COMS4060A/7056A: Assignment #2

Olatomiwa Akinlaja

Kudzai Sibanda

NYC Taxi Data

In [1]:
```
import warnings
warnings.filterwarnings('ignore')
import pandas as pd
import matplotlib.pyplot as plt
from shapely.geometry import Point
import geopandas as gpd
import folium
from folium.plugins import HeatMapWithTime
from folium.plugins import HeatMap
from branca.element import Figure
from shapely.geometry import Polygon, Point
import geoplot as gplt
import geoplot.crs as gcrs
import imageio
import pathlib
import mapclassify as mc
import geocoder
from folium import plugins
```

In [2]:
```
import hdbscan
import utm
from ipywidgets import interact, interactive
import ipywidgets as widgets
from folium.vector_layers import CircleMarker
from colour import Color
import numpy as np
import math
import seaborn as sns
```

1.1 Data Cleaning

In [3]:
```
nyc_data = pd.read_csv('nyc_taxis.csv')
```

In [4]:
```
nyc_data.isnull().sum()
```

Out[4]:
```
id                  0
vendor_id           0
pickup_datetime     0
dropoff_datetime    0
passenger_count     0
pickup_longitude    0
pickup_latitude     0
dropoff_longitude   0
dropoff_latitude    0
store_and_fwd_flag  0
trip_duration       0
dtype: int64
```

In [5]:
```
nyc_data.head()
```

Out[5]:

	id	vendor_id	pickup_datetime	dropoff_datetime	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	store_and_fwd_flag	trip_duration
0	id2875421	2	2016-03-14 17:24:55	2016-03-14 17:32:30	1	-73.982155	40.767937	-73.964630	40.765602	N	455
1	id2377394	1	2016-06-12 00:43:35	2016-06-12 00:54:38	1	-73.980415	40.738564	-73.999481	40.731152	N	663
2	id3858529	2	2016-01-19 11:35:24	2016-01-19 12:10:48	1	-73.979027	40.763939	-74.005333	40.710087	N	2124
3	id3504673	2	2016-04-06 19:32:31	2016-04-06 19:39:40	1	-74.010040	40.719971	-74.012268	40.706718	N	429
4	id2181028	2	2016-03-26 13:30:55	2016-03-26 13:38:10	1	-73.973053	40.793209	-73.972923	40.782520	N	435

In [6]:
```
nyc_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1458644 entries, 0 to 1458643
Data columns (total 11 columns):
 #   Column              Non-Null Count    Dtype
---  ------              --------------    -----
 0   id                  1458644 non-null  object
 1   vendor_id           1458644 non-null  int64
 2   pickup_datetime     1458644 non-null  object
 3   dropoff_datetime    1458644 non-null  object
 4   passenger_count     1458644 non-null  int64
 5   pickup_longitude    1458644 non-null  float64
 6   pickup_latitude     1458644 non-null  float64
 7   dropoff_longitude   1458644 non-null  float64
 8   dropoff_latitude    1458644 non-null  float64
 9   store_and_fwd_flag  1458644 non-null  object
 10  trip_duration       1458644 non-null  int64
dtypes: float64(4), int64(3), object(4)
memory usage: 122.4+ MB
```

In [7]:
```
len(nyc_data) # current number of entries pre-cleaning
```

Out[7]:
```
1458644
```

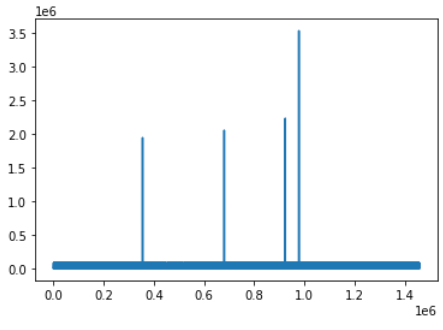
In [8]:
```
# Trip Duration Outliers
nyc_data['trip_duration'].max() # A trip taking over 3 million seconds is highly unlikely
```

Out[8]:
```
3526282
```

In [9]:
```
duration = ((nyc_data.trip_duration < 60) | # < 1 min
            (nyc_data.trip_duration > 3600*2)) # > 2 hours
print('Outliers in trip duration: {:.2f} %'.format((nyc_data[duration].shape[0] / nyc_data.shape[0]) * 100))
```

```
Outliers in trip duration: 0.74 %
```

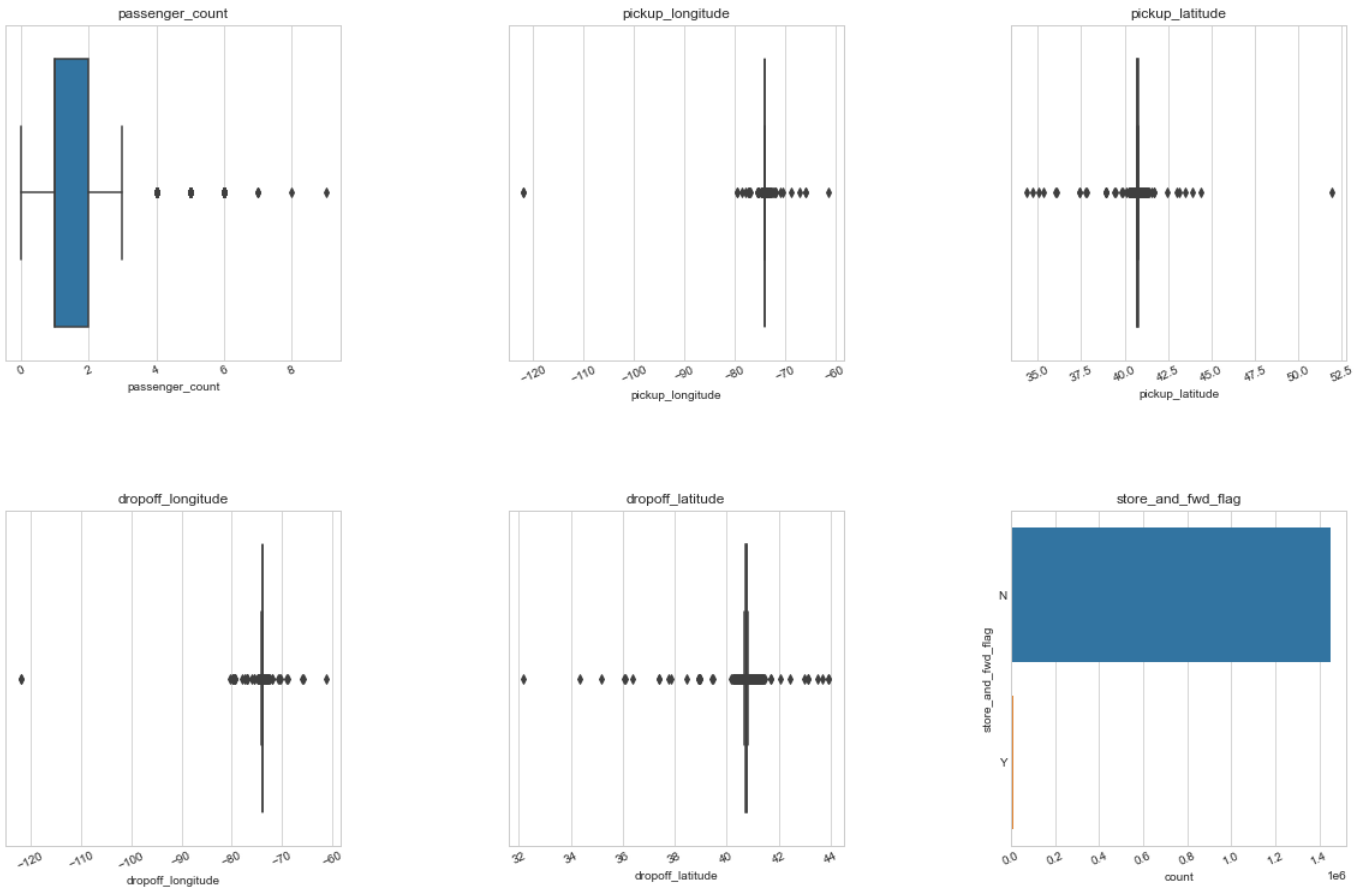
In [10]:
```
nyc_data['trip_duration'].plot()
plt.show()
```



In [11]:
```
# Well generally a trip duration should not be more than a day, which is 86400 seconds.
print('Trip duration in seconds: {} to {}'.format(nyc_data.trip_duration.min(), nyc_data.trip_duration.max()))
```

```
Trip duration in seconds: 1 to 3526282
```

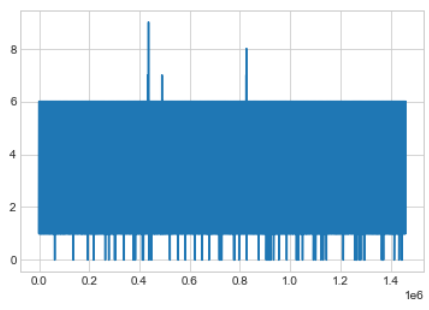
In [12]:
```
def plot_distribution(nyc_data, cols=9, width=20, height=15, hspace=0.2, wspace=0.5):
    plt.style.use('seaborn-whitegrid')
    fig = plt.figure(figsize=(width,height))
    fig.subplots_adjust(left=None, bottom=None, right=None, top=None, wspace=wspace, hspace=hspace)
    rows = math.ceil(float(nyc_data.shape[1]) / cols)
    for i, column in enumerate(nyc_data.columns):
        ax = fig.add_subplot(rows, cols, i + 1)
        ax.set_title(column)
        if nyc_data.dtypes[column] == np.object:
            g = sns.countplot(y=column, data=nyc_data)
            substrings = [s.get_text()[:18] for s in g.get_yticklabels()]
            g.set(yticklabels=substrings)
            plt.xticks(rotation=25)
        else:
            g = sns.boxplot(nyc_data[column])
            plt.xticks(rotation=25)

cols_to_plot = ['passenger_count', 'pickup_longitude', 'pickup_latitude', 'dropoff_longitude', 'dropoff_latitude', 'store_and_fwd_flag','trip_duration']
plot_distribution(nyc_data[cols_to_plot], cols=3, width=20, height=20, hspace=0.45, wspace=0.5)
```



In [13]:
```
# Passenger Count Outliers
nyc_data['passenger_count'].max() # A Taxi usually has space for a maximum of 4 passengers
```

Out[13]: 9

In [14]:
```
nyc_data['passenger_count'].plot()
plt.show() # Greater than 4 would be outliers in this case. zero passengers?
```



In [15]:
```
print('Passengers: {} to {}'.format(nyc_data.passenger_count.min(), nyc_data.passenger_count.max()))
```

```
Passengers: 0 to 9
```

In [16]:
```
# removing the outliers
nyc_data = nyc_data[~duration]
nyc_data.trip_duration = nyc_data.trip_duration.astype(np.uint16)
```

In [17]:
```
print('Trip duration in seconds: {} to {}'.format(nyc_data.trip_duration.min(), nyc_data.trip_duration.max())) # range after cleaning
```

```
Trip duration in seconds: 60 to 7191
```

In [18]:
```
# Removing trips with passenger count = 0
print('Empty trips: {}'.format(nyc_data[nyc_data.passenger_count == 0].shape[0]))
nyc_data = nyc_data[nyc_data.passenger_count > 0]
```

```
Empty trips: 17
```

In [19]:
```
len(nyc_data) # Number of entries post cleaning
```

Out[19]: 1447779

1.2 Feature Generation

- Distance of trip
- Day of week
- Average speed of trip

In [20]:
```
# gdf = geopandas.GeoDataFrame(nyc_data, geometry = geopandas.points_from_xy(df.Longitude, df.Latitude))
nyc_data['pickup_points'] = gpd.points_from_xy(nyc_data.pickup_longitude, nyc_data.pickup_latitude)
nyc_data['dropoff_points'] = gpd.points_from_xy(nyc_data.dropoff_longitude, nyc_data.dropoff_latitude)
```

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

Out[21]:

	id	vendor_id	pickup_datetime	dropoff_datetime	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	store_and_fwd_flag	trip_duration	pickup_points	dropoff_points
0	id2875421	2	2016-03-14 17:24:55	2016-03-14 17:32:30	1	-73.982155	40.767937	-73.964630	40.765602	N	455	POINT (-73.98215 40.76794)	POINT (-73.96463 40.76560)
1	id2377394	1	2016-06-12 00:43:35	2016-06-12 00:54:38	1	-73.980415	40.738564	-73.999481	40.731152	N	663	POINT (-73.98042 40.73856)	POINT (-73.99948 40.73115)
2	id3858529	2	2016-01-19 11:35:24	2016-01-19 12:10:48	1	-73.979027	40.763939	-74.005333	40.710087	N	2124	POINT (-73.97903 40.76394)	POINT (-74.00533 40.71009)
3	id3504673	2	2016-04-06 19:32:31	2016-04-06 19:39:40	1	-74.010040	40.719971	-74.012268	40.706718	N	429	POINT (-74.01004 40.71997)	POINT (-74.01227 40.70672)
4	id2181028	2	2016-03-26 13:30:55	2016-03-26 13:38:10	1	-73.973053	40.793209	-73.972923	40.782520	N	435	POINT (-73.97305 40.79321)	POINT (-73.97292 40.78252)

In [22]:
```
# Distance of trip
crs = 'EPSG:4326' # {'init':'EPSG:4326'} deprecated, try '<init>:<EPSG:4326>'
geometry_1 = [Point(xy) for xy in zip(nyc_data['pickup_longitude'], nyc_data['pickup_latitude'])]
geometry_2 = [Point(xy) for xy in zip(nyc_data['dropoff_longitude'], nyc_data['dropoff_latitude'])]

geo_df_pickup = gpd.GeoDataFrame(nyc_data[['pickup_longitude', 'pickup_latitude']], geometry = geometry_1, crs = crs)
geo_df_dropoff = gpd.GeoDataFrame(nyc_data[['dropoff_longitude', 'dropoff_latitude']], geometry = geometry_2, crs = crs)
```

In [23]:
```
# points_df2 = points_df.shift() # shift the dataframe by 1 to align pnt1 with pnt2
# Chose 2263 EPSG for New York
geo_df_pickup = geo_df_pickup.to_crs('EPSG:2263')
geo_df_dropoff = geo_df_dropoff.to_crs('EPSG:2263')
nyc_data['distance'] = geo_df_pickup.distance(geo_df_dropoff) # Distance in meters.
nyc_data.head()
```

Out[23]:

	id	vendor_id	pickup_datetime	dropoff_datetime	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	store_and_fwd_flag	trip_duration	pickup_points	dropoff_points	distance
0	id2875421	2	2016-03-14 17:24:55	2016-03-14 17:32:30	1	-73.982155	40.767937	-73.964630	40.765602	N	455	POINT (-73.98215 40.76794)	POINT (-73.96463 40.76560)	4928.355491
1	id2377394	1	2016-06-12 00:43:35	2016-06-12 00:54:38	1	-73.980415	40.738564	-73.999481	40.731152	N	663	POINT (-73.98042 40.73856)	POINT (-73.99948 40.73115)	5933.893605
2	id3858529	2	2016-01-19 11:35:24	2016-01-19 12:10:48	1	-73.979027	40.763939	-74.005333	40.710087	N	2124	POINT (-73.97903 40.76394)	POINT (-74.00533 40.71009)	20930.624947
3	id3504673	2	2016-04-06 19:32:31	2016-04-06 19:39:40	1	-74.010040	40.719971	-74.012268	40.706718	N	429	POINT (-74.01004 40.71997)	POINT (-74.01227 40.70672)	4867.539885
4	id2181028	2	2016-03-26 13:30:55	2016-03-26 13:38:10	1	-73.973053	40.793209	-73.972923	40.782520	N	435	POINT (-73.97305 40.79321)	POINT (-73.97292 40.78252)	3894.455150

In [24]:
```
# Day of the week
date_ny = pd.to_datetime(nyc_data['pickup_datetime'])
nyc_data['dayOfWeek'] = date_ny.dt.day_name()
```

In [25]:
```
nyc_data.head()
```

Out[25]:

	id	vendor_id	pickup_datetime	dropoff_datetime	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	store_and_fwd_flag	trip_duration	pickup_points	dropoff_points	distance	dayOfWeek
0	id2875421	2	2016-03-14 17:24:55	2016-03-14 17:32:30	1	-73.982155	40.767937	-73.964630	40.765602	N	455	POINT (-73.98215 40.76794)	POINT (-73.96463 40.76560)	4928.355491	Monday
1	id2377394	1	2016-06-12 00:43:35	2016-06-12 00:54:38	1	-73.980415	40.738564	-73.999481	40.731152	N	663	POINT (-73.98042 40.73856)	POINT (-73.99948 40.73115)	5933.893605	Sunday
2	id3858529	2	2016-01-19 11:35:24	2016-01-19 12:10:48	1	-73.979027	40.763939	-74.005333	40.710087	N	2124	POINT (-73.97903 40.76394)	POINT (-74.00533 40.71009)	20930.624947	Tuesday
3	id3504673	2	2016-04-06 19:32:31	2016-04-06 19:39:40	1	-74.010040	40.719971	-74.012268	40.706718	N	429	POINT (-74.01004 40.71997)	POINT (-74.01227 40.70672)	4867.539885	Wednesday
4	id2181028	2	2016-03-26 13:30:55	2016-03-26 13:38:10	1	-73.973053	40.793209	-73.972923	40.782520	N	435	POINT (-73.97305 40.79321)	POINT (-73.97292 40.78252)	3894.455150	Saturday

In [26]:
```
# Average Speed of trip >> Speed = Distance (m) / Time (s) m/s
nyc_data['Avg_speed'] = nyc_data['distance'] / nyc_data['trip_duration']
```

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

Out[27]:

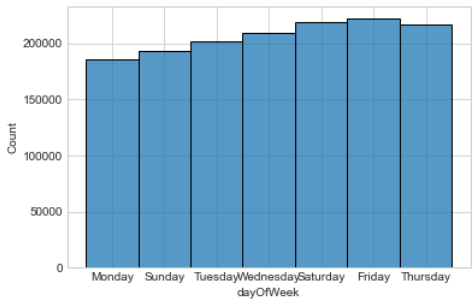
	id	vendor_id	pickup_datetime	dropoff_datetime	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	store_and_fwd_flag	trip_duration	pickup_points	dropoff_points	distance	dayOfWeek	Avg_speed
0	id2875421	2	2016-03-14 17:24:55	2016-03-14 17:32:30	1	-73.982155	40.767937	-73.964630	40.765602	N	455	POINT (-73.98215 40.76794)	POINT (-73.96463 40.76560)	4928.355491	Monday	10.831551
1	id2377394	1	2016-06-12 00:43:35	2016-06-12 00:54:38	1	-73.980415	40.738564	-73.999481	40.731152	N	663	POINT (-73.98042 40.73856)	POINT (-73.99948 40.73115)	5933.893605	Sunday	8.950066
2	id3858529	2	2016-01-19 11:35:24	2016-01-19 12:10:48	1	-73.979027	40.763939	-74.005333	40.710087	N	2124	POINT (-73.97903 40.76394)	POINT (-74.00533 40.71009)	20930.624947	Tuesday	9.854343
3	id3504673	2	2016-04-06 19:32:31	2016-04-06 19:39:40	1	-74.010040	40.719971	-74.012268	40.706718	N	429	POINT (-74.01004 40.71997)	POINT (-74.01227 40.70672)	4867.539885	Wednesday	11.346247
4	id2181028	2	2016-03-26 13:30:55	2016-03-26 13:38:10	1	-73.973053	40.793209	-73.972923	40.782520	N	435	POINT (-73.97305 40.79321)	POINT (-73.97292 40.78252)	3894.455150	Saturday	8.952770

1.3 Time-based questions

1.3.1

Which day of the week is the most popular

In [28]:
```
sns.histplot(data=nyc_data, x= 'dayOfWeek')
plt.show() # Friday is the most popular by a liltle margin.
```



1.3.2

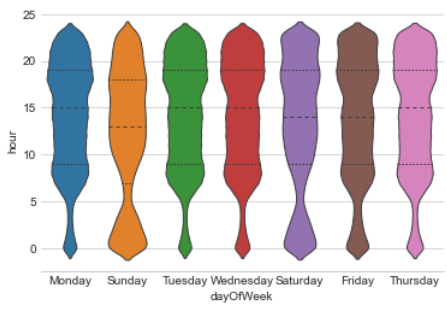
What hour of the day is the most popular on each day? Plot a distributions of the hours and make observations and give possible suggestions for why the data looks like it does.

In [29]:
```
nyc_data['hour'] = date_ny.dt.hour
```

In [30]:
```
for i, x in enumerate(nyc_data['dayOfWeek'].unique()):
    print(x, ": Popular Hour: ", int(nyc_data[nyc_data['dayOfWeek'] == x]['hour'].mode()))
```

```
Monday : Popular Hour:  18
Sunday : Popular Hour:  0
Tuesday : Popular Hour:  18
Wednesday : Popular Hour:  19
Saturday : Popular Hour:  23
Friday : Popular Hour:  19
Thursday : Popular Hour:  21
```

In [31]:
```
sns.violinplot(data=nyc_data, x="dayOfWeek", y="hour",
               split=True, inner="quart", linewidth=1,)
sns.despine(left=True)
plt.show()
```



The plot above shows the distribution of each hour during each day of the week. The Output before the plot shows the most popular hour for each day of the week.

- Monday (18:00), Tuesday (18:00) might be due to the end of business day, as most people might be going back home from work.
- Wednesday (19:00), Friday (19:00) alos probably due to end of business day rush.
- Thursday (21:00) Assuming rush hour is from 6pm to 9pm, The most popular time for thursday falls within this period.
- Saturday (23:00), Sunday (0:00) People were probably out on a saturday night, and the hours close to midnight might be a common time to take a cab back home.

1.3.3

Investigate the differences between weekdays and weekends. What would account for this?

People tend to take taxis during the earlier hours of the morning during the weekends when compared to the mornings of the weekdays.

The most popular hours been 18:00 - 21:00, shows a possible after work hours or a general close of business day, during weekdays. The use of taxis during the earlier ours during the weekends could suggest the use of cabs after party hours.

1.3.4

How do these patterns change on the major holidays (do they change?): St. Patrick's Day (17 March(03)), Easter(17 April(04)), Memorial Day (30 May(05)), for example?

In [32]:
```
nyc_data['month'] =  date_ny.dt.month
nyc_data['day'] =  date_ny.dt.day
```

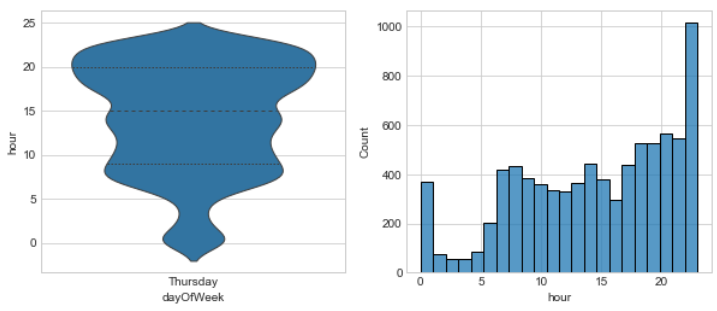
In [33]:
```
nyc_data.head()
```

Out[33]:

	id	vendor_id	pickup_datetime	dropoff_datetime	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	store_and_fwd_flag	trip_duration	pickup_points	dropoff_points	distance	dayOfWeek	Avg_speed	hour	month	day
0	id2875421	2	2016-03-14 17:24:55	2016-03-14 17:32:30	1	-73.982155	40.767937	-73.964630	40.765602	N	455	POINT (-73.98215 40.76794)	POINT (-73.96463 40.76560)	4928.355491	Monday	10.831551	17	3	14
1	id2377394	1	2016-06-12 00:43:35	2016-06-12 00:54:38	1	-73.980415	40.738564	-73.999481	40.731152	N	663	POINT (-73.98042 40.73856)	POINT (-73.99948 40.73115)	5933.893605	Sunday	8.950066	0	6	12
2	id3858529	2	2016-01-19 11:35:24	2016-01-19 12:10:48	1	-73.979027	40.763939	-74.005333	40.710087	N	2124	POINT (-73.97903 40.76394)	POINT (-74.00533 40.71009)	20930.624947	Tuesday	9.854343	11	1	19
3	id3504673	2	2016-04-06 19:32:31	2016-04-06 19:39:40	1	-74.010040	40.719971	-74.012268	40.706718	N	429	POINT (-74.01004 40.71997)	POINT (-74.01227 40.70672)	4867.539885	Wednesday	11.346247	19	4	6
4	id2181028	2	2016-03-26 13:30:55	2016-03-26 13:38:10	1	-73.973053	40.793209	-73.972923	40.782520	N	435	POINT (-73.97305 40.79321)	POINT (-73.97292 40.78252)	3894.455150	Saturday	8.952770	13	3	26

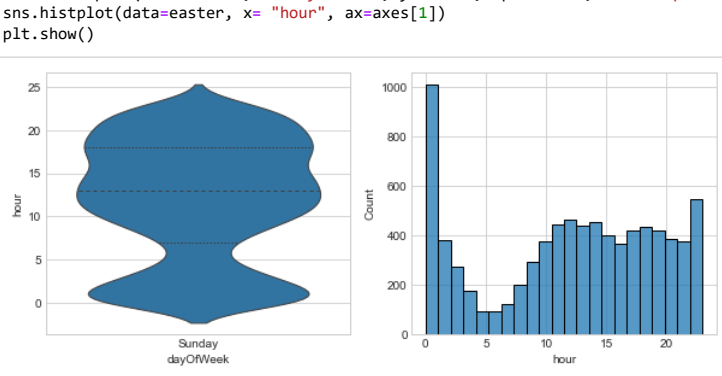
In [34]:
```
st_pat = nyc_data[(nyc_data['month'] == 3) & (nyc_data['day'] == 17)]
easter = nyc_data[(nyc_data['month'] == 4) & (nyc_data['day'] == 17)]
memorial = nyc_data[(nyc_data['month'] == 5) & (nyc_data['day'] == 30)]
```

In [35]:
```
# St Patrick's Day
f, axes = plt.subplots(1,2, figsize=(10, 4))
sns.violinplot(data=st_pat, x="dayOfWeek", y="hour", split=True, inner="quart", linewidth=1, ax=axes[0])
sns.histplot(data=st_pat, x= "hour", ax=axes[1])
plt.show()
```



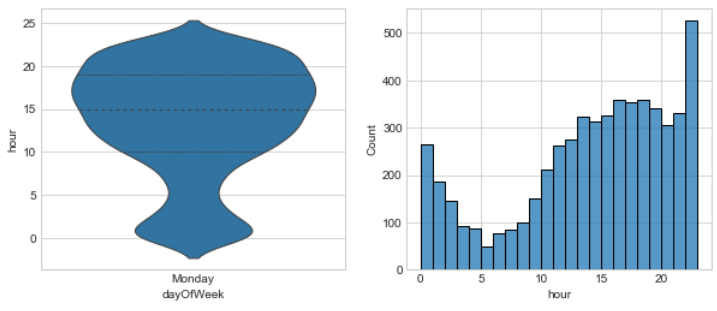
It is clear that taxis operate more during the later hours on st Patricks's day, Operations increase from 7:00 in the morning, The most opular time being around 22:00

In [36]:
```
# Easter
f, axes = plt.subplots(1,2, figsize=(10, 4))
sns.violinplot(data=easter, x="dayOfWeek", y="hour", split=True, inner="quart", linewidth=1, ax=axes[0])
sns.histplot(data=easter, x= "hour", ax=axes[1])
plt.show()
```



Taxis generally operate throughout all the hours of easter, mostly after 10:00 in the morning. The most popular being around Midnight

In [37]:
```
# Memorial Day
f, axes = plt.subplots(1,2, figsize=(10, 4))
sns.violinplot(data=memorial, x="dayOfWeek", y="hour", split=True, inner="quart", linewidth=1, ax=axes[0])
sns.histplot(data=memorial, x= "hour", ax=axes[1])
plt.show()
```

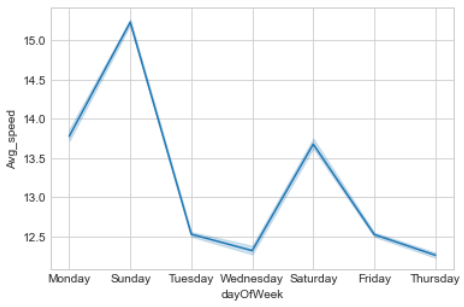


Taxis generally operate more from morning till the later hours of memorial day, a sharp increase in the morning from 10:00 in the morning, continues till the later hours of the day. The most popular hour being around 22:00

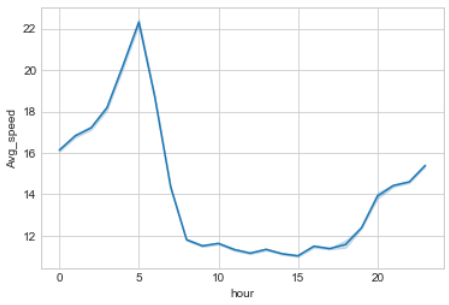
1.3.5

How does the average speed of trips change throughout the day? What time of day are trips fastest? Show plots to motivate your answer.

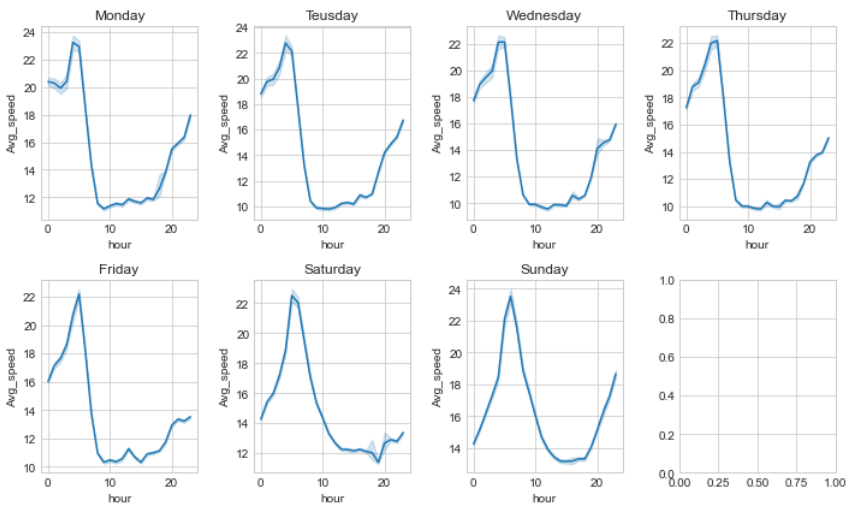
In [38]:
```
sns.lineplot(data=nyc_data, x="dayOfWeek", y="Avg_speed")
plt.show()
```



In [39]:
```
sns.lineplot(data=nyc_data, x="hour", y="Avg_speed")
plt.show()
```



In [40]:
```
# Checking each day of the week individually
f, axes = plt.subplots(2,4, figsize=(10, 6))

sns.lineplot(data=nyc_data[nyc_data['dayOfWeek'] == 'Monday'], x="hour", y="Avg_speed" , ax=axes[0,0]).set_title('Monday')
sns.lineplot(data=nyc_data[nyc_data['dayOfWeek'] == 'Tuesday'], x="hour", y="Avg_speed" , ax=axes[0,1]).set_title('Teusday')
sns.lineplot(data=nyc_data[nyc_data['dayOfWeek'] == 'Wednesday'], x="hour", y="Avg_speed" , ax=axes[0,2]).set_title('Wednesday')
sns.lineplot(data=nyc_data[nyc_data['dayOfWeek'] == 'Thursday'], x="hour", y="Avg_speed" , ax=axes[0,3]).set_title('Thursday')
sns.lineplot(data=nyc_data[nyc_data['dayOfWeek'] == 'Friday'], x="hour", y="Avg_speed" , ax=axes[1,0]).set_title('Friday')
sns.lineplot(data=nyc_data[nyc_data['dayOfWeek'] == 'Saturday'], x="hour", y="Avg_speed" , ax=axes[1,1]).set_title('Saturday')
sns.lineplot(data=nyc_data[nyc_data['dayOfWeek'] == 'Sunday'], x="hour", y="Avg_speed" , ax=axes[1,2]).set_title('Sunday')
plt.tight_layout()
plt.show()
```



The plots above show that the trips are usually faster during 5am in the morning, Followed by a sharp decline all through the day till around 15:00 pm in the afternoon where the average speed picks up once again.

1.4 Location Clusters

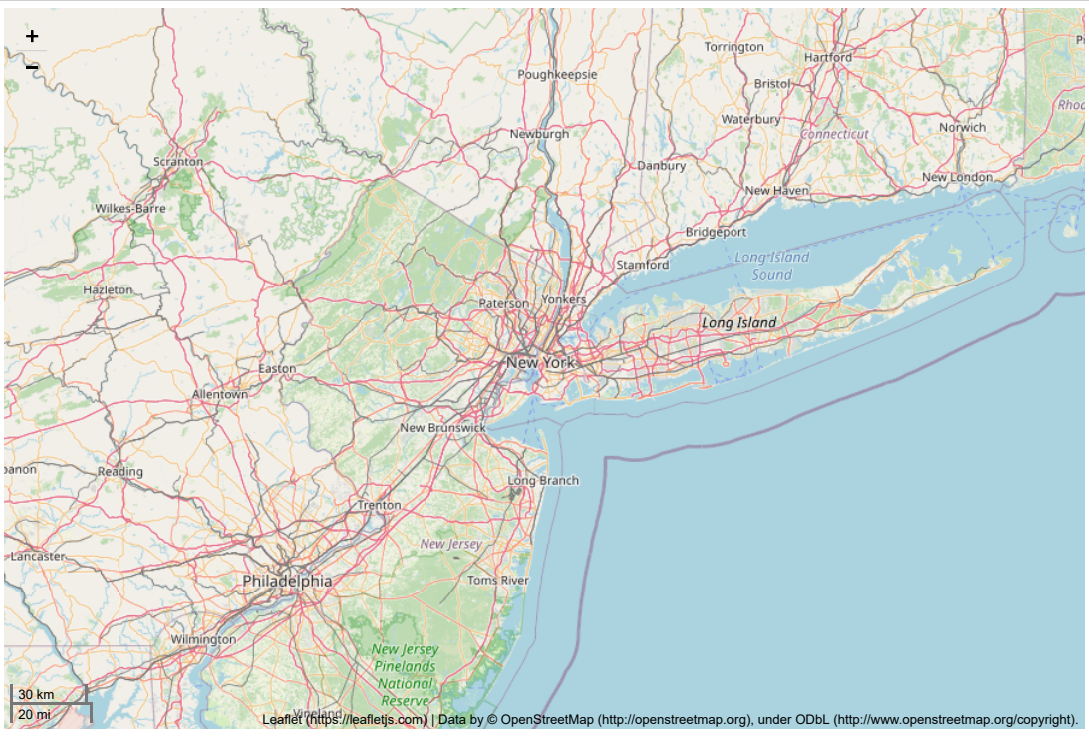
In [41]:
```
# weekdays and weekends
nyc_data_weekdays = nyc_data[(nyc_data['dayOfWeek'] != 'Saturday') & (nyc_data['dayOfWeek'] != 'Sunday')]
nyc_data_weekends = nyc_data[(nyc_data['dayOfWeek'] == 'Saturday') | (nyc_data['dayOfWeek'] == 'Sunday')]
```

In [42]:
```
# morning and evening (7-10 am)(16-19 pm)
nyc_data_mornings = nyc_data[(nyc_data['hour'] == 7) | (nyc_data['hour'] == 8) | (nyc_data['hour'] == 9) | (nyc_data['hour'] == 10)]
nyc_data_evenings = nyc_data[(nyc_data['hour'] == 16) | (nyc_data['hour'] == 17) | (nyc_data['hour'] == 18) | (nyc_data['hour'] == 19)]
```

In [43]:
```
def generateBaseMap(default_location=[40.693943, -73.985880], default_zoom_start=8):
    fig = Figure(width=900,height=600)
    base_map = folium.Map(location=default_location, control_scale=True, zoom_start=default_zoom_start)
    fig.add_child(base_map)
    return base_map
```

In [44]:
```
base_map = generateBaseMap()
base_map
```

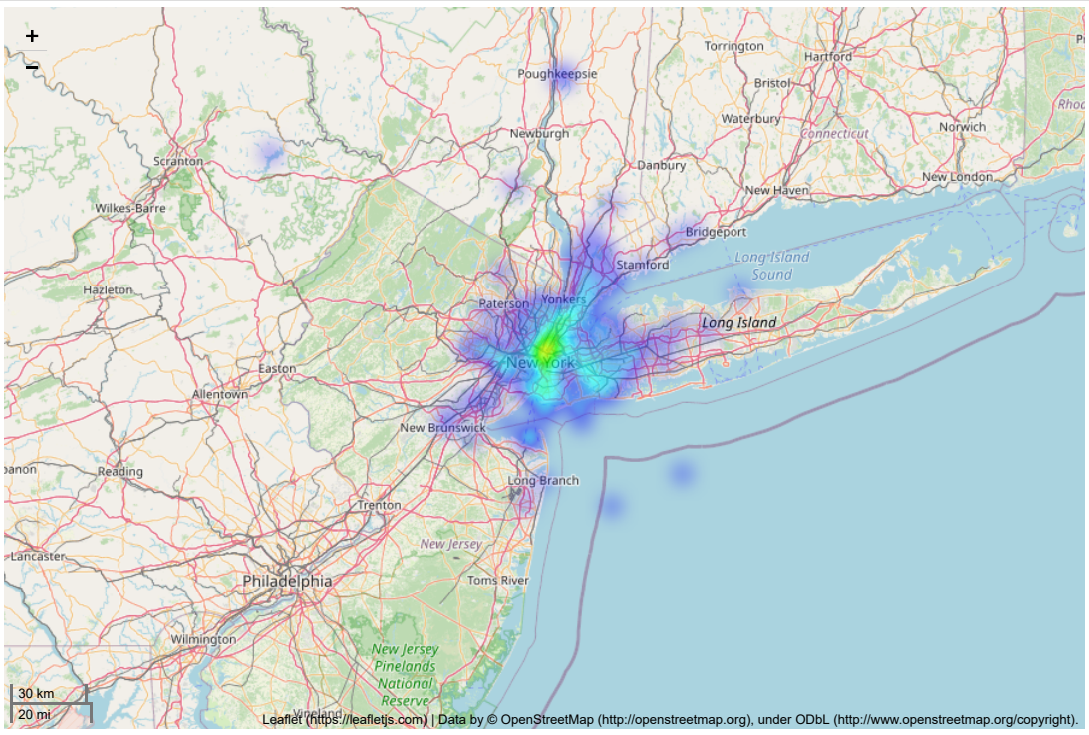
Out[44]:



In [45]:
```
nyc_data_weekdays['count'] = 1
nyc_data_weekends['count'] = 1

nyc_data_mornings['count'] = 1
nyc_data_evenings['count'] = 1
```

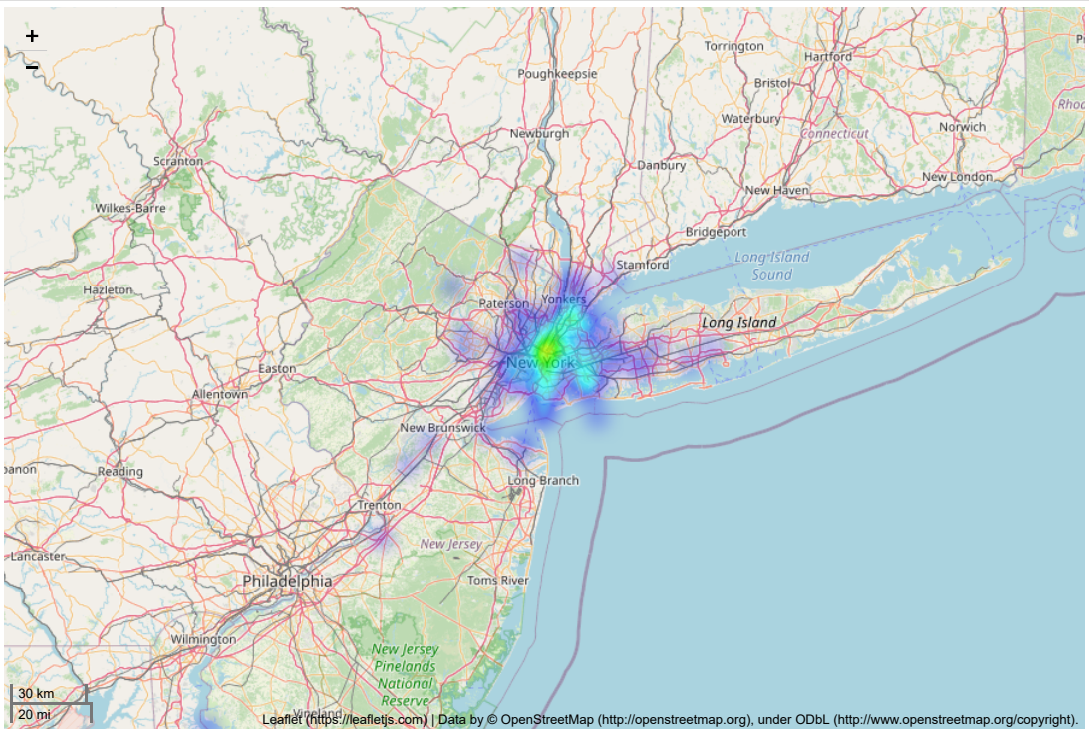
In [46]:
```
# weekdays
base_map = generateBaseMap()
HeatMap(data=nyc_data_weekdays[['pickup_latitude', 'pickup_longitude', 'count']].groupby(['pickup_latitude', 'pickup_longitude']).sum().reset_index().values.tolist(), radius=8, max_zoom=13).add_to(base_map)
base_map
```

Out[46]:



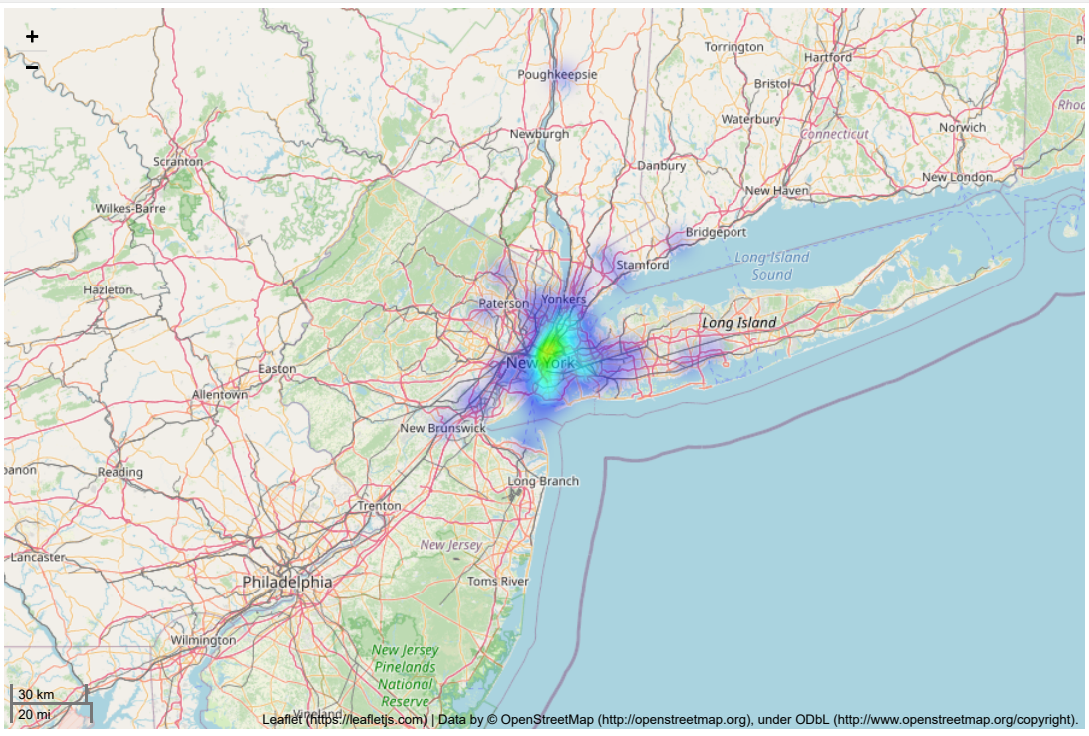
In [47]:
```
# Weekends
base_map = generateBaseMap()
HeatMap(data=nyc_data_weekends[['pickup_latitude', 'pickup_longitude', 'count']].groupby(['pickup_latitude', 'pickup_longitude']).sum().reset_index().values.tolist(), radius=8, max_zoom=13).add_to(base_map)
base_map
```

Out[47]:



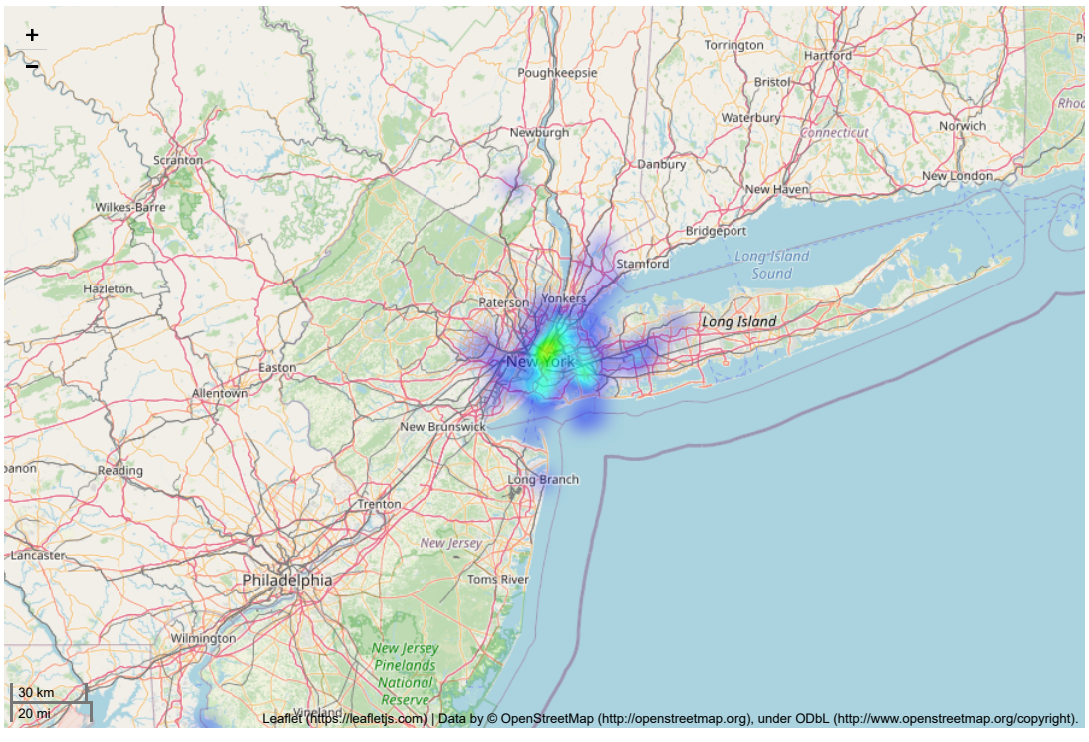
In [48]:
```
# Mornings
base_map = generateBaseMap()
HeatMap(data=nyc_data_mornings[['pickup_latitude', 'pickup_longitude', 'count']].groupby(['pickup_latitude', 'pickup_longitude']).sum().reset_index().values.tolist(), radius=8, max_zoom=13).add_to(base_map)
base_map
```

Out[48]:



In [49]:
```
# Evenings
base_map = generateBaseMap()
HeatMap(data=nyc_data_evenings[['pickup_latitude', 'pickup_longitude', 'count']].groupby(['pickup_latitude', 'pickup_longitude']).sum().reset_index().values.tolist(), radius=8, max_zoom=13).add_to(base_map)
base_map
```

Out[49]:



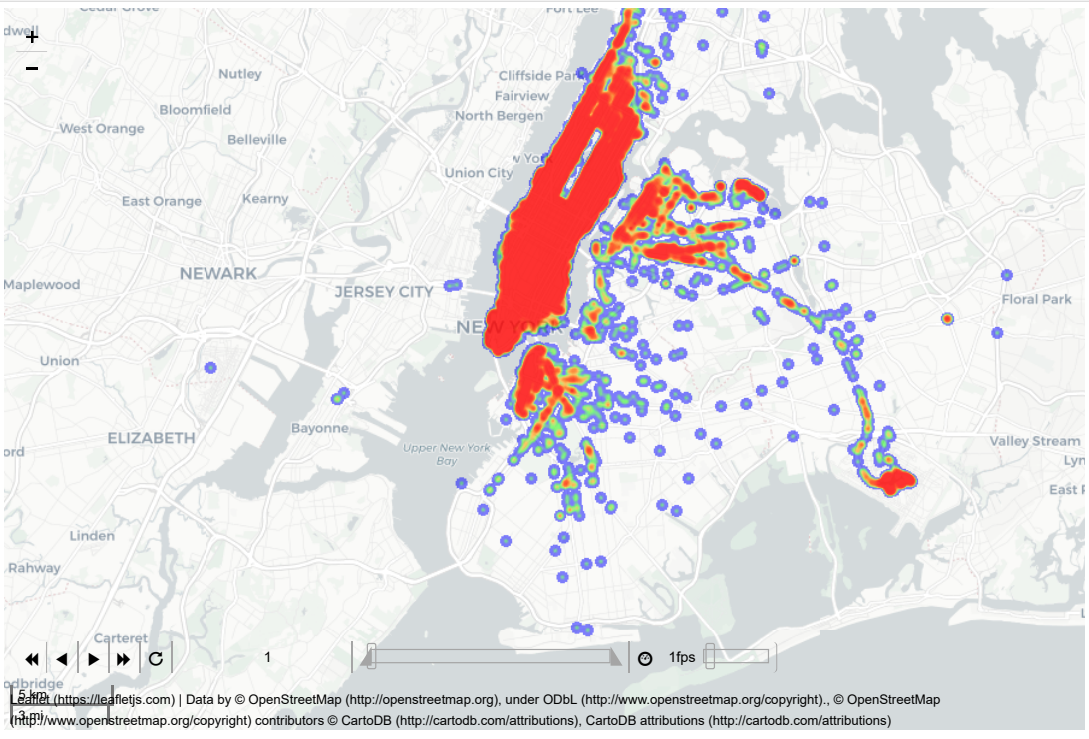
- Weekdays vs Weekends heatmaps: A higher concentration during weekdays compared to weekends.
- Mornings vs Evenings heatmaps: A higher concentration during evenings when compared to the evening.

Overall the pickup location are highly concentrated around New york. with the other pickups around the surrounding area being radomly distributed.

In [50]:
```
# Mornings, Plotting each hour of the Morning (7,8,9,10) am
df_hour_list = []
for hour in nyc_data_mornings.hour.sort_values().unique():
    df_hour_list.append(nyc_data_mornings.loc[nyc_data_mornings.hour == hour, ['pickup_latitude', 'pickup_longitude', 'count']].groupby(['pickup_latitude', 'pickup_longitude']).sum().reset_index().values.tolist())
```

In [51]:
```
base_map = generateBaseMap(default_zoom_start=11)
HeatMapWithTime(df_hour_list, radius=5, gradient={0.2: 'blue', 0.4: 'lime', 0.6: 'orange', 1: 'red'}, min_opacity=0.5, max_opacity=0.8, use_local_extrema=True).add_to(base_map)
folium.TileLayer('cartodbpositron').add_to(base_map)
base_map
```

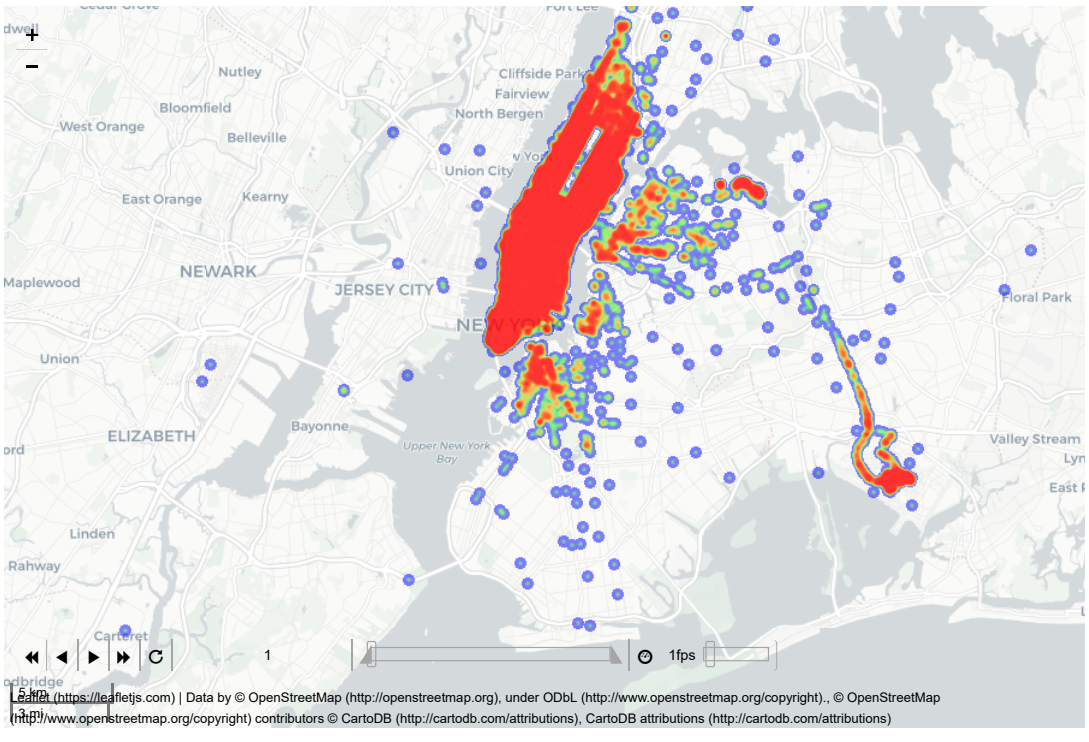
Out[51]:



In [52]:
```
# Evenings Plotting each hour of the Evening (16, 17, 18, 19) pm
df_hour_list = []
for hour in nyc_data_evenings.hour.sort_values().unique():
    df_hour_list.append(nyc_data_evenings.loc[nyc_data_evenings.hour == hour, ['pickup_latitude', 'pickup_longitude', 'count']].groupby(['pickup_latitude', 'pickup_longitude']).sum().reset_index().values.tolist())
```

In [53]:
```
base_map = generateBaseMap(default_zoom_start=11)
HeatMapWithTime(df_hour_list, radius=5, gradient={0.2: 'blue', 0.4: 'lime', 0.6: 'orange', 1: 'red'}, min_opacity=0.5, max_opacity=0.8, use_local_extrema=True).add_to(base_map)
folium.TileLayer('cartodbpositron').add_to(base_map)
base_map
```

Out[53]:



- From the most popular times for Friday night/Saturday morning (around midnight) and Thursday afternoon,

find hotspot locations (you will need to perform clustering). Using an algorithm like DBSCAN (available in sklearn) determines this for you, and works well on spatial data. **DBSCAN** has two configurable parameters: e - the maximum distance between any two points, and the minimum number of samples to determine a cluster.

- Your hotspot location might be defined as at least 15 pickups in that location in an hour, and
- locations might be required to be within 50 or 100 metres from each other (do motivate your choice of parameters).

Using **DBSCAN**, identify clusters and plot these on a map. How many clusters did you find?

In [54]:
```
# Selecting friday night/Saturday midnight (0)
db_selection_1 = nyc_data[((nyc_data['dayOfWeek'] == 'Saturday') | (nyc_data['dayOfWeek'] == 'Friday')) & (nyc_data['hour'] == 0)]
```

In [55]:
```
# Thursday Afternoon (12-16pm)
db_selection_2 = nyc_data[(nyc_data['dayOfWeek'] == 'Thursday') & ((nyc_data['hour'] == 12) | (nyc_data['hour'] == 13) | (nyc_data['hour'] == 14) | (nyc_data['hour'] == 15) | (nyc_data['hour'] == 16))]
```

In [56]:
```
loc_ini = db_selection_1[['pickup_latitude', 'pickup_longitude']].to_numpy()
loc_end = db_selection_1[['dropoff_latitude', 'dropoff_longitude']].to_numpy()
locations = np.vstack((loc_ini, loc_end))
```

- function to fit an HDBSCAN clusterer object. The fit_utm_clusterer function fits an HDBSCAN model using utm - Universal Transverse Mercator coordinate system

In [57]:
```
def fit_utm_clusterer(locations, min_cluster_size=20, min_samples=15, cluster_selection_epsilon = 49): #min_cluster_size = 15?, #cluster_selection_epsilon (distance)= 50 or 100?
    xyzz = [utm.from_latlon(ll[0], ll[1]) for ll in locations]
    pts = [[p[0], p[1]] for p in xyzz]

    clusterer = hdbscan.HDBSCAN(min_cluster_size=min_cluster_size,
                                min_samples=min_samples,
                                cluster_selection_epsilon=cluster_selection_epsilon,
                                metric='euclidean')
    clusterer.fit(pts)
    return clusterer
```

In [58]:
```
clusterer = fit_utm_clusterer(locations)
```

In [59]:
```
unique_clusters = np.unique(clusterer.labels_)[1:]
print("The initial number of clusters is: {0}".format(unique_clusters.shape[0]))
```

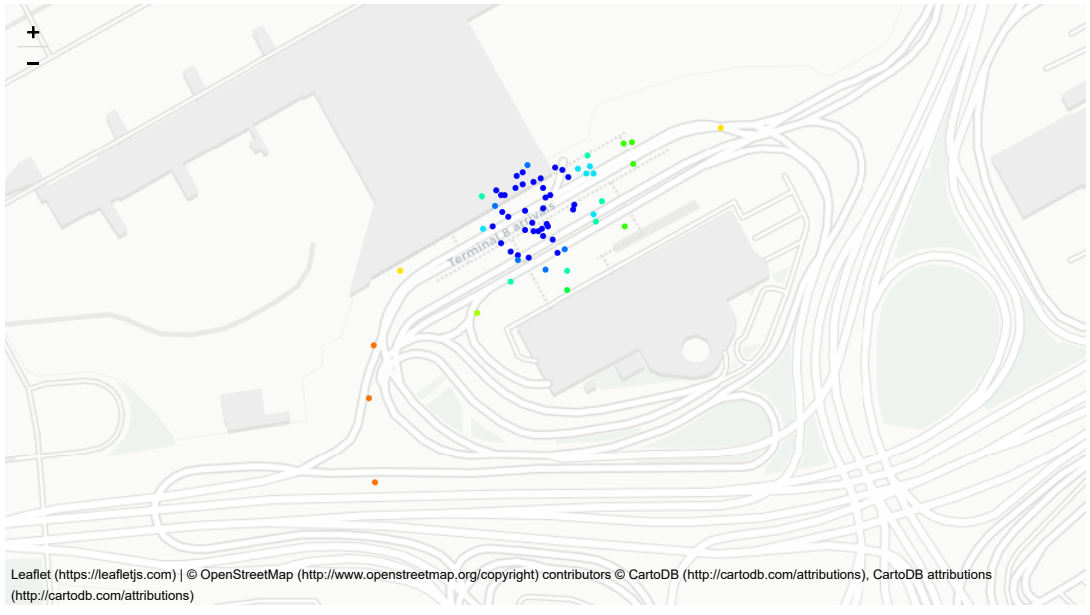
```
The initial number of clusters is: 213
```

In [60]:
```
def show_cluster_map(cluster_id):
    blue = Color("blue")
    red = Color("red")
    color_range = list(blue.range_to(red, 10))

    # base_map = generateBaseMap() # try base_map instead of map_
    map_ = folium.Map(width=900,height=500, prefer_canvas=True, tiles='CartoDB positron')

    clusters = clusterer.labels_
    outlier_scores = clusterer.outlier_scores_

    points = locations[clusters == cluster_id]
    scores = outlier_scores[clusters == cluster_id]


    for i in range(points.shape[0]):
        point = points[i]
        color = color_range[int(scores[i] * 10)]
        CircleMarker([point[0],point[1]], radius=1, color = color.hex, tooltip = "{:.2f}".format(scores[i])).add_to(map_)

    min_lat, max_lat = points[:, 0].min(), points[:, 0].max()
    min_lon, max_lon = points[:, 1].min(), points[:, 1].max()
    map_.fit_bounds([[min_lat, min_lon], [max_lat, max_lon]])

    return map_
```

- Friday night/Saturday midnight (0) Hotspots

In [61]:
```
ii = interact(show_cluster_map, cluster_id = widgets.IntText(min=0, max= clusterer.labels_.max(), step=1, value=0))
```

cluster_id 0



- Since there should be 15 pickups in the location in order to classify as a hotspot, the parameter minimum sample size (min_samples) is set to 15.
- A minimum cluster size of 20 is chosen as well. slightly bigger than the min_samples to capture a little more detail within clusters.
- cluster_selection_epsilon(distances between clusters) = 49 So that any point above 49 meters is not merged into the current cluster.
- Overall, with the parameters chosen, a total of **213 hotspot locations** were found around **midnight on friday and Saturday.**

In [62]:
```
loc_ini = db_selection_2[['pickup_latitude', 'pickup_longitude']].to_numpy()
loc_end = db_selection_2[['dropoff_latitude', 'dropoff_longitude']].to_numpy()
locations = np.vstack((loc_ini, loc_end))
```

In [63]:
```
def fit_utm_clusterer(locations,min_cluster_size=20, min_samples=15, cluster_selection_epsilon=49): #min_cluster_size = 15?, #cluster_selection_epsilon (distance)= 50 or 100?
    xyzz = [utm.from_latlon(ll[0], ll[1]) for ll in locations]
    pts = [[p[0], p[1]] for p in xyzz]

    clusterer = hdbscan.HDBSCAN(min_cluster_size=min_cluster_size,
                                min_samples=min_samples,
                                cluster_selection_epsilon=cluster_selection_epsilon,
                                metric='euclidean')
    clusterer.fit(pts)
    return clusterer
```

In [64]:
```
clusterer = fit_utm_clusterer(locations)
```

In [65]:
```
unique_clusters = np.unique(clusterer.labels_)[1:]
print("The initial number of clusters is: {0}".format(unique_clusters.shape[0]))
```

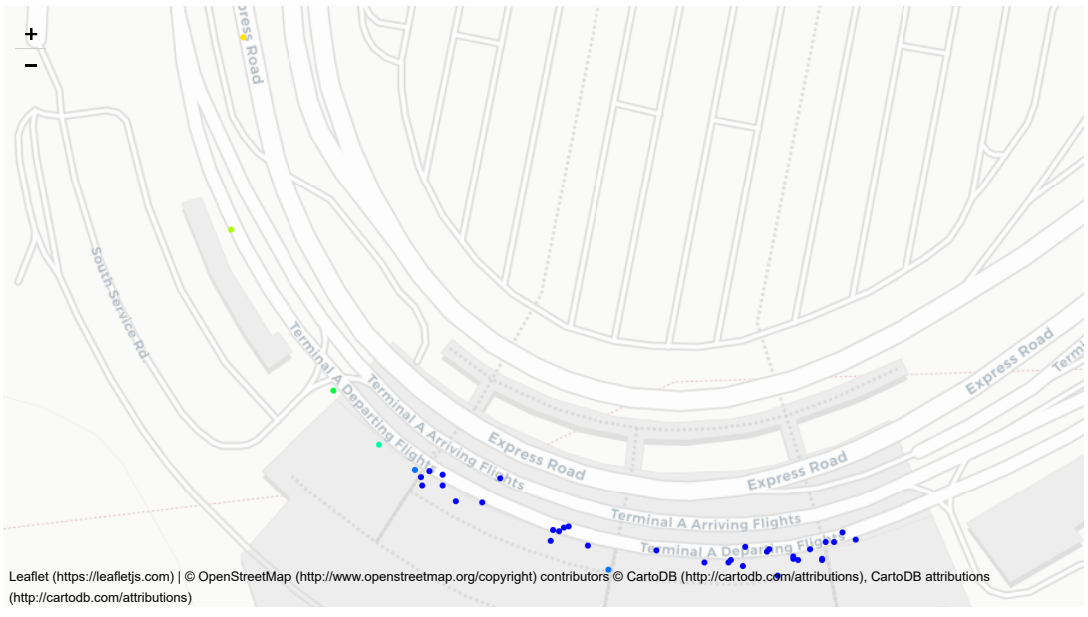
```
The initial number of clusters is: 115
```

In [66]:
```
def show_cluster_map(cluster_id):
    blue = Color("blue")
    red = Color("red")
    color_range = list(blue.range_to(red, 10))

    # base_map = generateBaseMap() # try base_map instead of map_
    map_ = folium.Map(width=900,height=500,prefer_canvas=True, tiles='CartoDB positron')

    clusters = clusterer.labels_
    outlier_scores = clusterer.outlier_scores_

    points = locations[clusters == cluster_id]
    scores = outlier_scores[clusters == cluster_id]


    for i in range(points.shape[0]):
        point = points[i]
        color = color_range[int(scores[i] * 10)]
        CircleMarker([point[0],point[1]], radius=1, color = color.hex, tooltip = "{:.2f}".format(scores[i])).add_to(map_)

    min_lat, max_lat = points[:, 0].min(), points[:, 0].max()
    min_lon, max_lon = points[:, 1].min(), points[:, 1].max()
    map_.fit_bounds([[min_lat, min_lon], [max_lat, max_lon]])

    return map_
```

- Thursday Afternoon (12-16pm) Hotspots

In [67]:
```
ii = interact(show_cluster_map, cluster_id = widgets.IntText(min=0, max= clusterer.labels_.max(), step=1, value=0))
```

cluster_id 0



In this case over **115 hotspot locations** were found from a period of **12-16 pm on a Thursday afternoon.** Using the parameters.

- minimum cluster size (min_samples) is set to 15
- A minimum cluster size of 20
- cluster_selection_epsilon(distances between clusters) = 49

1.5 Airports

JFK Airport (jfk) Latitude/Longitude: 40.6441666667, -73.7822222222 Queens, NY 11430, USA

Empire State Building (esb) Latitude/Longitude: 40.748817, -73.985428 (-73.985664) (Pickup) 20 W 34th St, New York, NY 10001, USA

Newark Airport (nwa) Latitude/Longitude: 40.6897222222, -74.17446239999998 (-74.175) 3 Brewster Rd, Newark, NJ 07114, USA

- esb -> jfk approx 51 mins (3060s)
- esb -> nwa approx 46 min (2760)
- using reverse goecoding to extract the locations

In [68]:
```
# Lat/Lon data is Lon/Lat
jfk = [40.644167, -73.782222]
esb = [40.748817, -73.985428]
nwa = [40.689722, -74.174462]
```

In [69]:
```
samp = nyc_data[(nyc_data.trip_duration >= 2760) & (nyc_data.trip_duration <= 3070)]
samp
```

Out[69]:

	id	vendor_id	pickup_datetime	dropoff_datetime	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	store_and_fwd_flag	trip_duration	pickup_points	dropoff_points	distance	dayOfWeek	Avg_speed	hour	month	day
98	id2102594	1	2016-03-30 16:14:29	2016-03-30 17:01:33	1	-73.789841	40.643559	-73.970665	40.687279	N	2824	POINT (-73.78984 40.64356)	POINT (-73.97066 40.68728)	52633.213474	Wednesday	18.637823	16	3	30
198	id3942917	1	2016-04-29 14:18:39	2016-04-29 15:05:36	4	-73.862778	40.769314	-73.986130	40.748581	N	2817	POINT (-73.86278 40.76931)	POINT (-73.98613 40.74858)	34997.797862	Friday	12.423783	14	4	29
488	id1506453	1	2016-06-16 10:23:25	2016-06-16 11:10:36	1	-73.990822	40.755753	-73.997528	40.722435	N	2831	POINT (-73.99082 40.75575)	POINT (-73.99753 40.72243)	12280.081633	Thursday	4.337719	10	6	16
839	id1437834	2	2016-05-03 10:28:30	2016-05-03 11:17:57	2	-73.870972	40.773819	-74.000717	40.757561	N	2967	POINT (-73.87097 40.77382)	POINT (-74.00072 40.75756)	36425.336771	Tuesday	12.276824	10	5	3
1008	id3039752	1	2016-04-14 15:08:54	2016-04-14 15:56:02	1	-73.987381	40.760708	-73.871986	40.774246	N	2828	POINT (-73.98738 40.76071)	POINT (-73.87199 40.77425)	32342.650919	Thursday	11.436581	15	4	14
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1458263	id3934738	2	2016-06-09 08:37:03	2016-06-09 09:27:50	5	-73.870918	40.773754	-73.998497	40.737244	N	3047	POINT (-73.87092 40.77375)	POINT (-73.99850 40.73724)	37765.847813	Thursday	12.394436	8	6	9
1458310	id1325943	1	2016-05-25 15:08:13	2016-05-25 15:58:24	1	-73.993568	40.724503	-73.955742	40.779602	N	3011	POINT (-73.99357 40.72450)	POINT (-73.95574 40.77960)	22645.639566	Wednesday	7.520970	15	5	25
1458333	id1758713	2	2016-04-17 14:56:46	2016-04-17 15:46:59	1	-73.782722	40.644966	-73.974808	40.750660	N	3013	POINT (-73.78272 40.64497)	POINT (-73.97481 40.75066)	65725.159856	Sunday	21.813860	14	4	17
1458363	id3404858	2	2016-05-18 12:39:49	2016-05-18 13:29:33	2	-73.975510	40.733070	-73.990242	40.762440	N	2984	POINT (-73.97551 40.73307)	POINT (-73.99024 40.76244)	11452.405204	Wednesday	3.837937	12	5	18
1458422	id0875221	1	2016-06-01 09:01:38	2016-06-01 09:48:04	1	-73.872955	40.774158	-73.985207	40.751099	N	2786	POINT (-73.87296 40.77416)	POINT (-73.98521 40.75110)	32210.847073	Wednesday	11.561682	9	6	1

9811 rows × 19 columns

In [70]:
```
# reverse geocode the pickup and dropoff locations to get the address
def reverse_geo_pick(x):
    p = geocoder.osm([x.pickup_latitude, x.pickup_longitude], method ='reverse').json
    if p:
        return p.get('address')
    else:
        return 'NA'

def reverse_geo_drop(x):
    d = geocoder.osm([x.dropoff_latitude, x.dropoff_longitude], method ='reverse').json
    if d:
        return d.get('address')
    else:
        return 'NA'
```

In [71]:
```
#samp['pickup_address'] = samp[['pickup_latitude', 'pickup_longitude']].apply(reverse_geo_pick, axis = 1)
#samp['dropoff_address'] = samp[['dropoff_latitude', 'dropoff_longitude']].apply(reverse_geo_drop, axis = 1)
```

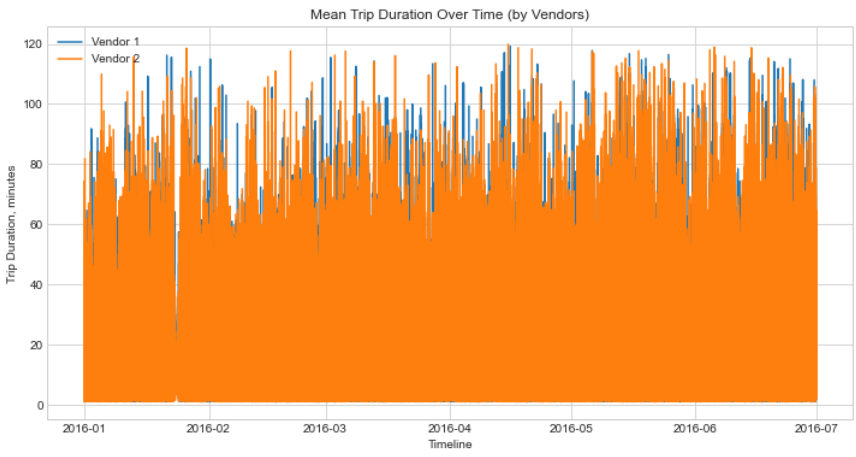
In [72]:
```
samp
```

Out[72]:

	id	vendor_id	pickup_datetime	dropoff_datetime	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	store_and_fwd_flag	trip_duration	pickup_points	dropoff_points	distance	dayOfWeek	Avg_speed	hour	month	day
98	id2102594	1	2016-03-30 16:14:29	2016-03-30 17:01:33	1	-73.789841	40.643559	-73.970665	40.687279	N	2824	POINT (-73.78984 40.64356)	POINT (-73.97066 40.68728)	52633.213474	Wednesday	18.637823	16	3	30
198	id3942917	1	2016-04-29 14:18:39	2016-04-29 15:05:36	4	-73.862778	40.769314	-73.986130	40.748581	N	2817	POINT (-73.86278 40.76931)	POINT (-73.98613 40.74858)	34997.797862	Friday	12.423783	14	4	29
488	id1506453	1	2016-06-16 10:23:25	2016-06-16 11:10:36	1	-73.990822	40.755753	-73.997528	40.722435	N	2831	POINT (-73.99082 40.75575)	POINT (-73.99753 40.72243)	12280.081633	Thursday	4.337719	10	6	16
839	id1437834	2	2016-05-03 10:28:30	2016-05-03 11:17:57	2	-73.870972	40.773819	-74.000717	40.757561	N	2967	POINT (-73.87097 40.77382)	POINT (-74.00072 40.75756)	36425.336771	Tuesday	12.276824	10	5	3
1008	id3039752	1	2016-04-14 15:08:54	2016-04-14 15:56:02	1	-73.987381	40.760708	-73.871986	40.774246	N	2828	POINT (-73.98738 40.76071)	POINT (-73.87199 40.77425)	32342.650919	Thursday	11.436581	15	4	14
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1458263	id3934738	2	2016-06-09 08:37:03	2016-06-09 09:27:50	5	-73.870918	40.773754	-73.998497	40.737244	N	3047	POINT (-73.87092 40.77375)	POINT (-73.99850 40.73724)	37765.847813	Thursday	12.394436	8	6	9
1458310	id1325943	1	2016-05-25 15:08:13	2016-05-25 15:58:24	1	-73.993568	40.724503	-73.955742	40.779602	N	3011	POINT (-73.99357 40.72450)	POINT (-73.95574 40.77960)	22645.639566	Wednesday	7.520970	15	5	25
1458333	id1758713	2	2016-04-17 14:56:46	2016-04-17 15:46:59	1	-73.782722	40.644966	-73.974808	40.750660	N	3013	POINT (-73.78272 40.64497)	POINT (-73.97481 40.75066)	65725.159856	Sunday	21.813860	14	4	17
1458363	id3404858	2	2016-05-18 12:39:49	2016-05-18 13:29:33	2	-73.975510	40.733070	-73.990242	40.762440	N	2984	POINT (-73.97551 40.73307)	POINT (-73.99024 40.76244)	11452.405204	Wednesday	3.837937	12	5	18
1458422	id0875221	1	2016-06-01 09:01:38	2016-06-01 09:48:04	1	-73.872955	40.774158	-73.985207	40.751099	N	2786	POINT (-73.87296 40.77416)	POINT (-73.98521 40.75110)	32210.847073	Wednesday	11.561682	9	6	1

9811 rows × 19 columns

In [73]:
```
data = pd.crosstab(index=date_ny,
                   columns=nyc_data.vendor_id,
                   values=nyc_data.trip_duration/60,
                   aggfunc='mean')
plt.figure(figsize=(12,6))
plt.title('Mean Trip Duration Over Time (by Vendors)')
plt.ylabel('Trip Duration, minutes') ; plt.xlabel('Timeline')
plt.plot(data)
plt.legend(['Vendor 1', 'Vendor 2'])
plt.show()
```



In [74]:
```
# nyc_data['pickup_address'] = nyc_data[['pickup_latitude', 'pickup_longitude']].apply(reverse_geo_pick, axis = 1)
```

In [75]:
```
# nyc_data['dropoff_address'] = nyc_data[['dropoff_latitude', 'dropoff_longitude']].apply(reverse_geo_drop, axis = 1)
```

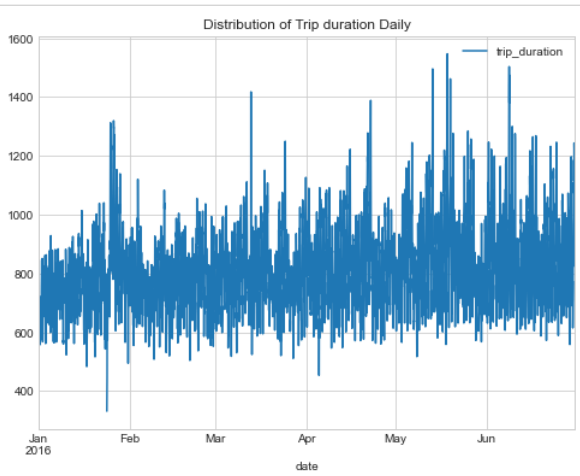
In [76]:
```
sub = nyc_data[['pickup_datetime', 'trip_duration']].copy()
sub['date'] = pd.to_datetime(sub['pickup_datetime'], format='%Y-%m-%d')
sub.set_index(sub['date'], inplace=True)
```

In [77]:
```
sub.head()
```

Out[77]:

	pickup_datetime	trip_duration	date
date			
2016-03-14 17:24:55	2016-03-14 17:24:55	455	2016-03-14 17:24:55
2016-06-12 00:43:35	2016-06-12 00:43:35	663	2016-06-12 00:43:35
2016-01-19 11:35:24	2016-01-19 11:35:24	2124	2016-01-19 11:35:24
2016-04-06 19:32:31	2016-04-06 19:32:31	429	2016-04-06 19:32:31
2016-03-26 13:30:55	2016-03-26 13:30:55	435	2016-03-26 13:30:55

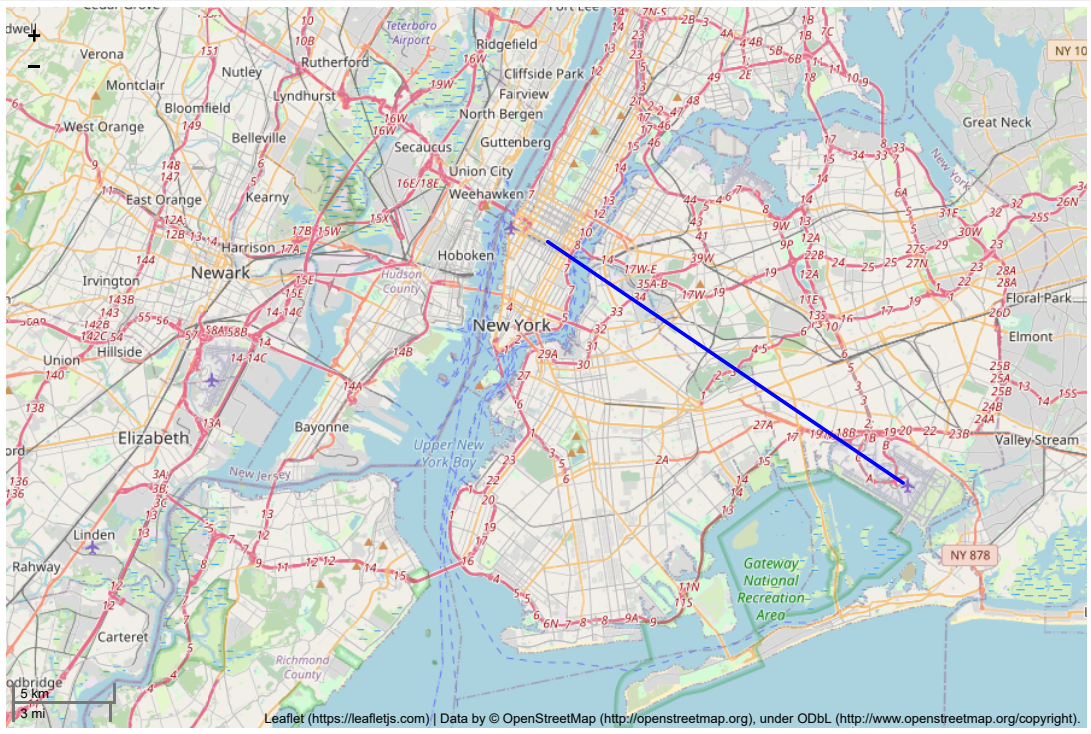
In [78]:
```
fig, ax = plt.subplots(1,1)
fig.set_size_inches(8, 6)
sub.resample('H').mean().plot(ax=ax) # Daily
plt.title("Distribution of Trip duration Daily")
plt.show()
```



In [79]:
```
# Look for the locations from the address in the dataset
```

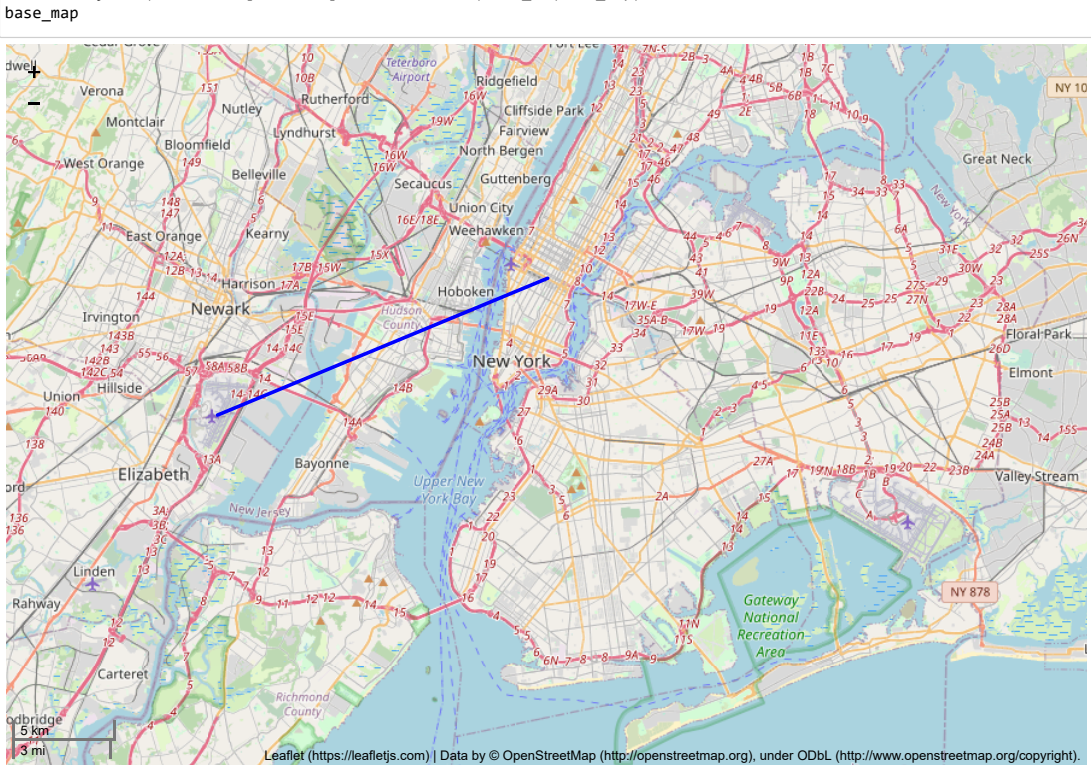
In [80]:
```
# esb to jfk
base_map = generateBaseMap(default_zoom_start=11)
folium.PolyLine(locations=[esb, jfk], color= 'blue').add_to(base_map)
base_map
```

Out[80]:



In [81]:
```
# esb to nwa
base_map = generateBaseMap(default_zoom_start=11)
folium.PolyLine(locations=[esb, nwa], color= 'blue').add_to(base_map)
base_map
```

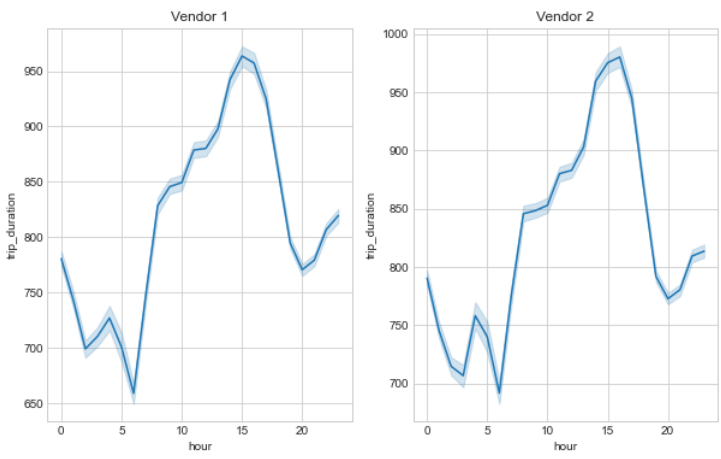
Out[81]:



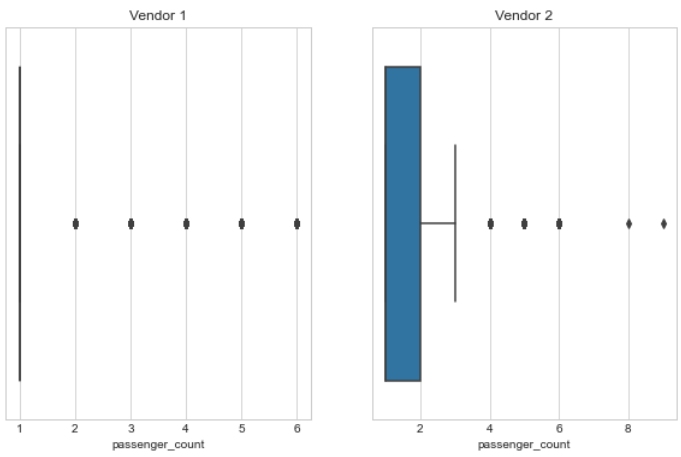
1.6 Vendors

In [82]:
```
vendor_1 = nyc_data[nyc_data.vendor_id == 1]
vendor_2 = nyc_data[nyc_data.vendor_id == 2]
```

In [83]:
```
#duration of trips vendor_1, vendor_2
f, axes = plt.subplots(1,2, figsize=(10, 6))
sns.lineplot(data=vendor_1, x="hour", y="trip_duration" , ax=axes[0]).set_title('Vendor 1')
sns.lineplot(data=vendor_2, x="hour", y="trip_duration" , ax=axes[1]).set_title('Vendor 2')
plt.show()
```



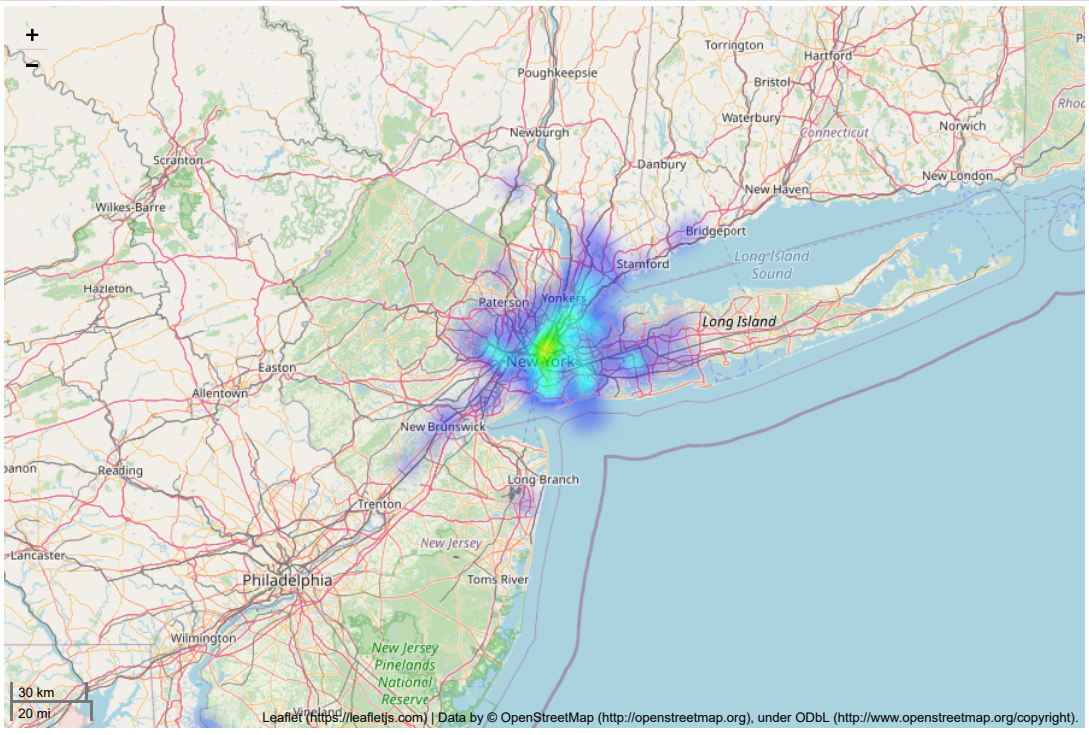
In [84]:
```
# Passenger Count Boxplot
#duration of trips vendor_1, vendor_2
f, axes = plt.subplots(1,2, figsize=(10, 6))

sns.boxplot(vendor_1['passenger_count'], ax=axes[0]).set_title('Vendor 1')
sns.boxplot(vendor_2['passenger_count'], ax=axes[1]).set_title('Vendor 2')
plt.show()
```



In [85]:
```
vendor_1['count'] = 1
vendor_2['count'] = 1
```

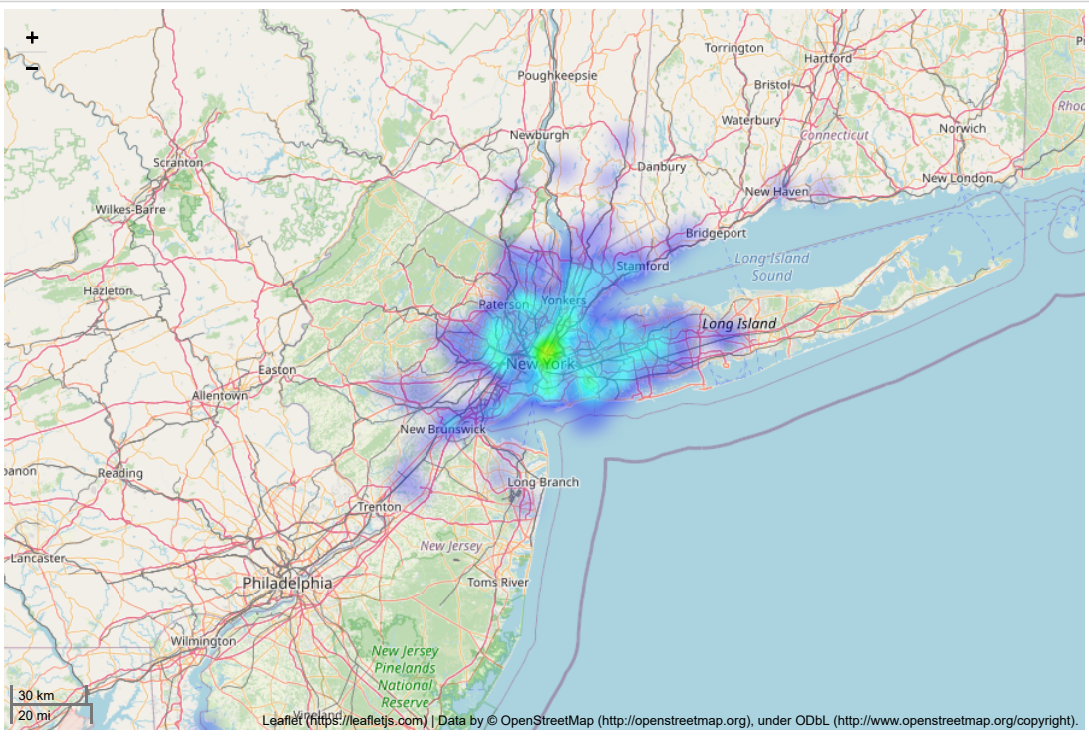
In [86]:
```
# Vendor 1 pickup locations
base_map = generateBaseMap()
HeatMap(data=vendor_1[['pickup_latitude', 'pickup_longitude', 'count']].groupby(['pickup_latitude', 'pickup_longitude']).sum().reset_index().values.tolist(), radius=8, max_zoom=13).add_to(base_map)
base_map
```

Out[86]:



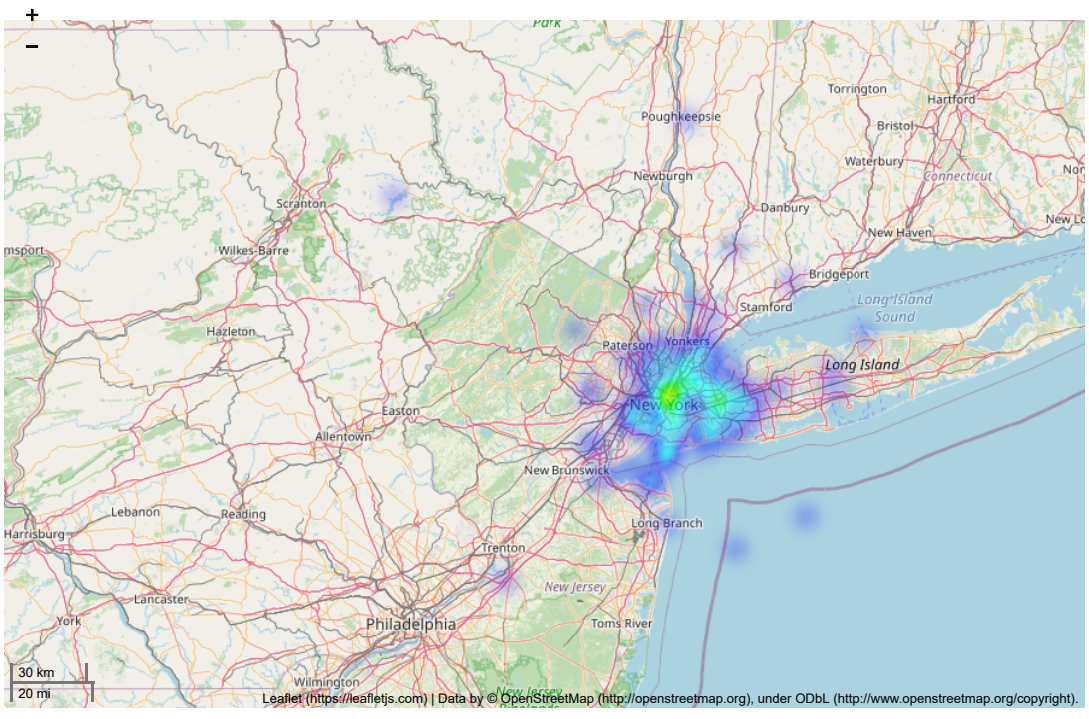
In [87]:
```
# Vendor 1 Dropoff Locations
base_map = generateBaseMap()
HeatMap(data=vendor_1[['dropoff_latitude', 'dropoff_longitude', 'count']].groupby(['dropoff_latitude', 'dropoff_longitude']).sum().reset_index().values.tolist(), radius=8, max_zoom=13).add_to(base_map)
base_map
```

Out[87]:



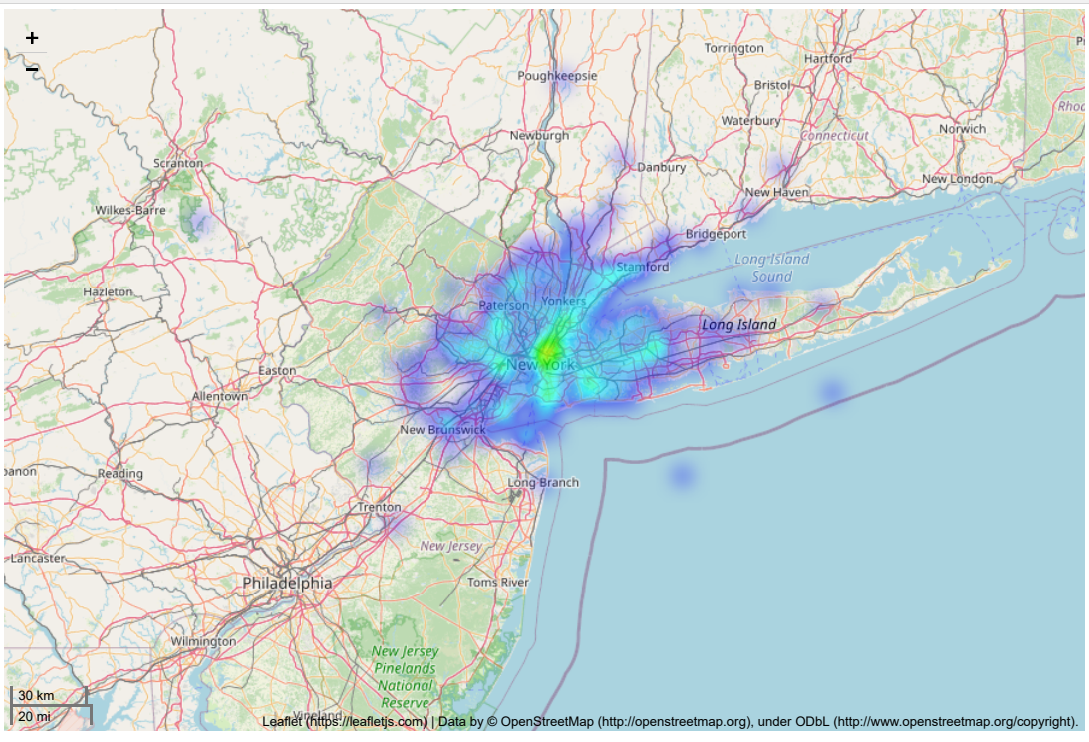
In [88]:
```
# Vendor 2 pickup locations
base_map = generateBaseMap()
HeatMap(data=vendor_2[['pickup_latitude', 'pickup_longitude', 'count']].groupby(['pickup_latitude', 'pickup_longitude']).sum().reset_index().values.tolist(), radius=8, max_zoom=13).add_to(base_map)
base_map
```

Out[88]:



In [89]:
```
# Vendor 2 Dropoff Locations
base_map = generateBaseMap()
HeatMap(data=vendor_2[['dropoff_latitude', 'dropoff_longitude', 'count']].groupby(['dropoff_latitude', 'dropoff_longitude']).sum().reset_index().values.tolist(), radius=8, max_zoom=13).add_to(base_map)
base_map
```

Out[89]:



- Vendor 2 has the highest trip durations per hour compared to vendor 1.
- Vendor 1 pickup and dropoff locations are very widely spread, showing a larger area of operation.
- The number of passenger for Vendor 1 can not be clearly distinguished in the box plot, while Vendor 2 is clearly visible, a maximum of 3 passengers, with a few exceptions.
- Vendor 2 shows a more intense areas with regards to hotspots, meaning vendor 2 has a higher pickup and dropoff rate. within the intense area.

Therefore:

- **Vendor 1 = Yellow Cabs**
- **Vendor 2 = boro taxis**

1.7 Boroughs

1.7.1

In [111]:
```
# load shapefile
nyc = gpd.read_file("geo_export_da24b28d-b7fe-488f-b052-8bec92c4ac08.shp")
```

In [112]:
```
nyc.head()
```

Out[112]:

	boro_code	boro_name	county_fip	ntacode	ntaname	shape_area	shape_leng	geometry
0	3.0	Brooklyn	047	BK88	Borough Park	5.400502e+07	39247.227887	POLYGON ((-73.97605 40.63128, -73.97717 40.630...
1	4.0	Queens	081	QN51	Murray Hill	5.248828e+07	33266.904952	POLYGON ((-73.80379 40.77561, -73.80099 40.775...
2	4.0	Queens	081	QN27	East Elmhurst	1.972685e+07	19816.711764	POLYGON ((-73.86110 40.76366, -73.85993 40.762...
3	3.0	Brooklyn	047	BK23	West Brighton	8.738769e+06	14113.581565	POLYGON ((-73.96889 40.57526, -73.96895 40.575...
4	4.0	Queens	081	QN41	Fresh Meadows-Utopia	2.777485e+07	22106.431272	POLYGON ((-73.77758 40.73019, -73.77849 40.729...

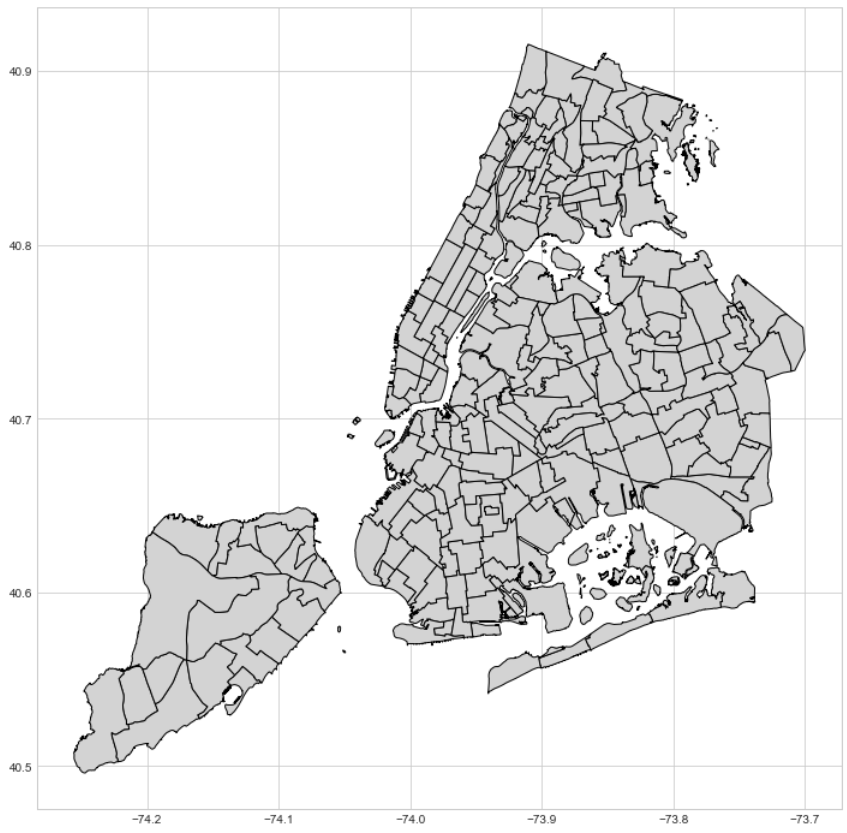
In [113]:
```
# Boroughs
nyc.boro_name.unique()
```

Out[113]:
```
array(['Brooklyn', 'Queens', 'Bronx', 'Staten Island', 'Manhattan'],
      dtype=object)
```

In [114]:
```
# Neighbourhoods
nyc.ntaname.unique()
```

Out[114]:
```
array(['Borough Park', 'Murray Hill', 'East Elmhurst', 'West Brighton',
       'Fresh Meadows-Utopia', 'St. Albans', 'Clinton Hill',
       'Pelham Bay-Country Club-City Island', 'Gravesend',
       'Ocean Parkway South', 'Soundview-Bruckner', 'Glendale',
       'Van Cortlandt Village', 'Highbridge', 'Middle Village',
       'South Ozone Park', 'Schuylerville-Throgs Neck-Edgewater Park',
       'Mount Hope', 'Windsor Terrace', 'Canarsie',
       'Soundview-Castle Hill-Clason Point-Harding Park',
       'Rossville-Woodrow', 'Gramercy', 'Upper West Side', 'Norwood',
       'Bedford Park-Fordham North', 'Sunset Park East',
       'Oakland Gardens', 'North Corona', 'Rego Park', 'Woodside',
       'Fort Greene', 'Starrett City', 'Briarwood-Jamaica Hills',
       'Williamsbridge-Olinville', 'Whitestone', 'Ozone Park',
       'Springfield Gardens South-Brookville',
       'Springfield Gardens North', 'Laurelton', 'Longwood', 'Ocean Hill',
       'Melrose South-Mott Haven North',
       'Prospect Lefferts Gardens-Wingate', 'Queensboro Hill',
       'Morrisania-Melrose', 'Crotona Park East', 'Kew Gardens Hills',
       'Pomonok-Flushing Heights-Hillcrest', 'East Flushing',
       'West Farms-Bronx River', 'Kingsbridge Heights',
       'University Heights-Morris Heights', 'Williamsburg',
       'North Riverdale-Fieldston-Riverdale', 'Madison', 'Hollis',
       'Rosedale', 'Queens Village', 'Grymes Hill-Clifton-Fox Hills',
       'South Jamaica', 'Baisley Park', 'Erasmus',
       'Spuyten Duyvil-Kingsbridge', 'Steinway', 'Richmond Hill',
       'Rikers Island', 'Auburndale', 'Astoria',
       'Jamaica Estates-Holliswood', 'Bushwick North', 'Jamaica',
       'Corona', 'Ridgewood', 'Maspeth', 'Elmhurst', 'Elmhurst-Maspeth',
       'Stuyvesant Heights', 'Belmont', 'East Tremont',
       'East Williamsburg', 'Midtown-Midtown South',
       'Kensington-Ocean Parkway', 'Claremont-Bathgate', 'Fordham South',
       'Washington Heights North',
       'Hudson Yards-Chelsea-Flatiron-Union Square', 'Clinton',
       'Yorkville', 'Marble Hill-Inwood', 'Bedford',
       'Ft. Totten-Bay Terrace-Clearview', 'Bushwick South',
       'Morningside Heights', 'Stapleton-Rosebank',
       'Douglas Manor-Douglaston-Little Neck', 'East Harlem North',
       'Central Harlem South', 'Flushing', 'Forest Hills',
       'Crown Heights South', 'Cambria Heights', 'Bayside-Bayside Hills',
       'New Brighton-Silver Lake', 'Bellerose',
       'East New York (Pennsylvania Ave)',
       'Glen Oaks-Floral Park-New Hyde Park', 'Jackson Heights',
       'Midwood', 'Chinatown', 'Old Astoria', 'Crown Heights North',
       'Brownsville', 'East Flatbush-Farragut', 'Rugby-Remsen Village',
       'Flatlands', 'Parkchester',
       'SoHo-TriBeCa-Civic Center-Little Italy',
       'Battery Park City-Lower Manhattan', 'Sunset Park West',
       'Murray Hill-Kips Bay', 'Stuyvesant Town-Cooper Village',
       'Far Rockaway-Bayswater', 'College Point',
       'Brooklyn Heights-Cobble Hill',
       'DUMBO-Vinegar Hill-Downtown Brooklyn-Boerum Hill', 'Airport',
       'North Side-South Side', 'Lower East Side',
       'Lenox Hill-Roosevelt Island',
       'Queensbridge-Ravenswood-Long Island City',
       'Charleston-Richmond Valley-Tottenville',
       "Annadale-Huguenot-Prince's Bay-Eltingville", 'Great Kills',
       'Old Town-Dongan Hills-South Beach',
       'Grasmere-Arrochar-Ft. Wadsworth', 'New Dorp-Midland Beach',
       'East Harlem South', 'Arden Heights',
       'Georgetown-Marine Park-Bergen Beach-Mill Basin', 'Bath Beach',
       'Bensonhurst West', 'Bensonhurst East', 'Oakwood-Oakwood Beach',
       'park-cemetery-etc-Staten Island', 'Manhattanville', 'Hunts Point',
       'Mott Haven-Port Morris', 'Bronxdale', 'Kew Gardens',
       'Lincoln Square', 'Sheepshead Bay-Gerritsen Beach-Manhattan Beach',
       'Prospect Heights', 'Dyker Heights', 'Bay Ridge',
       'Carroll Gardens-Columbia Street-Red Hook', 'Park Slope-Gowanus',
       'Brighton Beach', 'Homecrest', 'Seagate-Coney Island',
       "Mariner's Harbor-Arlington-Port Ivory-Graniteville",
       'Port Richmond', 'Woodhaven', 'Westerleigh',
       'West New Brighton-New Brighton-St. George',
       'Turtle Bay-East Midtown', 'Woodlawn-Wakefield',
       'park-cemetery-etc-Bronx', 'East New York',
       'Lindenwood-Howard Beach',
       'Todt Hill-Emerson Hill-Heartland Village-Lighthouse Hill',
       'Allerton-Pelham Gardens', 'Greenpoint',
       'Hunters Point-Sunnyside-West Maspeth', 'Washington Heights South',
       'Upper East Side-Carnegie Hill', 'park-cemetery-etc-Manhattan',
       'Van Nest-Morris Park-Westchester Square', 'Westchester-Unionport',
       'New Springville-Bloomfield-Travis',
       'Eastchester-Edenwald-Baychester', 'Co-op City',
       'Central Harlem North-Polo Grounds', 'Hamilton Heights',
       'Pelham Parkway', 'Cypress Hills-City Line',
       'park-cemetery-etc-Queens', 'Flatbush',
       'park-cemetery-etc-Brooklyn',
       'Breezy Point-Belle Harbor-Rockaway Park-Broad Channel',
       'Hammels-Arverne-Edgemere', 'East Village', 'West Village',
       'East Concourse-Concourse Village', 'West Concourse'], dtype=object)
```

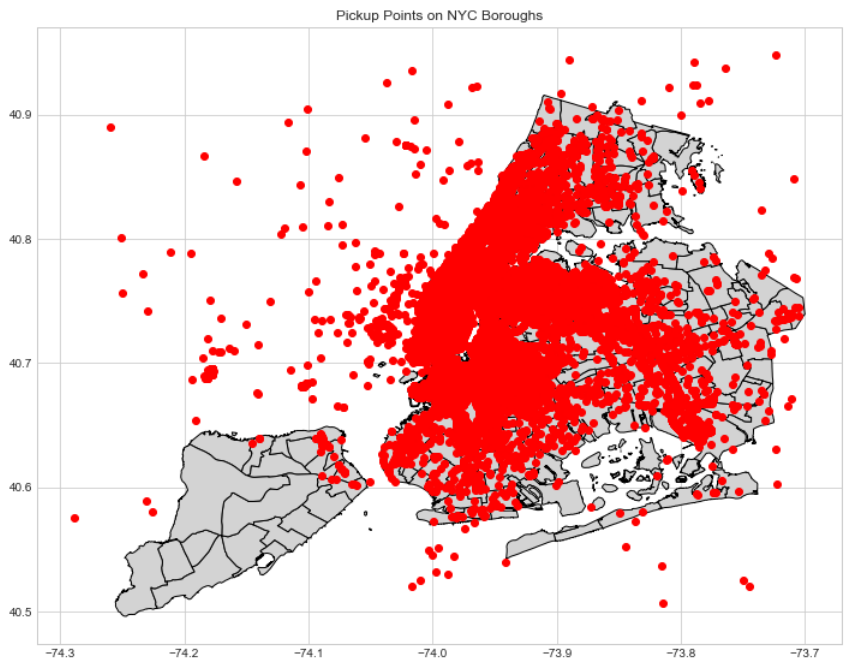
In [115]:
```
fig, ax = plt.subplots(figsize = (12,14))
nyc.plot(ax= ax, color='lightgrey', edgecolor='black')
plt.show()
```



In [116]:
```
gdf_pickup = gpd.GeoDataFrame(
    nyc_data, geometry=gpd.points_from_xy(nyc_data.pickup_longitude, nyc_data.pickup_latitude))
```

In [117]:
```
# removing outliers that fall outside the boroughs
gdf_pickup = gdf_pickup[(gdf_pickup['pickup_longitude'] > -74.3) & (gdf_pickup['pickup_longitude'] < -73.7)]
gdf_pickup = gdf_pickup[(gdf_pickup['pickup_latitude'] < 40.95) & (gdf_pickup['pickup_latitude'] > 40.5)]
```

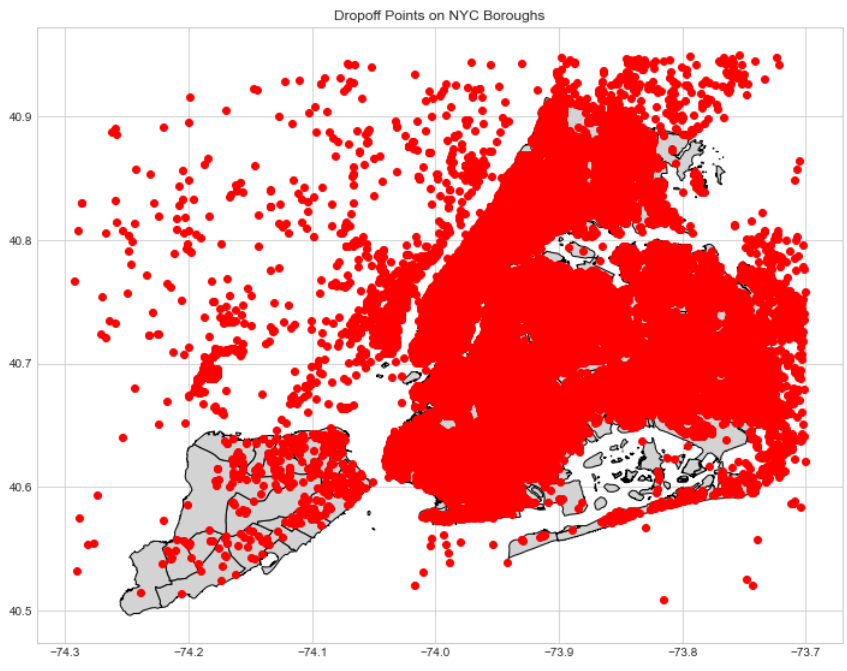
In [118]:
```
# Neighbourhoods for trip start locations
fig, ax = plt.subplots(figsize = (12,14))
nyc.plot(ax=ax, color='lightgrey', edgecolor='black') # .to_crs(epsg=2263)  4326
gdf_pickup.plot(ax=ax, color='red') # using geo_df_pickup
ax.set_title('Pickup Points on NYC Boroughs')
plt.show()
```



In [119]:
```
gdf_dropoff = gpd.GeoDataFrame(
    nyc_data, geometry=gpd.points_from_xy(nyc_data.dropoff_longitude, nyc_data.dropoff_latitude))
```

In [120]:
```
# removing outliers that fall outside the boroughs
gdf_dropoff = gdf_dropoff[(gdf_dropoff['dropoff_longitude'] > -74.3) & (gdf_dropoff['dropoff_longitude'] < -73.7)]
gdf_dropoff = gdf_dropoff[(gdf_dropoff['dropoff_latitude'] < 40.95) & (gdf_dropoff['dropoff_latitude'] > 40.5)]
```

In [121]:
```
# Neighbourhoods to trip end locations
fig, ax = plt.subplots(figsize = (12,14))
nyc.plot(ax=ax, color='lightgrey', edgecolor='black') # .to_crs(epsg=2263)  4326
gdf_dropoff.plot(ax=ax, color='red') # using geo_df_pickup
ax.set_title('Dropoff Points on NYC Boroughs')
plt.show()
```



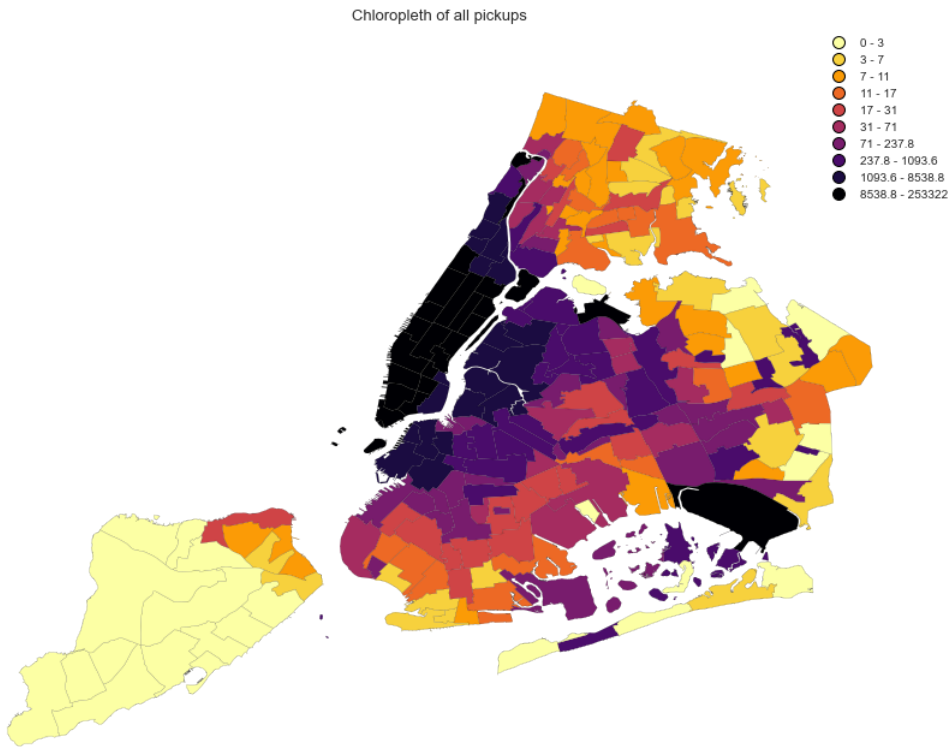
1.7.2

In [14]:
```
gdf_pickup = gpd.GeoDataFrame(
    nyc_data[['pickup_latitude', 'pickup_longitude']], geometry=gpd.points_from_xy(nyc_data.pickup_longitude, nyc_data.pickup_latitude))
```

In [15]:
```
gdf_pickup['count'] = 1
```

In [16]:
```
# gdf_pickup creating sum column in nyc shapefile from pickup locations sum = pickup
sum_hex = []
spatial_index = gdf_pickup.sindex

for index, row in nyc.iterrows():
    polygon = row.geometry
    possible_matches_index = list(spatial_index.intersection(polygon.bounds))
    possible_matches = gdf_pickup.iloc[possible_matches_index]
    precise_matches = possible_matches[possible_matches.within(polygon)]
    sum_hex.append(sum(precise_matches['count']))

nyc['sum'] = sum_hex
```

In [124]:
```
fig, ax = plt.subplots(1, 1, figsize=(16, 12))

# Set up the color sheme:
scheme = mc.Quantiles(nyc['sum'], k=10)

# Map
gplt.choropleth(nyc,
    hue="sum",
    linewidth=.1,
    scheme=scheme, cmap='inferno_r',
    legend=True,
    edgecolor='black',
    ax=ax
);

ax.set_title('Chloropleth of all pickups', fontsize=13);
```

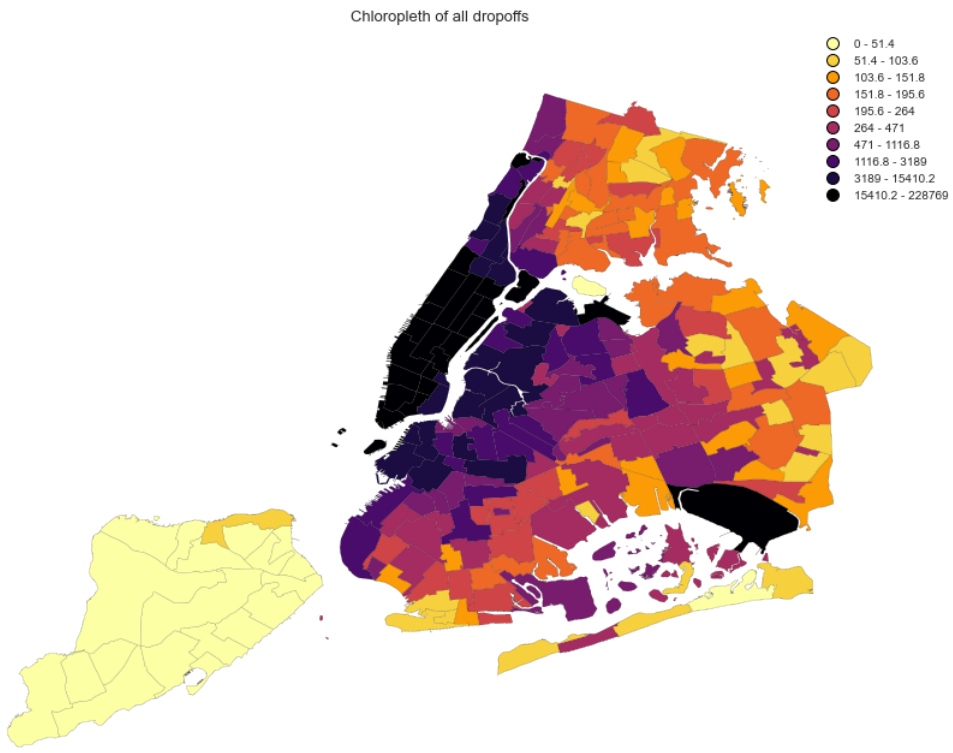


In [125]:
```
gdf_dropoff = gpd.GeoDataFrame(
    nyc_data[['dropoff_latitude', 'dropoff_longitude']], geometry=gpd.points_from_xy(nyc_data.dropoff_longitude, nyc_data.dropoff_latitude))
```

In [30]:
```
gdf_dropoff['count'] = 1
```

In [31]:
```
# gdf_dropoff creating sum2 column in nyc shapefile from pickup locations sum2 = dropoff
sum_hex2 = []
spatial_index2 = gdf_dropoff.sindex

for index, row in nyc.iterrows():
    polygon2 = row.geometry
    possible_matches_index2 = list(spatial_index2.intersection(polygon2.bounds))
    possible_matches2 = gdf_dropoff.iloc[possible_matches_index2]
    precise_matches2 = possible_matches2[possible_matches2.within(polygon2)]
    sum_hex2.append(sum(precise_matches2['count']))

nyc['sum2'] = sum_hex2
```

In [126]:
```
fig, ax = plt.subplots(1, 1, figsize=(16, 12))

# Set up the color sheme:
scheme2 = mc.Quantiles(nyc['sum2'], k=10)

# Map
gplt.choropleth(nyc,
    hue="sum2",
    linewidth=.1,
    scheme=scheme2, cmap='inferno_r',
    legend=True,
    edgecolor='black',
    ax=ax
);

ax.set_title('Chloropleth of all dropoffs', fontsize=13);
```



Difference in distribution? between pickup and dropoff chloropeths?

- There is a higher distribution of dropoffs than pickups within the boroughs

1.7.3

In [127]:
```
# most incoming/outgoing trips
nyc['sum'].groupby(nyc['boro_name']).sum().reset_index().values
```

Out[127]:
```
array([['Bronx', 1257],
       ['Brooklyn', 26394],
       ['Manhattan', 1343792],
       ['Queens', 85921],
       ['Staten Island', 57]], dtype=object)
```

- Which boroughs have the most incoming and **outgoing** trips?

  Since these are pickups therefore:

  Manhattan has the most outgoing trips, followed by Queens

In [128]:
```
# most incoming/outgoing trips
nyc['sum2'].groupby(nyc['boro_name']).sum().reset_index().values
```

Out[128]:
```
array([['Bronx', 9389],
       ['Brooklyn', 77516],
       ['Manhattan', 1288941],
       ['Queens', 76089],
       ['Staten Island', 376]], dtype=object)
```

- Which boroughs have the most **incoming** and outgoing trips?

  Since these are dropoffs therefore:

  Manhattan has the most incoming trips, followed by Brooklyn

1.7.4 & 1.7.5

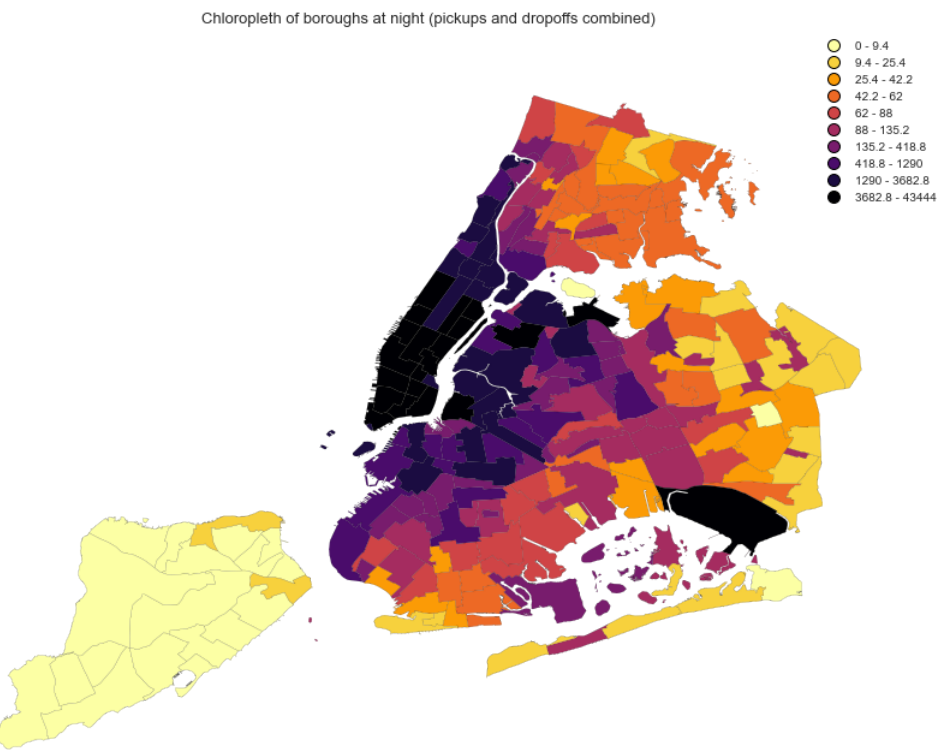
In [129]:
```
# selecting night between 0-5am, (concat pickup and dropoff)
# plot chloropleth to determine the boroughs that are busiest and quietest?
night = nyc_data[(nyc_data.hour >= 0) & (nyc_data.hour <= 5)]
```

In [130]:
```
gdf_night_1 = gpd.GeoDataFrame(
    night[['pickup_latitude', 'pickup_longitude']], geometry=gpd.points_from_xy(night.pickup_longitude, night.pickup_latitude))
gdf_night_2 = gpd.GeoDataFrame(
    night[['dropoff_latitude', 'dropoff_longitude']], geometry=gpd.points_from_xy(night.dropoff_longitude, night.dropoff_latitude))

gdf_night = pd.concat([gdf_night_1, gdf_night_2])
```

In [48]:
```
gdf_night['count'] = 1
```

In [49]:
```
sum_hex3 = []
spatial_index3 = gdf_night.sindex

for index, row in nyc.iterrows():
    polygon3 = row.geometry
    possible_matches_index3 = list(spatial_index3.intersection(polygon3.bounds))
    possible_matches3 = gdf_night.iloc[possible_matches_index3]
    precise_matches3 = possible_matches3[possible_matches3.within(polygon3)]
    sum_hex3.append(sum(precise_matches3['count']))

nyc['sum3'] = sum_hex3
```

In [131]:
```
fig, ax = plt.subplots(1, 1, figsize=(16, 12))

# Set up the color sheme:
scheme3 = mc.Quantiles(nyc['sum3'], k=10)

# Map
gplt.choropleth(nyc,
    hue="sum3",
    linewidth=.1,
    scheme=scheme3, cmap='inferno_r',
    legend=True,
    edgecolor='black',
    ax=ax
);

ax.set_title('Chloropleth of boroughs at night (pickups and dropoffs combined)', fontsize=13);
```



In [132]:
```
nyc['sum3'].groupby(nyc['boro_name']).sum().reset_index().values
```

Out[132]:
```
array([['Bronx', 3230],
       ['Brooklyn', 30863],
       ['Manhattan', 280793],
       ['Queens', 26474],
       ['Staten Island', 94]], dtype=object)
```

- Which boroughs are quietest at night?

  Staten Island is the the quitest at night, followed by Bronx

- Which boroughs are busiest at night?

  Manhattan is the busiest at night, followed by Brooklyn