks(rotation=45)
plt.show()
```



- **Though Brooklyn had the highest cases of car accidents in 2020, but State Island recorded the highest death cases of car accidents in New York.**

In [12]:
```
car.columns
```

Out[12]:
```
Index(['crash date', 'crash time', 'borough', 'zip code', 'latitude',
       'longitude', 'location', 'on street name', 'cross street name',
       'off street name', 'number of persons injured',
       'number of persons killed', 'number of pedestrians injured',
       'number of pedestrians killed', 'number of cyclist injured',
       'number of cyclist killed', 'number of motorist injured',
       'number of motorist killed', 'contributing factor vehicle 1',
       'contributing factor vehicle 2', 'contributing factor vehicle 3',
       'contributing factor vehicle 4', 'contributing factor vehicle 5',
       'collision_id', 'vehicle type code 1', 'vehicle type code 2',
       'vehicle type code 3', 'vehicle type code 4', 'vehicle type code 5'],
      dtype='object')
```

In [13]:
```
# selecting the needed columns for further analysis
car_1 = car[['crash date', 'crash time', 'borough', 'zip code', 'latitude',
       'longitude', 'location', 'on street name', 'cross street name',
       'off street name']]
```

In [14]:
```
car_1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 74881 entries, 0 to 74880
Data columns (total 10 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   crash date         74881 non-null  object 
 1   crash time         74881 non-null  object 
 2   borough            49140 non-null  object 
 3   zip code           49134 non-null  float64
 4   latitude           68935 non-null  float64
 5   longitude          68935 non-null  float64
 6   location           68935 non-null  object 
 7   on street name     55444 non-null  object 
 8   cross street name  35681 non-null  object 
 9   off street name    19437 non-null  object 
dtypes: float64(3), object(7)
memory usage: 5.7+ MB
```

In [15]:
```
# filling the nan with the most frequent values in all the categorical columns
car_1.fillna(car_1.mode().iloc[0], inplace=True)
```

In [16]:
```
car_1.isna().sum()
```

Out[16]:
```
crash date           0
crash time           0
borough              0
zip code             0
latitude             0
longitude            0
location             0
on street name       0
cross street name    0
off street name      0
dtype: int64
```

In [17]:
```
# checking duplicates
car_1.duplicated().sum()
```

Out[17]:
```
114
```

In [18]:
```
# dropping duplicated in the data
car_3 = car_1.drop_duplicates()
```

In [19]:
```
# making a copy to ensure effecient changes in case
accidents = car_3.copy()
```

- ## Geospatial Data Analysis

In [20]:
```
# convert pandas dataframe to geodataframe
df_1 = gpd.GeoDataFrame(accidents,
                         crs='EPSG:4326',
                         geometry=gpd.points_from_xy(accidents.longitude, accidents.latitude))
```

In [21]:
```
df_1.head()
```

Out[21]:

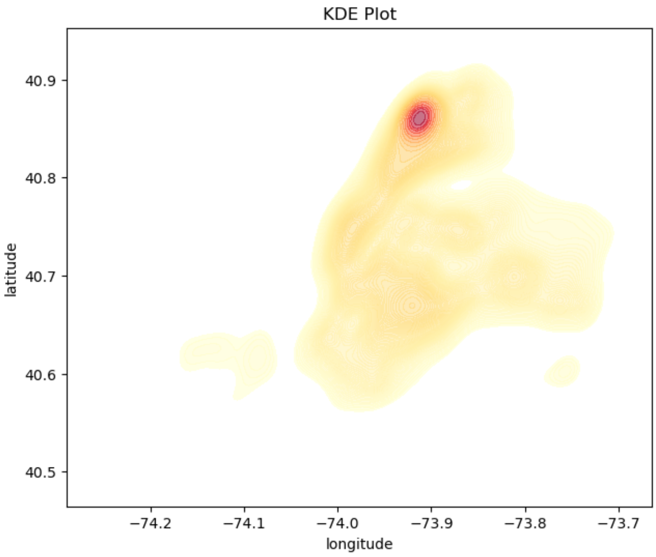
	crash date	crash time	borough	zip code	latitude	longitude	location	on street name	cross street name	of
0	2020-08-29	15:40:00	BRONX	10466.0	40.89210	-73.833760	POINT (-73.83376 40.8921)	PRATT AVENUE	STRANG AVENUE	EDG
1	2020-08-29	21:00:00	BROOKLYN	11221.0	40.69050	-73.919914	POINT (-73.919914 40.6905)	BUSHWICK AVENUE	PALMETTO STREET	EDG
2	2020-08-29	18:20:00	BROOKLYN	11207.0	40.81650	-73.946556	POINT (-73.946556 40.8165)	8 AVENUE	3 AVENUE	EDG
3	2020-08-29	00:00:00	BRONX	10459.0	40.82472	-73.892960	POINT (-73.89296 40.82472)	BELT PARKWAY	3 AVENUE	SI
4	2020-08-29	17:10:00	BROOKLYN	11203.0	40.64989	-73.933890	POINT (-73.93389 40.64989)	BELT PARKWAY	3 AVENUE	S

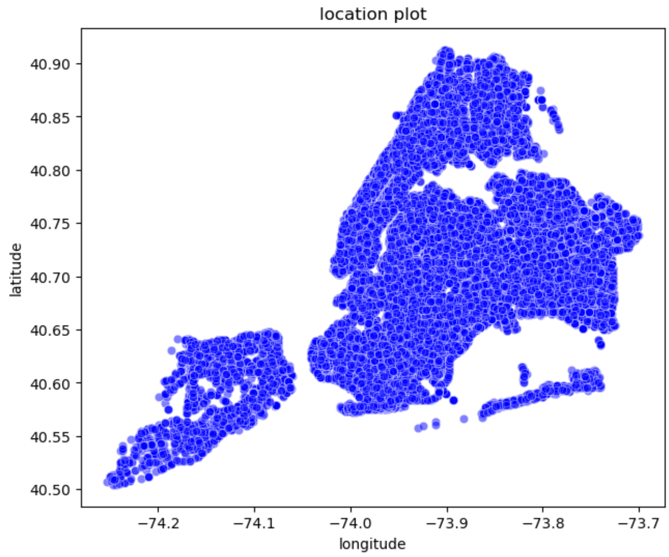
In [22]:

```
# drop the unmapped rows
df_1 = df_1[df_1.longitude!=0]
```

In [23]:

```
# plot the kernel density map
# Set up the figure and axes
fig, axs = plt.subplots(1, 2, figsize=(16, 6))
# Generate and add the KDE plot to the first axis
sns.kdeplot(
data=df_1,
   x="longitude",
   y="latitude",
   n_levels=75,
   shade=True,
   fill=True,
   levels=90,
   alpha=0.55,
   linewidths=1.5,
   cmap="YlOrRd",
   ax=axs[0]
)
# Generate and add the scatter plot to the second axis
sns.scatterplot(
    data=df_1,
    x="longitude",
    y="latitude",
    alpha=0.5,
    color='blue',
    ax=axs[1]
)
# Set titles for the plots
axs[0].set_title("KDE Plot")
axs[1].set_title("location plot")
# Display the plot
plt.show()
```





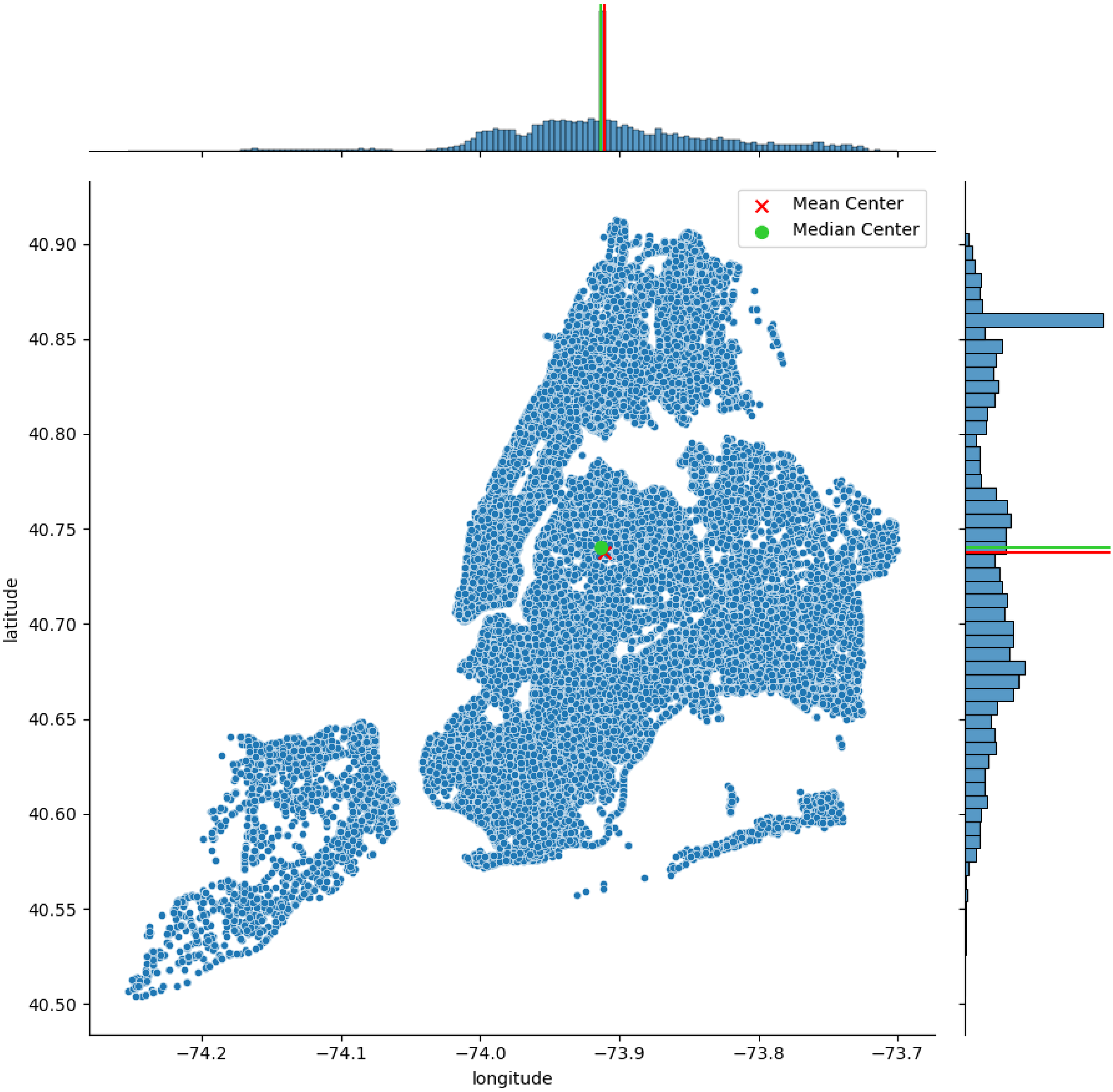
- **Accident occurrence in New York through the kernel density map proves that occurrence of accident are higher in the northern part of New York, while the southern part recorded few cases.**

In [24]:
```
mean_center = centrography.mean_center(df_1[["longitude","latitude"]])
```

In [25]:
```
med_center = centrography.euclidean_median(df_1[["longitude","latitude"]])
print(f"The mean center is : {mean_center}")
print(f"The median center is: {med_center}")
```

```
The mean center is : [-73.91105714  40.73773049]
The median center is: [-73.91362603  40.74042753]
```

In [26]:
```
# Dispaly the mean and median value on a plot
# Generate scatter plot
joint_axes = sns.jointplot(
df_1,
   x="longitude",
   y="latitude",
   s=20,
    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()
# Clean axes
# joint_axes.ax_joint.set_axis_off()
# Display
plt.show()
```



- Though the kernel density shows a higher cluster of car accidents at the north, meanwhile the mean and median accident cases in the geographic area almost overlaps at the center of the map. This may suggests that despite the majority of car accidents are recorded in the north, the central tendency of the distribution is not heavily influenced by the northern outliers. Also there might be spatial variation in the dataset, while the northern part has high density , the distribution is relatively symmetric to balance.

In [27]:
```
# reading in the borough shapefile
boroughs = gpd.read_file("2020 Census Tracts - Tabular_20240110.geojson")
```

In [28]:
```
boroughs.head()
```

Out[28]:

	shape_area	ntaname	cdtaname	shape_leng	boroname	ct2020	nta2020	borocode	cdel
0	1843004.52241	The Battery-Governors Island-Ellis Island-Libe...	MN01 Financial District-Tribeca (CD 1 Equivalent)	10833.0439286	Manhattan	000100	MN0191	1	
1	972312.140355	Chinatown-Two Bridges	MN03 Lower East Side-Chinatown (CD 3 Equivalent)	4754.49524739	Manhattan	000201	MN0301	1	
2	2582705.15746	Chinatown-Two Bridges	MN03 Lower East Side-Chinatown (CD 3 Equivalent)	6976.28621477	Manhattan	000600	MN0301	1	
3	1006116.58429	Lower East Side	MN03 Lower East Side-Chinatown (CD 3 Equivalent)	5075.33199978	Manhattan	001401	MN0302	1	
4	1226206.24719	Lower East Side	MN03 Lower East Side-Chinatown (CD 3 Equivalent)	4459.1560187	Manhattan	001402	MN0302	1	

In [29]:

```
#checking the columns
boroughs.columns
```

Out[29]:

```
Index(['shape_area', 'ntaname', 'cdtaname', 'shape_leng', 'boroname', 'ct2020',
       'nta2020', 'borocode', 'cdeligibil', 'geoid', 'boroct2020', 'cdta2020',
       'ctlabel', 'geometry'],
      dtype='object')
```

In [30]:

```
# selecting the needed columns for further analysis
geo_df = boroughs[["geoid", "shape_area", "boroname", "geometry"]]
```

In [31]:

```
geo_df.head()
```

Out[31]:

	geoid	shape_area	boroname	geometry
0	36061000100	1843004.52241	Manhattan	MULTIPOLYGON (((-74.04388 40.69019, -74.04351 ...
1	36061000201	972312.140355	Manhattan	MULTIPOLYGON (((-73.98450 40.70951, -73.98655 ...
2	36061000600	2582705.15746	Manhattan	MULTIPOLYGON (((-73.99022 40.71440, -73.98934 ...
3	36061001401	1006116.58429	Manhattan	MULTIPOLYGON (((-73.98837 40.71645, -73.98754 ...
4	36061001402	1226206.24719	Manhattan	MULTIPOLYGON (((-73.98507 40.71908, -73.98423 ...

In [32]:
```
# checking the projection of the borogh data
geo_df.crs
```

Out[32]:
```
<Geographic 2D CRS: EPSG:4326>
Name: WGS 84
Axis Info [ellipsoidal]:
- Lat[north]: Geodetic latitude (degree)
- Lon[east]: Geodetic longitude (degree)
Area of Use:
- name: World.
- bounds: (-180.0, -90.0, 180.0, 90.0)
Datum: World Geodetic System 1984 ensemble
- Ellipsoid: WGS 84
- Prime Meridian: Greenwich
```

In [33]:
```
# ensuring the both dataset has the same projection
geo_df.crs==df_1.crs
```

Out[33]:
```
True
```

In [34]:
```
# Do the spatial join
join = gpd.sjoin(geo_df, df_1, how='left')
join.head()
```

Out[34]:

	geoid	shape_area	boroname	geometry	index_right	crash date	crash time	borough
0	36061000100	1843004.52241	Manhattan	MULTIPOLYGON (((-74.04388 40.69019, -74.04351 ...	NaN	NaN	NaN	NaN
1	36061000201	972312.140355	Manhattan	MULTIPOLYGON (((-73.98450 40.70951, -73.98655 ...	7160.0	2020-08-05	05:00:00	MANHATTAN
1	36061000201	972312.140355	Manhattan	MULTIPOLYGON (((-73.98450 40.70951, -73.98655 ...	29491.0	2020-05-13	00:00:00	BROOKLYN
1	36061000201	972312.140355	Manhattan	MULTIPOLYGON (((-73.98450 40.70951, -73.98655 ...	33910.0	2020-04-15	18:19:00	BROOKLYN
1	36061000201	972312.140355	Manhattan	MULTIPOLYGON (((-73.98450 40.70951, -73.98655 ...	9347.0	2020-07-30	22:40:00	MANHATTAN

In [35]:
```
# calculating the number of car accidents in each census tracts
accidents_gdf = join["geoid"].value_counts().rename_axis('geoid').reset_index(name='total_acc
```

In [36]:
```
accidents_gdf.head()
```

Out[36]:

	geoid	total_accident
0	36005025700	6052
1	36081038302	445
2	36081030600	330
3	36005001902	278
4	36005006301	231

In [37]:

```
# merging the summary table back to the gdf
gdf_1=geo_df.merge(accidents_gdf,on='geoid')
```

In [38]:

```
gdf_1.head()
```

Out[38]:

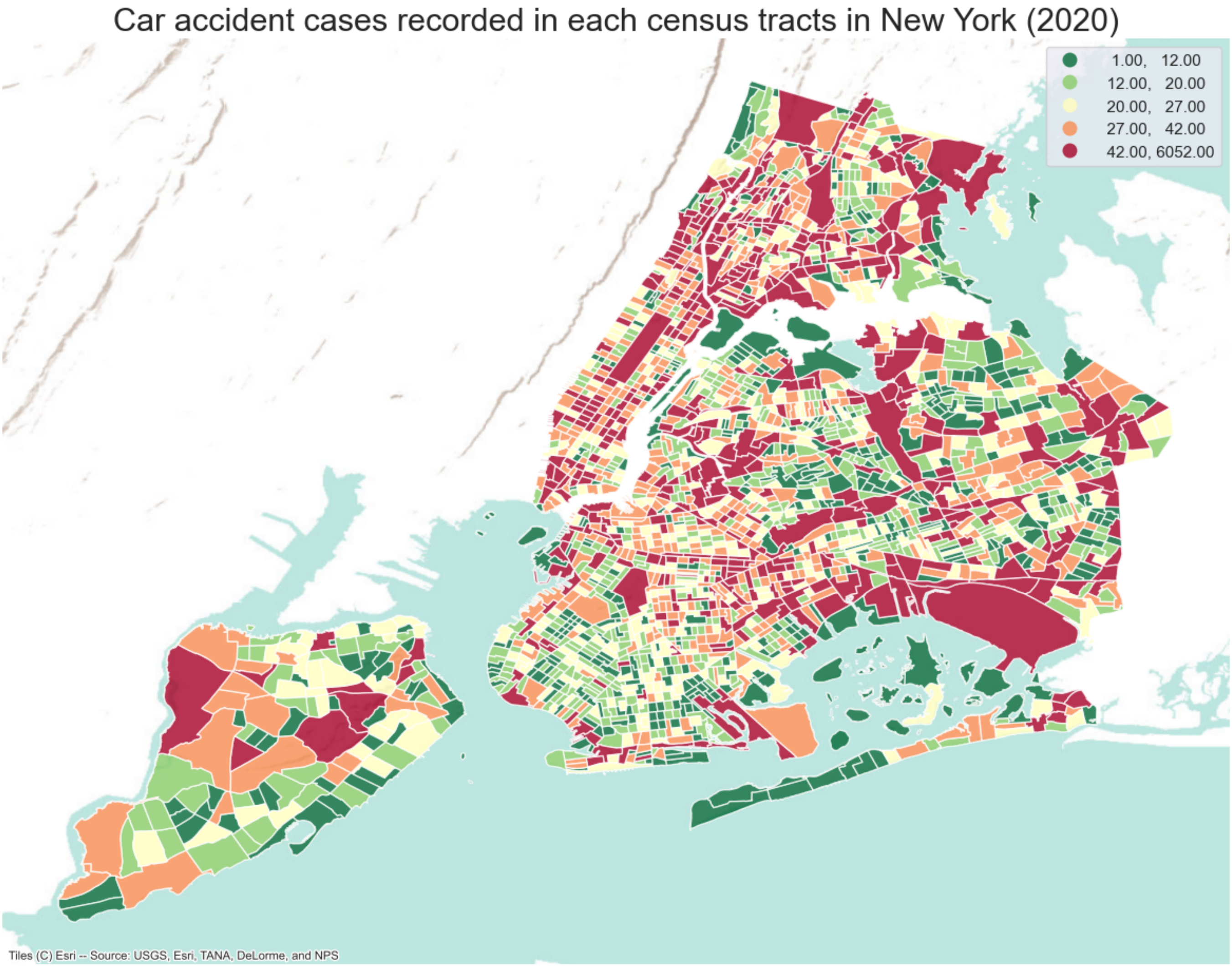
	geoid	shape_area	boroname	geometry	total_accident
0	36061000100	1843004.52241	Manhattan	MULTIPOLYGON (((-74.04388 40.69019, -74.04351 ...	1
1	36061000201	972312.140355	Manhattan	MULTIPOLYGON (((-73.98450 40.70951, -73.98655 ...	21
2	36061000600	2582705.15746	Manhattan	MULTIPOLYGON (((-73.99022 40.71440, -73.98934 ...	30
3	36061001401	1006116.58429	Manhattan	MULTIPOLYGON (((-73.98837 40.71645, -73.98754 ...	12
4	36061001402	1226206.24719	Manhattan	MULTIPOLYGON (((-73.98507 40.71908, -73.98423 ...	38

In [39]:

```
basemap = ctx.providers.OpenStreetMap.Mapnik(style='dark_all')
```

In [118…

```
fig,ax = plt.subplots(figsize=(15,15))

gdf_1.plot(ax=ax,
        column='total_accident', # this makes it a choropleth
        legend=True,
        alpha=0.8,
        cmap='RdYlGn_r', # a diverging color scheme
        scheme='quantiles') # how to break the data into bins

ax.axis('off')
ax.set_title('Car accident cases recorded in each census tracts in New York (2020)',fontsize=
# Add basemap
ctx.add_basemap(
    ax,
    crs=gdf_1.crs,
    source=ctx.providers.Esri.WorldTerrain,)
```



- Map showing the distribution of car accidents in each census tracts in New York

- # Exploratory Spatial Data Analysis

In [41]:
```
# calculate spatial weight
wq =  lps.weights.KNN.from_dataframe(gdf_1,k=8)

# Row-standardization
wq.transform = 'r'
```

- Spatial weight matrix is a technique needed to choose the nearest neighbor for further analysis which can be rook contiguity, Queen, and KNN matrix this project use KNN matrix.

In [42]:
```
# create a new column for the spatial lag
gdf_1['accident_lag'] = lps.weights.lag_spatial(wq, gdf_1['total_accident'])
```

- Spatial lag is a byproduct of the spatial weights. Spatial lag is a variable that average the values of the nearest neighbors selected by the spatial weights. This will smooth the selected point to set a pattern on the map.

In [43]:
```
gdf_1.head()
```

Out[43]:

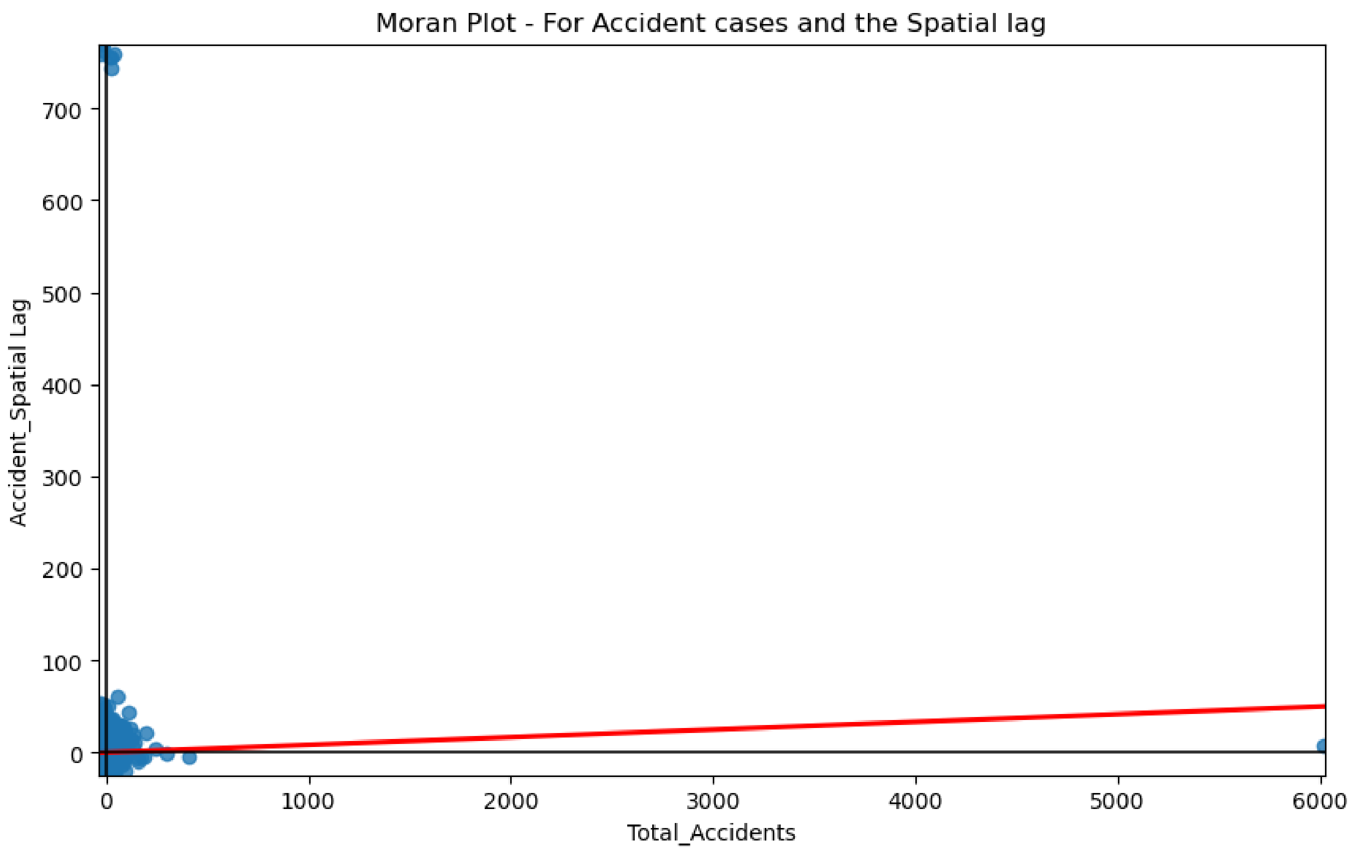
	geoid	shape_area	boroname	geometry	total_accident	accident_lag
0	36061000100	1843004.52241	Manhattan	MULTIPOLYGON (((-74.04388 40.69019, -74.04351 ...	1	20.000
1	36061000201	972312.140355	Manhattan	MULTIPOLYGON (((-73.98450 40.70951, -73.98655 ...	21	29.875
2	36061000600	2582705.15746	Manhattan	MULTIPOLYGON (((-73.99022 40.71440, -73.98934 ...	30	42.125
3	36061001401	1006116.58429	Manhattan	MULTIPOLYGON (((-73.98837 40.71645, -73.98754 ...	12	45.375
4	36061001402	1226206.24719	Manhattan	MULTIPOLYGON (((-73.98507 40.71908, -73.98423 ...	38	33.625

In [44]:

```
# Standardize the variable of interest and the lag variable
gdf_1['accident_std'] = (gdf_1['total_accident'] -
gdf_1['total_accident'].mean())
gdf_1['accident_lag_std'] = (gdf_1['accident_lag'] -
gdf_1['accident_lag'].mean())
```

In [45]:

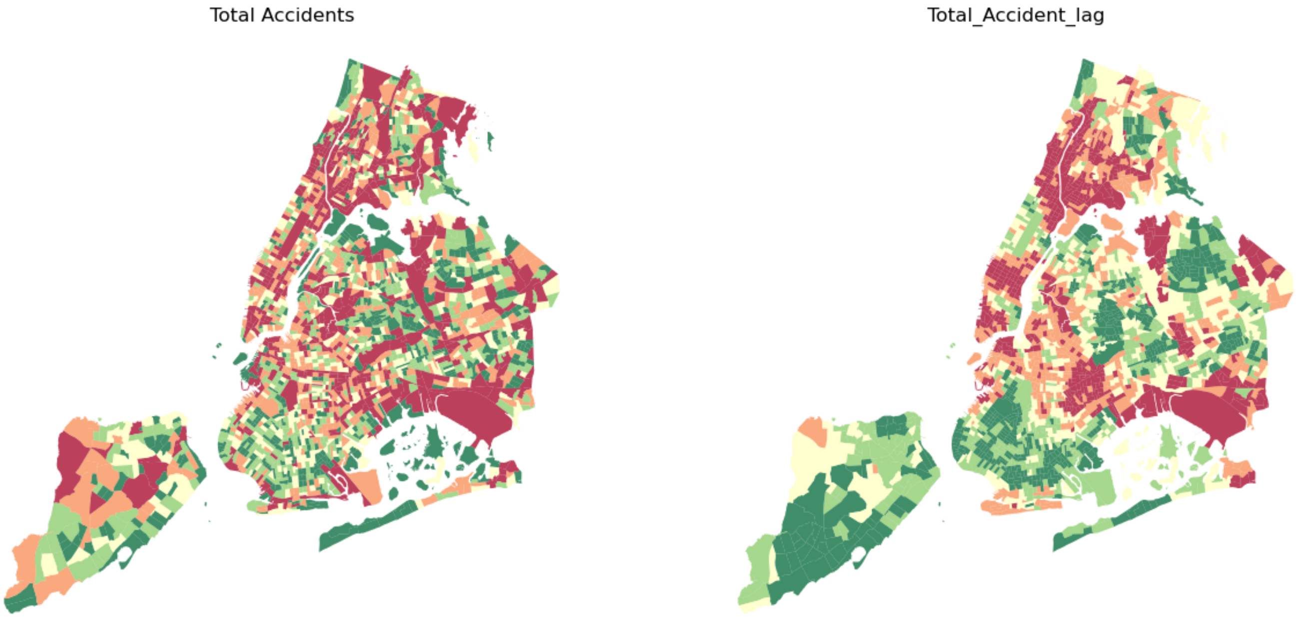
```
f, ax = plt.subplots(1, figsize=(10, 6))
sns.regplot(
x='accident_std', # variable of interest
y='accident_lag_std', # spatial lag
ci=None, # suppress the plotting of the confidence interval
data=gdf_1, # dataset
line_kws={'color':'r'}
)

# Rescale the plot to smaller x and y values
# Adjust the x-axis and y-axis limits to zoom out
plt.xlim(left=min(gdf_1['accident_std']) - 5, right=max(gdf_1['accident_std']) + 5)
plt.ylim(bottom=min(gdf_1['accident_lag_std']) - 1, top=max(gdf_1['accident_lag_std']) + 1)

ax.axvline(0, c='k', alpha=0.8)
ax.axhline(0, c='k', alpha=0.8)
ax.set_title('Moran Plot - For Accident cases and the Spatial lag')
ax.set_xlabel('Total_Accidents')
ax.set_ylabel('Accident_Spatial Lag')
plt.show()
```



- From the graph above, the data points are spread upper-right quadrant and the lower-right quadrant base on the best fit line. The graph also implies that there is a positive spatial autocorrelation suggesting a cluster pattern where neighboring location exhibits similar values. This graph will be much importance when analyzing Local Indicators of Spatial Association in our further analysis.

In [46]:
```
# create the 1x2 subplots
fig, ax = plt.subplots(1, 2, figsize=(15, 8))

# two subplots produces ax[0] (left) and ax[1] (right)

# regular count map on the left
gdf_1.plot(ax=ax[0], # this assigns the map to the left subplot
         column='total_accident',
         cmap='RdYlGn_r',
         scheme='quantiles',
         k=5,
         edgecolor='white',
         linewidth=0,
         alpha=0.75,
           )


ax[0].axis("off")
ax[0].set_title("Total Accidents")

# spatial lag map on the right
gdf_1.plot(ax=ax[1], # this assigns the map to the right subplot
         column='accident_lag',
         cmap='RdYlGn_r',
         scheme='quantiles',
         k=5,
         edgecolor='white',
         linewidth=0,
         alpha=0.75
           )
```

```
ax[1].axis("off")
ax[1].set_title("Total_Accident_lag")

plt.show()
```



- From the maps above we can observed the difference between the actual accident cases map and the accident spatial lag map.The patterns in the actual accident cases are roughly clustering in various geographical space whiles the spatial lag map has some smooth observation in the pattern and clustering of cases in particular location.

In [47]:
```
morans_stat = esda.moran.Moran(gdf_1['total_accident'], wq)
display(Markdown(f"""**Morans I:** {morans_stat.I}"""))
display(Markdown(f"""**p-value:** {morans_stat.p_sim}"""))
```

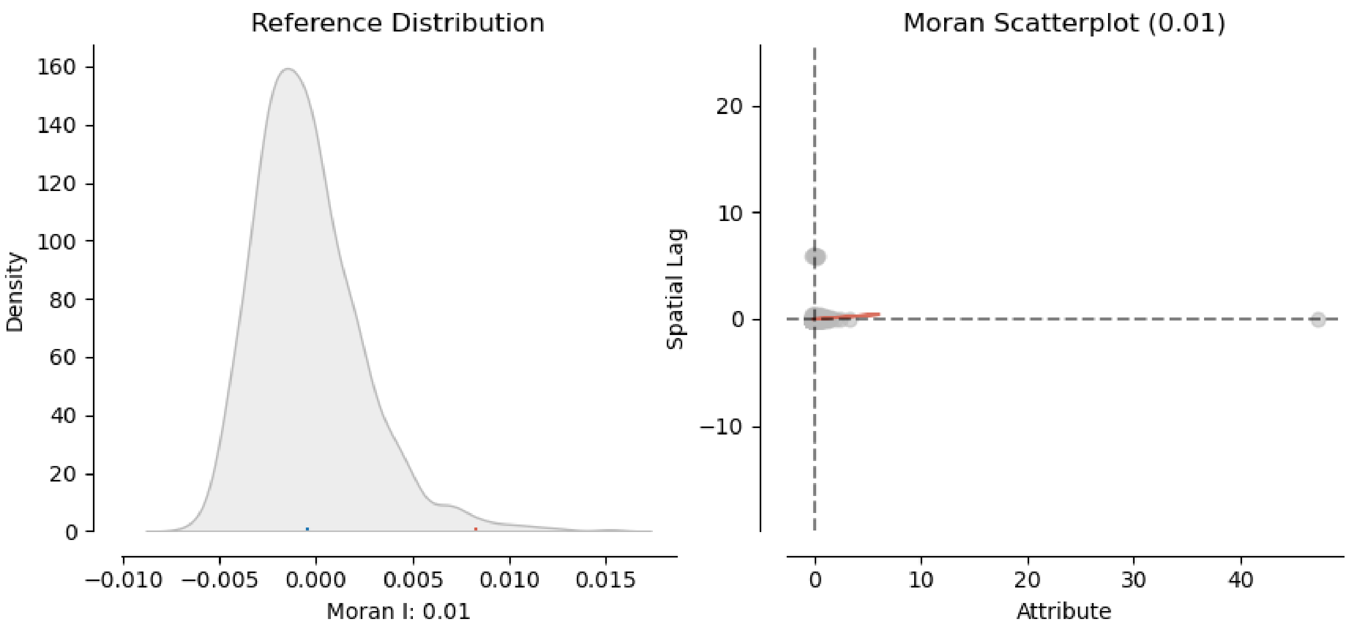
Morans I: 0.008322801690229836

p-value: 0.012

- The Morans I ranges from -1 to 1.A positive values suggest positive spatial autocorrelation, meaning similar values tends to be closer to each other. In this project the Moran's value is (0.0083) indicating a weak positive spatial autocorrelation.
- The p-value is associate with a hypothesis test. It test the null hypothesis. The p-value in this project is less than the significance level of 0.05, suggesting that we REJECT THE NULL HYPOTHESIS. Therefore we can say that there is a spatial autocorrelation.
- So both the Moaron I and the p-value together proves that there is evidence of positive spatial autocorrelation, hence accident points are likely not due to random chance.

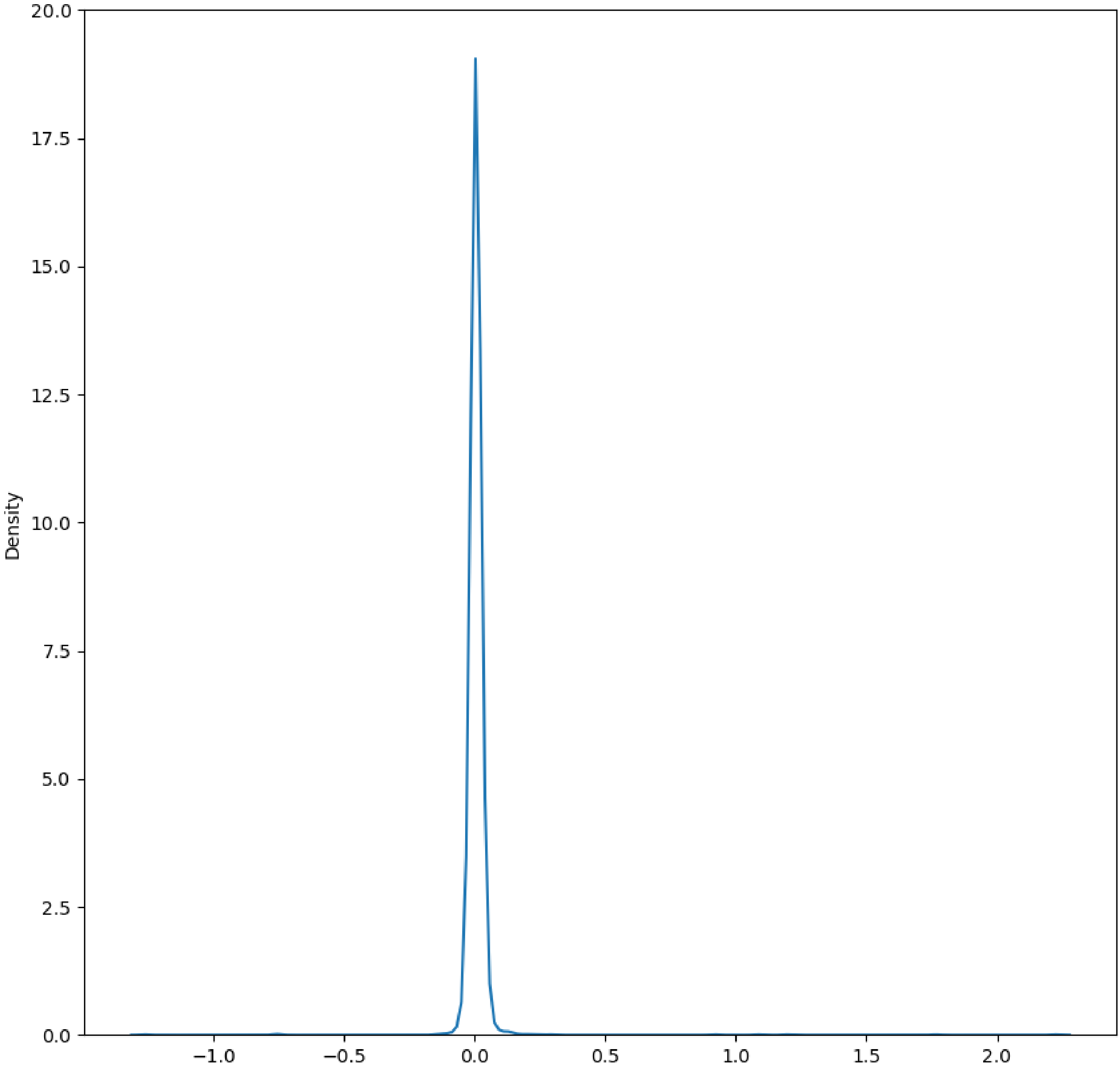
In [48]:
```
plot_moran(morans_stat)
```

Out[48]:
```
(<Figure size 1000x400 with 2 Axes>,
 array([<Axes: title={'center': 'Reference Distribution'}, xlabel='Moran I: 0.01', ylabel='De
nsity'>,
        <Axes: title={'center': 'Moran Scatterplot (0.01)'}, xlabel='Attribute', ylabel='Spat
ial Lag'>],
       dtype=object))
```



In [49]:
```
accident_lisa = esda.moran.Moran_Local(gdf_1["total_accident"], wq)
```

In [50]:
```
# Draw KDE line
f, ax = plt.subplots(1, figsize=(10, 10))
sns.kdeplot(accident_lisa.Is, ax=ax)
plt.show()
```

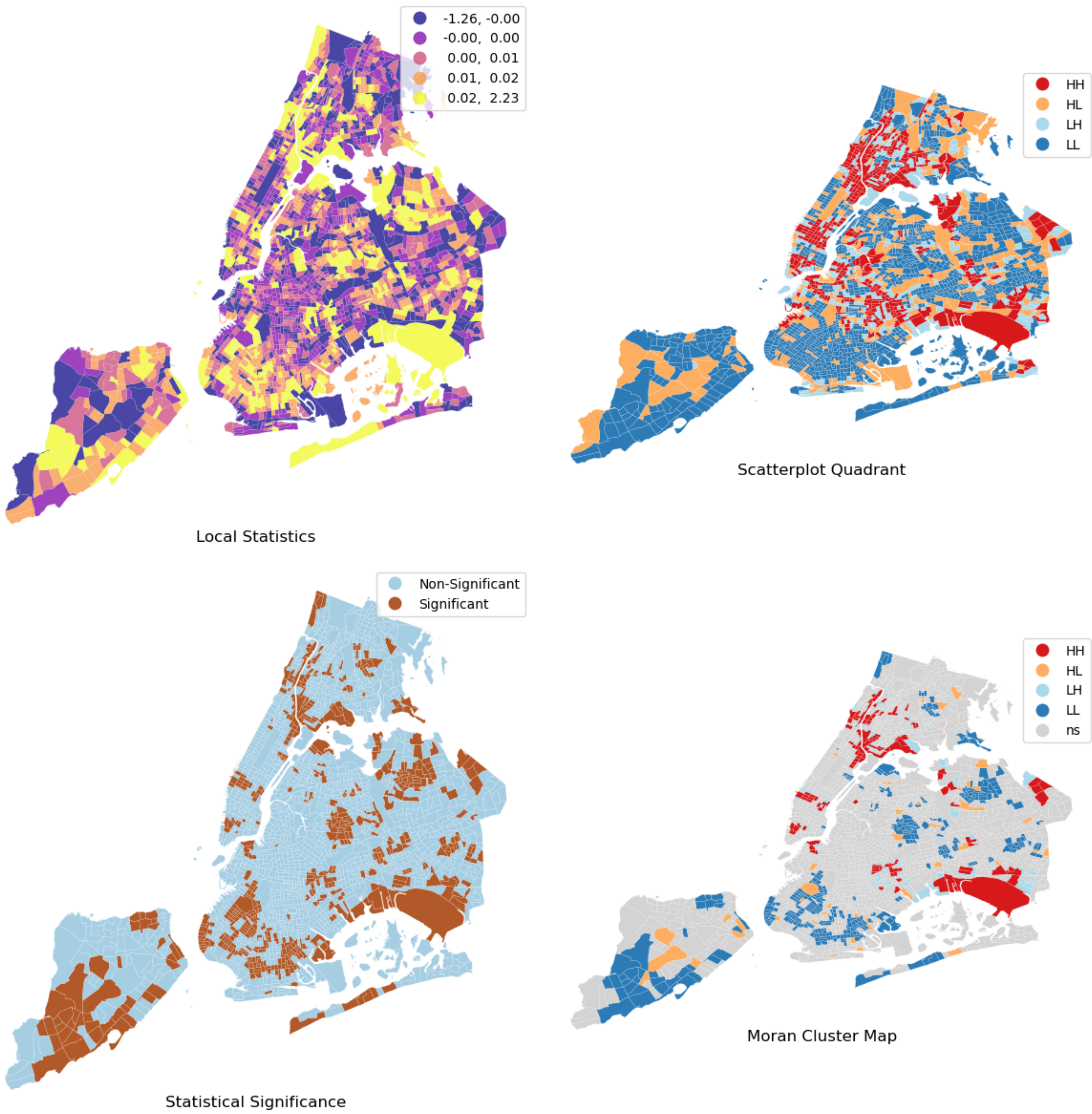


- From the distribution we can observed that there is a massive spike in the data around 0, with along right tail. This is mainly due to the presence of large number of observations with positive spatial autocorrelation. This is in line with what we observed from the global measures.

In [53]:
```
# Set up figure and axes
f, axs = plt.subplots(nrows=2, ncols=2, figsize=(12, 12))
# Make the axes accessible with single indexing
axs = axs.flatten()

# Subplot 1 #
# Choropleth of local statistics
# Grab first axis in the figure
ax = axs[0]
# Assign new column with local statistics on-the-fly
gdf_1.assign(
    ML_Is=accident_lisa.Is
    # Plot choropleth of local statistics
).plot(
    column="ML_Is",
    cmap="plasma",
    scheme="quantiles",
    k=5,
```

```python
    edgecolor="white",
    linewidth=0.1,
    alpha=0.75,
    legend=True,
    ax=ax,
)

# Subplot 2 #
# Quadrant categories
# Grab second axis of local statistics
ax = axs[1]
# Plot Quadrant colors (note to ensure all polygons are assigned a
# quadrant, we "trick" the function by setting significance level to
# 1 so all observations are treated as "significant" and thus assigned
# a quadrant color
esdaplot.lisa_cluster(accident_lisa, gdf_1, p=1, ax=ax)

# Subplot 3 #
# Significance map
# Grab third axis of local statistics
ax = axs[2]
#
# Find out significant observations
labels = pd.Series(
    1 * (accident_lisa.p_sim < 0.05),  # Assign 1 if significant, 0 otherwise
    index=gdf_1.index  # Use the index in the original data
    # Recode 1 to "Significant and 0 to "Non-significant"
).map({1: "Significant", 0: "Non-Significant"})
# Assign labels to `db` on the fly
gdf_1.assign(
    cl=labels
    # Plot choropleth of (non-)significant areas
).plot(
    column="cl",
    categorical=True,
    k=2,
    cmap="Paired",
    linewidth=0.1,
    edgecolor="white",
    legend=True,
    ax=ax,
)


# Subplot 4 #
# Cluster map
# Grab second axis of local statistics
ax = axs[3]
# Plot Quadrant colors In this case, we use a 5% significance
# level to select polygons as part of statistically significant
# clusters
esdaplot.lisa_cluster(accident_lisa, gdf_1, p=0.05, ax=ax)

# Figure styling #
# Set title to each subplot
for i, ax in enumerate(axs.flatten()):
    ax.set_axis_off()
    ax.set_title(
        [
            "Local Statistics",
            "Scatterplot Quadrant",
            "Statistical Significance",
            "Moran Cluster Map",
```

```
        ][i],
        y=0,
    )
# Tight layout to minimize in-between white space
f.tight_layout()

# Display the figure
plt.show()
```



In [124…

```
counts = pd.value_counts(accident_lisa.q)
counts
```

Out[124…

```
3    1205
1     401
2     396
4     323
Name: count, dtype: int64
```

In [122…

```
(accident_lisa.p_sim < 0.05).sum() * 100 / len(accident_lisa.p_sim)
```

Out[122…

```
24.731182795698924
```

- The count shows that lower-lower(LL-3) followers by high-high (HH -1) values are the predominant in this project. In the first two maps the statistical significant of the local values are not considered so care must be taking on it interpretation, since we just mapped the original LISA values alongside the quadrant.
- The bottom left map differentiate census tracts whose p-values are above (Non-significant) or below (Significant) the threshold values of 0.05.
- As low as over 24.7% of the census tracts were considered by this analysis to be part of spatial cluster.

- # Conclusions

- In the bustling streets of New York City in 2020, a concerning pattern emerged as car accidents became a prominent issue, particularly in Brooklyn and Queens. The analysis revealed that the top five streets prone to accidents were Belt Parkway, Long Island Expressway, Brooklyn Queens Expressway, FDR Drive, and Major Deegan Expressway. Unspecified reasons, driver inattention, following too closely, and improper passing lane usage were identified as the leading factors contributing to these incidents.
- The temporal analysis indicated that car accidents peaked between 14:00 and 18:00, corresponding to rush hour traffic. Furthermore, January, February, and March recorded the highest number of accidents, with Fridays and Thursdays being the riskiest days. Surprisingly, although Brooklyn led in overall accident cases, Staten Island suffered the highest number of fatalities.
- The spatial autocorrelation analysis unveiled a cluster pattern, with higher accident occurrences in the northern part of New York. However, the central tendency analysis suggested that despite the high density in the north, the overall distribution remained relatively symmetric, indicating a balance.
- Examining census tracts through kernel density and spatial lag maps highlighted the importance of local indicators of spatial association. The analysis revealed positive spatial autocorrelation, suggesting that neighboring locations exhibited similar accident values, and the presence of clusters was not due to random chance.
- Moran's I value of 0.0083 confirmed weak positive spatial autocorrelation, further supported by a p-value less than 0.012. This collective evidence indicated that accidents were not randomly distributed, emphasizing the need for targeted interventions.
- The distribution analysis showed a spike in data around 0 with a right tail, indicating a substantial number of observations with positive spatial autocorrelation. Lower-lower (LL-3) and high-high (HH-1) values were predominant, emphasizing the clustering pattern observed in the analysis.
- The final map differentiating between significant and non-significant p-values identified over 24.7% of census tracts as part of a spatial cluster. This information is crucial for policymakers, as it points towards areas where targeted interventions and policies can be implemented to reduce car accidents and enhance road safety.
- In conclusion, understanding the spatial autocorrelation of car accidents in New York City in 2020 allows for informed policy recommendations. Focusing on the identified clusters and addressing specific contributing factors, such as driver inattention and improper lane usage, can contribute to a safer and more secure transportation environment for all residents.

@ Benjamin Ababio

In [ ]: