DVE_assignment2

June 21, 2022

1 COMS7056A Assignment 2

1.1 Dataset Overview:

You will analyse a public dataset from Uber available on Kaggle, available here: https://www.kaggle.com/c/nyctaxi-trip-duration. It is also available on Moodle for download. Your primary dataset is one released by the NYC Taxi and Limousine Commission (TLC), which includes pickup time, geo-coordinates, number of passengers, and several other variables for 1.5 million trips between 2016-01-01 and 2016-06-30. Note that for this analysis, just use the training sample. - `id` - a unique identifier for each trip - `vendor_id` - a code indicating the provider associated with the trip record - `pickup_datetime` - date and time when the meter was engaged - `dropoff_datetime` - date and time when the meter was disengaged - `passenger_count` - the number of passengers in the vehicle (driver entered value) - `pickup_longitude` - the longitude where the meter was engaged - `pickup_latitude` - the latitude where the meter was engaged - `dropoff_longitude` - the longitude where the meter was disengaged - `dropoff_latitude` - the latitude where the meter was disengaged - `store_and_fwd_flag` - This flag indicates whether the trip record was held in vehicle memory before sending to the vendor because the vehicle did not have a connection to the server - Y=store and forward; N=not a store and forward trip - `trip_duration` - duration of the trip in seconds There is more up-to-date data available from TLC, but the datasets are large (10GB+ per year since 2018). They do include additional fields, however. For this exercise, you can assume the centre of New York City is at the long/lat coordinates (40.716662,-74.009899).

[17]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import datetime as datetime
import scipy.stats as stats
import warnings
import matplotlib
import matplotlib.cm as cm
from shapely.geometry import Point
import geopandas as gpd # install on google colab
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 # install on colab
import geoplot.crs as gcrs
import imageio
import pathlib
import mapclassify as mc
import geocoder # install on colab
from folium import plugins
import hdbscan # install on colab
import utm # install on colab
from ipywidgets import interact, interactive
import ipywidgets as widgets
from folium.vector_layers import CircleMarker
from colour import Color # install on colab
#sns.set(rc={'figure.figsize':(20,5)})
matplotlib.rcParams['figure.figsize'] = (20,5)
warnings.simplefilter(action='ignore', category= FutureWarning)
from pandas.core.common import SettingWithCopyWarning
warnings.simplefilter(action="ignore", category=SettingWithCopyWarning)
plt.style.use('seaborn-whitegrid')
```

[ ]:
```
# install: geopandas, geoplot, geocoder, hdbscan, utm, colour, #uncomment code␣
↪below to install packages
#!pip install geocoder
```

[ ]:
```
#!pip uninstall rtree
#!sudo apt install libspatialindex-dev
#!pip install rtree
```

[ ]:
```
# Upload data
#from google.colab import files
#uploaded = files.upload()
```

[ ]:
```
#from google.colab import drive
#drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

2 1 Data Cleaning

[2]:
```
df = pd.read_csv('C:/Users/Mfund/Downloads/nyc_taxis.csv')
```

[3]:
```
df.head()
```

[3]:
```
          id  vendor_id      pickup_datetime     dropoff_datetime  \
0  id2875421          2  2016-03-14 17:24:55  2016-03-14 17:32:30
```

```
1  id2377394          1  2016-06-12 00:43:35  2016-06-12 00:54:38
2  id3858529          2  2016-01-19 11:35:24  2016-01-19 12:10:48
3  id3504673          2  2016-04-06 19:32:31  2016-04-06 19:39:40
4  id2181028          2  2016-03-26 13:30:55  2016-03-26 13:38:10

   passenger_count  pickup_longitude  pickup_latitude  dropoff_longitude  \
0                1        -73.982155        40.767937         -73.964630
1                1        -73.980415        40.738564         -73.999481
2                1        -73.979027        40.763939         -74.005333
3                1        -74.010040        40.719971         -74.012268
4                1        -73.973053        40.793209         -73.972923

   dropoff_latitude store_and_fwd_flag  trip_duration
0         40.765602                  N            455
1         40.731152                  N            663
2         40.710087                  N           2124
3         40.706718                  N            429
4         40.782520                  N            435
```

[5]:
```
# Data information
df.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
```

[6]:
```
# Number of missing values in each column
df.isnull().sum()
```

[6]:
```
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
```

[7]:
```
# Number of unique values in each column
df.nunique()
```

[7]:
```
id                    1458644
vendor_id                   2
pickup_datetime       1380222
dropoff_datetime      1380377
passenger_count            10
pickup_longitude        23047
pickup_latitude         45245
dropoff_longitude       33821
dropoff_latitude        62519
store_and_fwd_flag          2
trip_duration            7417
dtype: int64
```

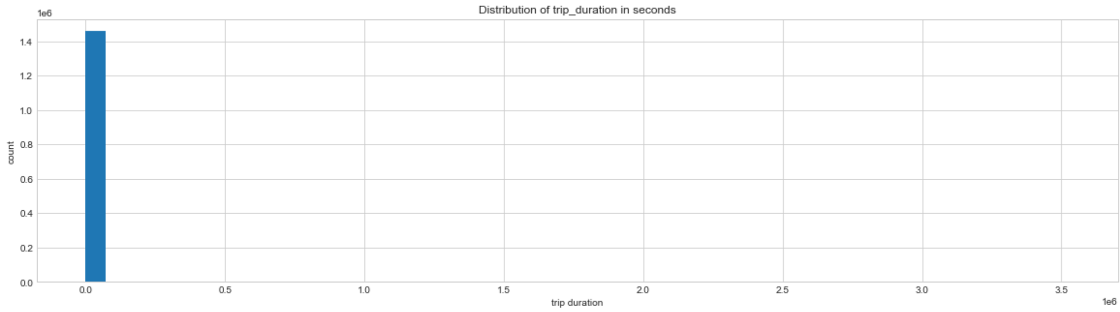
There are no missing values in the dataset as shown above

Outlier Detection and Handling

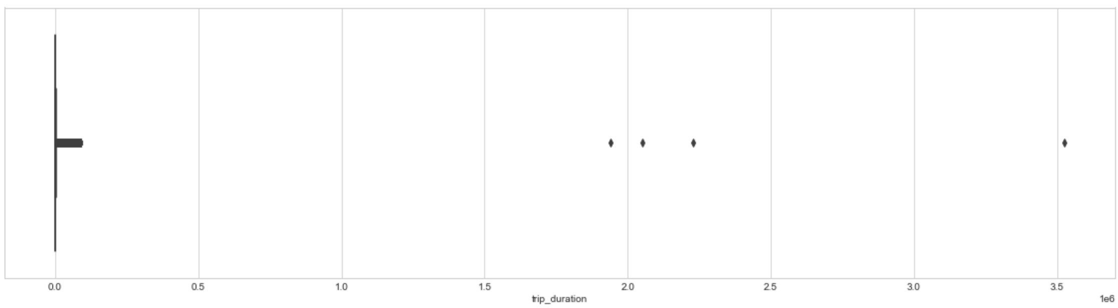
[8]:
```
def haversine_np(lon1, lat1, lon2, lat2):
    """
    Calculate the great circle distance between two points
    on the earth (specified in decimal degrees)
    All args must be of equal length.
    """
    lon1, lat1, lon2, lat2 = map(np.radians, [lon1, lat1, lon2, lat2])
    dlon = lon2 - lon1
    dlat = lat2 - lat1
    a = np.sin(dlat/2.0)**2 + np.cos(lat1) * np.cos(lat2) * np.sin(dlon/2.0)**2
    c = 2 * np.arcsin(np.sqrt(a))
    km = 6367 * c
    return km
```

[18]:
```
plt.hist(df['trip_duration'].values, bins=50)
plt.xlabel('trip duration')
plt.ylabel('count')
plt.title("Distribution of trip_duration in seconds");
```

```
plt.show()
```



[19]:
```
sns.boxplot(df['trip_duration']);
plt.show()
```



[11]:
```
# Statistical summary of trips
# pd.options.display.float_format = '{:.1f}'.format
df['trip_duration'].describe().apply(lambda x: '%.5f' % x)
```

[11]:
```
count    1458644.00000
mean         959.49227
std         5237.43172
min            1.00000
25%          397.00000
50%          662.00000
75%         1075.00000
max      3526282.00000
Name: trip_duration, dtype: object
```

[12]:
```
# Number of Trip durations longer than a day
# A day in seconds = 24*60*60 = 86400
len(df[df['trip_duration'] > 86400])
```

[12]:
```
4
```

[13]:
```
# Number of Trip durations less than a minute
len(df[df['trip_duration'] < 60])
```

[13]: 8595

[20]:
```
# Outlier Detection for Trip duration
columns = ['trip_duration']
for column in df[columns]:
  df1_Q1 = df[column].describe()['25%']
  print("Q1", df1_Q1)

  df1_Q3 = df[column].describe()['75%']
  print("Q3", df1_Q3)
  df1_IQR = df1_Q3 - df1_Q1
  print("IQR", df1_IQR)

  # Any number greater than this is a suspected outlier.
  df1_ub = (df1_Q3 +1.5*df1_IQR)
  print("Upper Bound", df1_ub)

  # Any number less than this is a suspected outlier.
  df1_lb = (df1_Q1 -1.5*df1_IQR)
  print("Lower Bound", df1_lb)
```

```
Q1 397.0
Q3 1075.0
IQR 678.0
Upper Bound 2092.0
Lower Bound -620.0
```

[21]:
```
# Number of Trips that take longer than 4 hours
len(df[df['trip_duration'] > 14400])
```

[21]: 2077

[22]:
```
# Percentage of Trips that take longer than 4 hours
len(df[df['trip_duration'] > 14400])/len(df) * 100
```

[22]: 0.14239252346700088

Using the IQR (Inter Quartile Range) Method for outliers, trip durations that are more than 2092 seconds (34,86 mins) are considered to be outliers, however trips may even take longer. Trips that take longer than 4 hours will be considered as outliers and removed

From the above statistical summary and data exploration, it can be observed that there are trip durations that exceed 24 hours and there also several trips have a duration of less than a minute. These values are not intuitive in the context of this dataset and therefore the values were removed

Outlier Detection and Handling for passenger_count According to nyc.gov The maximum amount of passengers allowed in a yellow taxicab by law is four (4) in a four (4) passenger taxicab or five (5) passengers in a five (5) passenger taxicab, except that an additional passenger must be accepted if such passenger is under the age of seven (7) and is held on the lap of an adult passenger seated in the rear.

[23]:
```
df.passenger_count.value_counts()
```

[23]:
```
1    1033540
2     210318
5      78088
3      59896
6      48333
4      28404
0         60
7          3
9          1
8          1
Name: passenger_count, dtype: int64
```

[24]:
```
sns.countplot(x='passenger_count',data=df);
plt.title("Number of Passengers in NYC Taxi");
plt.xlabel("Number Passenger seats");
plt.show()
```

Intuitively a taxi trip cannot have 0 passengers, and more than 5 passengers according to the NYC Taxi & Limousine Commission, so these values will be removed from the dataset

[25]:
```
# Removing trips with durations less than 60 seconds and greater than 4 hours
df=df[df['trip_duration'] > 60]
df=df[df['trip_duration'] < 14400]

# Removing trips with 0 and and more than 6 passengers
df=df[df['passenger_count']!=0]
df=df[df['passenger_count']<6]
```

[26]:
```
len(df)
```

[26]:
```
1399739
```

3 2 Feature generation

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

Distance of trip

[27]:
```
# Distance of trip
df["trip_distance"] = df.apply(lambda x: haversine_np(x.pickup_longitude, x.
↪pickup_latitude, x.dropoff_longitude, x.dropoff_latitude), axis=1)
```

Day of week

[28]:
```
# Day of week
df['pickup_datetime']=pd.to_datetime(df['pickup_datetime'])
df['dropoff_datetime']=pd.to_datetime(df['dropoff_datetime'])

df['pickup_day']=df['pickup_datetime'].dt.day_name()
df['dropoff_day']=df['dropoff_datetime'].dt.day_name()

df['pickup_day_no']=df['pickup_datetime'].dt.weekday
df['dropoff_day_no']=df['dropoff_datetime'].dt.weekday
```

Average speed of trip

[29]:
```
# Average speed of trip
# Average Speed = (Total distance)/(Total Time)
df['trip_duration_hr'] = df['trip_duration']/3600
df['Average_speed_km/sec'] = df['trip_distance']/df['trip_duration']
df['Average_speed_km/hr'] = df['trip_distance']/df['trip_duration_hr']
```

[30]:
```
df.head()
```

[30]:
```
          id  vendor_id     pickup_datetime    dropoff_datetime  \
0  id2875421          2 2016-03-14 17:24:55 2016-03-14 17:32:30
1  id2377394          1 2016-06-12 00:43:35 2016-06-12 00:54:38
2  id3858529          2 2016-01-19 11:35:24 2016-01-19 12:10:48
3  id3504673          2 2016-04-06 19:32:31 2016-04-06 19:39:40
4  id2181028          2 2016-03-26 13:30:55 2016-03-26 13:38:10

   passenger_count  pickup_longitude  pickup_latitude  dropoff_longitude  \
0                1        -73.982155        40.767937         -73.964630
1                1        -73.980415        40.738564         -73.999481
```

```
2                1       -73.979027       40.763939        -74.005333
3                1       -74.010040       40.719971        -74.012268
4                1       -73.973053       40.793209        -73.972923

   dropoff_latitude store_and_fwd_flag  trip_duration  trip_distance  \
0         40.765602                  N            455       1.497580
1         40.731152                  N            663       1.804374
2         40.710087                  N           2124       6.381090
3         40.706718                  N            429       1.484566
4         40.782520                  N            435       1.187842

  pickup_day dropoff_day  pickup_day_no  dropoff_day_no  trip_duration_hr  \
0     Monday      Monday              0               0          0.126389
1     Sunday      Sunday              6               6          0.184167
2    Tuesday     Tuesday              1               1          0.590000
3  Wednesday   Wednesday              2               2          0.119167
4   Saturday    Saturday              5               5          0.120833

   Average_speed_km/sec  Average_speed_km/hr
0              0.003291            11.848984
1              0.002722             9.797504
2              0.003004            10.815406
3              0.003461            12.457894
4              0.002731             9.830418
```

Additional Features

[31]:
```
df['pickup_hour']=df['pickup_datetime'].dt.hour
df['dropoff_hour']=df['dropoff_datetime'].dt.hour

df['pickup_month']=df['pickup_datetime'].dt.month
df['dropoff_month']=df['dropoff_datetime'].dt.month
```

Determining what time of the day the ride was taken function

- `Morning`: from 6:00 am to 11:59 pm
- `Afternoon`: from 12 noon to 3:59 pm
- `Evening`: from 4:00 pm to 9:59 pm
- `Late Night` from 10:00 pm to 5:59 am

[32]:
```
def time_of_day(x):
    if x in range(6,12):
        return 'Morning'
    elif x in range(12,16):
        return 'Afternoon'
    elif x in range(16,22):
        return 'Evening'
    else:
```

```
        return 'Late night'
```

[33]:
```
df['pickup_timeofday']=df['pickup_hour'].apply(time_of_day)
df['dropoff_timeofday']=df['dropoff_hour'].apply(time_of_day)
```

[34]:
```
df.head()
```

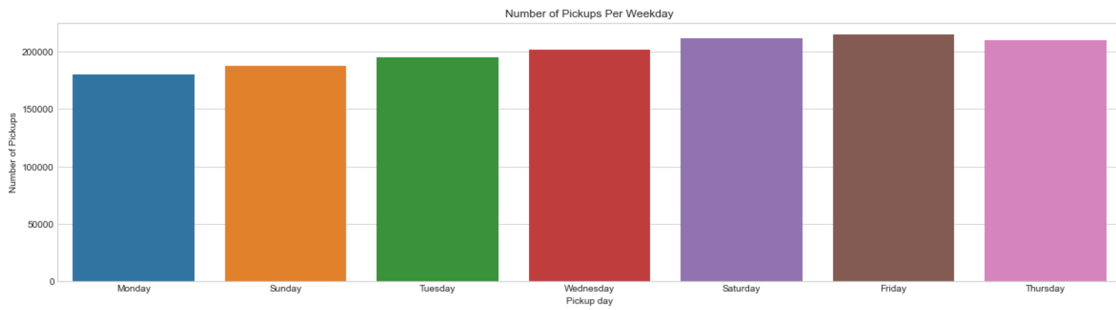
[34]:
```
          id  vendor_id     pickup_datetime    dropoff_datetime  \
0  id2875421          2 2016-03-14 17:24:55 2016-03-14 17:32:30
1  id2377394          1 2016-06-12 00:43:35 2016-06-12 00:54:38
2  id3858529          2 2016-01-19 11:35:24 2016-01-19 12:10:48
3  id3504673          2 2016-04-06 19:32:31 2016-04-06 19:39:40
4  id2181028          2 2016-03-26 13:30:55 2016-03-26 13:38:10

   passenger_count  pickup_longitude  pickup_latitude  dropoff_longitude  \
0                1        -73.982155        40.767937         -73.964630
1                1        -73.980415        40.738564         -73.999481
2                1        -73.979027        40.763939         -74.005333
3                1        -74.010040        40.719971         -74.012268
4                1        -73.973053        40.793209         -73.972923

   dropoff_latitude store_and_fwd_flag  ...  dropoff_day_no  trip_duration_hr  \
0         40.765602                  N  ...               0          0.126389
1         40.731152                  N  ...               6          0.184167
2         40.710087                  N  ...               1          0.590000
3         40.706718                  N  ...               2          0.119167
4         40.782520                  N  ...               5          0.120833

  Average_speed_km/sec Average_speed_km/hr  pickup_hour  dropoff_hour  \
0             0.003291           11.848984           17            17
1             0.002722            9.797504            0             0
2             0.003004           10.815406           11            12
3             0.003461           12.457894           19            19
4             0.002731            9.830418           13            13

   pickup_month  dropoff_month  pickup_timeofday  dropoff_timeofday
0             3              3           Evening            Evening
1             6              6        Late night         Late night
2             1              1           Morning          Afternoon
3             4              4           Evening            Evening
4             3              3         Afternoon          Afternoon

[5 rows x 25 columns]
```

4 3 Time-based

- Assume pickup time unless otherwise specified

4.1 1. Which day of the week is the most popular? Show plots to motivate your answer.

[35]:
```
sns.countplot(x="pickup_day",data=df);
plt.title("Number of Pickups Per Weekday")
plt.xlabel("Pickup day")
plt.ylabel("Number of Pickups")
plt.show()
```

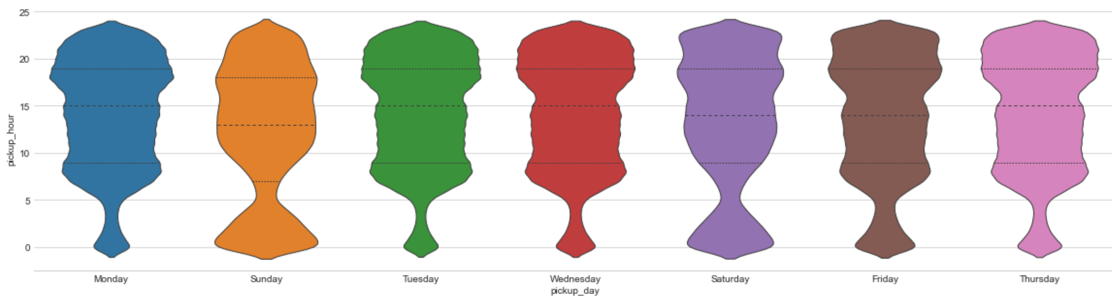


4.2 2. What hour of the day is the most popular on each day? Plot a distributions of the hours and make observations and give possible suggestions for why the data looks like it does

[36]:
```
for i, x in enumerate(df['pickup_day'].unique()):
 print(x, ": Popular Hour: ", int(df[df['pickup_day'] == x]['pickup_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
```

[37]:
```
sns.violinplot(data=df, x="pickup_day", y="pickup_hour",
 split=True, inner="quart", linewidth=1,)
sns.despine(left=True)
plt.show()
```



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

- Use 3.1 and 3.2 observations to explain

4.4 4 Look at how these patterns change on the major holidays (do they change?). Look at the following: St. Patrick's Day, Easter, Memorial Day, Valentine's Day, Martin Luther King Day. Make sure you use the correct dates for these for the relevant year

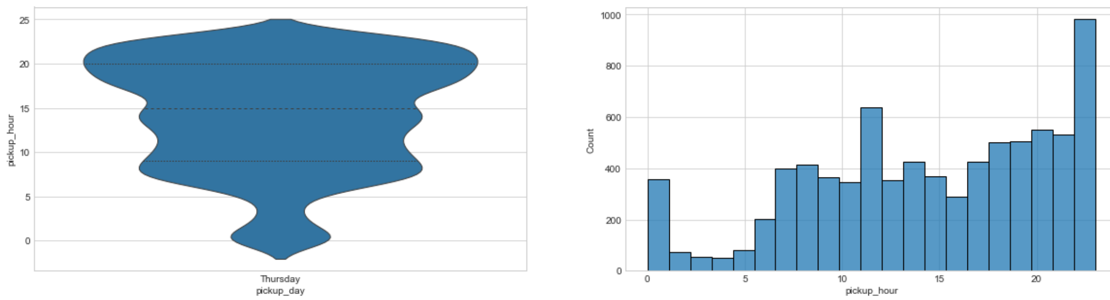
[38]:
```
df['pickup_month'].sort_values().unique()
```

[38]:
```
array([1, 2, 3, 4, 5, 6], dtype=int64)
```

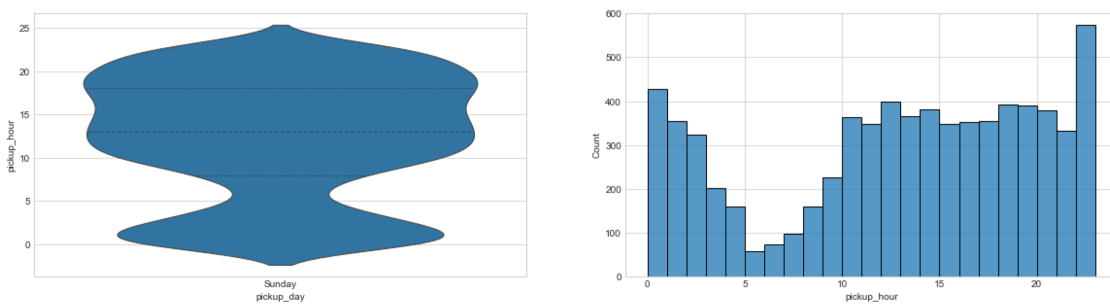
[39]:
```
# Pickdate Column Addition
df['pickup_date'] = df['pickup_datetime'].dt.strftime('%m-%d')
```

[ ]:
```
# St Patrick's Day: Thursday, March 17th 2016
#st_pat = df[df['pickup_date']== '03-17']
# Easter: Sunday, March 27th 2016
#easter = df[df['pickup_date']== '03-27']
# Memorial Day: Monday, May 30 2016
#Memorial = df[df['pickup_date']== '05-30']
# Valentine's Day: Sunday, February 14th 2016
#valentine = df[df['pickup_date']== '02-14']
# Martin Luther King Day: Monday, January 18th
#mlk = df[df['pickup_date']== '01-18']
```

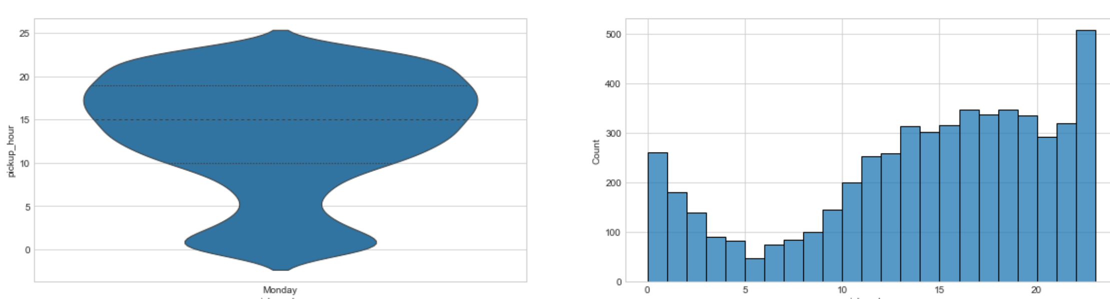
[40]:
```
# St Patrick's Day: Thursday, March 17th 2016
f,axes = plt.subplots(1,2, figsize=(20, 5))
sns.violinplot(data=df[df['pickup_date']== '03-17'], x="pickup_day",␣
↪y="pickup_hour", split=True, inner="quart", linewidth=1, ax= axes[0])
sns.histplot(data=df[df['pickup_date']== '03-17'], x= "pickup_hour", ax =␣
↪axes[1])
plt.show()
```



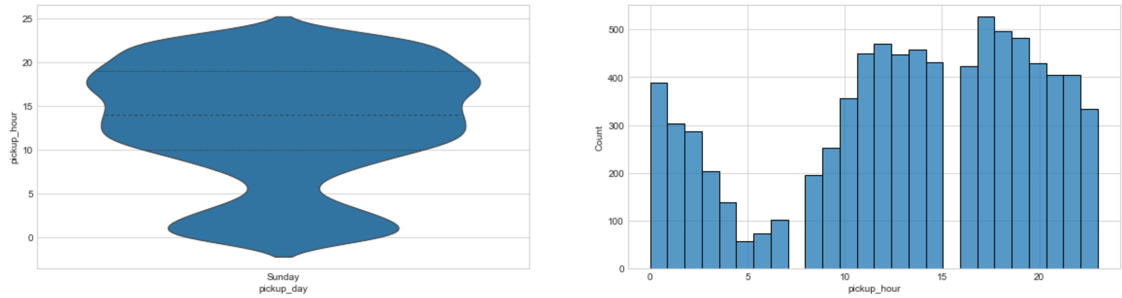
[41]:
```
# Easter: Sunday, March 27th 2016
f,axes = plt.subplots(1,2, figsize=(20, 5))
sns.violinplot(data=df[df['pickup_date']== '03-27'], x="pickup_day",␣
↪y="pickup_hour", split=True, inner="quart", linewidth=1, ax= axes[0])
sns.histplot(data=df[df['pickup_date']== '03-27'], x= "pickup_hour", ax =␣
↪axes[1])
plt.show()
```



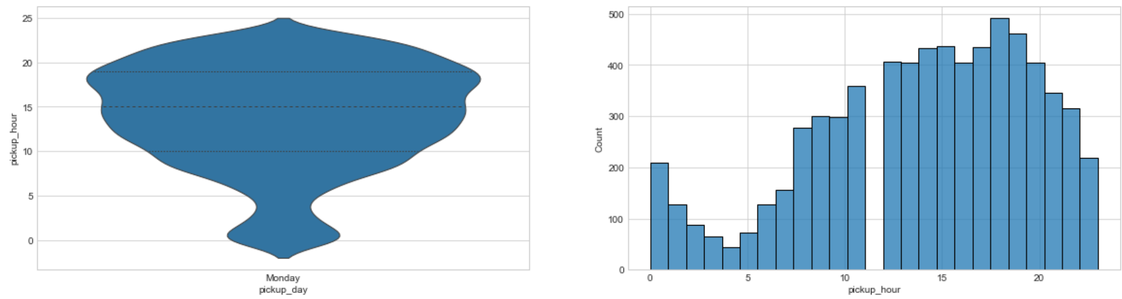
[42]:
```
# Memorial Day: Monday, May 30 2016
f,axes = plt.subplots(1,2, figsize=(20, 5))
sns.violinplot(data=df[df['pickup_date']== '05-30'], x="pickup_day",␣
↪y="pickup_hour", split=True, inner="quart", linewidth=1, ax= axes[0])
sns.histplot(data=df[df['pickup_date']== '05-30'], x= "pickup_hour", ax =␣
↪axes[1])
plt.show()
```



[43]:
```
# Valentine's Day: Sunday, February 14th 2016
f,axes = plt.subplots(1,2, figsize=(20, 5))
sns.violinplot(data=df[df['pickup_date']== '02-14'], x="pickup_day",␣
↪y="pickup_hour", split=True, inner="quart", linewidth=1, ax= axes[0])
sns.histplot(data=df[df['pickup_date']== '02-14'], x= "pickup_hour", ax =␣
↪axes[1])
plt.show()
```

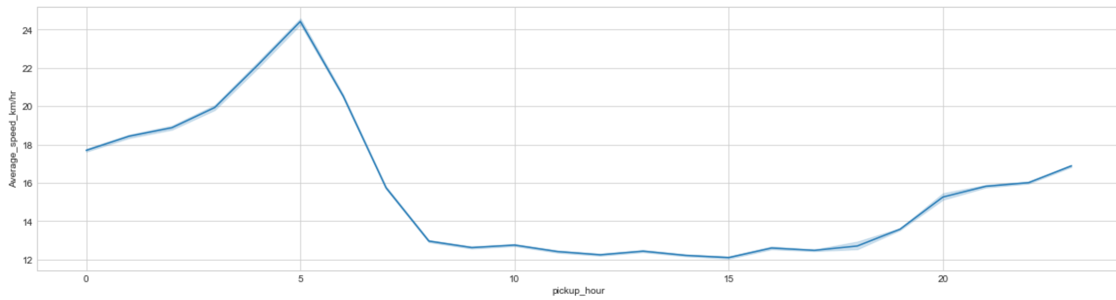


[44]:
```
# Martin Luther King Day: Monday, January 18th
f,axes = plt.subplots(1,2, figsize=(20, 5))
sns.violinplot(data=df[df['pickup_date']== '01-18'], x="pickup_day",␣
↪y="pickup_hour", split=True, inner="quart", linewidth=1, ax= axes[0])
sns.histplot(data=df[df['pickup_date']== '01-18'], x= "pickup_hour", ax =␣
↪axes[1])
plt.show()
```

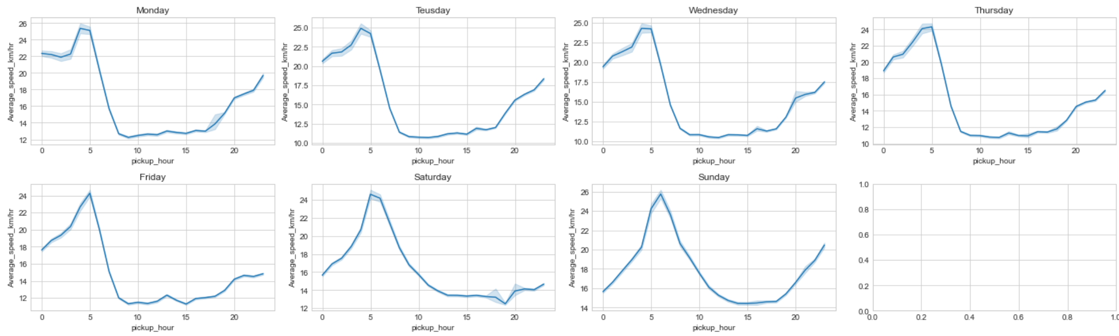


4.5 5 How does the average speed of trips change throughout the day? What time of day are trips fastest? Show plots to motivate your answer

[45]:
```
sns.lineplot(data=df, x="pickup_hour", y="Average_speed_km/hr")
plt.show()
```



[46]:
```
# Observing for each weekday
f, axes = plt.subplots(2,4, figsize=(20, 6))
sns.lineplot(data=df[df['pickup_day'] == 'Monday'], x="pickup_hour",␣
 ↪y="Average_speed_km/hr" , ax=axes[0,0]).set_title('Monday')
sns.lineplot(data=df[df['pickup_day'] == 'Tuesday'], x="pickup_hour",␣
 ↪y="Average_speed_km/hr" , ax=axes[0,1]).set_title('Teusday')
sns.lineplot(data=df[df['pickup_day'] == 'Wednesday'], x="pickup_hour",␣
 ↪y="Average_speed_km/hr" , ax=axes[0,2]).set_title('Wednesday')
sns.lineplot(data=df[df['pickup_day'] == 'Thursday'], x="pickup_hour",␣
 ↪y="Average_speed_km/hr" , ax=axes[0,3]).set_title('Thursday')
sns.lineplot(data=df[df['pickup_day'] == 'Friday'], x="pickup_hour",␣
 ↪y="Average_speed_km/hr" , ax=axes[1,0]).set_title('Friday')
sns.lineplot(data=df[df['pickup_day'] == 'Saturday'], x="pickup_hour",␣
 ↪y="Average_speed_km/hr" , ax=axes[1,1]).set_title('Saturday')
sns.lineplot(data=df[df['pickup_day'] == 'Sunday'], x="pickup_hour",␣
 ↪y="Average_speed_km/hr" , ax=axes[1,2]).set_title('Sunday')
plt.tight_layout()
plt.show()
```



5 4 Location clusters

6 4.1 Heatmaps

6.0.1 weekdays and weekends

[47]:
```
# weekdays and weekends
df_weekdays = df[(df['pickup_day'] != 'Saturday') & (df['pickup_day'] !=
 ↪'Sunday')]
df_weekends = df[(df['pickup_day'] == 'Saturday') | (df['pickup_day'] ==
 ↪'Sunday')]
```

[48]:
```
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
```

[49]:
```
base_map = generateBaseMap()
base_map
```

[49]: <folium.folium.Map at 0x22fbf625a20>

[ ]:
```
#base_map.save('4_basemap.html')
```

[50]:
```
df_weekdays['count'] = 1
df_weekends['count'] = 1
```

6.1 Weekdays

[51]:
```
# weekdays
base_map = generateBaseMap()
HeatMap(data=df_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
```

[51]: <folium.folium.Map at 0x22fb84ee0b0>

[ ]:
```
base_map.save('4_1_1_weekdays_heatmap.html') # weekdays_heatmap
```

Weekends

```
[52]: # Weekends
base_map = generateBaseMap()
HeatMap(data=df_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
```

```
[52]: <folium.folium.Map at 0x22fb81c3b80>
```

```
[ ]: base_map.save('4_1_1_weekends_heatmap.html')
```

6.2 morning and evening

- Morning: from `6:00 am to 11:59 pm`
- Evening: from `4:00 pm to 9:59 pm` From the `time_of_day()`

```
[53]: df['pickup_timeofday'].value_counts()
```

```
[53]: Evening       471345
Morning       342518
Late night    308752
Afternoon     277124
Name: pickup_timeofday, dtype: int64
```

```
[54]: df_morning = df[df['pickup_timeofday'] == "Morning"]
df_evening = df[df['pickup_timeofday'] == "Evening"]
```

```
[55]: df_morning['count'] = 1
df_evening['count'] = 1
```

Mornings

```
[56]: # Mornings
base_map = generateBaseMap()
HeatMap(data=df_morning[['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
```

```
[56]: <folium.folium.Map at 0x22fb81c0ee0>
```

```
[57]: base_map.save('4_1_2_morning_heatmap.html')
```

Evenings

```
[58]: # Evenings
base_map = generateBaseMap()
```

```
HeatMap(data=df_evening[['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
```

[58]: <folium.folium.Map at 0x22fb81c37f0>

```
[ ]: base_map.save('4_1_2_evening_heatmap.html')
```

```
[ ]: # Plotting Morning hours
#df_hour_list = []
#for hour in df_morning.pickup_hour.sort_values().unique():
#  df_hour_list.append(df_morning.loc[df_morning.pickup_hour == hour,
↪['pickup_latitude', 'pickup_longitude', 'count']].groupby(['pickup_latitude',
↪'pickup_longitude']).sum().reset_index().values.tolist())
```

```
[ ]: #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
```

```
[ ]: #base_map.save('morning_heatmap_w_time.html')
```

```
[ ]: # Plotting Evening hours
#df_hour_list_evening = []
#for hour in df_evening['pickup_hour'].sort_values().unique():
# df_hour_list_evening.append(df_evening.loc[df_evening['pickup_hour'] == hour,
↪['pickup_latitude', 'pickup_longitude', 'count']].groupby(['pickup_latitude',
↪'pickup_longitude']).sum().reset_index().values.tolist())
```

```
[ ]: #base_map = generateBaseMap(default_zoom_start=11)
#HeatMapWithTime(df_hour_list_evening, 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
```

7 4.2 Hotspots

- If you were a taxi driver wanting to plan your evenings so that you could get the most trips, you would want to know where the popular areas are.
- Looking at the time periods 23:00 on a Friday evening to 02:00 on a Saturday morning, and between 17:00 and 20:00 on a Thursday, find hotspot locations (areas where there are a large number of trips happening).

- If you were to use k-means, you would define the number of clusters.
- However, here the number of clusters is not at all clear.
- DBSCAN (available in sklearn) determines this for you, and works well on spatial data.
- DBSCAN has two configurable parameters: - 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 (motivate your choice of parameters).
- Using DBSCAN, identify clusters and plot these on a map. How many clusters did you find?

[59]:
```
df['pickup_hour'].sort_values().unique()
```

[59]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23], dtype=int64)
```

[60]:
```
df['pickup_day'].unique()
```

[60]:
```
array(['Monday', 'Sunday', 'Tuesday', 'Wednesday', 'Saturday', 'Friday',
       'Thursday'], dtype=object)
```

Time periods 23:00 on a Friday evening to 02:00 on a Saturday morning

[61]:
```
fri_time = df[(df['pickup_day'] == 'Friday') & (df['pickup_hour'] == 23)]
```

[62]:
```
sat_time = df[(df['pickup_day'] == 'Saturday') & ((df['pickup_hour'] == 0) |␣
↪(df['pickup_hour'] == 1))]
```

[63]:
```
# Filtering Friday night/Saturday midnight (23pm-2am)
fri_sat_morning = pd.concat([fri_time, sat_time])
```

Time periods between 17:00 and 20:00 on a Thursday

[64]:
```
# Thursday (17pm-20pm)
thur_evening = df[(df['pickup_day'] == 'Thursday') & ((df['pickup_hour'] == 17)␣
↪| (df['pickup_hour'] == 18)
| (df['pickup_hour'] == 19) | (df['pickup_hour'] == 20))]
```

Friday 11pm to Saturday 2am

[65]:
```
loc_ini = fri_sat_morning[['pickup_latitude', 'pickup_longitude']].to_numpy()
loc_end = fri_sat_morning[['dropoff_latitude', 'dropoff_longitude']].to_numpy()
locations = np.vstack((loc_ini, loc_end))
```

[66]:
```
def fit_utm_clusterer(locations, min_cluster_size=20, min_samples=15,␣
↪cluster_selection_epsilon = 49):
  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
```

[67]:
```
clusterer = fit_utm_clusterer(locations)
```

[68]:
```
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: 211
```

[69]:
```
def show_cluster_map(cluster_id):
  blue = Color("blue")
  red = Color("red")
  color_range = list(blue.range_to(red, 10))

  #base_map = generateBaseMap() # uncomment to 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_)
    #CircleMarker([point[0],point[1]], radius=1, color = color.hex, tooltip = "{:
↪.2f}".format(scores[i])).add_to(base_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]])
  #base_map.fit_bounds([[min_lat, min_lon], [max_lat, max_lon]])

  return map_
```

[70]:
```
# Interactive cluster map of Friday 11pm to Saturday 2am Hotspots
ii = interact(show_cluster_map, cluster_id = widgets.IntText(min=0, max=␣
↪clusterer.labels_.max(), step=1, value=0))
```

```
interactive(children=(IntText(value=0, description='cluster_id'), Output()),␣
↪_dom_classes=('widget-interact',)...
```

[71]:
```
c1 = show_cluster_map(clusterer.labels_.min())
c1
```

[71]: <folium.folium.Map at 0x22fbf6bf760>

[ ]:
```
c1.save('4_2_Friday_23pm_to_Saturday_2am_hotspots.html')
```

Thursday 17pm-20pm

[72]:
```
loc_ini = thur_evening[['pickup_latitude', 'pickup_longitude']].to_numpy()
loc_end = thur_evening[['dropoff_latitude', 'dropoff_longitude']].to_numpy()
locations = np.vstack((loc_ini, loc_end))
```

[73]:
```
def fit_utm_clusterer(locations,min_cluster_size=20, min_samples=15,␣
↪cluster_selection_epsilon=49):
  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
```

[74]:
```
clusterer = fit_utm_clusterer(locations)
```

[75]:
```
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: 118
```

[76]:
```
def show_cluster_map(cluster_id):
  blue = Color("blue")
  red = Color("red")
  color_range = list(blue.range_to(red, 10))

  #base_map = generateBaseMap() # uncomment to 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_)
      #CircleMarker([point[0],point[1]], radius=1, color = color.hex, tooltip = "{:
↪.2f}".format(scores[i])).add_to(base_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]])
    #base_map.fit_bounds([[min_lat, min_lon], [max_lat, max_lon]])

    return map_
```

[77]:
```
ii = interact(show_cluster_map, cluster_id = widgets.IntText(min=0, max=␣
↪clusterer.labels_.max(), step=1, value=0))
```

```
interactive(children=(IntText(value=0, description='cluster_id'), Output()),␣
↪_dom_classes=('widget-interact',)...
```

8 Airports

- Find out how long it takes, on average, to travel to JFK airport from the Empire State Building.
- Produce a plot showing the travel time by time of day.
- How does this compare with Newark Airport? Assume the following coordinates for the centre point of the locations (long, lat):
- `JFK Airport`: (40.647929, -73.777813): `Terminal 5, Queens, NY 11430, USA`
- `Empire State Building`: (40.756724, -73.983806): `111 W 44th St, New York, NY 10036, USA`
- `Newark Airport`: (40.689442, -74.173242): `E.W.R. (EWR), 3 Brewster Rd, Newark, NJ 07114, USA`
- According to Rome2Rio it takes approximately 22 minutes (1320 seconds) (27.5 km) to travel to JFK airport from the Empire State Building by a taxi drive
- According to Rome2Rio it takes approximately 19 minutes (1140 seconds) (26.3km) to travel to JFK airport from the Empire State Building by a taxi drive

[78]:
```
# [lat, lon]:
jfk = [40.647929, -73.777813]
esb = [40.756724, -73.983806]
nwa = [40.689442, -74.173242]
```

```
# The nyc data is [lon, lat]
```

[79]:
```
# Pickup distances from Empire State Building (comeback): How far each pickup␣
↪point is from the ESB centre point
df['esb_distance'] = haversine_np(esb[1], esb[0], df['pickup_longitude'],␣
↪df['pickup_latitude'])

# Dropoff distance from JFK Airport: How far each dropoff point is from the JFK␣
↪centre point
df['jfk_distance'] = haversine_np(jfk[1], jfk[0], df['dropoff_longitude'],␣
↪df['dropoff_latitude'])

# Dropoff distance from Newark Airport: How far each dropoff point is from the␣
↪NWA centre point
df['nwa_distance'] = haversine_np(nwa[1], nwa[0], df['dropoff_longitude'],␣
↪df['dropoff_latitude'])
```

[80]:
```
print("Empire State Building surrounding distances (km) statistical summary:")
df['esb_distance'].describe().apply(lambda x: '%.5f' % x)
```

```
Empire State Building surrounding distances (km) statistical summary:
```

[80]:
```
count    1399739.00000
mean           3.12065
std            6.44135
min            0.00270
25%            1.20000
50%            2.31658
75%            3.66262
max         4102.10217
Name: esb_distance, dtype: object
```

[81]:
```
print("JFK Airport surrounding distances (km) statistical summary:")
df['jfk_distance'].describe().apply(lambda x: '%.5f' % x)
```

```
JFK Airport surrounding distances (km) statistical summary:
```

[81]:
```
count    1399739.00000
mean          20.48571
std            6.17223
min            0.10997
25%           20.10871
50%           20.78483
75%           21.52520
max         4121.43129
Name: jfk_distance, dtype: object
```

[82]:
```python
print("Newark Airport Airport surrounding distances (km) statistical summary:")
df['nwa_distance'].describe().apply(lambda x: '%.5f' % x)
```

```
Newark Airport Airport surrounding distances (km) statistical summary:
```

[82]:
```
count    1399739.00000
mean          18.49023
std            6.44379
min            0.37322
25%           16.43162
50%           17.95545
75%           19.93326
max         4087.83121
Name: nwa_distance, dtype: object
```

Determining if a GPS coordinate is at location (1 km radius)

Points within 1km Radius of Empire State Building, JFK Airport and Newark Airport will be filtered out

[83]:
```python
# Checking Points within the Empire State Building radius: Use 1km radius
conditions_esb = [
    (df['esb_distance'] <= 1),
    (df['esb_distance'] > 1)
]

values_esb = ['yes', 'no']

df['Within_esb_radius'] = np.select(conditions_esb, values_esb, default='other')

# Checking Points within the JFK Airport radius: Use 1km radius
conditions_jfk = [
    (df['jfk_distance'] <= 1),
    (df['jfk_distance'] > 1)
]

values_jfk = ['yes', 'no']

df['Within_jfk_radius'] = np.select(conditions_jfk, values_jfk, default='other')

# Checking Points within the Newark Airport radius: Use 1km radius
conditions_nwa = [
    (df['nwa_distance'] <= 1),
    (df['nwa_distance'] > 1)
]

values_nwa = ['yes', 'no']
```

```
df['Within_nwa_radius'] = np.select(conditions_nwa, values_nwa, default='other')
```

[84]:
```
print(df['Within_esb_radius'].value_counts())
print(df['Within_jfk_radius'].value_counts())
print(df['Within_nwa_radius'].value_counts())
```

```
no     1154183
yes     245556
Name: Within_esb_radius, dtype: int64
no     1392672
yes       7067
Name: Within_jfk_radius, dtype: int64
no     1397330
yes       2409
Name: Within_nwa_radius, dtype: int64
```

[85]:
```
from_esb_to_jfk_df = df[(df['Within_esb_radius'] == 'yes') &␣
 ↪(df['Within_jfk_radius'] == 'yes')]
```

How long it takes, on average, to travel to JFK/Newark airport from the Empire State Building

[86]:
```
print("Average Travel Time from Empire State Building to JFK Airport:",␣
 ↪from_esb_to_jfk_df['trip_duration'].mean(), "seconds")
print("Average Travel Time from Empire State Building to JFK Airport:",␣
 ↪(from_esb_to_jfk_df['trip_duration'].mean())/60, "minutes")
```

```
Average Travel Time from Empire State Building to JFK Airport: 2876.207991242474
seconds
Average Travel Time from Empire State Building to JFK Airport: 47.93679985404123
minutes
```

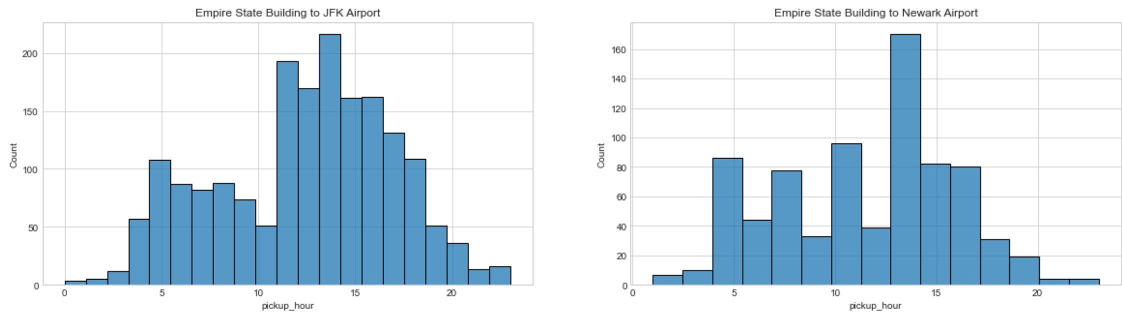
[87]:
```
from_esb_to_nwa_df = df[(df['Within_esb_radius'] == 'yes') &␣
 ↪(df['Within_nwa_radius'] == 'yes')]
```

[88]:
```
print("Average Travel Time from Empire State Building to Newark Airport:",␣
 ↪from_esb_to_nwa_df['trip_duration'].mean(), "seconds")
print("Average Travel Time from Empire State Building to Newark Airport:",␣
 ↪(from_esb_to_nwa_df['trip_duration'].mean())/60, "minutes")
```

```
Average Travel Time from Empire State Building to Newark Airport:
2277.4738186462323 seconds
Average Travel Time from Empire State Building to Newark Airport:
37.95789697743721 minutes
```

Travel time by time of day From the above outputs, much time is taken on average to travel to JFK Airport from the Empire State Building as compared to Newark Airport

[89]:
```
f,axes = plt.subplots(1,2, figsize=(20, 5))
sns.histplot(data=from_esb_to_jfk_df, x= "pickup_hour", ax = axes[0]).
 ↪set_title('Empire State Building to JFK Airport')
sns.histplot(data=from_esb_to_nwa_df, x= "pickup_hour", ax = axes[1]).
 ↪set_title('Empire State Building to Newark Airport')
plt.show()
```



[90]:
```
# esb to jfk: Red line
# esb to nwa: Blue line
base_map = generateBaseMap(default_zoom_start=11)
folium.PolyLine(locations=[esb, jfk], color= 'red').add_child(folium.
 ↪Popup('Empire State Building to JFK Airport')).add_to(base_map)
folium.PolyLine(locations=[esb, nwa], color= 'blue').add_child(folium.
 ↪Popup('Empire State Building to Newark Airport')).add_to(base_map)
base_map
```

[90]:
```
<folium.folium.Map at 0x22fb0b3d750>
```

9 Boroughs

9.0.1 1 Using this shapefile find the neighbourhoods for the trip start and end locations (try geopandas, shapely, or fiona, for example)

[95]:
```
data = gpd.read_file('C:/Users/Mfund/Downloads/2010 Neighborhood Tabulation␣
 ↪Areas (NTAs)/geo_export_3871484a-e150-46de-a8d7-525e6709e130.shp')
SHAPE_RESTORE_SHX =True
```

[96]:
```
data.head()
```

[96]:
```
   boro_code boro_name county_fip ntacode                ntaname    shape_area  \
0        4.0    Queens        081    QN51            Murray Hill  5.248828e+07
1        4.0    Queens        081    QN27          East Elmhurst  1.972685e+07
2        4.0    Queens        081    QN41  Fresh Meadows-Utopia  2.777485e+07
3        4.0    Queens        081    QN08             St. Albans  7.741275e+07
```

```
4        3.0  Brooklyn        047    BK69          Clinton Hill  2.052820e+07

     shape_leng                                          geometry
0  33266.904856  POLYGON ((-73.80379 40.77561, -73.80099 40.775...
1  19816.711758  POLYGON ((-73.86110 40.76366, -73.85993 40.762...
2  22106.431272  POLYGON ((-73.77758 40.73019, -73.77849 40.729...
3  45401.316786  POLYGON ((-73.75205 40.70523, -73.75174 40.704...
4  23971.466236  POLYGON ((-73.95337 40.68064, -73.95328 40.680...
```

[97]:
```python
data.boro_name.unique()# boroughs
```

[97]:
```
array(['Queens', 'Brooklyn', 'Bronx', 'Staten Island', 'Manhattan'],
      dtype=object)
```

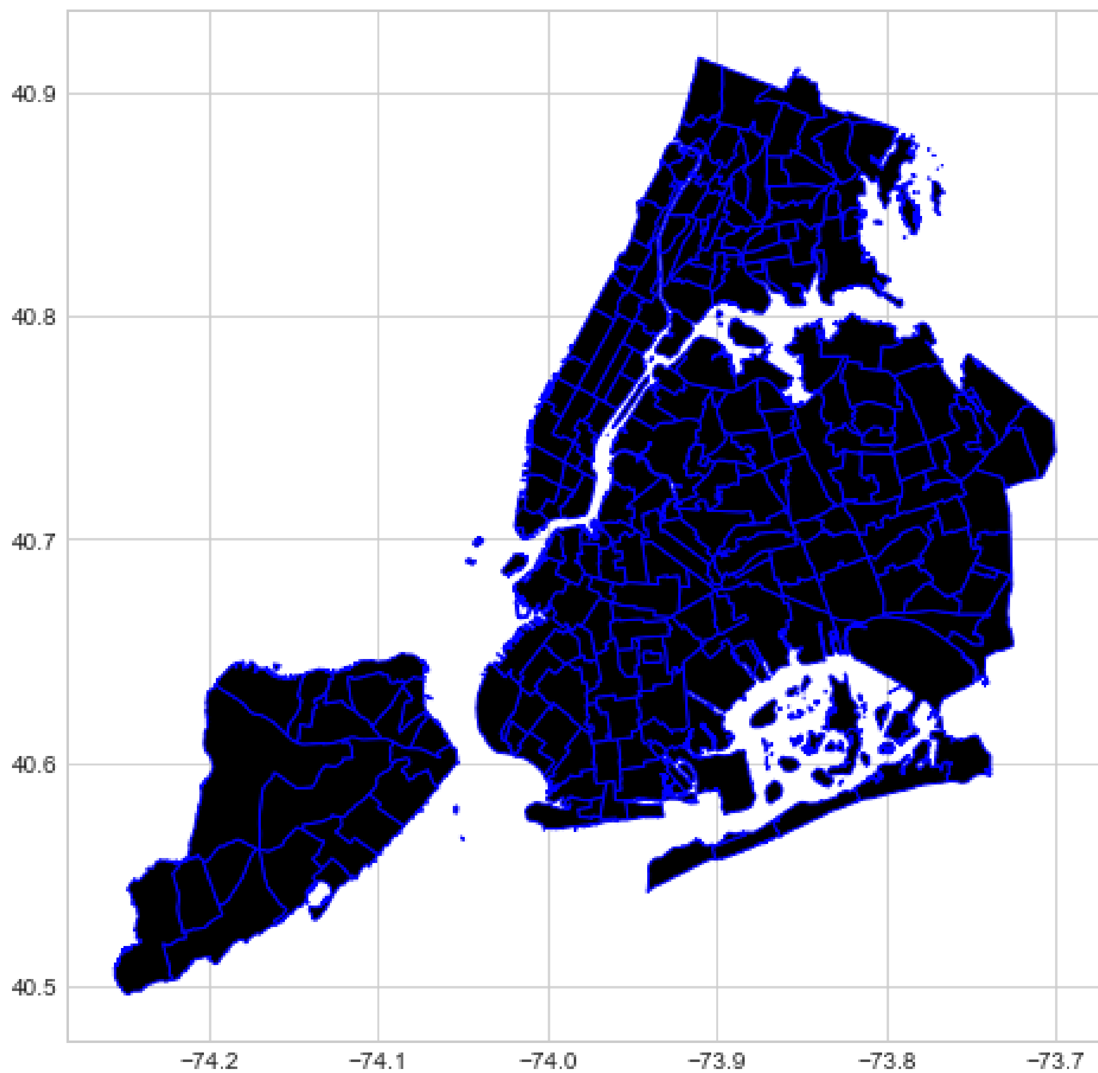
[98]:
```python
#neighbourhoods
data.ntaname.unique()
```

[98]:
```
array(['Murray Hill', 'East Elmhurst', 'Fresh Meadows-Utopia',
       'St. Albans', 'Clinton Hill', 'Gravesend', 'Ocean Parkway South',
       'Van Cortlandt Village', 'South Ozone Park', 'Windsor Terrace',
       'Canarsie', 'Rossville-Woodrow', 'Upper West Side', 'Norwood',
       'Bedford Park-Fordham North', 'Mount Hope', 'North Corona',
       'West Brighton', 'Rego Park', 'Whitestone', 'Ozone Park',
       'Springfield Gardens South-Brookville', 'Fort Greene',
       'Starrett City', 'Gramercy', 'Ocean Hill',
       'Pomonok-Flushing Heights-Hillcrest', 'East Flushing',
       'Kingsbridge Heights', 'University Heights-Morris Heights',
       'Williamsburg', 'Madison', 'South Jamaica', 'Erasmus',
       'Rikers Island', 'Hollis', 'Rosedale', 'Richmond Hill',
       'Auburndale', 'Jamaica Estates-Holliswood', 'Jamaica', 'Corona',
       'Astoria', 'Bushwick North', 'Ridgewood', 'Elmhurst-Maspeth',
       'Stuyvesant Heights', 'East Tremont', 'East Williamsburg',
       'Midtown-Midtown South', 'Kensington-Ocean Parkway',
       'Fordham South', 'Hudson Yards-Chelsea-Flatiron-Union Square',
       'Clinton', 'Yorkville', 'Marble Hill-Inwood', 'Bedford',
       'Bushwick South', 'Stapleton-Rosebank',
       'Douglas Manor-Douglaston-Little Neck', 'Morningside Heights',
       'Central Harlem South', 'Cambria Heights', 'Bayside-Bayside Hills',
       'Bellerose', 'Glen Oaks-Floral Park-New Hyde Park',
       'Jackson Heights', 'Chinatown', 'Old Astoria',
       'Crown Heights North', 'Rugby-Remsen Village', 'Parkchester',
       'SoHo-TriBeCa-Civic Center-Little Italy',
       'Battery Park City-Lower Manhattan', 'Lower East Side',
       'Queensbridge-Ravenswood-Long Island City',
       'Old Town-Dongan Hills-South Beach',
       'Grasmere-Arrochar-Ft. Wadsworth', 'College Point', 'Airport',
       'East Harlem South',
```

```
'Georgetown-Marine Park-Bergen Beach-Mill Basin', 'Bath Beach',
'Bensonhurst West', 'Bensonhurst East',
'park-cemetery-etc-Staten Island', 'Manhattanville', 'Bronxdale',
'Lincoln Square', 'Prospect Heights', 'Dyker Heights', 'Bay Ridge',
'Carroll Gardens-Columbia Street-Red Hook', 'Homecrest',
'Seagate-Coney Island', 'Westerleigh',
'West New Brighton-New Brighton-St. George', 'East New York',
'Lindenwood-Howard Beach', 'Greenpoint', 'Woodhaven',
'Turtle Bay-East Midtown', 'Westchester-Unionport',
'Todt Hill-Emerson Hill-Heartland Village-Lighthouse Hill',
'Upper East Side-Carnegie Hill',
'Central Harlem North-Polo Grounds', 'Hamilton Heights',
'Cypress Hills-City Line', 'East Village', 'West Village',
'Middle Village', 'Elmhurst', 'North Side-South Side',
'Washington Heights South', 'park-cemetery-etc-Manhattan',
'West Farms-Bronx River', 'Queensboro Hill', 'Borough Park',
'Washington Heights North', 'Flushing', 'Allerton-Pelham Gardens',
'Oakwood-Oakwood Beach', 'Great Kills',
"Annadale-Huguenot-Prince's Bay-Eltingville",
'Charleston-Richmond Valley-Tottenville', 'Far Rockaway-Bayswater',
'New Dorp-Midland Beach', 'Grymes Hill-Clifton-Fox Hills',
'New Brighton-Silver Lake', 'Brooklyn Heights-Cobble Hill',
'Arden Heights',
'Breezy Point-Belle Harbor-Rockaway Park-Broad Channel',
'Hammels-Arverne-Edgemere', 'Murray Hill-Kips Bay',
'Stuyvesant Town-Cooper Village', 'Laurelton',
'Pelham Bay-Country Club-City Island',
'Schuylerville-Throgs Neck-Edgewater Park',
'Hunters Point-Sunnyside-West Maspeth', 'Woodside', 'Maspeth',
'Ft. Totten-Bay Terrace-Clearview',
'Prospect Lefferts Gardens-Wingate', 'Crown Heights South',
'Steinway', 'Longwood', 'Springfield Gardens North',
'Baisley Park', 'Forest Hills', 'Port Richmond', 'Queens Village',
'Oakland Gardens', 'Van Nest-Morris Park-Westchester Square',
'Pelham Parkway', 'Sunset Park West', 'Soundview-Bruckner',
'Crotona Park East', 'Kew Gardens Hills',
'North Riverdale-Fieldston-Riverdale',
'Spuyten Duyvil-Kingsbridge', 'Glendale',
'Lenox Hill-Roosevelt Island', 'Morrisania-Melrose',
'Woodlawn-Wakefield', 'Flatbush', 'Midwood',
'Eastchester-Edenwald-Baychester', 'Co-op City', 'Highbridge',
'Claremont-Bathgate', 'Belmont', 'Flatlands',
'East Flatbush-Farragut', 'West Concourse',
'Sheepshead Bay-Gerritsen Beach-Manhattan Beach', 'Brighton Beach',
'Sunset Park East', 'park-cemetery-etc-Brooklyn', 'Brownsville',
'East New York (Pennsylvania Ave)', 'Kew Gardens',
'Briarwood-Jamaica Hills', 'park-cemetery-etc-Queens',
```

```
'Park Slope-Gowanus',
'DUMBO-Vinegar Hill-Downtown Brooklyn-Boerum Hill',
'Williamsbridge-Olinville', 'park-cemetery-etc-Bronx',
'New Springville-Bloomfield-Travis',
"Mariner's Harbor-Arlington-Port Ivory-Graniteville",
'Soundview-Castle Hill-Clason Point-Harding Park', 'Hunts Point',
'Mott Haven-Port Morris', 'East Harlem North',
'East Concourse-Concourse Village',
'Melrose South-Mott Haven North'], dtype=object)
```

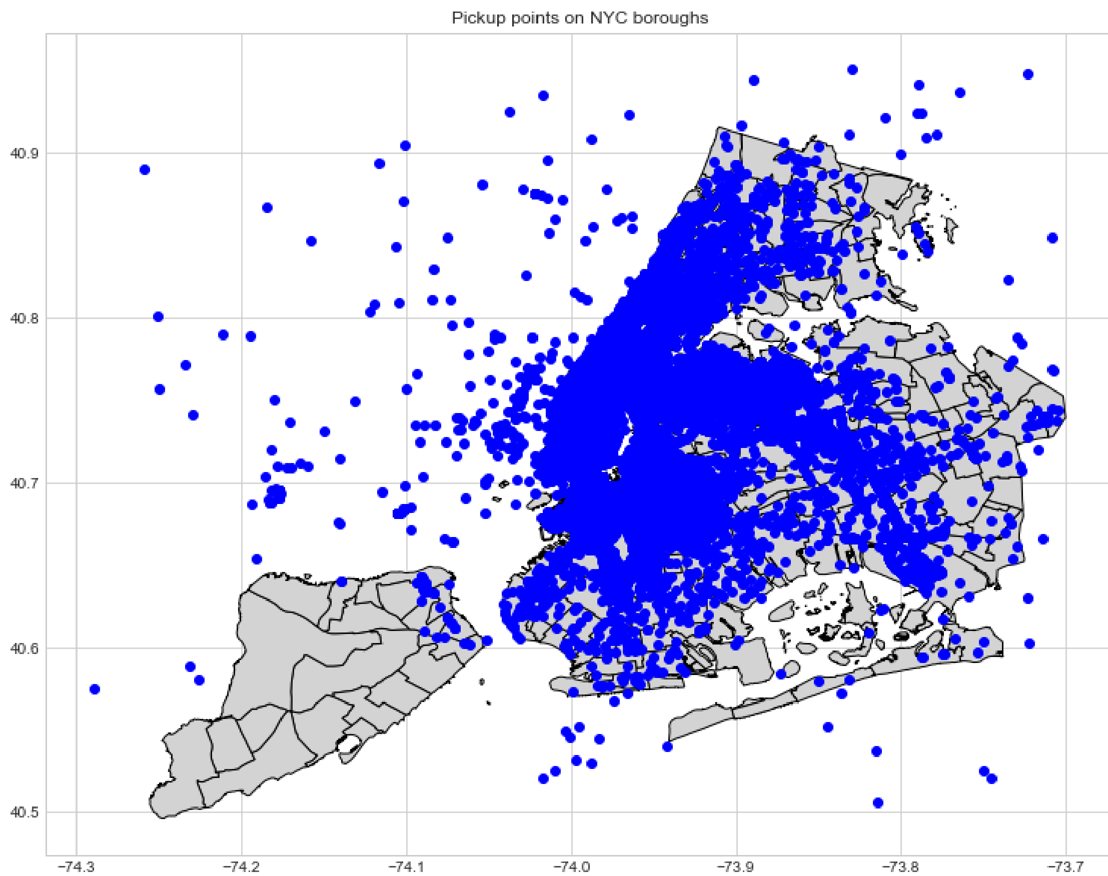
[99]:
```
fig,ax=plt.subplots(figsize=(16,8))
data.plot(ax=ax,color='black', edgecolor='blue')
plt.show()
```



[100]:
```python
data_pickup = gpd.GeoDataFrame(
    df, geometry=gpd.points_from_xy(df.pickup_longitude, df.pickup_latitude)
)
```

[101]:
```python
# Eliminating outliers that falls outside boroughs
data_pickup = data_pickup[(data_pickup['pickup_longitude']>-74.3) &␣
 ↪(data_pickup['pickup_longitude']< -73.7)]
data_pickup = data_pickup[(data_pickup['pickup_latitude']<40.97) &␣
 ↪(data_pickup['pickup_latitude']> 40.5)]
```

[102]:
```python
# Neighbourhoods for trip start locations
fig, ax = plt.subplots(figsize=(16,10))
data.plot(ax=ax, color='lightgrey', edgecolor='black')
data_pickup.plot(ax=ax, color='blue')
ax.set_title("Pickup points on NYC boroughs")
plt.show()
```

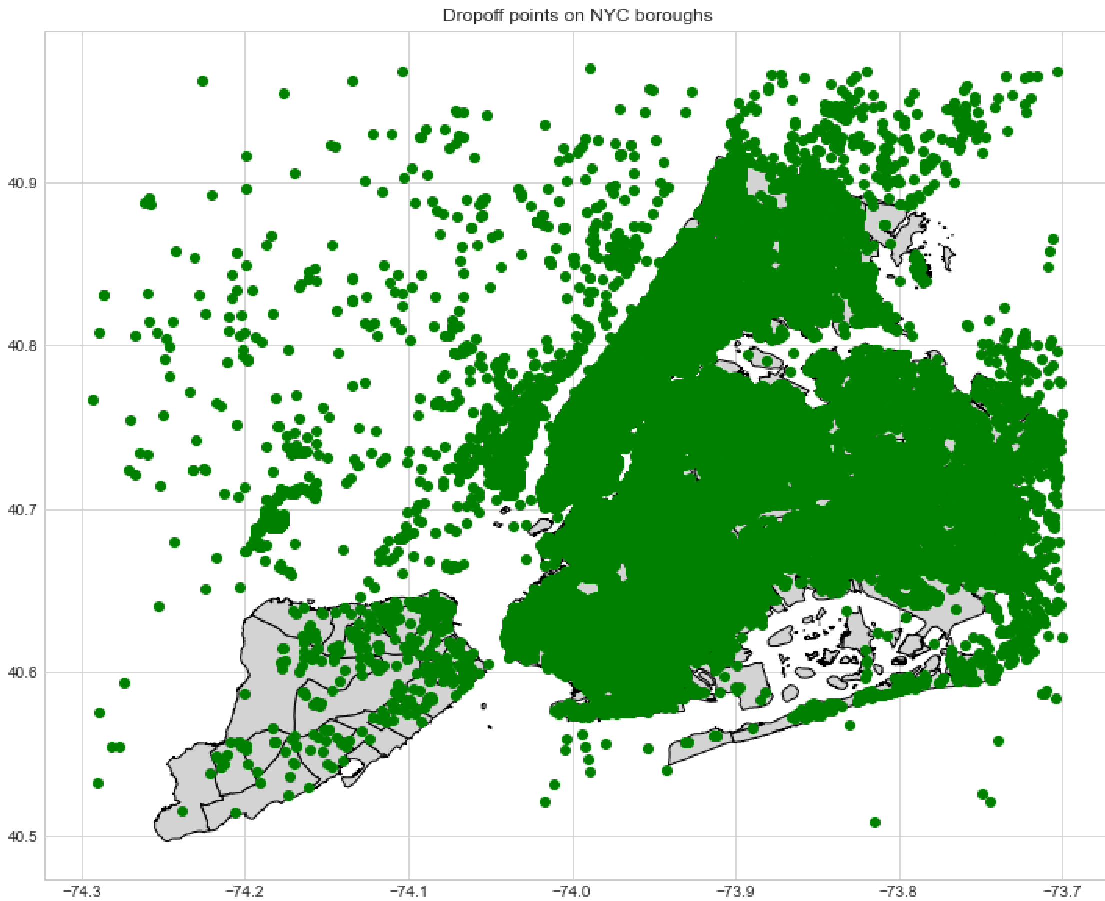


[103]:
```python
data_dropoff = gpd.GeoDataFrame(
   df, geometry=gpd.points_from_xy(df.dropoff_longitude, df.dropoff_latitude)
```

```
)
```

[104]:
```
#remove outliers that falls outside boroughs
data_dropoff = data_dropoff[(data_dropoff['dropoff_longitude']>-74.3) &
↪(data_dropoff['dropoff_longitude']< -73.7)]
data_dropoff = data_dropoff[(data_dropoff['dropoff_latitude']<40.97) &
↪(data_dropoff['dropoff_latitude']> 40.5)]
```

[105]:
```
#neighbourhoods for trip stary locations
fig, ax = plt.subplots(figsize=(16,10))
data.plot(ax=ax, color='lightgrey', edgecolor='black')
data_dropoff.plot(ax=ax, color='green')
ax.set_title("Dropoff points on NYC boroughs")
plt.show()
```



9.0.2 2. Plot a chloropleth of all pickups and all dropoffs in NYC. What do you notice about the difference in distribution

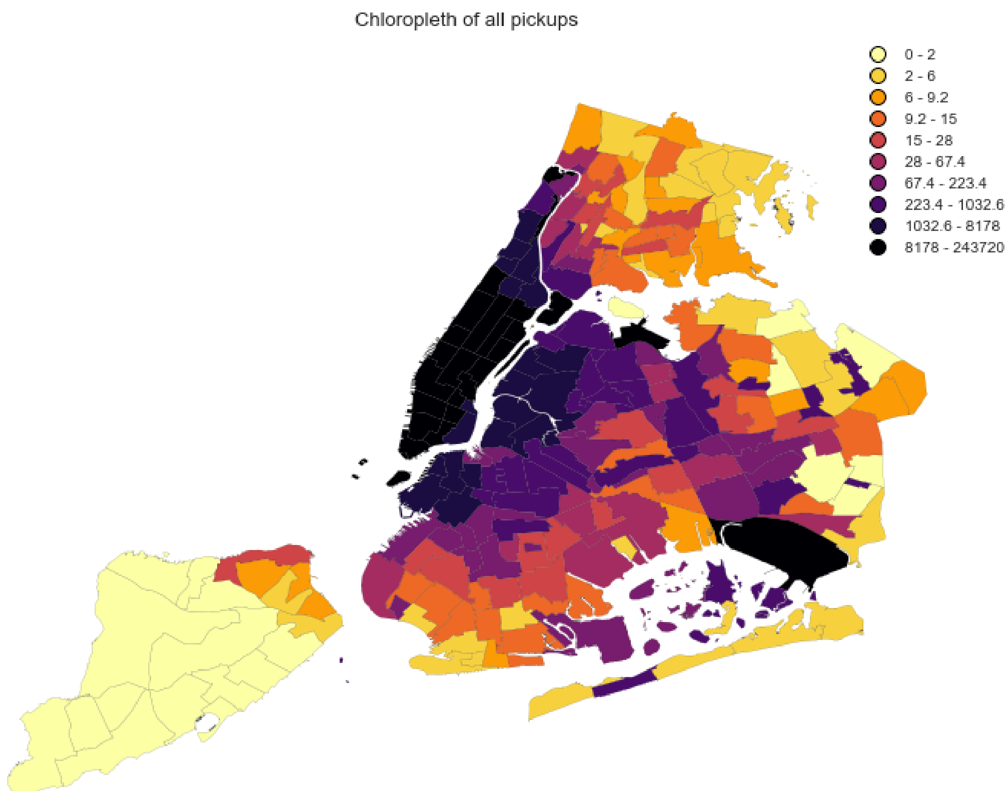
[106]:
```
data_pickup = gpd.GeoDataFrame(
    df[['pickup_latitude','pickup_longitude']], geometry = gpd.points_from_xy(df.
 ↪pickup_longitude, df.pickup_latitude)
)
```

[107]:
```
data_pickup['count'] =1
```

[ ]:
```
#!pip uninstall rtree
#!sudo apt install libspatialindex-dev
#!pip install rtree
```

[108]:
```
# gdf_pickup creating sum column in nyc shapefile from pickup locations sum =␣
 ↪pickup
#before loading this cell run this !pip install rtree, pygeos  in a separate cell
sum_hex = []
spatial_index = data_pickup.sindex
for index, row in data.iterrows():
 polygon = row.geometry
 possible_matches_index = list(spatial_index.intersection(polygon.bounds))
 possible_matches = data_pickup.iloc[possible_matches_index]
 precise_matches = possible_matches[possible_matches.within(polygon)]
 sum_hex.append(sum(precise_matches['count']))
data['sum'] = sum_hex
```

[109]:
```
fig, ax = plt.subplots(1, 1, figsize=(16, 10))
# Set up the color sheme:
scheme = mc.Quantiles(data['sum'], k=10)
# Map
gplt.choropleth(data,
 hue="sum",
 linewidth=.1,
 scheme=scheme, cmap='inferno_r',
 legend=True,
 edgecolor='black',
 ax=ax
);
ax.set_title('Chloropleth of all pickups', fontsize=13);
```



[110]:
```
data_dropoff = gpd.GeoDataFrame(
 df[['dropoff_latitude', 'dropoff_longitude']], geometry=gpd.points_from_xy(df.
↪dropoff_longitude, df.dropoff_latitude))
```

[111]:
```
data_dropoff['count'] = 1
```

[ ]:
```
# gdf_dropoff creating sum2 column in nyc shapefile from pickup locations sum2 =␣
↪dropoff
sum_hex2 = []
spatial_index2 = data_dropoff.sindex
for index, row in data.iterrows():
 polygon2 = row.geometry
 possible_matches_index2 = list(spatial_index2.intersection(polygon2.bounds))
 possible_matches2 = data_dropoff.iloc[possible_matches_index2]
 precise_matches2 = possible_matches2[possible_matches2.within(polygon2)]
 sum_hex2.append(sum(precise_matches2['count']))
data['sum2'] = sum_hex2
```

[ ]:
```
fig, ax = plt.subplots(1, 1, figsize=(16, 12))
# Set up the color sheme:
scheme2 = mc.Quantiles(data['sum2'], k=10)
# Map
gplt.choropleth(data,
 hue="sum2",
 linewidth=.1,
 scheme=scheme2, cmap='inferno_r',
 legend=True,
 edgecolor='black',
 ax=ax
);
ax.set_title('Chloropleth of all dropoffs', fontsize=13);
```

9.0.3 3. Which boroughs have the most incoming trips and the most outgoing trips?

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

[ ]:
```
#most incoming/outcoming
data['sum2'].groupby(data['boro_name']).sum().reset_index().values
```

9.0.4 4 Which neighbourhood(s) is/are the quietest at night, between midnight and 5AM? (Not everyonewants to party)

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

[ ]:
```
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])
```

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

[ ]:
```
sum_hex3 = []
spatial_index3 = gdf_night.sindex
for index, row in data.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']))
data['sum3'] = sum_hex3
```

[ ]:
```
fig, ax = plt.subplots(1, 1, figsize=(16, 8))
# Set up the color sheme:
scheme3 = mc.Quantiles(data['sum3'], k=10)
# Map
gplt.choropleth(data,
 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);
```

[ ]:
```
data['sum3'].groupby(data['boro_name']).sum().reset_index().values
```

9.0.5 5 Which neighbourhood(s) is/are the busiest at night, between midnight and 5AM? (Some people party, well, only Rod actually)

[ ]:
```
data['sum3'].groupby(data['boro_name']).sum().reset_index().values
```