Before you turn in the homework, make sure everything runs as expected. To do so, select **Kernel→ Restart & Run All** in the toolbar above. Remember to submit both on **DataHub** and **Gradescope**.

Please fill in your name and include a list of your collaborators below.

In [1]:
```
NAME = "Junsheng Pei"
COLLABORATORS = ""
```

Project 2: NYC Taxi Rides

Part 2: EDA, Visualization, Feature Engineering

In this part, we will conduct EDA on the NYC Taxi dataset that we cleaned and train/validation split in part 1. We will also guide you through the engineering of some features that hopefully will help our model to accurately understand the data.

Imports

Let us start by loading the Python libraries and custom tools we will use in this part.

In [2]:
```
import pandas as pd
import numpy as np
import random
import matplotlib.pyplot as plt
import seaborn as sns
import os
from pathlib import Path

plt.rcParams['figure.figsize'] = (12, 9)
plt.rcParams['font.size'] = 12

sns.set(style="whitegrid", palette="muted")
%matplotlib inline
```

Loading & Formatting data

The following code loads the data into a pandas DataFrame.

In [3]:
```
# Run this cell to load the data.
data_file = Path("data/part1", "cleaned_data.hdf")
train_df = pd.read_hdf(data_file, "train")
```

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

Out[4]:

	record_id	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	pickup_longitude	pickup_latitude	RatecodeID	store_and_fwd_flag	...	dropoff_latitude	payment_type	fare_amount	ex
16434	8614300	2	2016-01-21 17:37:12	2016-01-21 18:37:56	2	10.89	-73.863403	40.769432	1	N	...	40.688210	1	41.5	
21929	7230200	2	2016-01-29 23:22:26	2016-01-29 23:31:23	2	1.00	-74.008087	40.739365	1	N	...	40.729271	1	7.0	
3370	9830300	2	2016-01-05 18:50:16	2016-01-05 18:56:00	2	0.56	-73.972923	40.755650	1	N	...	40.758469	1	5.0	
21975	7251500	2	2016-01-30 00:14:34	2016-01-30 00:47:13	1	6.65	-73.992027	40.718662	1	N	...	40.661118	1	25.5	
13758	6168000	1	2016-01-18 13:25:24	2016-01-18 13:38:51	1	2.10	-73.953125	40.784538	1	N	...	40.760792	2	11.5	

5 rows × 21 columns

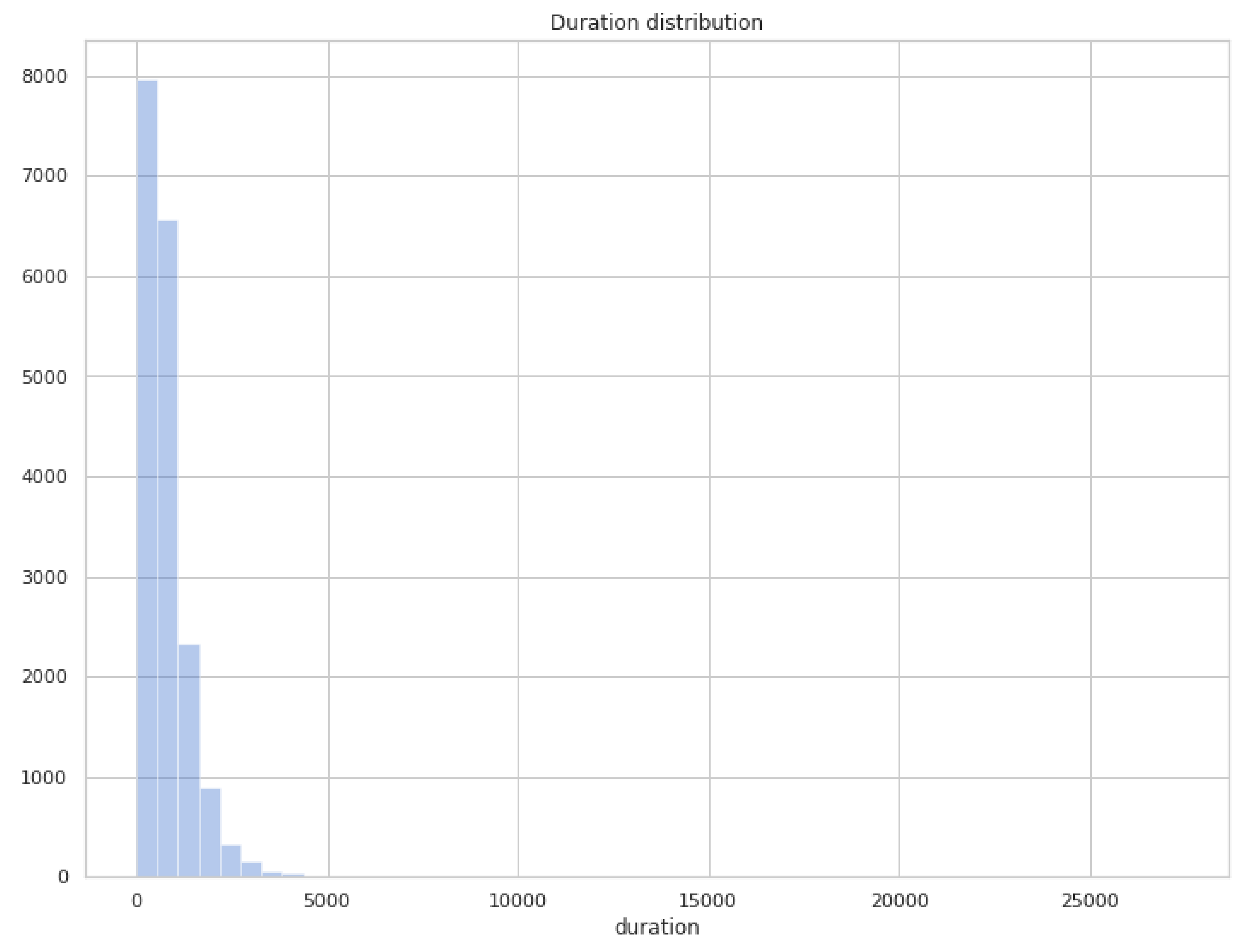
1: Data Overview

As a reminder, the raw taxi data contains the following columns:

- `recordID` : primary key of this database
- `VendorID` : a code indicating the provider associated with the trip record
- `passenger_count` : the number of passengers in the vehicle (driver entered value)
- `trip_distance` : trip distance
- `tpep_dropoff_datetime` : date and time when the meter was engaged
- `tpep_pickup_datetime` : date and time when the meter was disengaged
- `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
- `duration` : duration of the trip in seconds
- `payment_type` : the payment type
- `fare_amount` : the time-and-distance fare calculated by the meter
- `extra` : miscellaneous extras and surcharges
- `mta_tax` : MTA tax that is automatically triggered based on the metered rate in use
- `tip_amount` : the amount of credit card tips, cash tips are not included
- `tolls_amount` : amount paid for tolls
- `improvement_surcharge` : fixed fee
- `total_amount` : total amount paid by passengers, cash tips are not included

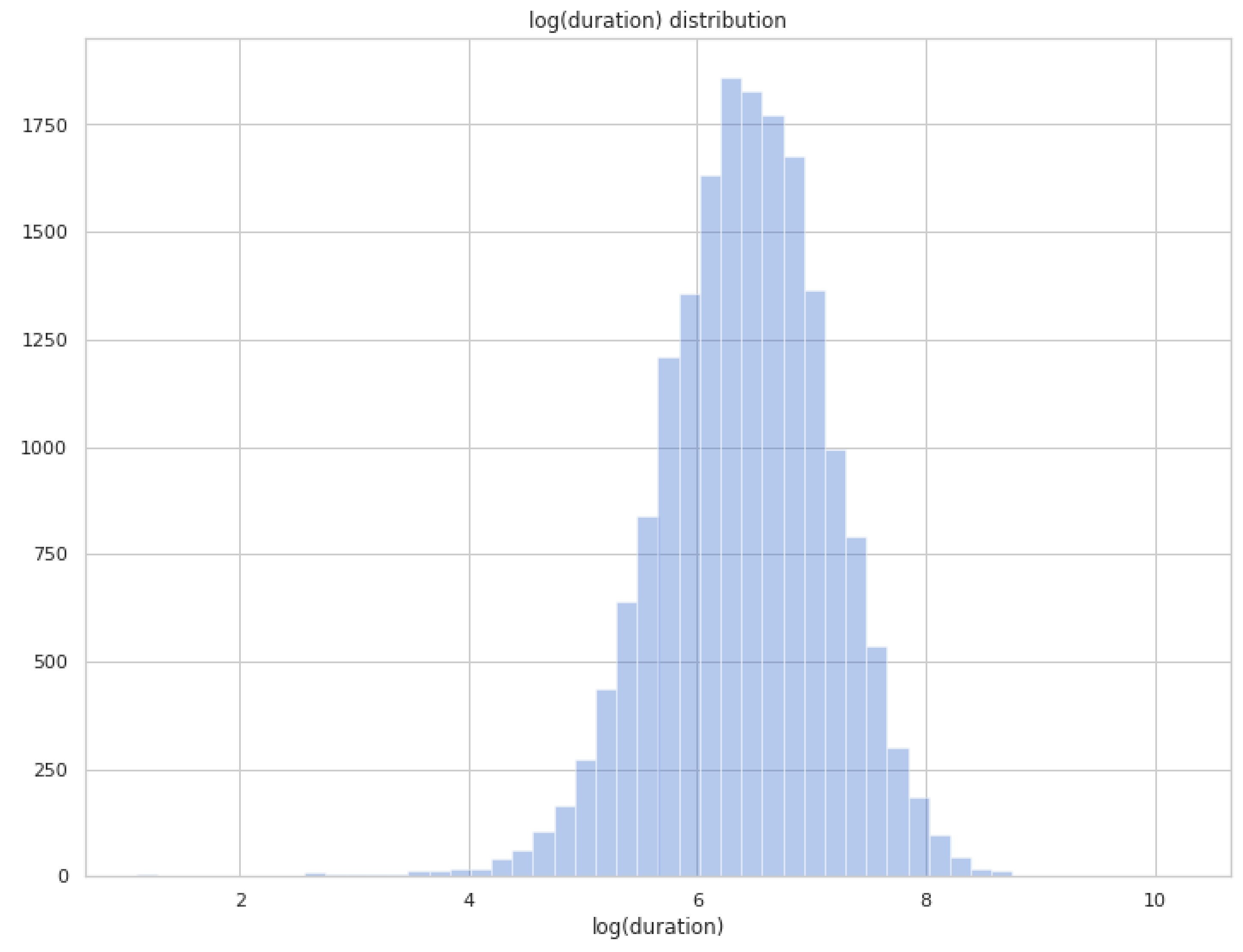
Let us take a closer look at the target `duration` variable. In the cell below, we plot its distribution using `sns.distplot` . This should give us an idea about whether we have some outliers in our data.

In [5]:
```
fig, ax = plt.subplots(figsize=(12, 9))

# Plot the distribution of duration using sns.distplot
# You can fill `ax=ax` to sns.distplot to plot in the ax object created above

sns.distplot(train_df['duration'], ax=ax, kde=False)

plt.title('Duration distribution');
```



As expected for a positive valued variable, we observe a skewed distribution. Note that we seem to have a handful of very long trips within our data. Use an appropriate data transformation to squeeze this highly-skewed distribution. Plot a `sns.distplot` of the transformed duration data for `train_df` .

In [6]:
```
fig, ax = plt.subplots(figsize=(12, 9))

# Use a log transformation to squeeze the distribution
# You can add + 1 to all values before taking the log to handle possible 0 values for distribution

sns.distplot(np.log(train_df['duration'] + 1),
             ax=ax,
             axlabel='log(duration)',
             kde=False)

plt.title('log(duration) distribution');
```



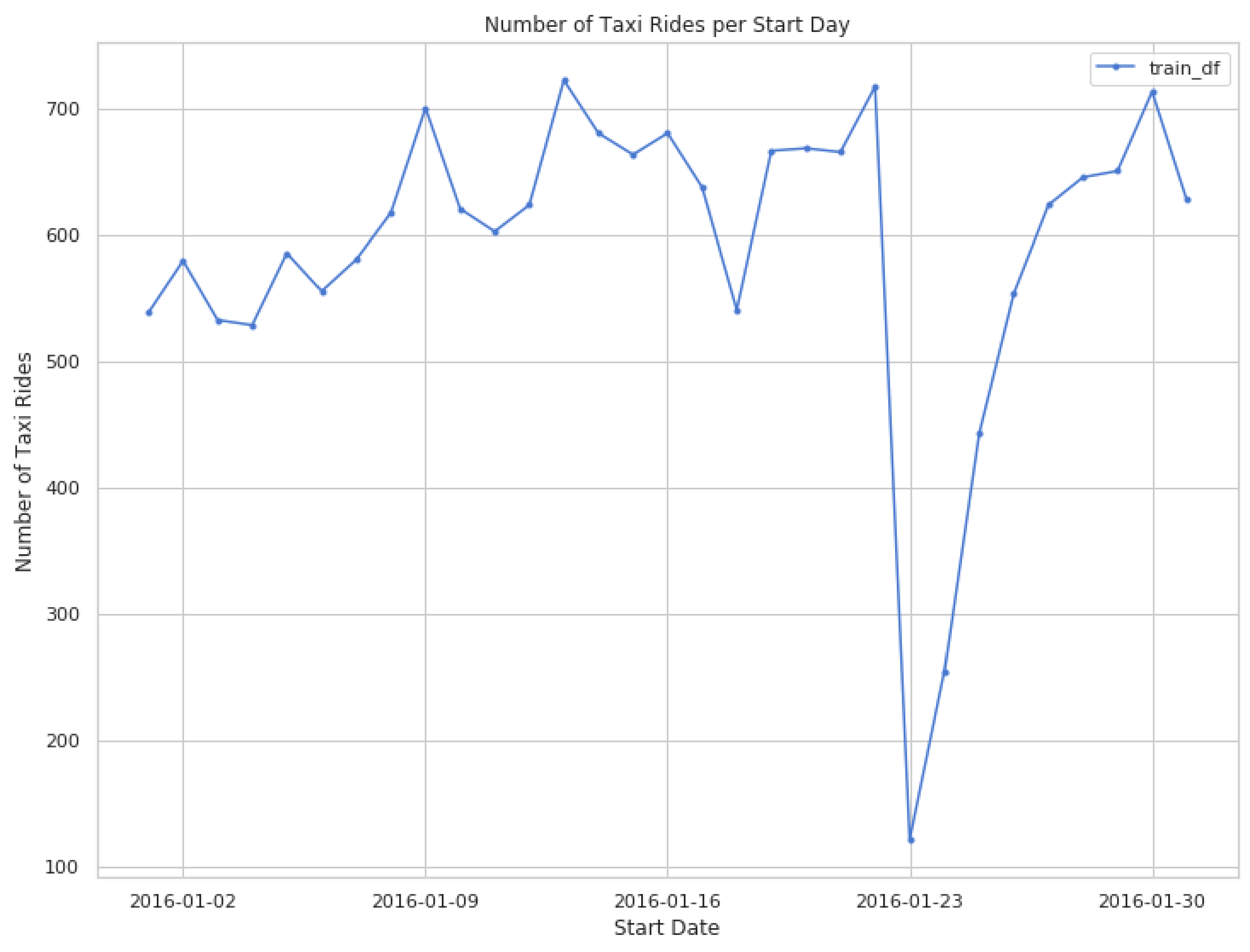
After transforming our data, we should immediately observe that we are dealing with what seems to be log-normal distribution for the target variable `duration` . We can see the behavior of shorter rides better, whereas before they were lumped in a bar near 0. This is a nice result, since it can facilitate modeling later.

Note: Keep in mind that we want to avoid peeking at our validation data because it may introduce bias. Therefore, we will be focusing on analyzing the training data for the remainder of this notebook.

2: Date Analysis

In order to understand the general pattern/trends of our taxi ride data, we will plot the number of taxi rides requested over time. Please run the following cell.

In [7]:
```python
plt.figure(figsize=(12, 9))

# Make a temporary copy of our datasets
tmp_train = train_df.copy()
tmp_train['date'] = tmp_train['tpep_pickup_datetime'].dt.date
tmp_train = tmp_train.groupby('date').count()['pickup_longitude']

# Plot the temporal overlap
plt.plot(tmp_train, '.-', label='train_df')

plt.title('Number of Taxi Rides per Start Day')
plt.xlabel("Start Date")
plt.legend()
plt.ylabel('Number of Taxi Rides');
```



Question 2a

Taking a closer look at the plot above, we notice a drastic drop in taxi rides towards the end of Janurary. What is the date corresponding to the lowest number of taxi rides? Enter your answer as a string in the format MM-DD-YYYY.

In [8]:
```python
lowest_rides_date = "01-23-2016"

# YOUR CODE HERE
#raise NotImplementedError()

print(lowest_rides_date)
```

```
01-23-2016
```

In [9]:
```python
# Hidden test!
```

Question 2b

What event could have caused this drop in taxi rides? Feel free to use Google.

In [10]:
```python
q2b_answer = r"""

There was a severe snowstorm on that day.

"""
# YOUR CODE HERE
#raise NotImplementedError()

print(q2b_answer)
```

```


There was a severe snowstorm on that day.
```

3. Spatial/Locational Analysis

We are curious about the distribution of taxi pickup/dropoff coordinates. We also may be interested in observing whether this distribution changes as we condition of longer/shorter taxi rides. In the cells below, we will categorize our data into long and short rides based on duration. Then we will plot the latitude and longitude coordinates of rides conditioned on these categories.

First you may want to familiarize yourself with a map of Manhattan (https://www.google.com/maps/place/Manhattan,+New+York,+NY/@40.7590402,-74.0394431,12z/data=!3m1!4b1!4m5!3m4!1s0x89c2588f046ee661:0xa0b3281fcecc08c!8m2!3d40.7830603!4d-73.9712488).

Here we split `train_df` into two data frames, one called `short_rides` and one called `long_rides`. `short_rides` should contain all rides less than or equal to 15 minutes and `long_rides` should contain rides more than 15 minutes.

Note: We chose 15 minutes because the mean duration of a ride is roughly 700 seconds ~ 12 minutes. We then round up to the nearest nice multiple of 5. Note that you should adjust how you determine short/long rides and outliers when feature engineering.

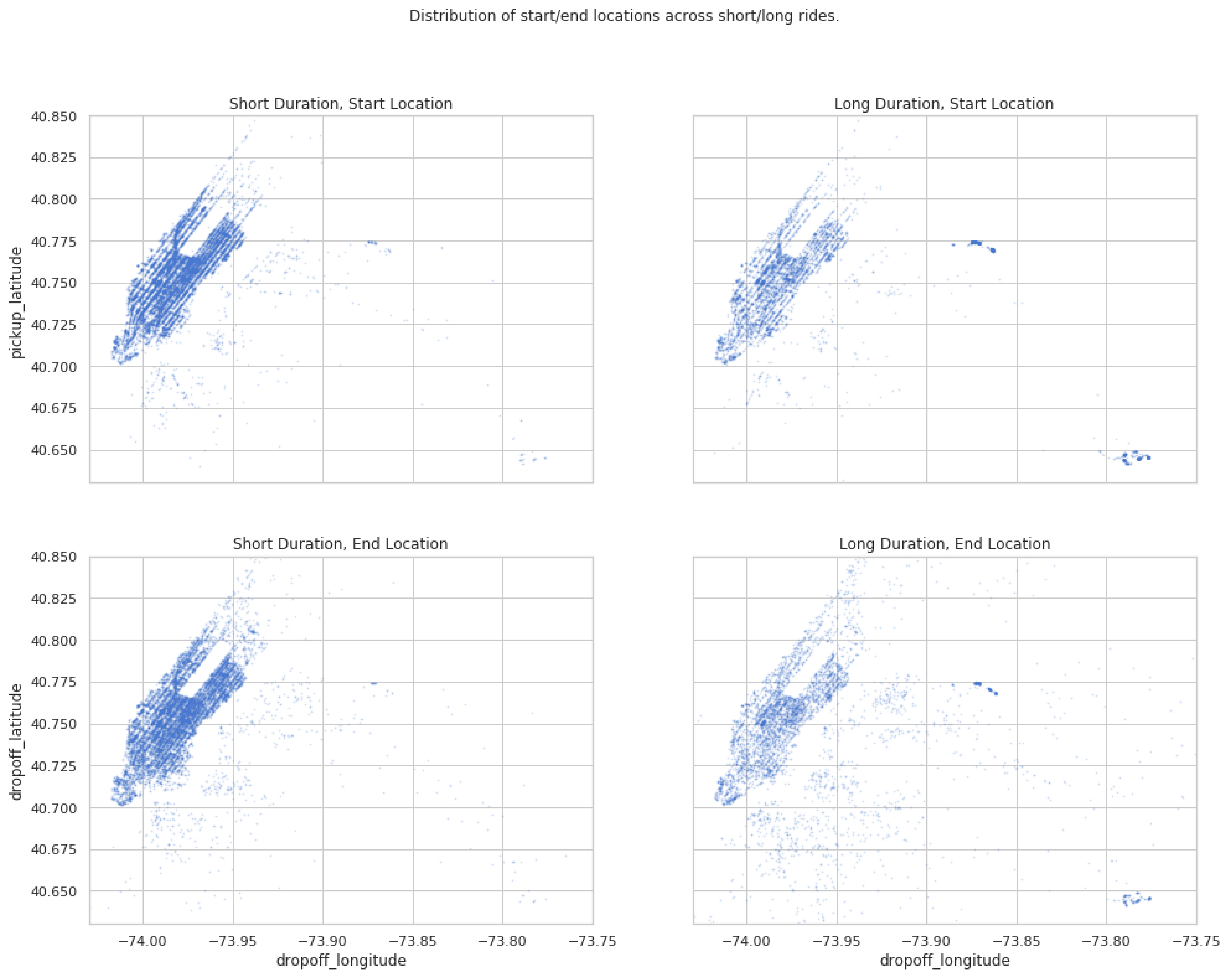
In [11]:
```python
short_rides = train_df[train_df["duration"] <= 900] # rides less than or equal to 15 mins
long_rides = train_df[train_df["duration"] > 900] # rides more than 15 minutes
```

In [12]:
```python
assert len(short_rides) == 12830
assert len(long_rides) == 5524
```

Below we generate 4 scatter plots. The scatter plots are ordered as follows:

- ax1: plot the **start** location of short duration rides
- ax2: plot the **start** location of long duration rides
- ax3: plot the **end** location of short duration rides
- ax4: plot the **end** location of long duration rides

In [13]:
```python
# Set random seed of reproducibility
random.seed(42)

# City boundaries
city_long_border = (-74.03, -73.75)
city_lat_border = (40.63, 40.85)

# Define figure
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(ncols=2, nrows = 2, figsize=(16, 12), sharex=True, sharey=True)
alpha = 0.15 # make sure to include these as an argument
s = 1 # make sure to include this as an argument

short_rides.plot(kind = "scatter", x = "pickup_longitude", y = "pickup_latitude",
                     ax = ax1, alpha = alpha, s = s, title='Short Duration, Start Location')
long_rides.plot(kind = "scatter", x = "pickup_longitude", y = "pickup_latitude",
                    ax = ax2, alpha = alpha, s = s, title='Long Duration, Start Location')
short_rides.plot(kind = "scatter", x = "dropoff_longitude", y = "dropoff_latitude",
                     ax = ax3, alpha = alpha, s = s , title='Short Duration, End Location')
long_rides.plot(kind = "scatter", x = "dropoff_longitude", y = "dropoff_latitude",
                    ax = ax4, alpha = alpha, s = s, title='Long Duration, End Location')


fig.suptitle('Distribution of start/end locations across short/long rides.')


plt.ylim(city_lat_border)
plt.xlim(city_long_border);
```

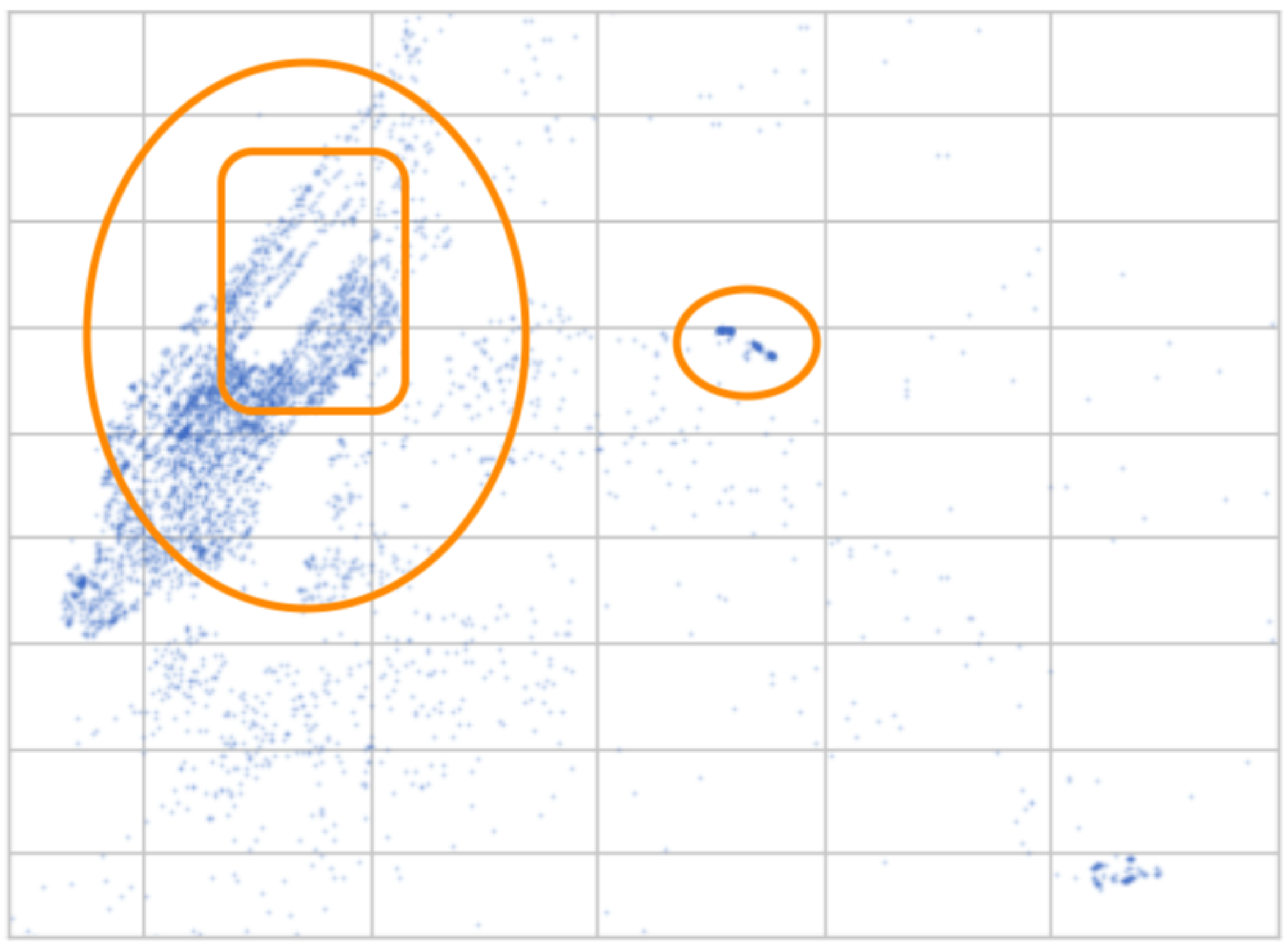


Question 3a

What do the plots above look like?

In particular:

- Find what the following circled regions correspond to:



Hint: Here is a [page (https://www.google.com/maps/place/Manhattan,+New+York,+NY/@40.7590402,-74.0394431,12z/data=!3m1!4b1!4m5!3m4!1s0x89c2588f046ee661:0xa0b3281fcecc08c!8m2!3d40.7830603!4d-73.9712488)](https://www.google.com/maps/place/Manhattan,+New+York,+NY/@40.7590402,-74.0394431,12z/data=!3m1!4b1!4m5!3m4!1s0x89c2588f046ee661:0xa0b3281fcecc08c!8m2!3d40.7830603!4d-73.9712488) that may be useful.

In [14]:
```
q3a_answer = r"""
The large circle is Manhattan
The squre is New York Central Park
The small circle is LaGuardia Airport.
"""

# YOUR CODE HERE
#raise NotImplementedError()

print(q3a_answer)
```

```
The large circle is Manhattan
The squre is New York Central Park
The small circle is LaGuardia Airport.
```

Question 3b

In each scatter plot above, why are there no points contained within the small rectangular region (towards the top left between the blue points)? Could this be an error/mistake in our data?

In [15]:
```
q3b_answer = r"""
It's not an error.
The rectangular region in New York Central Park. You can't drive inside the park.

"""

# YOUR CODE HERE
#raise NotImplementedError()

print(q3b_answer)
```

```
It's not an error.
The rectangular region in New York Central Park. You can't drive inside the park.
```

Question 3c

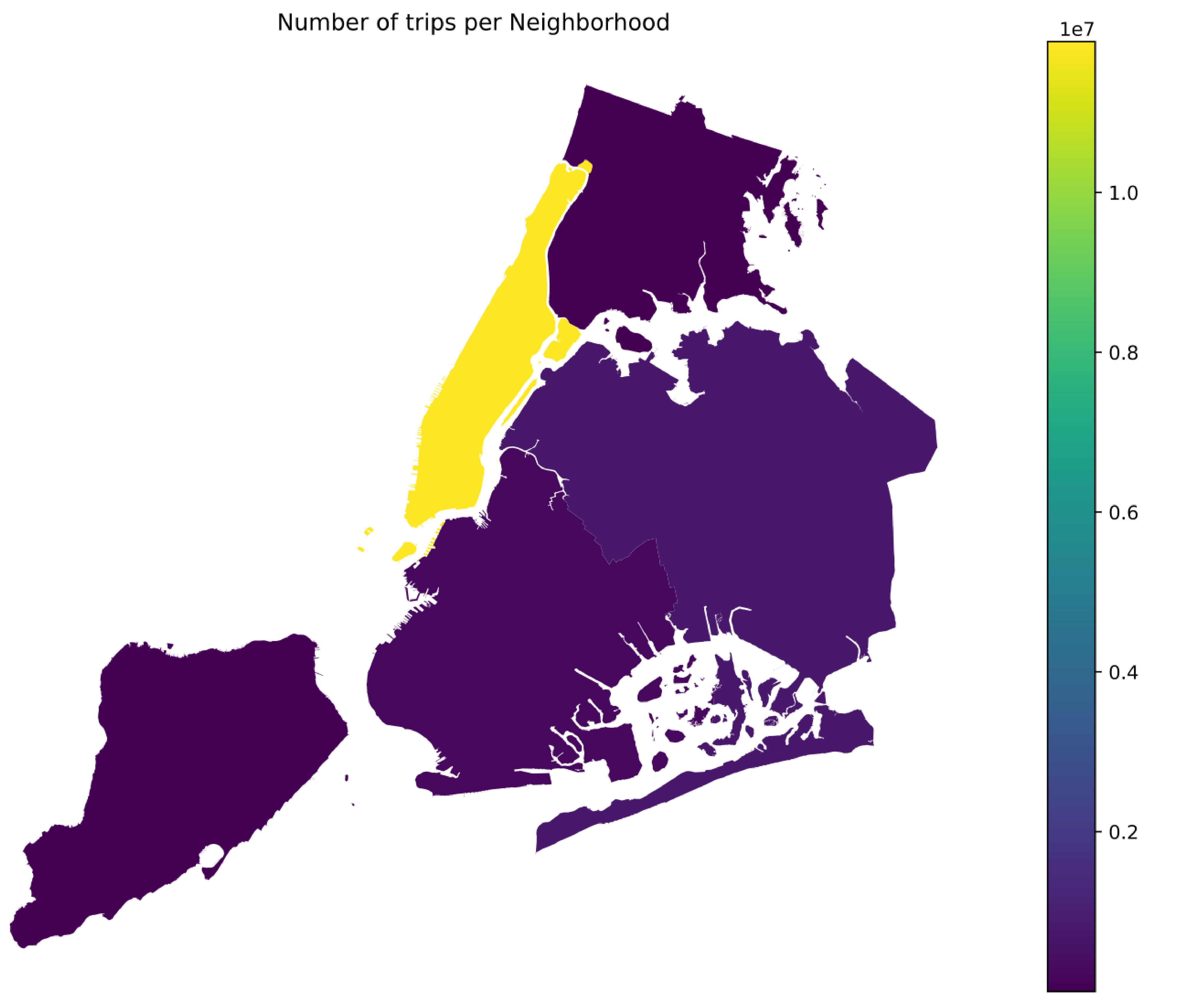
What observations/conclusions do you make based on the scatter plots above? In particular, how are trip duration and pickup/dropoff location related?

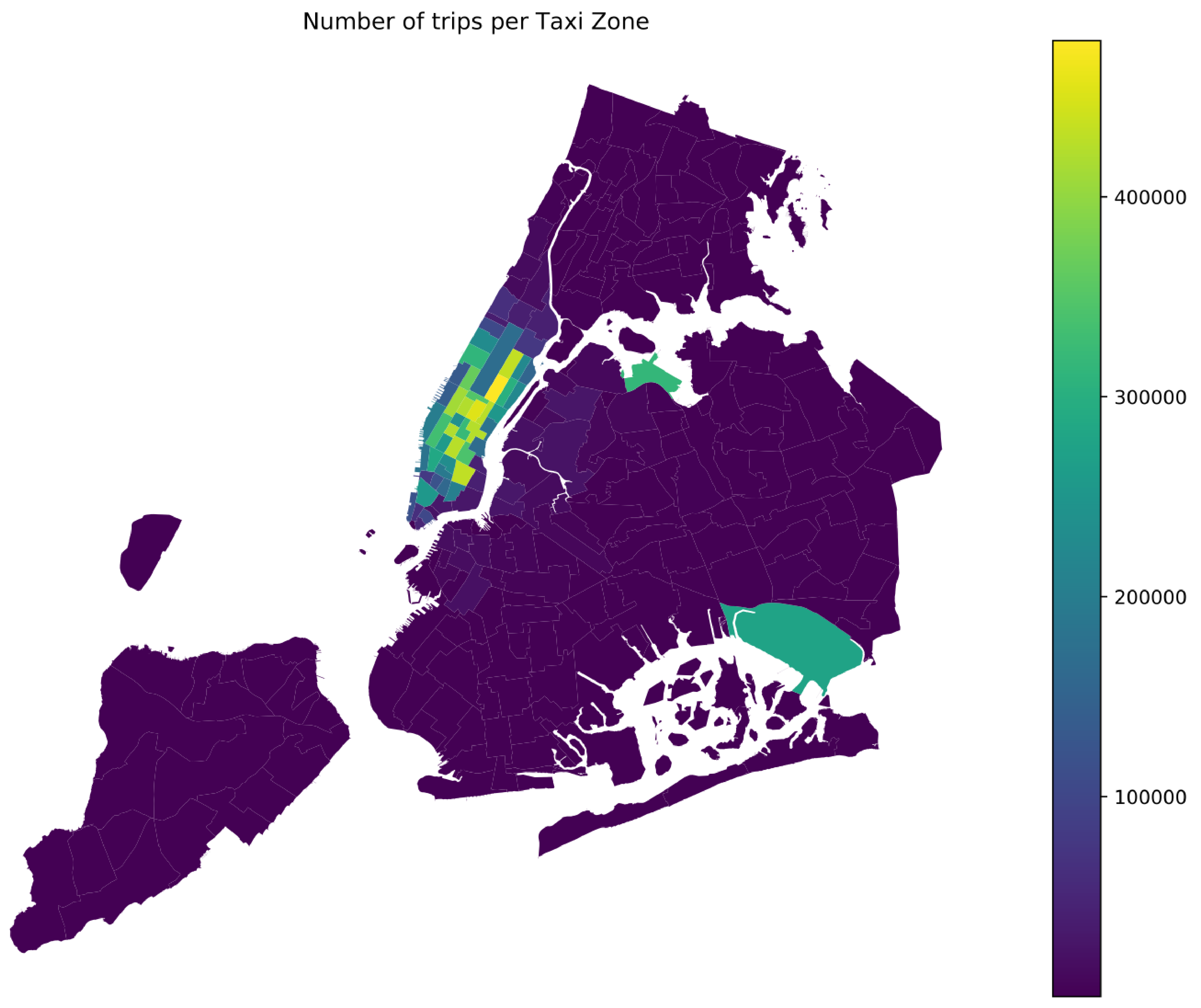
In [16]:
```
q3c_answer = r"""

Most of pickup and dropoff locations for short duration rides are in Manhattan, and very few of them are outside
Manhattan. However, more pickup locations and dropoff location for long duration rides are ouside Manhattan,
especially for dropoff locations.

So we can conclude that People New York mainly live in Manhattan.
"""

# YOUR CODE HERE
#raise NotImplementedError()

print(q3c_answer)
```

```

Most of pickup and dropoff locations for short duration rides are in Manhattan, and very few of them are outside
Manhattan. However, more pickup locations and dropoff location for long duration rides are ouside Manhattan,
especially for dropoff locations.

So we can conclude that People New York mainly live in Manhattan.
```

This confirms that the trips are localized in NYC, with a very strong concentration in Manhattan **and** on the way to LaGuardia Airport. This might give you ideas of relevant features for feature engineering.

Another way to visualize ride coordinates is using a **heat map** (this also helps us avoid overplotting). The following plots count the number of trips for NYC neighborhoods and areas, plotting with the `geopandas` package and theses shapefiles (https://geo.nyu.edu/catalog/nyu_2451_36743) (do not mind the values on the colorbar). If you are curious about how to create the figures below, feel free to check out `geopandas` (http://geopandas.org/).





4: Temporal features

We can utilize the `start_timestamp` column to design a lot of interesting features.

We implement the following temporal (related to time) features using the `add_time_columns` function below.

- `month` derived from `start_timestamp`.
- `week_of_year` derived from `start_timestamp`.
- `day_of_month` derived from `start_timestamp`.
- `day_of_week` derived from `start_timestamp`.
- `hour` derived from `start_timestamp`.
- `week_hour` derived from `start_timestamp`.

Note 1: You can use the `dt` attribute of the `start_timestamp` column to convert the entry into a `DateTime` object.

Note 2: We set `df.is_copy = False` to explicitly write back to the original dataframe, `df`, that is being passed into the `add_time_columns` function. Otherwise `pandas` will complain.

In [17]:
```python
def add_time_columns(df):
    """
    Add temporal features to df
    """
    df.is_copy = False
    df.loc[:, 'month'] = df['tpep_pickup_datetime'].dt.month
    df.loc[:, 'week_of_year'] = df['tpep_pickup_datetime'].dt.weekofyear
    df.loc[:, 'day_of_month'] = df['tpep_pickup_datetime'].dt.day
    df.loc[:, 'day_of_week'] = df['tpep_pickup_datetime'].dt.dayofweek
    df.loc[:, 'hour'] = df['tpep_pickup_datetime'].dt.hour
    df.loc[:, 'week_hour'] = df['tpep_pickup_datetime'].dt.weekday * 24 + df['hour']

    # No real need to return here, but we harmonize with remove_outliers for later pipelinezation
    return df
```

In [18]:
```python
# Note that we are applying this transformation to train_df, short_rides and long_rides
train_df = add_time_columns(train_df)
short_rides = add_time_columns(short_rides)
long_rides = add_time_columns(long_rides)
```

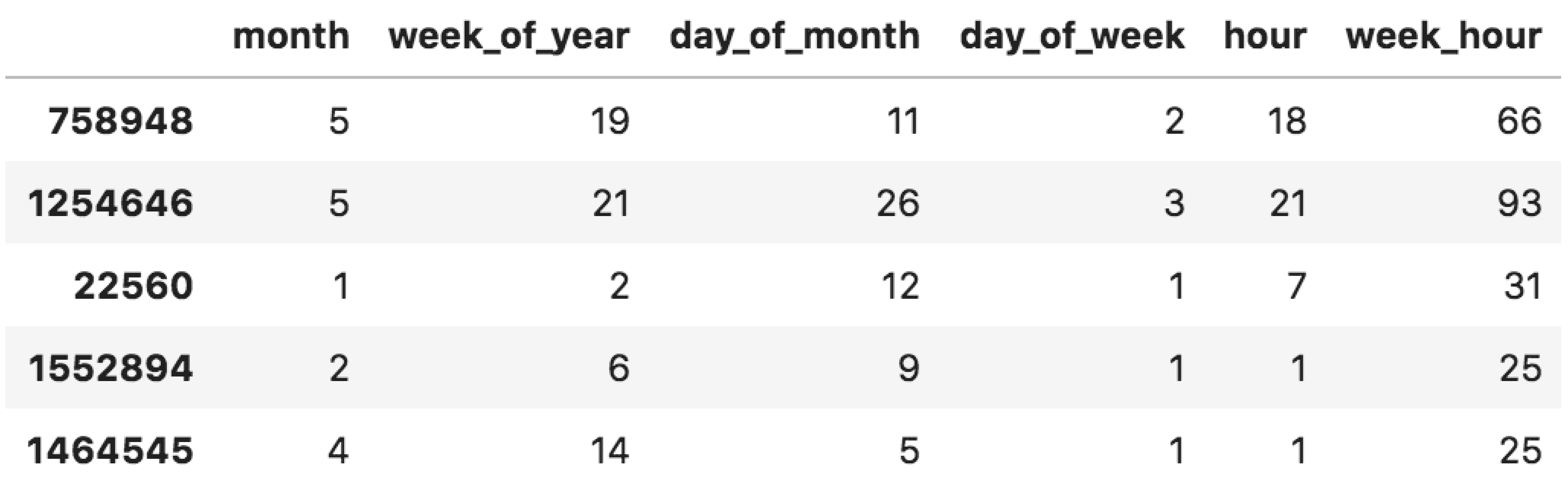
```
/srv/conda/envs/data100/lib/python3.6/site-packages/pandas/core/generic.py:4388: FutureWarning: Attribute 'is_copy' is deprecated and will be removed in a future ver
sion.
  object.__getattribute__(self, name)
/srv/conda/envs/data100/lib/python3.6/site-packages/pandas/core/generic.py:4389: FutureWarning: Attribute 'is_copy' is deprecated and will be removed in a future ver
sion.
  return object.__setattr__(self, name, value)
```

In [19]:
```python
train_df[['month', 'week_of_year', 'day_of_month', 'day_of_week', 'hour', 'week_hour']].head()
```

Out[19]:

	month	week_of_year	day_of_month	day_of_week	hour	week_hour
16434	1	3	21	3	17	89
21929	1	4	29	4	23	119
3370	1	1	5	1	18	42
21975	1	4	30	5	0	120
13758	1	3	18	0	13	13

Your `train_df.head()` should look like this, although the ordering of the data in `id` might be different:



	month	week_of_year	day_of_month	day_of_week	hour	week_hour
758948	5	19	11	2	18	66
1254646	5	21	26	3	21	93
22560	1	2	12	1	7	31
1552894	2	6	9	1	1	25
1464545	4	14	5	1	1	25

In [20]:
```python
time_columns = ['month',
                'week_of_year',
                'day_of_month',
                'day_of_week',
                'hour',
                'week_hour']


# Check columns were created
assert all(column in train_df.columns for column in time_columns)

# Check type
assert train_df[time_columns].dtypes.nunique() == 1

assert train_df[time_columns].dtypes.nunique() == 1
```

Visualizing Temporal Features

Question 4a

Let us now use the features we created to plot some histograms and visualize patterns in our dataset. We will analyze the distribution of the number of taxi rides across months and days of the week. This can help us visualize and understand patterns and trends within our data.

This is a open ended question. Create 2 plots that visualize temporal information from our dataset. At least one of them must visualize the hour of each day. Aside from that you can use any column from `time_columns`.

You can use the same column multiple times, but if the plots are redundant you will not receive full credit. This will be graded based on how informative each plot is and how "good" the visualization is (remember what good/bad visualizations look like for different kinds of data!).

Visualization 1

In [21]:
```python
train_df['tpep_pickup_datetime'].min().weekday()
```

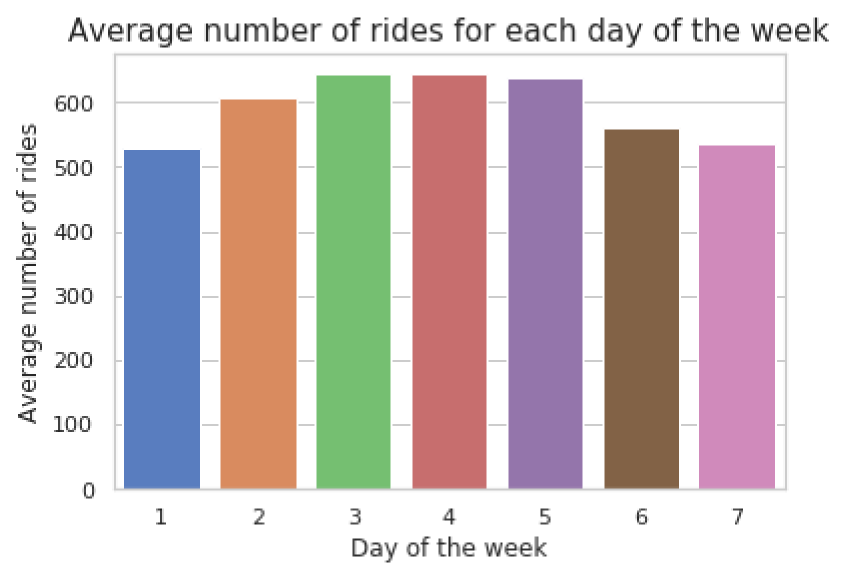
Out[21]:
```
4
```

In [22]:
```python

len(train_df['day_of_month'].unique())
```

Out[22]:
```
31
```

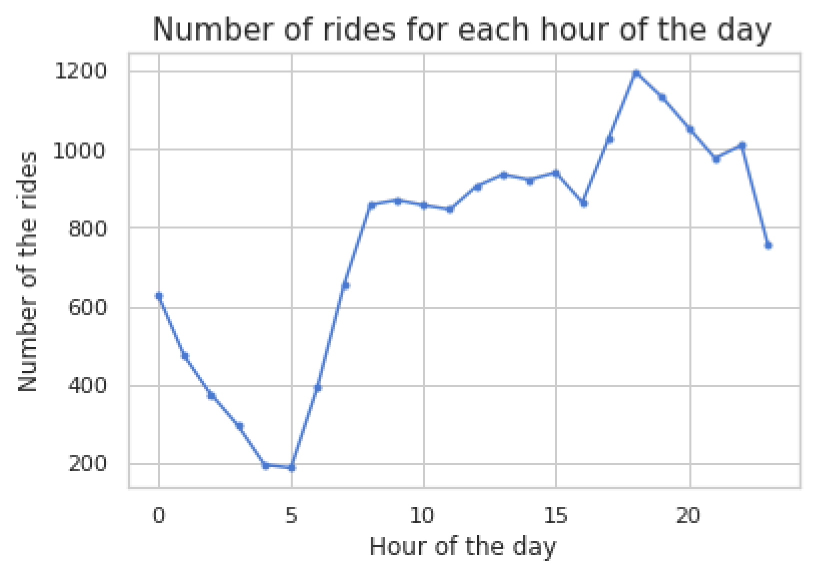
In [23]:
```
week_of_firstday = train_df['tpep_pickup_datetime'].min().weekday()
number_of_weekday = [0]*7
for i in range(31):
    number_of_weekday[(i+week_of_firstday) % 7] +=1
```

In [24]:
```
# Visualization 1
# YOUR CODE HERE
#raise NotImplementedError()
# We want to know the average number of rides for each day of the week, for January we have 31 days, 3 days of
# the day have 5 days, 4 have 4 days. 31 = 5*3 + 4*4
week_of_firstday = train_df['tpep_pickup_datetime'].min().weekday()
number_of_weekday = [0]*7
for i in range(31):
    number_of_weekday[(i+week_of_firstday) % 7] +=1
train_df['tpep_pickup_datetime'].unique()
train_df_groupby_week = train_df.groupby("day_of_week")["day_of_week"].count() / number_of_weekday
sns.barplot(x =np.array(train_df_groupby_week.index)+1, y = train_df_groupby_week)
plt.xlabel("Day of the week")
plt.ylabel('Average number of rides')
plt.title("Average number of rides for each day of the week", size = 15)
plt.show()
```



Visualization 2

In [25]:
```
# Visualization 2
# YOUR CODE HERE
#raise NotImplementedError()
train_df_groupby_hour = train_df.groupby('hour')['hour'].count()
plt.plot(train_df_groupby_hour, '.-')
plt.xlabel("Hour of the day")
plt.ylabel('Number of the rides')
plt.title('Number of rides for each hour of the day', size =15)
plt.show()
```



Question 4b

Briefly explain for each plot

1. What feature you're visualization
2. Why you chose this feature
3. Why you chose this visualization method

In [26]:
```
q4b_answer = r"""

In plot one:
1. I want to visualize the average number of riders in each of the week.
2. I choose this feature, bacause we want to show whether different days of the week affect the number of rides,
    especially the differences between weekdays and weekends.
3. I use bar plot beacause there are only 7 days in a week, and we can show the average number of each day
    clearly.

In plot two:
1. I want to visualize the total number of riders in each hour of a day.
2. I choose this feature because I want to see the trend of taking a taxi across a day like when there are most
    people taking taxis and when there are least least people taking taxis
3. I use line plot because it's easy to see the changes of the numbers of rides in 24 hours.
"""

# YOUR CODE HERE
#raise NotImplementedError()

print(q4b_answer)
```

```
In plot one:
1. I want to visualize the average number of riders in each of the week.
2. I choose this feature, bacause we want to show whether different days of the week affect the number of rides,
    especially the differences between weekdays and weekends.
3. I use bar plot beacause there are only 7 days in a week, and we can show the average number of each day
    clearly.

In plot two:
1. I want to visualize the total number of riders in each hour of a day.
2. I choose this feature because I want to see the trend of taking a taxi across a day like when there are most
    people taking taxis and when there are least least people taking taxis
3. I use line plot because it's easy to see the changes of the numbers of rides in 24 hours.
```

Question 4c

From the various plots above, what conclusions can you draw about the temporal aspects of our data? How does this relate to duration?

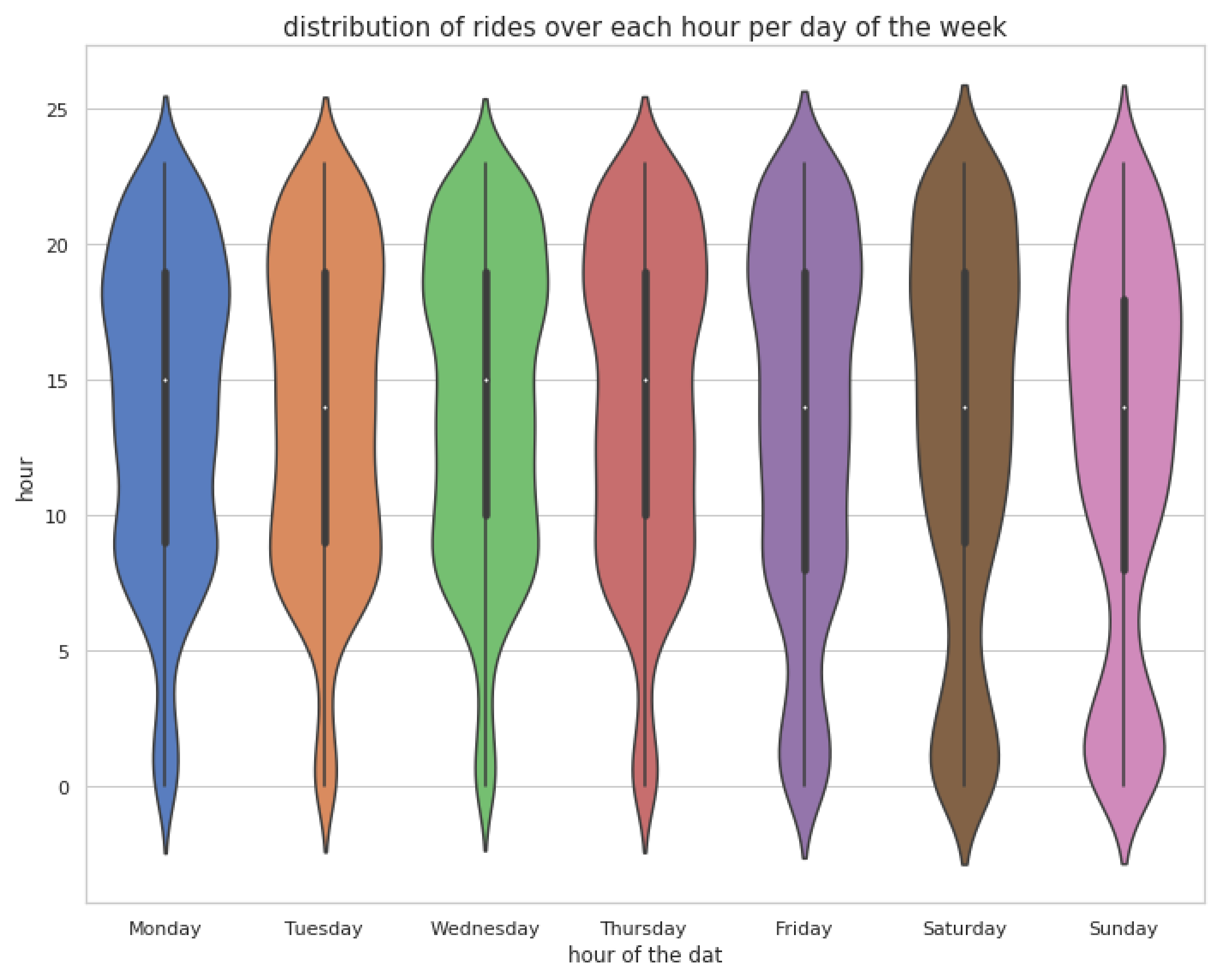
In [27]:
```python
q4c_answer = r"""

In plot one:
    There are more people that will take taxis on weekdays than weekends. And people take most taxis on Tuesday,
    Wednesday and Thurdays.

In plot two:
    There most people who will take taxis at 8pm, and least people at 4-5am

It's hard to relate to duration because we only count data for pickup time.

"""

# YOUR CODE HERE
#raise NotImplementedError()

print(q4c_answer)
```

```
In plot one:
    There are more people that will take taxis on weekdays than weekends. And people take most taxis on Tuesday,
    Wednesday and Thurdays.

In plot two:
    There most people who will take taxis at 8pm, and least people at 4-5am

It's hard to relate to duration because we only count data for pickup time.
```

Question 4d

Previously, we have analyzed the temporal features `hour` and `day_of_week` independently, but these features may in fact have a relationship between each other. Determining the extent to their relationship may be useful in helping us create new features in our model. Create a violin plot that displays distribution of rides over each hour per day of the week.

In [28]:
```python
fig, axes = plt.subplots(1, 1, figsize=(10, 8))
days_of_week = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"]


# YOUR CODE HERE

sns.violinplot(x = train_df['day_of_week'], y = train_df['hour'])
plt.xticks(range(7), days_of_week)
plt.xlabel("day of the week")
plt.xlabel('hour of the dat')
plt.title('distribution of rides over each hour per day of the week',size = 15)
#raise NotImplementedError()
plt.tight_layout()
plt.show()
```



Question 4e

Do you notice anything interesting about your visualization? How would you explain this plot to a lay person? What are the features/patterns of interest?

In [29]:
```python
q4e_answer = r"""
The shapes of the distritions for 7 days in the week are similar. And the shapes of Monday to Thursday are more
similar to each other. The shapes of Friday to Sunday are more similar to each other.

I will tell the lay person that each voilin correspond to a distribution distribution of rides over each hour
one day of the weel. The wider the part of voilin is, the more people will take taxis at this time.

All the voilins are narrowest at around 4 am, and widest at around 8pm.
For Friday, Saturday and Sunday, the voilins are more wider from 0am to 5 am.

"""

# YOUR CODE HERE
#raise NotImplementedError()

print(q4e_answer)
```

```
The shapes of the distritions for 7 days in the week are similar. And the shapes of Monday to Thursday are more
similar to each other. The shapes of Friday to Sunday are more similar to each other.

I will tell the lay person that each voilin correspond to a distribution distribution of rides over each hour
one day of the weel. The wider the part of voilin is, the more people will take taxis at this time.

All the voilins are narrowest at around 4 am, and widest at around 8pm.
For Friday, Saturday and Sunday, the voilins are more wider from 0am to 5 am.
```

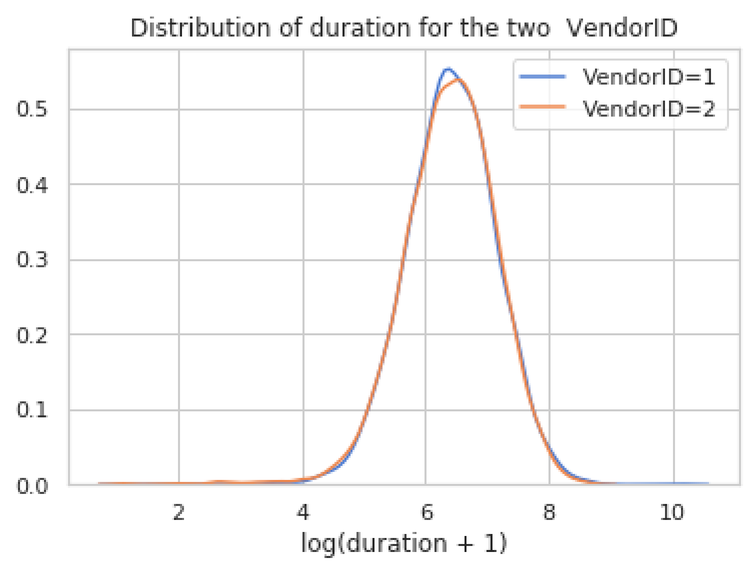
5: Vendors

Recall that in Part 1, we found that there are only two unique vendors represented in the dataset. We may wonder if the vendor feature can be useful when trying to understand taxi ride duration.

Question 5a

Visualize the VendorID feature. Create at least one plot that gives insight as to whether this feature would be useful or not in our model.

In [30]:
```python
# Visualization

# YOUR CODE HERE
sns.distplot(np.log(train_df[train_df['VendorID'] == 2]['duration']+1),hist=False,label= 'VendorID=1')
sns.distplot(np.log(train_df[train_df['VendorID'] == 1]['duration']+1),hist=False,label= 'VendorID=2')
plt.xlabel('log(duration + 1)')
plt.title("Distribution of duration for the two  VendorID")
plt.show()
#raise NotImplementedError()
```



Question 5b

Justify why you chose this visualization method and how it helps determine whether `vendor_id` is useful in our model or not.

In [31]:
```python
q5b_answer = r"""

I chost KDE distribution to visualize because we care about the proportion of each duration to see the
distribution. And we don't care about the actual number for each duration, so we won't use histogram

From the distribution plot of the two VendorIds, we can see the plots are mostly overlapped. So we can determine
that vendor_id is not useful in our model.

"""

# YOUR CODE HERE
#raise NotImplementedError()

print(q5b_answer)
```

```
I chost KDE distribution to visualize because we care about the proportion of each duration to see the
distribution. And we don't care about the actual number for each duration, so we won't use histogram

From the distribution plot of the two VendorIds, we can see the plots are mostly overlapped. So we can determine
that vendor_id is not useful in our model.
```

Question 5c

From the plot above, do you think vendor_id will help us understand duration? Why or why not?

In [32]:
```python
q5c_answer = r"""

I don't think vendor_if will help us understand duration. Because both the two vendor ids have the distribution
of duration.

"""

# YOUR CODE HERE
#raise NotImplementedError()

print(q5c_answer)
```

```
I don't think vendor_if will help us understand duration. Because both the two vendor ids have the distribution
of duration.
```

6: Distance features

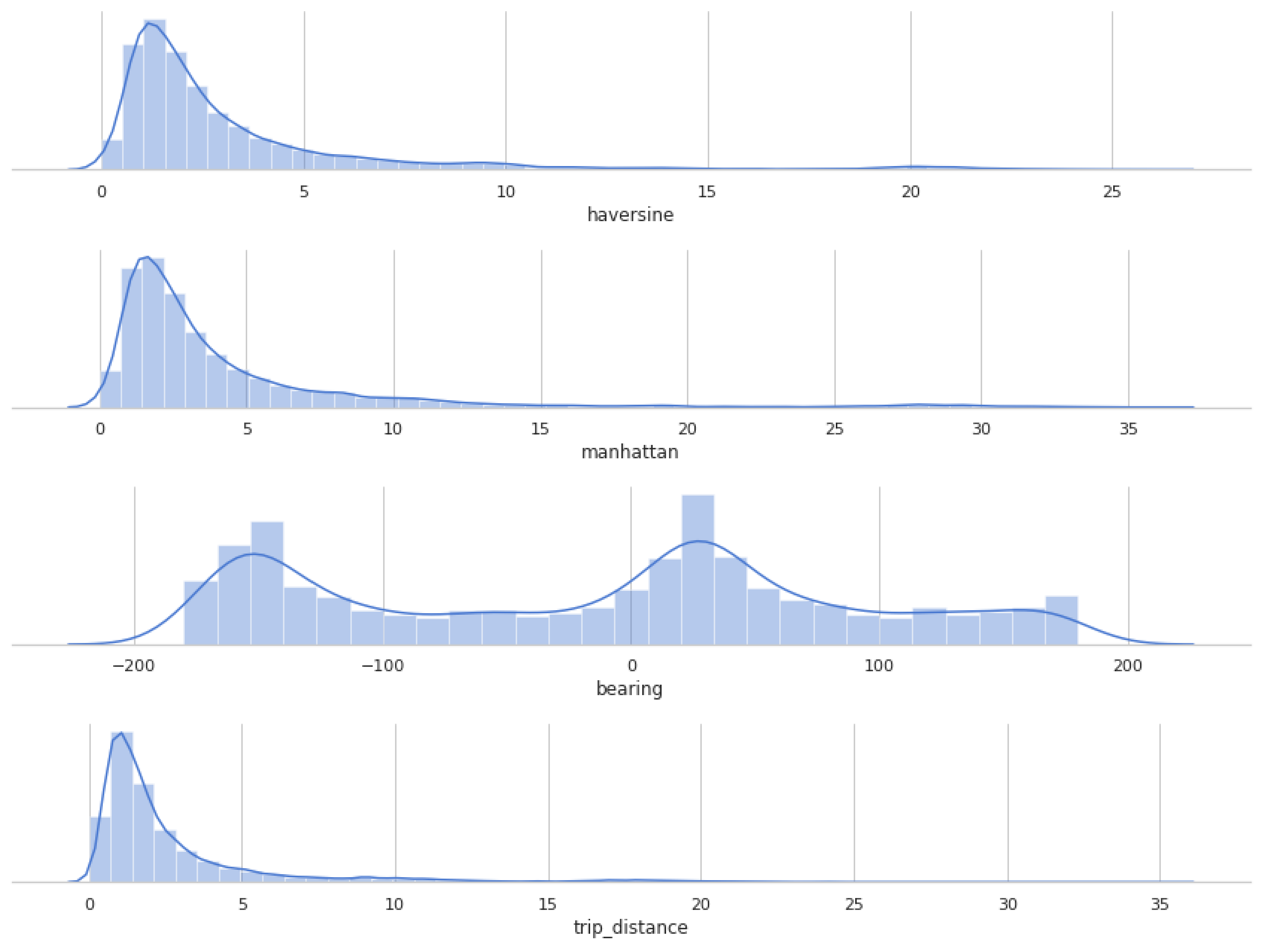
We can also use the coordinates information to compute distance features. This will allow us to compute speed related features.
We will compute the haversine (https://en.wikipedia.org/wiki/Haversine_formula) distance, the manhattan (https://en.wikipedia.org/wiki/Taxicab_geometry) distance and the bearing (http://www.mathsteacher.com.au/year7/ch08_angles/07_bear/bearing.htm) angle.

In [33]:
```python
# These functions are implemented for you
def haversine(lat1, lng1, lat2, lng2):
    """
    Compute haversine distance

    The haversine formula determines the great-circle distance between two points
    on a sphere given their longitudes and latitudes. Important in navigation, it
    is a special case of a more general formula in spherical trigonometry,
    the law of haversines, that relates the sides and angles of spherical triangles.
    """
    lat1, lng1, lat2, lng2 = map(np.radians, (lat1, lng1, lat2, lng2))
    average_earth_radius = 6371
    lat = lat2 - lat1
    lng = lng2 - lng1
    d = np.sin(lat * 0.5) ** 2 + np.cos(lat1) * np.cos(lat2) * np.sin(lng * 0.5) ** 2
    h = 2 * average_earth_radius * np.arcsin(np.sqrt(d))
    return h

def manhattan_distance(lat1, lng1, lat2, lng2):
    """
    Computes Manhattan distance

    The name alludes to the grid layout of most streets on the island of Manhattan,
    which causes the shortest path a car could take between two intersections in the borough
    to have length equal to the intersections' distance in taxicab geometry.
    """
    a = haversine(lat1, lng1, lat1, lng2)
    b = haversine(lat1, lng1, lat2, lng1)
    return a + b

def bearing(lat1, lng1, lat2, lng2):
    """
    Compute the bearing, or angle, from (lat1, lng1) to (lat2, lng2).
    A bearing of 0 refers to a NORTH orientation.
    """
    lng_delta_rad = np.radians(lng2 - lng1)
    lat1, lng1, lat2, lng2 = map(np.radians, (lat1, lng1, lat2, lng2))
    y = np.sin(lng_delta_rad) * np.cos(lat2)
    x = np.cos(lat1) * np.sin(lat2) - np.sin(lat1) * np.cos(lat2) * np.cos(lng_delta_rad)
    return np.degrees(np.arctan2(y, x))
```

In [34]:
```python
def add_distance_columns(df):
    df.loc[:, 'manhattan'] = manhattan_distance(lat1=df['pickup_latitude'],
                                                 lng1=df['pickup_longitude'],
                                                 lat2=df['dropoff_latitude'],
                                                 lng2=df['dropoff_longitude'])

    df.loc[:, 'bearing'] = bearing(lat1=df['pickup_latitude'],
                                   lng1=df['pickup_longitude'],
                                   lat2=df['dropoff_latitude'],
                                   lng2=df['dropoff_longitude'])
    df.loc[:, 'haversine'] = haversine(lat1=df['pickup_latitude'],
                                   lng1=df['pickup_longitude'],
                                   lat2=df['dropoff_latitude'],
                                   lng2=df['dropoff_longitude'])

    return df
```

In [35]:
```python
train_df = add_distance_columns(train_df)
short_rides = add_distance_columns(short_rides)
long_rides = add_distance_columns(long_rides)
```

In [36]:
```python
fig, axes = plt.subplots(4, 1, figsize=(12, 9))
sns.distplot(train_df['haversine'], ax=axes[0], axlabel='haversine');
sns.distplot(train_df['manhattan'], ax=axes[1], axlabel='manhattan');
sns.distplot(train_df['bearing'], ax=axes[2], axlabel='bearing');
sns.distplot(train_df['trip_distance'], ax=axes[3], axlabel='trip_distance');

sns.despine(left=True);
plt.setp(axes, yticks=[]);
plt.tight_layout();

```



Question 6a

The `bearing` direction is angle, the initial direction of the trip.
The bearing direction has two prominent peaks around 30 and -150 degrees.
Can you relate these peaks to the orientation of Manhattan? What do you notice about these angles?

Hint: This [wikipedia article (https://en.wikipedia.org/wiki/Commissioners%27_Plan_of_1811)](https://en.wikipedia.org/wiki/Commissioners%27_Plan_of_1811) has the answer, although it may take some digging. Alternatively, try to look at a map of Manhattan.

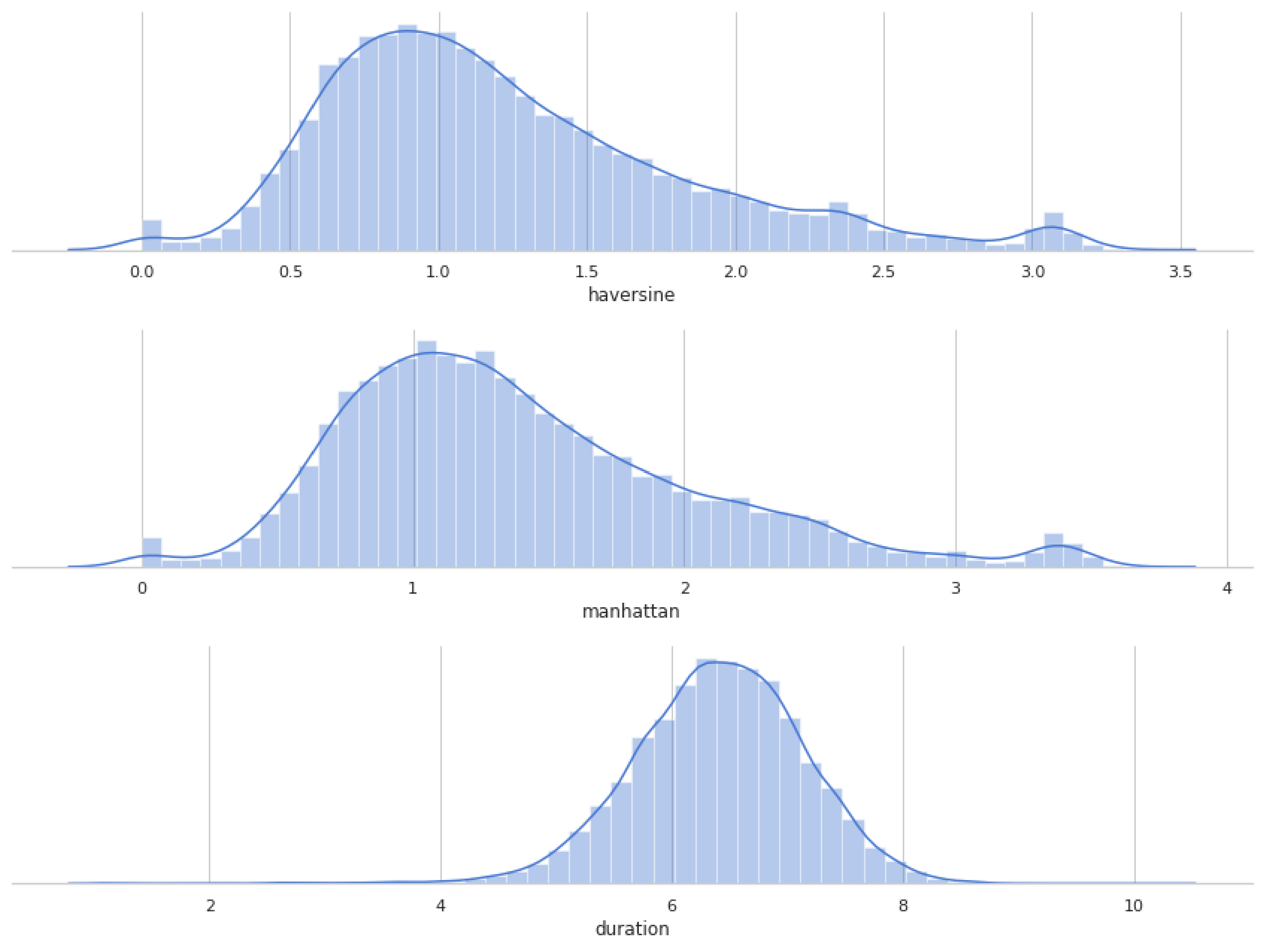
In [37]:
```
q6a_answer = r"""

As we can observe Manhattan in the map, the peninsula has an orientation around 30 degrees. And the main avenues
also have an orientation around 30 degrees. In fact 30 and -150 degrees correspond to the orientation of
Manhattan. So it's nost strange that The bearing direction has two prominent peaks around 30 and -150 degrees.

"""

# YOUR CODE HERE
#raise NotImplementedError()

print(q6a_answer)
```

```
As we can observe Manhattan in the map, the peninsula has an orientation around 30 degrees. And the main avenues
also have an orientation around 30 degrees. In fact 30 and -150 degrees correspond to the orientation of
Manhattan. So it's nost strange that The bearing direction has two prominent peaks around 30 and -150 degrees.
```

Question 6b

For haversine and manhattan distances, it is probably more helpful to look at the log distribution. We are also curious about whether these distance features can help us understand duration. Create at least one plot that compares haversine and manhattan distances and gives insight as to whether this would be a useful feature in our model.

In [38]:
```
# Visualization
# YOUR CODE HERE
fig, axes = plt.subplots(3, 1, figsize=(12, 9))
sns.distplot(np.log(train_df['haversine']+1), ax=axes[0], axlabel='haversine');
sns.distplot(np.log(train_df['manhattan']+1), ax=axes[1], axlabel='manhattan');
sns.distplot(np.log(train_df['duration']+1), ax=axes[2], axlabel='duration');

sns.despine(left=True);
plt.setp(axes, yticks=[]);
plt.tight_layout();
#raise NotImplementedError()
```



Question 6c

Justify why you chose this visualization method and how it helps inform you about using manhattan/haversine distance as a feature for predicting trip duration.

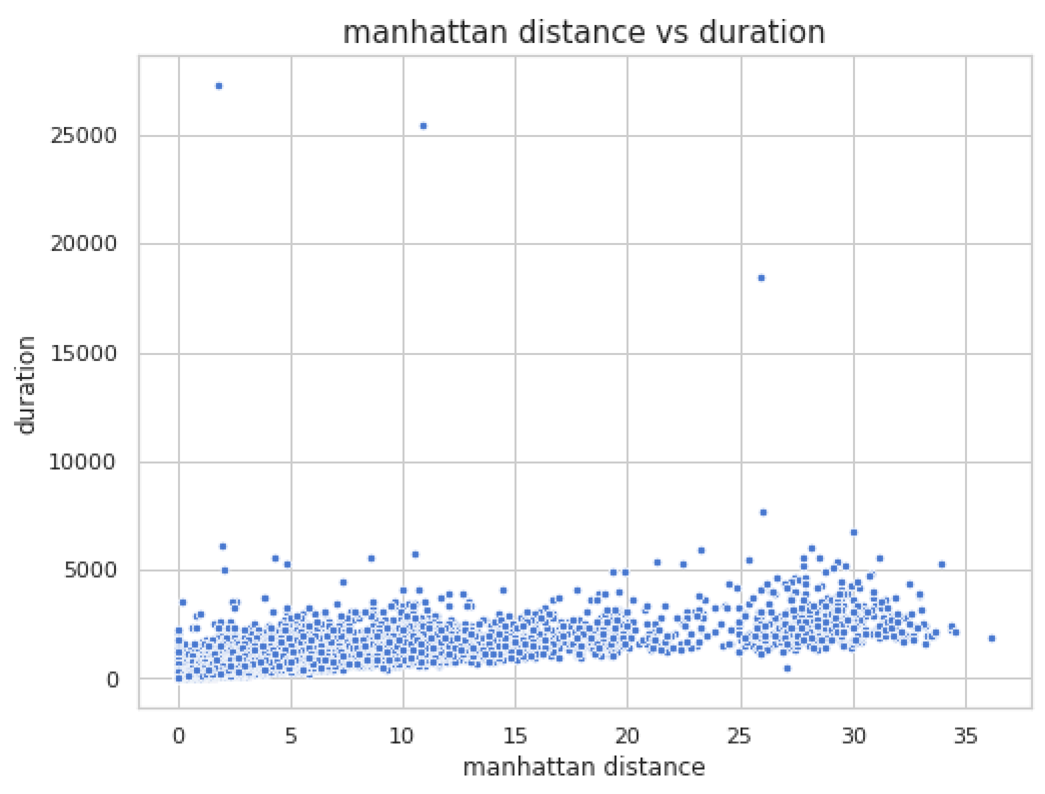
In [39]:
```
q6c_answer = r"""
I chose histograms and KDE to visualize our distances and duration, and made a comparison. Instead of viewing
the raw data, we show them as log distribution, in this way we can slove the right skew problem and compare those
distributions more clearly.
From the above plot, we can see that the distribution of harversine and that of manhanttan distances are
pretty similiar. And They both have an analogous distrition as the disrtibution ofduration. So we can use either
one of harversine and manhanttan distances for predicting trip duration


"""

# YOUR CODE HERE
#raise NotImplementedError()

print(q6c_answer)
```

```
I chose histograms and KDE to visualize our distances and duration, and made a comparison. Instead of viewing
the raw data, we show them as log distribution, in this way we can slove the right skew problem and compare those
distributions more clearly.
From the above plot, we can see that the distribution of harversine and that of manhanttan distances are
pretty similiar. And They both have an analogous distrition as the disrtibution ofduration. So we can use either
one of harversine and manhanttan distances for predicting trip duration
```

Question 6d

Fill in the code below to plot a scatter plot of manhattan distance vs duration.

In [40]:
```
# YOUR CODE HERE
plt.figure(figsize=(8,6))
sns.scatterplot(train_df['manhattan'], train_df['duration'],s = 20)
plt.xlabel('manhattan distance')
plt.ylabel('duration')
plt.title('manhattan distance vs duration',size = 15)
plt.show()
#raise NotImplementedError()
```



Question 6e

According to the plot above, there are a few outliers in both duration and manhattan distance. **Which type of outliers is most likely to be a mistake in our data?**

In [41]:
```
q6e_answer = r"""
The outliers of duration are most likely to be mistake. For some points, The manhattan distances are not larger,
however the durations are more than 25000 seconds, almost 8 hours, it is very unusual.
"""

# YOUR CODE HERE
#raise NotImplementedError()

print(q6e_answer)
```

```
The outliers of duration are most likely to be mistake. For some points, The manhattan distances are not larger,
however the durations are more than 25000 seconds, almost 8 hours, it is very unusual.
```

7: Advanced features

You do not need to incorporate these features into your model, although it may help lower your error. You are required to read through this portion and respond to the questions. All of the code is provided, please skim through it and try to understand what each cell is doing.

Clustering

Clustering (https://en.wikipedia.org/wiki/Cluster_analysis) is the task of grouping objects such that members within each group are more similar to each other than members of other groups. Clustering is a powerful tool used in many fields, including machine learning, pattern recognition, image analysis, information retrieval, bioinformatics, data compression, and computer graphics. Recall cluster sampling, which we learned earlier in the semester. We will use a simple clustering method (clustering by spatial locality) to reveal some more advanced features.

Speed features

For `train_df` , we have the `duration` and now some distance information.
This is enough for us to compute average speed and try to better understand our data.

For `test_df` , we cannot use `duration` as a feature because it is what we are trying to predict. One clever way to include speed information for modeling would be as follows:

1. Cluster the observations in `train_df` by rounding the latitude and longitudes.
2. Compute the average speed per pickup cluster and dropoff cluster.
3. Match each observation in `test_df` to its pickup cluster and dropoff cluster based off the latitude and longitude, thus assigning the average speed for the pickup and dropoff cluster.
4. We have added speed information as features for `test_df` .

Therefore, we have propagated information computed in the `train_df` into the `test_df` via clustering. This is not something we will do in this notebook, although you can try it for yourself!

Other information that could be added based on clustering (both pickup cluster and dropoff cluster):

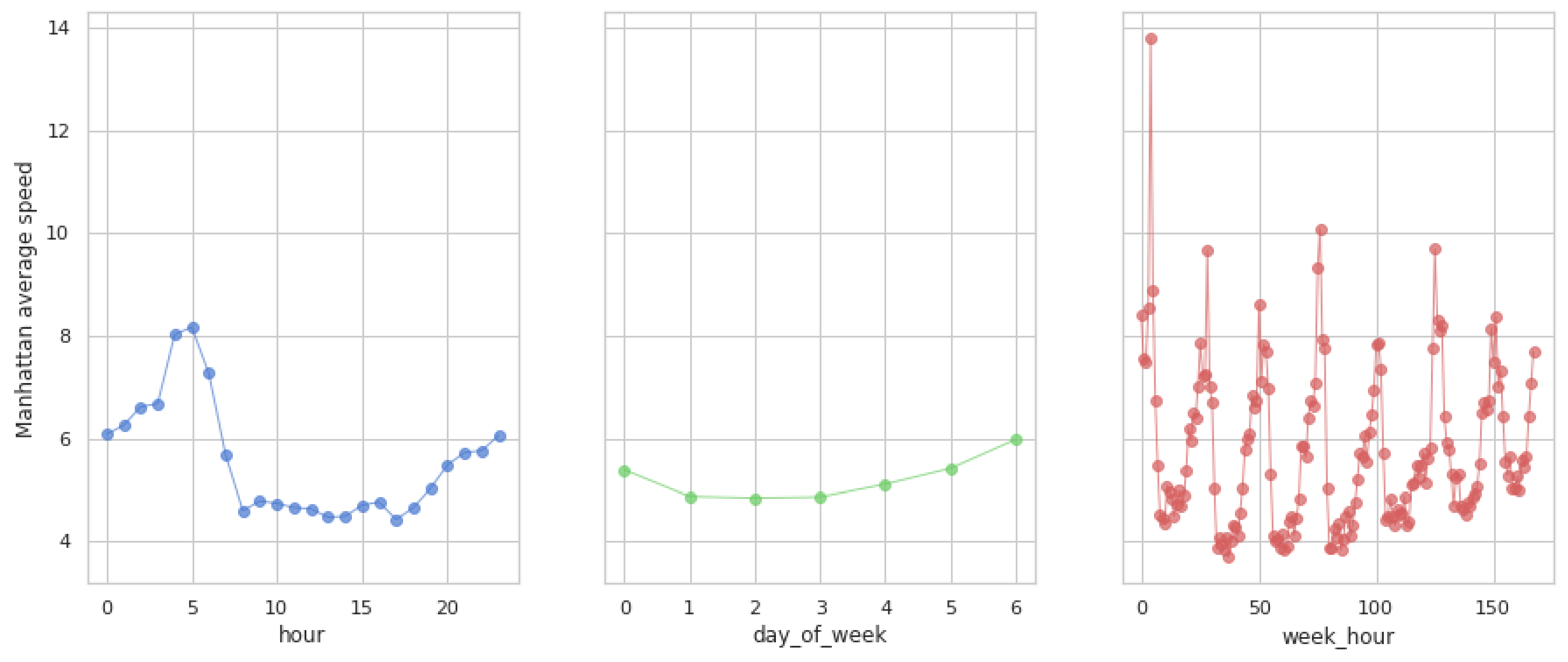
- Average of `avg_speed_h` per cluster.
- Average of `duration` per cluster.
- Average of `avg_speed_h` per cluster and hour.
- Average of `duration` per cluster and hour.
- In-cluster flow of trips for 60 min period.
- Out-cluster flow of trips for 60 min period.

In [42]:
```
# Calculate average manhattan speed
train_df['avg_speed_m'] = 1000 * train_df['manhattan'] / train_df['duration']
train_df['avg_speed_m'] = train_df['avg_speed_m'][train_df['avg_speed_m'] < 100]
train_df['avg_speed_m'].fillna(train_df['avg_speed_m'].median(), inplace=True)
```

In [43]:
```
train_df['avg_speed_m'].describe()
```

Out[43]:
```
count    18354.000000
mean         5.210825
std          2.883174
min          0.000000
25%          3.287328
50%          4.617264
75%          6.413992
max         59.225577
Name: avg_speed_m, dtype: float64
```

In [44]:
```python
# Visualize average manhattan speed by hour, day of week and week hour
fig, axes = plt.subplots(ncols=3, figsize=(15, 6), sharey=True)

axes[0].plot(train_df.groupby('hour').mean()['avg_speed_m'], 'bo-', lw=1, alpha=0.7)
axes[1].plot(train_df.groupby('day_of_week').mean()['avg_speed_m'], 'go-', lw=1, alpha=0.7)
axes[2].plot(train_df.groupby('week_hour').mean()['avg_speed_m'], 'ro-', lw=1, alpha=0.7)

axes[0].set_xlabel('hour')
axes[1].set_xlabel('day_of_week')
axes[2].set_xlabel('week_hour')
axes[0].set_ylabel('Manhattan average speed');
```



Question 7a

Based off of these visualizations, provide 2-3 insights on the average speed.

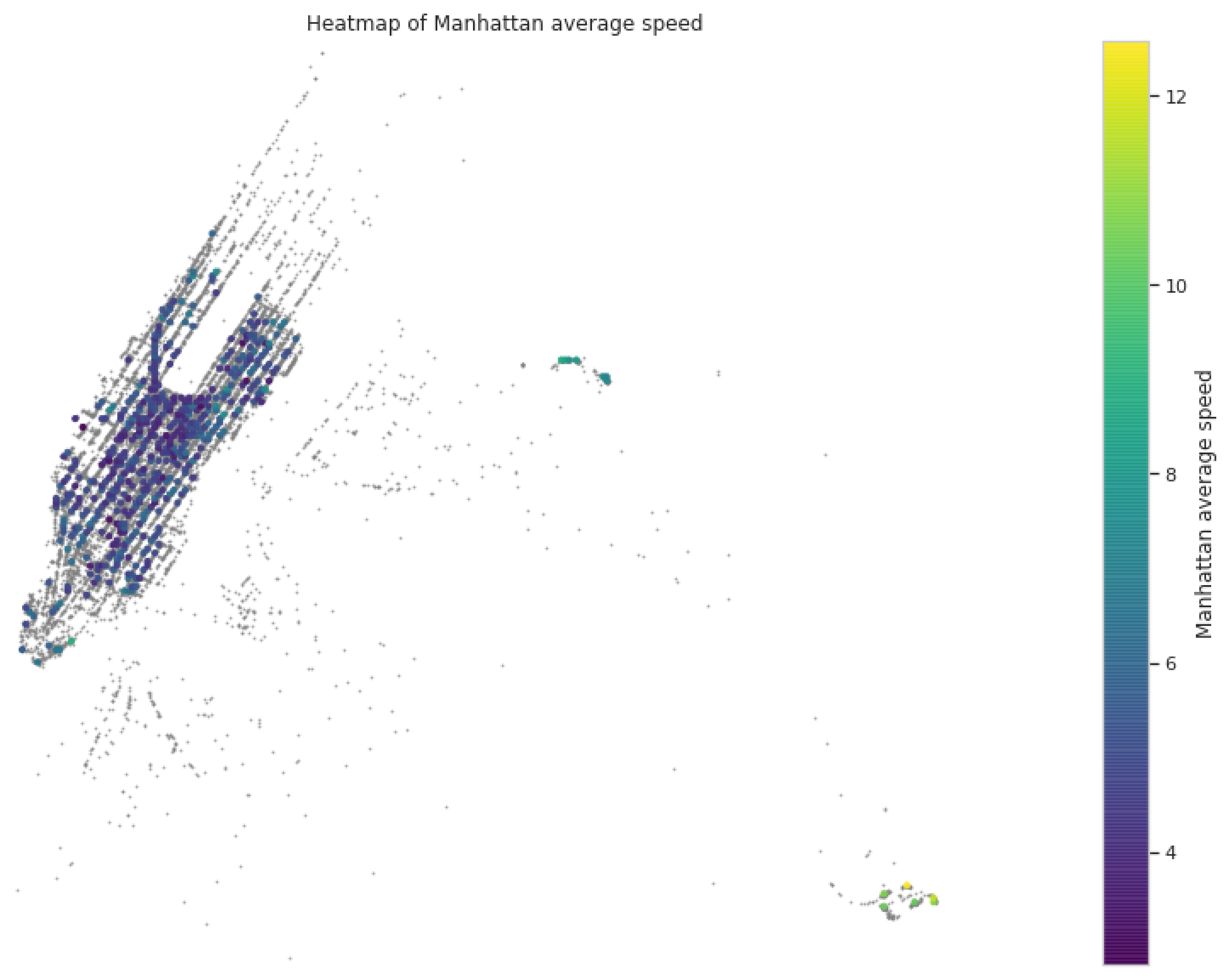
In [45]:
```python
q7a_answer = r"""

1. The graph of speed across hours is very like the inversion of the graph of numbers of rides across hours.
And the graph of speed across days in the week is very like the inversion of the graph of numbers of
rides across days in the week. The prominent peak of the speed correspont the valley of the number of rides.
So we could conclude that the speed might in inversely proportional to the number of rides.

2. In the third the period of speeds related to week hours is about 24 hours. So the speeds of taxis hav the
similar pattern every day.

"""

# YOUR CODE HERE
#raise NotImplementedError()

print(q7a_answer)
```

```
1. The graph of speed across hours is very like the inversion of the graph of numbers of rides across hours.
And the graph of speed across days in the week is very like the inversion of the graph of numbers of
rides across days in the week. The prominent peak of the speed correspont the valley of the number of rides.
So we could conclude that the speed might in inversely proportional to the number of rides.

2. In the third the period of speeds related to week hours is about 24 hours. So the speeds of taxis hav the
similar pattern every day.
```

We are now going to visualize the average speed per region. Here we define regions as a very basic classical clustering based on rounding of spatial coordinates.

In [46]:
```python
# Round / bin the latitude and longitudes
train_df['start_lat_bin'] = np.round(train_df['pickup_latitude'], 3)
train_df['start_lng_bin'] = np.round(train_df['pickup_longitude'], 3)

# Average speed for regions
gby_cols = ['start_lat_bin', 'start_lng_bin']

coord_stats = (train_df.groupby(gby_cols)
               .agg({'avg_speed_m': 'mean', 'manhattan': 'count'})
               .reset_index())

coord_stats = coord_stats[coord_stats['manhattan'] > 10]
```

In [47]:
```python
# Visualize the average speed per region
city_long_border = (-74.03, -73.75)
city_lat_border = (40.63, 40.85)
fig, ax = plt.subplots(ncols=1, nrows=1, figsize=(14, 10))

scatter_trips = ax.scatter(train_df['pickup_longitude'].values,
                           train_df['pickup_latitude'].values,
                           color='grey', s=1, alpha=0.5)

scatter_cmap = ax.scatter(coord_stats['start_lng_bin'].values,
                          coord_stats['start_lat_bin'].values,
                          c=coord_stats['avg_speed_m'].values,
                          cmap='viridis', s=10, alpha=0.9)

cbar = fig.colorbar(scatter_cmap)
cbar.set_label("Manhattan average speed")
ax.set_xlim(city_long_border)
ax.set_ylim(city_lat_border)
ax.set_xlabel('Longitude')
ax.set_ylabel('Latitude')
plt.title('Heatmap of Manhattan average speed')
plt.axis('off');
```



Question 7b

In 2-3 sentences, describe how we can use the clustering visualization above to gain insight on the speed. Do you think spatial clustering would be useful in reducing the error of our model?

In [48]:
```python
q7b_answer = r"""

1. The more central the pickup points are in Manhattan, the speed will be shower. The farther the pickup points
are away form the downtown in Manhattan, the speed will be faster.

2.The speeds in cantral Manhattan are very close to each other, and they merge together.

3.The speeds in the two airports of New York are the highest.

I think it would be useful in reducing the error, because we can predict the speed by the location, and combined
with the distance, we can calculate the duration time. But the improvement will be very limited. Since in the
most crowded location in New York, the speeds are very close, and it's easy to make a mistake when finding the \
cluster.
"""

# YOUR CODE HERE
#raise NotImplementedError()

print(q7b_answer)
```

```
1. The more central the pickup points are in Manhattan, the speed will be shower. The farther the pickup points
are away form the downtown in Manhattan, the speed will be faster.

2.The speeds in cantral Manhattan are very close to each other, and they merge together.

3.The speeds in the two airports of New York are the highest.

I think it would be useful in reducing the error, because we can predict the speed by the location, and combined
with the distance, we can calculate the duration time. But the improvement will be very limited. Since in the
most crowded location in New York, the speeds are very close, and it's easy to make a mistake when finding the \
cluster.
```

Part 2 Exports

We are not requiring you to export anything from this notebook, but you may find it useful to do so. There is a space below for you to export anything you wish.

In [49]:
```python
Path("data/part2").mkdir(parents=True, exist_ok=True)
data_file = Path("data/part2", "data_part2.hdf") # Path of hdf file
...
```

Out[49]:
```
Ellipsis
```

Part 2 Conclusions

We now have a good understanding of the taxi data we are working with. Visualizing large amounts of data can be a difficult task. One helpful tool is datashader (https://github.com/bokeh/datashader), a data rasterization pipeline for automating the process of creating meaningful representations of large amounts of data. Using the geopandas (http://geopandas.org/) package also makes working with geospatial data easier. We encourage you to explore these tools if you are interested in learning more about visualization!

Within our taxi data set, we have explored different features and their relationship with ride duration. Now, we are ready to incorporate more data in order to add to our set of features.

Please proceed to part 3 where we will be engineering more features and building our models using a processing pipeline.

Submission

You're almost done!

Before submitting this assignment, ensure that you have:

1. Restarted the Kernel (in the menubar, select Kernel→Restart & Run All)
2. Validated the notebook by clicking the "Validate" button.

Then,

1. **Submit** the assignment via the Assignments tab in **Datahub**
2. **Upload and tag** the manually reviewed portions of the assignment on **Gradescope**