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 = "Matthew Brennan"
COLLABORATORS = "Connor McCormick"
```

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
16434	8614300	2	2016-01-21 17:37:12	2016-01-21 18:37:56	2	10.89	-73.863403	40.769432
21929	7230200	2	2016-01-29 23:22:26	2016-01-29 23:31:23	2	1.00	-74.008087	40.739365
3370	9830300	2	2016-01-05 18:50:16	2016-01-05 18:56:00	2	0.56	-73.972923	40.755650
21975	7251500	2	2016-01-30 00:14:34	2016-01-30 00:47:13	1	6.65	-73.992027	40.718662
13758	6168000	1	2016-01-18 13:25:24	2016-01-18 13:38:51	1	2.10	-73.953125	40.784538

5 rows × 21 columns

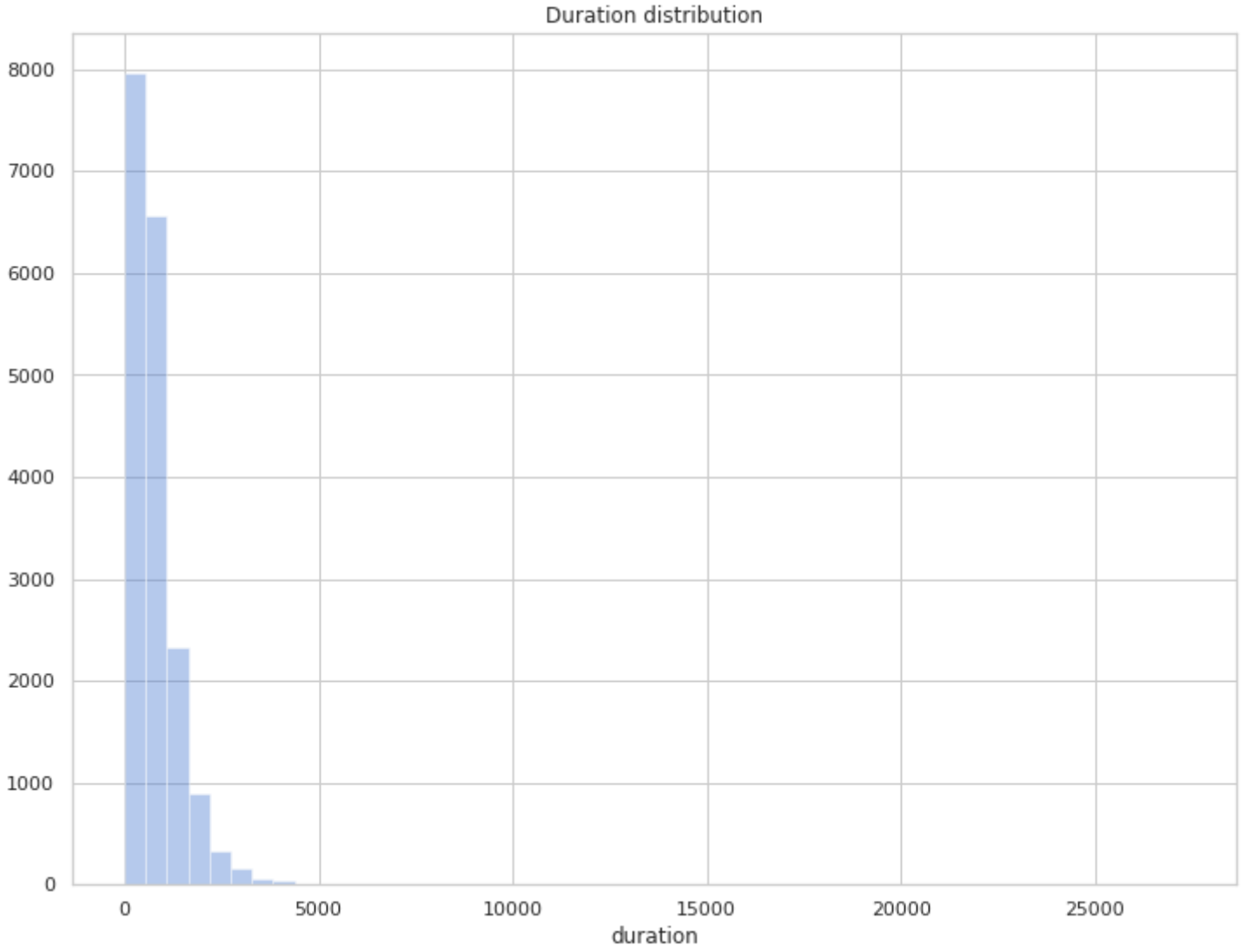
1: Data Overview

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

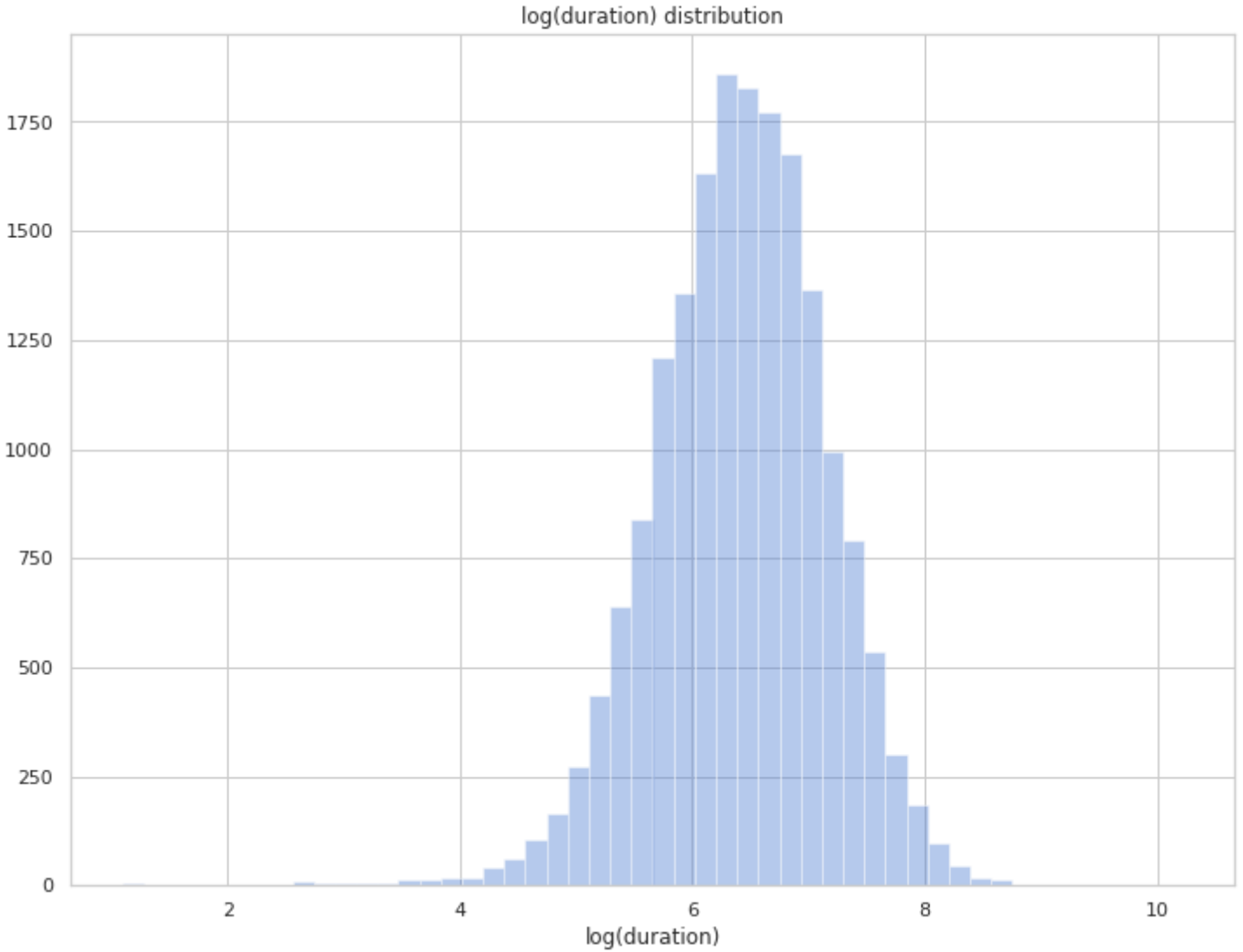
Let us take a closer look at the target `duration` variable. In the cell below, we plot its distribution using `sns.distplot` . This should give us an idea about whether we have some outliers in our data.

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');
```



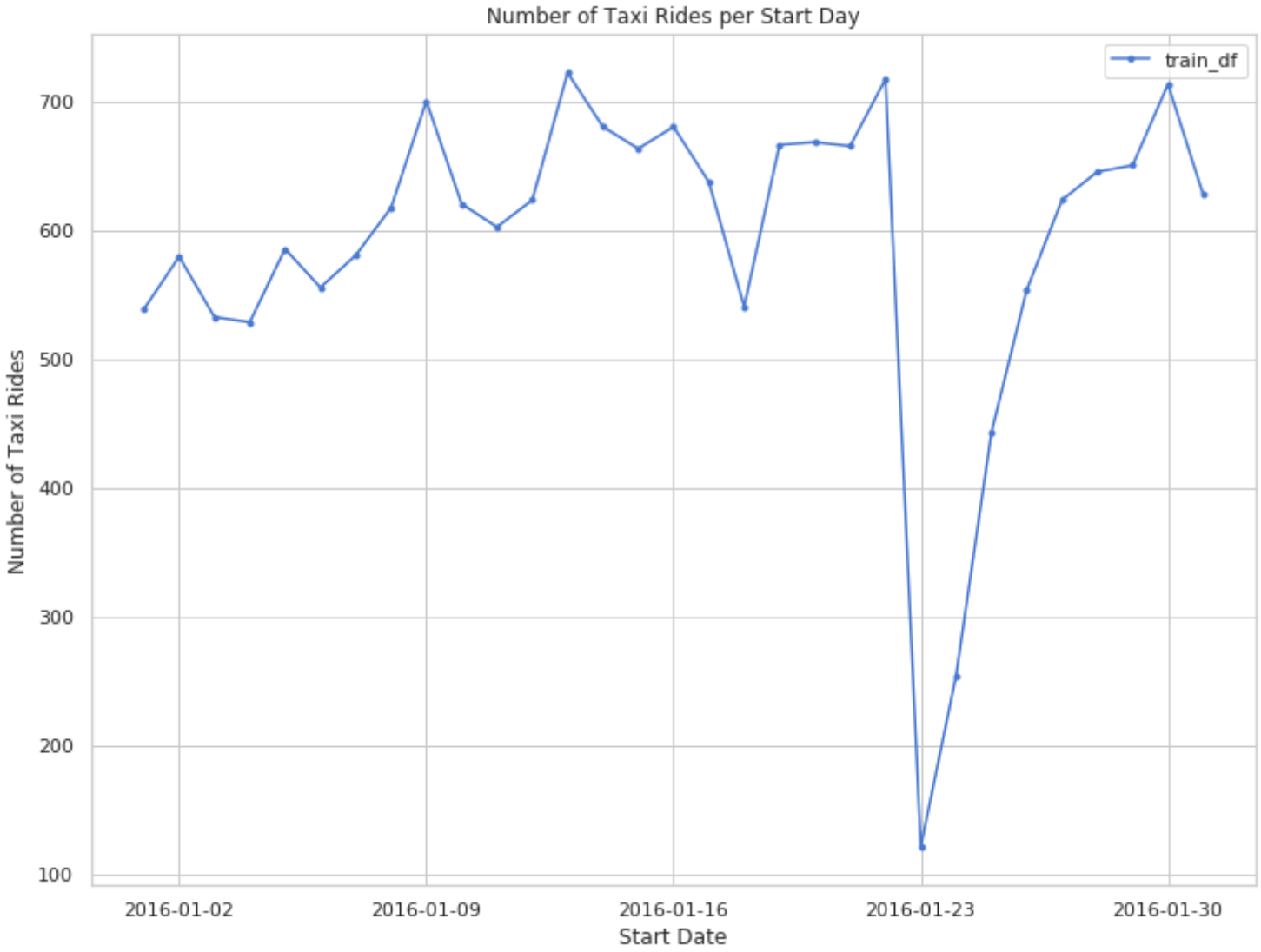
After transforming our data, we should immediately observe that we are dealing with what seems to be log-normal distribution for the target variable `duration`. We can see the behavior of shorter rides better, whereas before they were lumped in a bar near 0. This is a nice result, since it can facilitate 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]:
```
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]:
```
lowest_rides_date = "01-23-2016"

# YOUR CODE HERE
#raise NotImplementedError()

print(lowest_rides_date)
```

```
01-23-2016
```

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

Question 2b

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

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

According to Google there was an incredibly bad blizzard on this occasion which explains why there was s

"""

# YOUR CODE HERE
#raise NotImplementedError()

print(q2b_answer)
```

```
According to Google there was an incredibly bad blizzard on this occasion which explains why there was
such a drop in the amount of taxis. Further, this was also my 19th birthday which could account for an
other variable for the low amount of taxis as no one wanted to come.
```

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!1s0x89c2588f0 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.

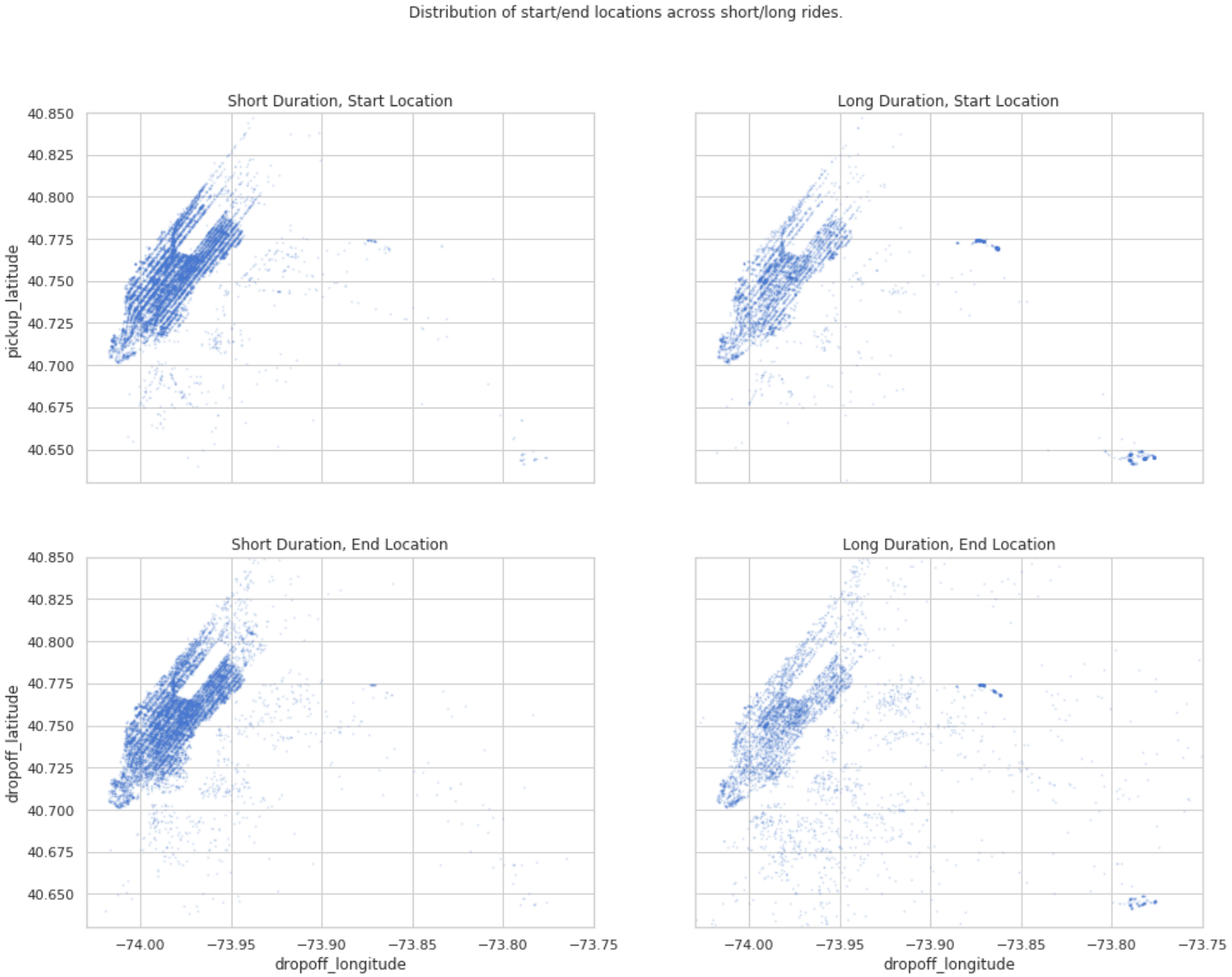
In [11]:
```
short_rides = train_df[train_df["duration"] <= 900] # rides less than or equal to 15 mins
long_rides = train_df[train_df["duration"] > 900] # rides more than 15 minutes
```

In [12]:
```
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]:
```
# 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=T
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);
```

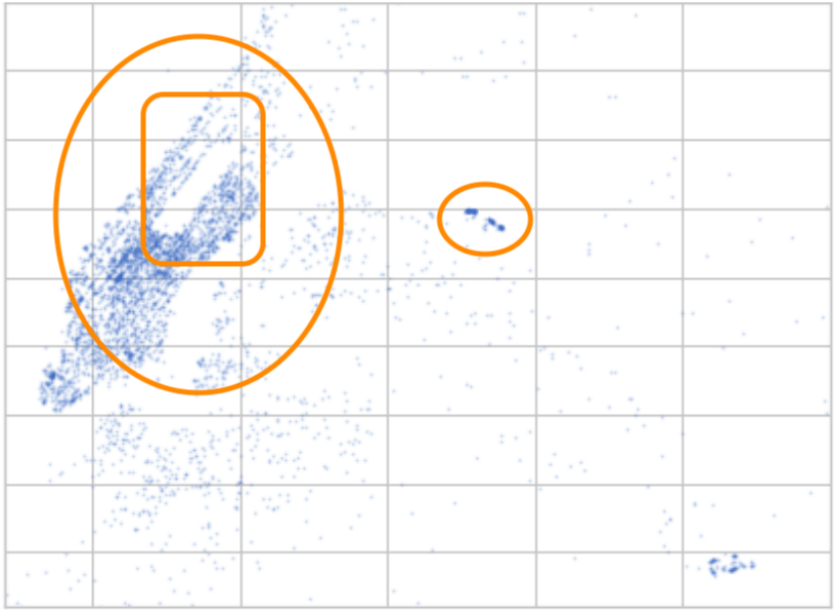

Distribution of start/end locations across short/long rides.
Short Duration, Start Location
Long Duration, Start Location
Short Duration, End Location
Long Duration, End Location
pickup_latitude
dropoff_latitude
dropoff_longitude
40.850
40.825
40.800
40.775
40.750
40.725
40.700
40.675
40.650
−74.00
−73.95
−73.90
−73.85
−73.80
−73.75

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!1s0x89c2 73.9712488) that may be useful.

In [14]:
```
q3a_answer = r"""

We can see that the plot as a whole refers to all of New York. The inner circled region corresponds to c

"""

# YOUR CODE HERE
#raise NotImplementedError()

print(q3a_answer)
```

```
We can see that the plot as a whole refers to all of New York. The inner circled region corresponds to
central park whereas the outercircle corresponds to the rest of Manhattan and then water on the white
areas. The circle on the far right is La Gaurdia airport which is a major destination for people flyin
g out of New York and therefore receives an immense amount of taxi traffic.
```

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"""

The outline in the top left with no points represents Central Park where there is no available rodes fo

"""

# YOUR CODE HERE
#raise NotImplementedError()

print(q3b_answer)
```

```
The outline in the top left with no points represents Central Park where there is no available rodes f
or taxis to pickup or dropoff people. This is not a mistake in the data but in fact reflects the adroi
tness of the code to pinpoint locations.
```

Question 3c

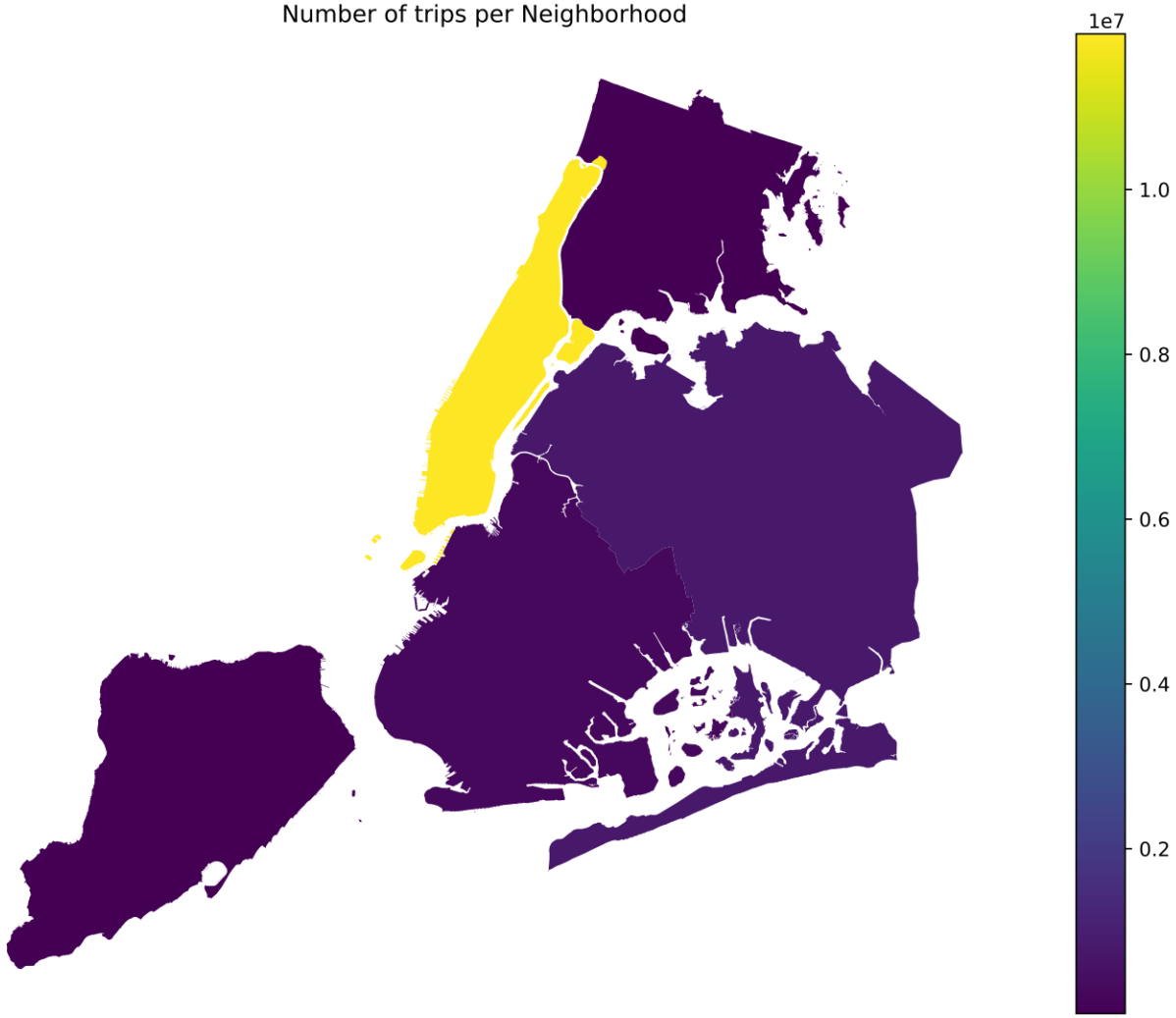
What observations/conclusions do you make based on the scatter plots above? In particular, how are trip duration and pickup/dropoff location related?

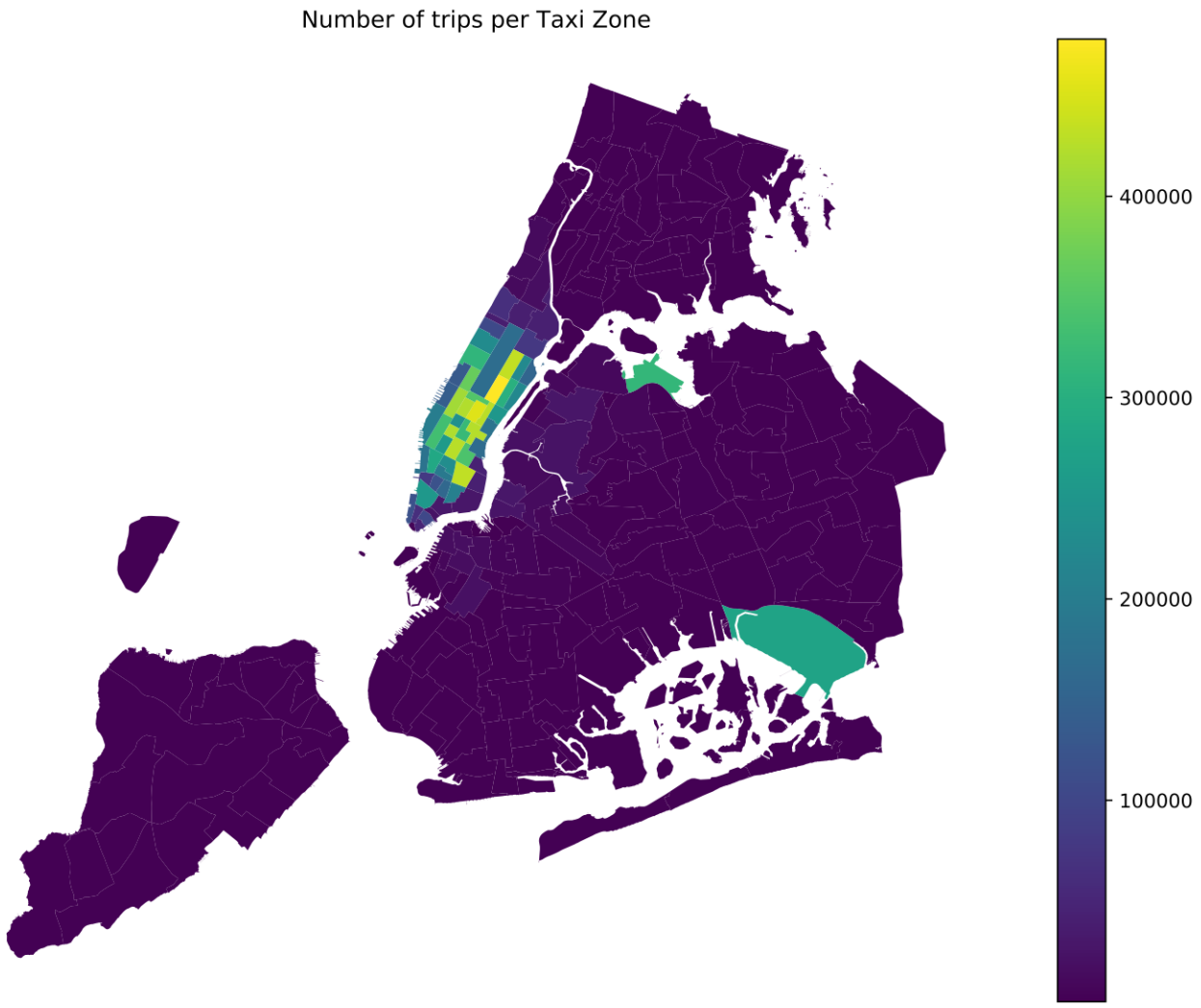
In [16]:
```
q3c_answer = r"""

Trip duration and pickup/dropoff location highlight the immense amount of short duration trips that occu

"""

# YOUR CODE HERE
#raise NotImplementedError()

print(q3c_answer)
```

```
Trip duration and pickup/dropoff location highlight the immense amount of short duration trips that oc
cur within Manahattan specifically. We can observe in the short duration plots that people tend to tak
e taxis in abundance all over the city and that are more inclined to take long distance taxis away fro
m the city to La Guardia as opposed to any other area in the region. The lesser amount of long duratio
n taxis centered in Manahttan reflects the lack of ambition of New Yorker to leave Manhattan when they
are already there, predominantly because New York has everything that is needed.
```

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/).


Number of trips per Neighborhood
1e7
1.0
0.8
0.6
0.4
0.2



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

In [19]:
```
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]:
```
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` .

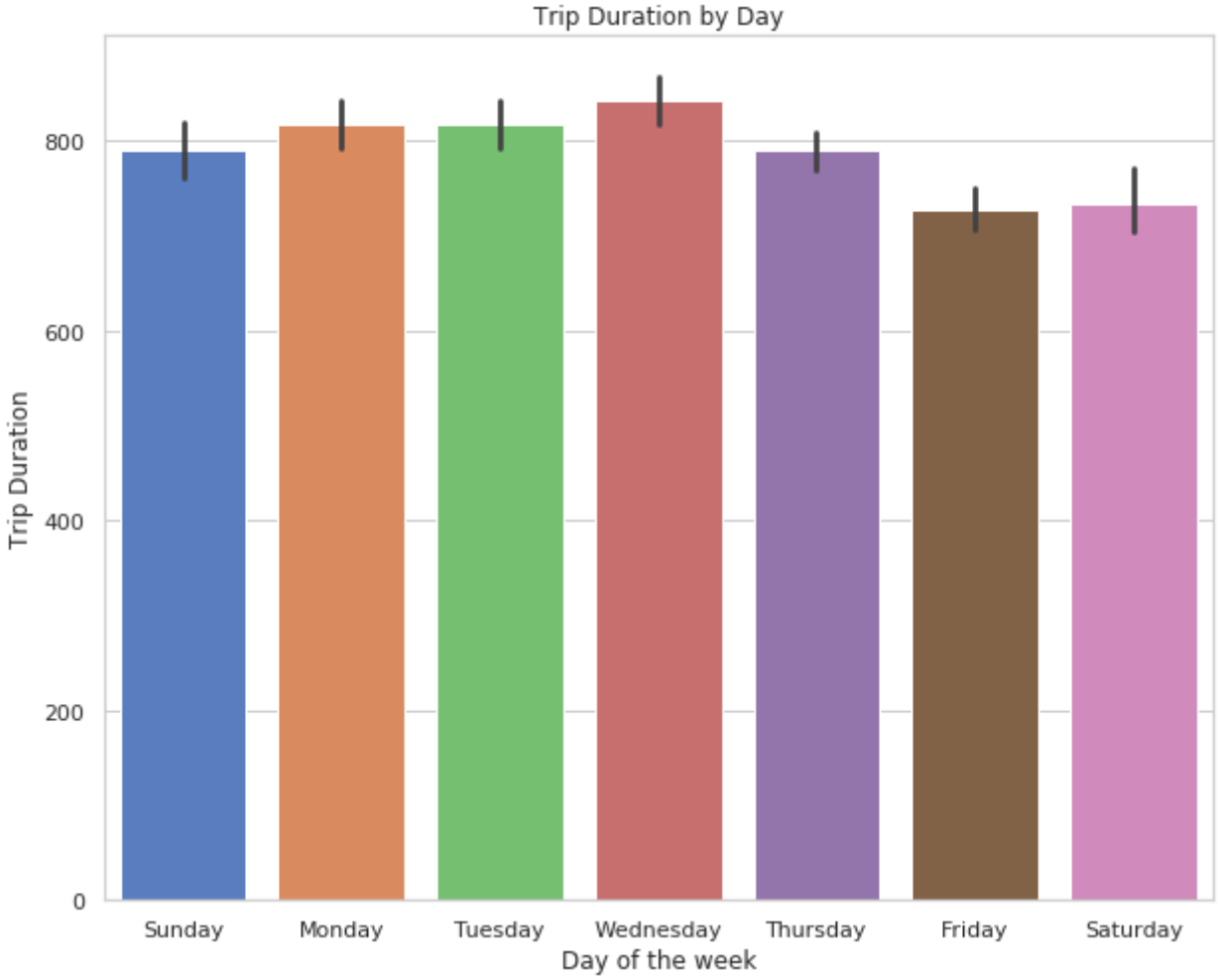
You can use the same column multiple times, but if the plots are redundant you will not receive full credit. This will be graded based on how informative each plot is and how "good" the visualization is (remember what good/bad visualizations look like for different kinds of data!).

Visualization 1

In [21]:
```
# Visualization 1
fig, ax = plt.subplots()
sns.lineplot('hour', 'duration', data = train_df)
plt.xlabel('hour of the day')
plt.ylabel('duration')
plt.title('Duration of taxi rides for each hour');

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

Visualization 2

In [22]:
```
# Visualization 2
fig, ax = plt.subplots()
ax = sns.barplot(x="day_of_week", y="duration", data=train_df)
fig.set_size_inches(10,8)
plt.xlabel('Day of the week')
plt.ylabel('Trip Duration')
plt.title('Trip Duration by Day')
ax.set_xticklabels(['Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday']);

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



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 [23]:
```
q4b_answer = r"""
For plot 1, I opted to utilize the hour feature in order to compare the duration of taxi rides per hour

"""

# YOUR CODE HERE
#raise NotImplementedError()

print(q4b_answer)
```

```
For plot 1, I opted to utilize the hour feature in order to compare the duration of taxi rides per hou
r. The reasoning behind doing this was to identify the highest duration times of taxi usage per day an
d to therefore see if they corresponded with normal times of commuting. In addition, this could enable
me to examine the odd hours of the day to see the discrepancies in price or distance traveled on these
occasions as likely a lesser supply of taxis would cause prices to increase. The visualization method
I used was a Lineplot because this allows me to accurately gauge the distribution across hours and to
identify the most active times. For visualization 2, I decided to use day of the week as the feature i
n order to determine if longer drives of taxis were more prevalant on certain days. For this reason I
decided to examine the train_df dataframe rather than the others in order to really focus on all of th
e results so that we could identify standout days. In constructing the barplot for this feature, this
structure made the most sense for comparison as I am able to easily compare the days of the week as no
rmally nominal data is represented the easiest in a barplot.
```

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?

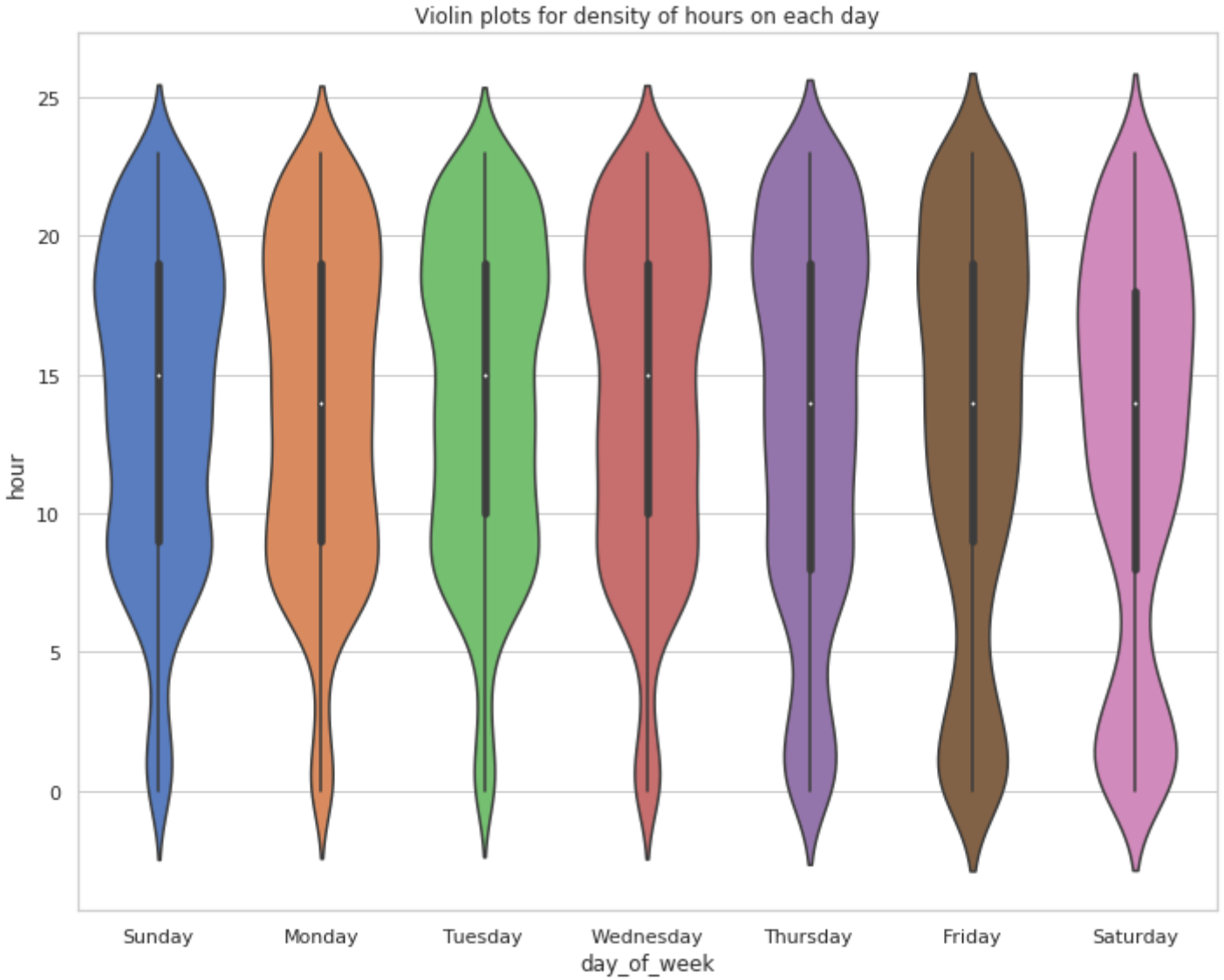
In [24]:
```
q4c_answer = r"""
The duration of the trip appears to be at its highest on Wednesdays when it is in the early afternoon a

"""

# YOUR CODE HERE
#raise NotImplementedError()

print(q4c_answer)
```

```
The duration of the trip appears to be at its highest on Wednesdays when it is in the early afternoon
according to the temporal aspects that I accounted for. Additionally, we see that the duration time de
creases significantly on the weekends and during the early morning and late night hours. Although this
information is useful, we really cannot draw that many significant conclusions about the overall durat
ion from this information other than modelling at peak hours where we know duration will be significan
t, and yet this also feels like guessing rather than drawing grand conclusions.
```

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 [25]:
```
fig, axes = plt.subplots(1, 1, figsize=(10, 8))
days_of_week = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"]
sns.violinplot(x = 'day_of_week', y = 'hour', data = train_df)
axes.set_xticklabels(['Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday']);
plt.title("Violin plots for density of hours on each day")
# YOUR CODE HERE
#raise NotImplementedError()

plt.tight_layout();
```



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 [26]:
```
q4e_answer = r"""
We observe that during rush hours the density is a little heightened relative to the other hours and tha
"""

# YOUR CODE HERE
#raise NotImplementedError()

print(q4e_answer)
```

```
We observe that during rush hours the density is a little heightened relative to the other hours and t
hat this feature is most noticeable during the work week. Also, we can see that on the weekends the am
ount of late night (12 am, 1 am etc,..) taxi usage is heightened relative to the work week. In talking
with a lay person, I would explain that the fatter the violin plot at certain points the more taxi usa
ge is occuring relative to other times and therefore would explain the aforementioned aspects with reg
ards to fat and skinny parts of each violin plot. The pattern of interest to me is that the violinplot
literally resemebles the day of a human: the time some people wake up versus when we know everyone is
basically awake, the lack of taxi usage needed while people are at work, when people get off work, and
when people start to hibernate in their residences at night.
```

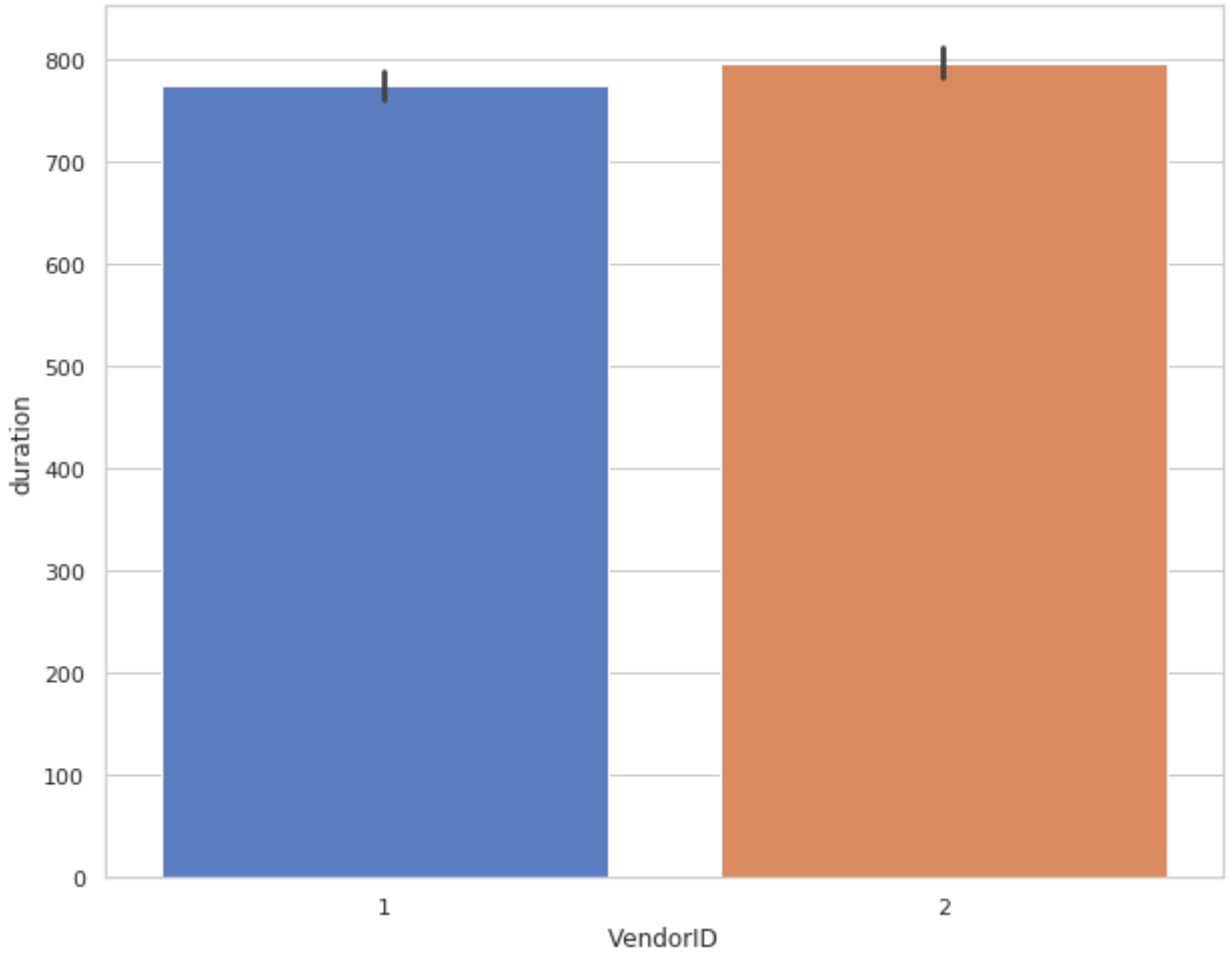
5: Vendors

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 [27]:
```
# Visualization
fig, ax = plt.subplots()
ax = sns.barplot(x="VendorID", y="duration", data=train_df)
fig.set_size_inches(10,8)
# YOUR CODE HERE
#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 [28]:
```
q5b_answer = r"""

Since VendorID is ordinal data, I used a bar chart in order to determine whether the VendorID had any a


"""

# YOUR CODE HERE
#raise NotImplementedError()

print(q5b_answer)
```

```
Since VendorID is ordinal data, I used a bar chart in order to determine whether the VendorID had any
affect on the duration of the trip. In general, we see that VendorID 2 has longer durations, yet we ca
nnot conclude that this is necessarilly a useful feature as far as our model as they appear to not dif
fer by too much. Therefore, I would conclude that vendor_id is not a useful feature and that there are
likely other features that appear less arbitrary.
```

Question 5c

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

In [29]:
```
q5c_answer = r"""
I do not think that vendor_id will help us understand duration. Although there is some discrepancy in tl
"""

# YOUR CODE HERE
#raise NotImplementedError()

print(q5c_answer)
```

```
I do not think that vendor_id will help us understand duration. Although there is some discrepancy in
the duration for vendor 1 and vendor 2, this overall spike does not cause me to beleive that this is a
nything other than coincidence. Logically, I assume that people either take whatever taxi is closest o
r call in an taxi ahead of time, but do not base their distance expectations on the vendor and therefo
re the duration. Further, I assume driving in Manhattan is virtually the same for all taxi drivers and
that one vendor does not have some abnormal advantage over the other.
```

6: Distance features

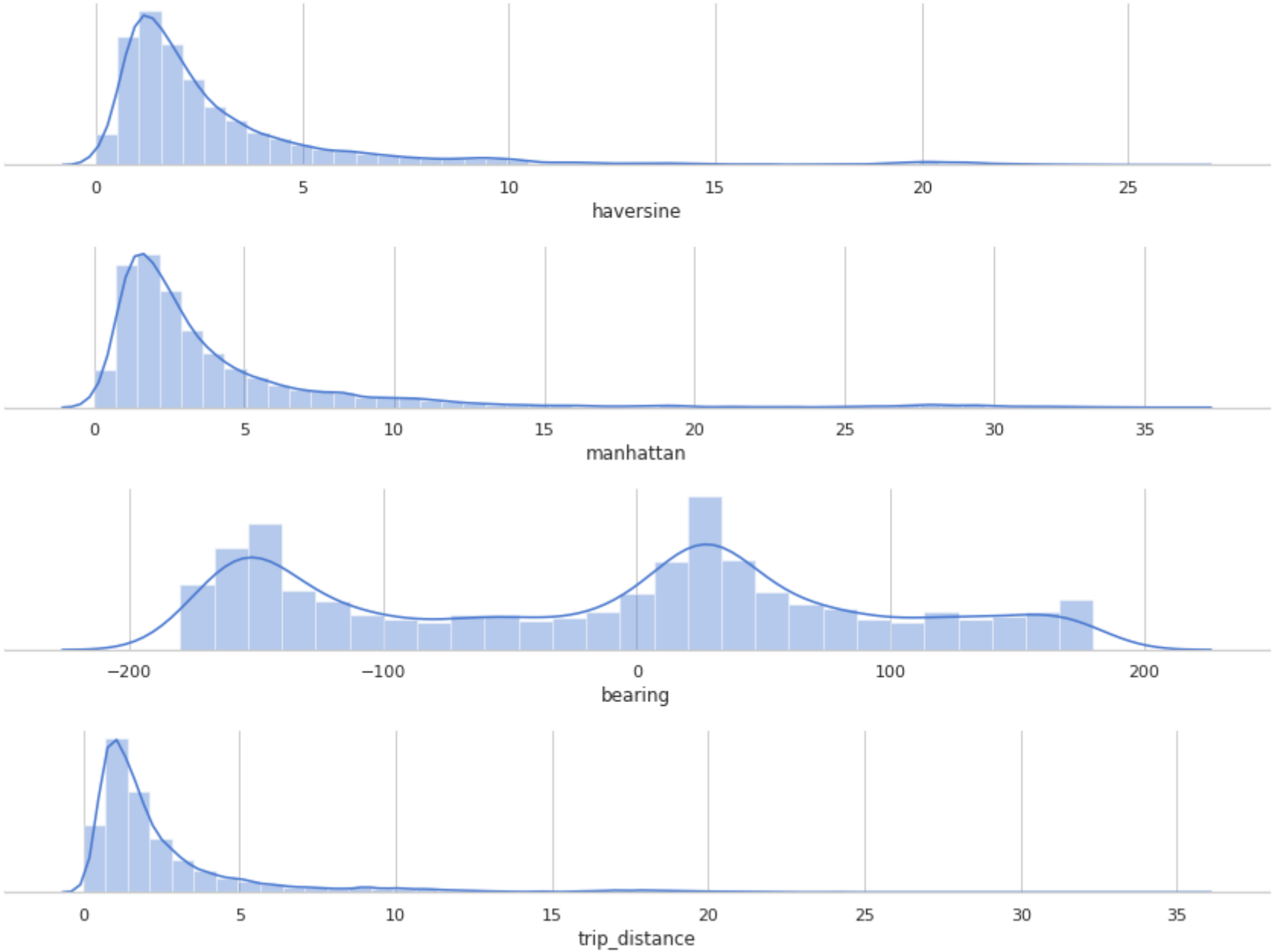
We can also use the coordinates information to compute distance features. This will allow us to compute speed related features.
We will compute the haversine (https://en.wikipedia.org/wiki/Haversine_formula) distance, the manhattan (https://en.wikipedia.org/wiki/Taxicab_geometry) distance and the bearing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

In [30]:
```
# 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 [31]:
```
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 [32]:
```
train_df = add_distance_columns(train_df)
short_rides = add_distance_columns(short_rides)
long_rides = add_distance_columns(long_rides)
```

In [33]:
```
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();
```


0
5
10
15
20
25
haversine
0
5
10
15
20
25
30
35
manhattan
−200
−100
0
100
200
bearing
0
5
10
15
20
25
30
35
trip_distance

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) has the answer, although it may take some digging. Alternatively, try to look at a map of Manhattan.

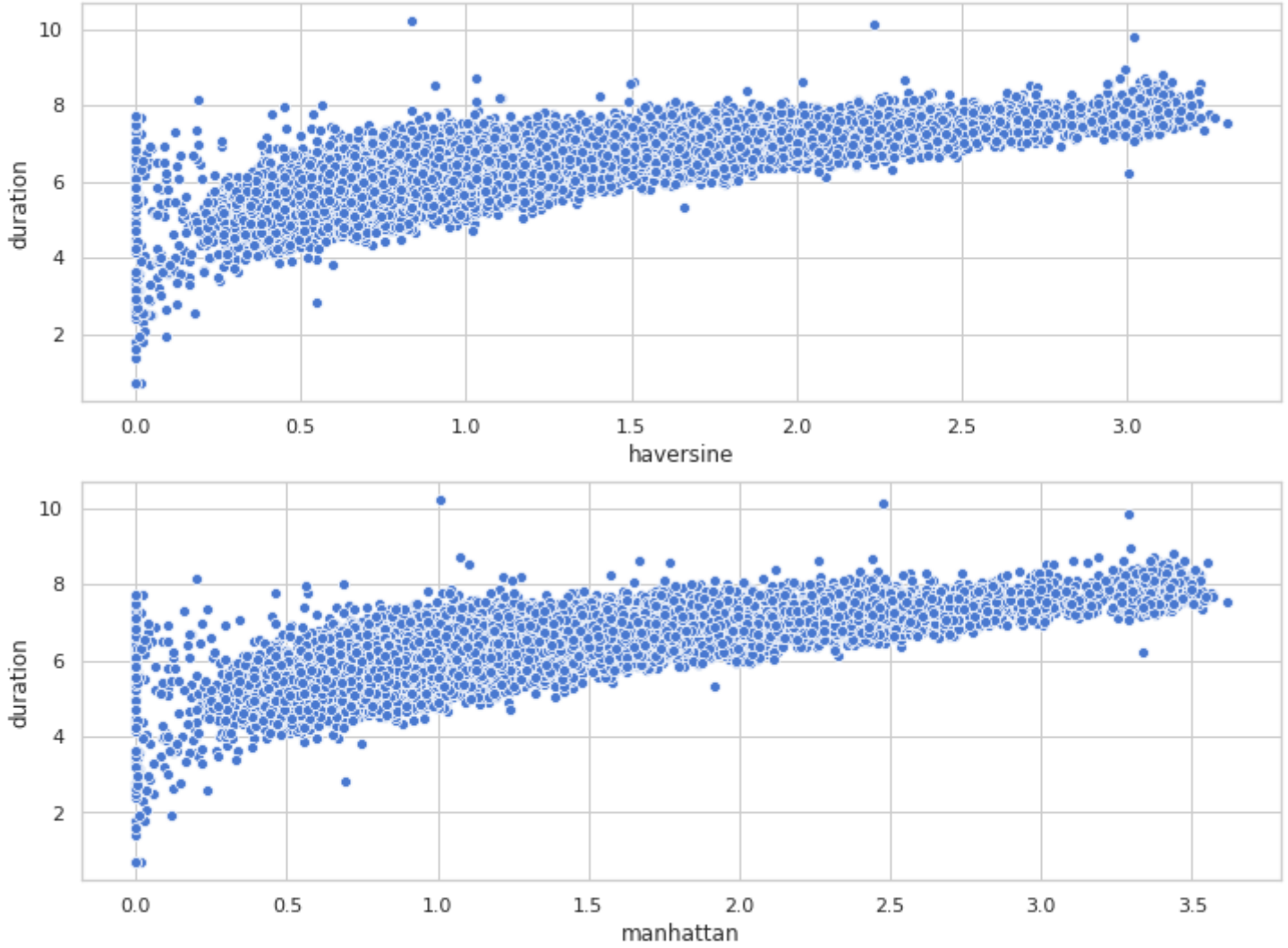
In [34]:
```
q6a_answer = r"""
The shape of Manhattan aligns with that of 30 and -150 degrees, meaning that in order to go further alo
"""

# YOUR CODE HERE
#raise NotImplementedError()

print(q6a_answer)
```

```
The shape of Manhattan aligns with that of 30 and -150 degrees, meaning that in order to go further al
ong the stretch of Manhattan one would have to head in either of these directions. This coincidentally
coincides with the geographical orientation of manhattan as far as angles on a map. With this being sa
id, in order to travel from time's square to central park one would have to go in the initial directio
n of 30 degrees and the opposite route would occur at -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 [35]:
```
# Visualization
# YOUR CODE HERE
fig, axes = plt.subplots(2, 1, figsize=(12, 9))
sns.scatterplot(np.log(train_df['haversine']+1), np.log(train_df['duration']), ax=axes[0]);
sns.scatterplot(np.log(train_df['manhattan']+1), np.log(train_df['duration']), ax=axes[1]);

#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.

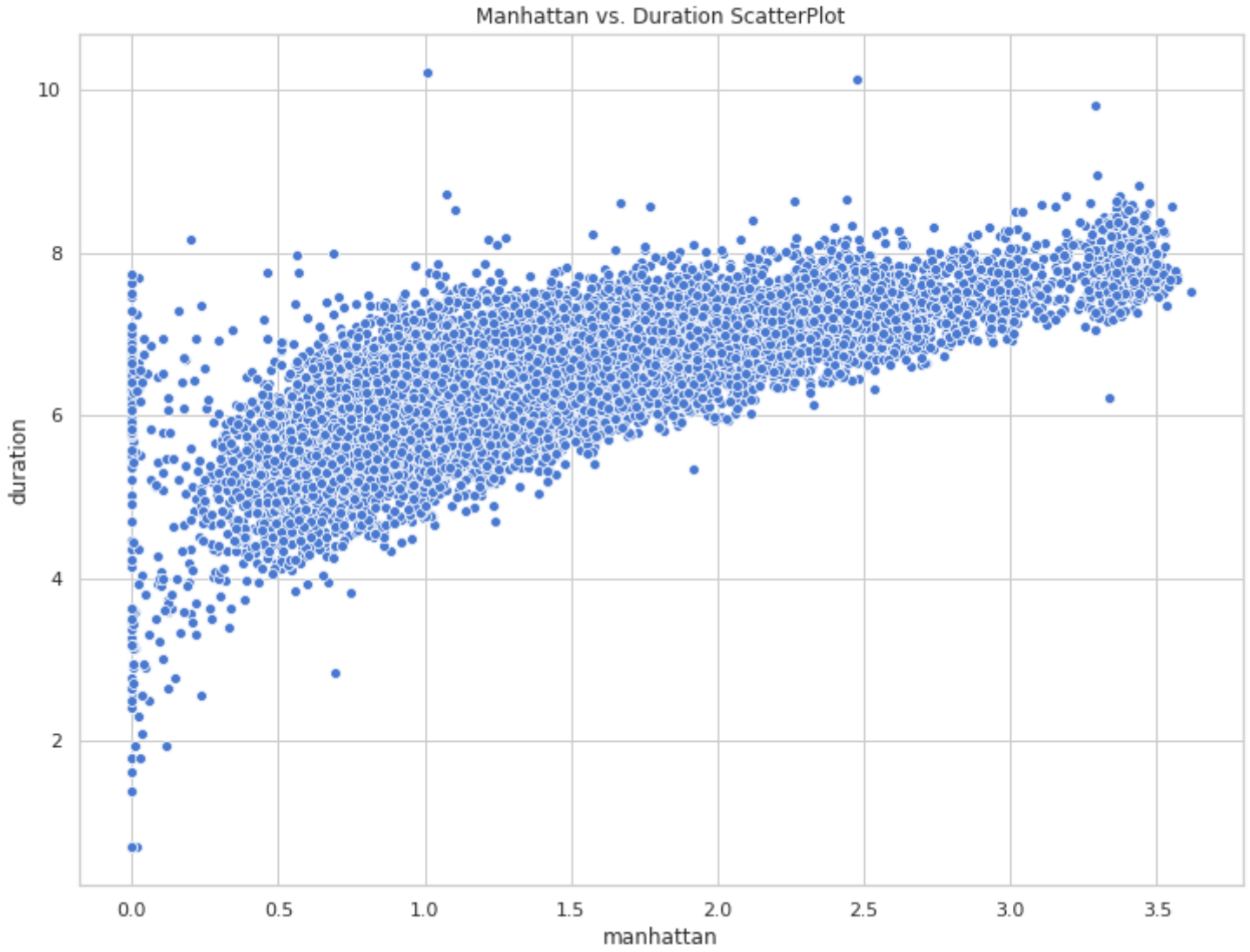
In [36]:
```
q6c_answer = r"""

For a comparison of quantitative data, a scatterplot is a very appropriate means to identify this relat
"""

# YOUR CODE HERE
#raise NotImplementedError()

print(q6c_answer)
```

```
For a comparison of quantitative data, a scatterplot is a very appropriate means to identify this rela
tionship. From the scatterplot we are definitely able to see that as the manhattan/haversine distance
increases, so does that of the duration of the trip. This trend is evident in the slope of the overall
plot which suggests that the relationship is positive.
```

Question 6d

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

In [37]:
```
# YOUR CODE HERE
fig, axes = plt.subplots(1, 1, figsize=(12, 9))
sns.scatterplot(np.log(train_df['manhattan']+1), np.log(train_df['duration']))
plt.title("Manhattan vs. Duration ScatterPlot");
#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 [38]:
```
q6e_answer = r"""

All of the points where the Manhattan distance is 0 yet the duration of the taxi ride is greater than 0
"""

# YOUR CODE HERE
#raise NotImplementedError()

print(q6e_answer)
```

```
All of the points where the Manhattan distance is 0 yet the duration of the taxi ride is greater than
0 would likely be the result of human error. The reason I assume stems from the lack of feasibility th
at one moves no distance from pickup location to dropoff location intentionally in a taxi. Therefore,
this is likely the result of either human error or simply one got into a taxi and then opted to not mo
ve because of waiting for someone or because of a change of heart yet still got penalized for the time
and thus is in the records.
```

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):

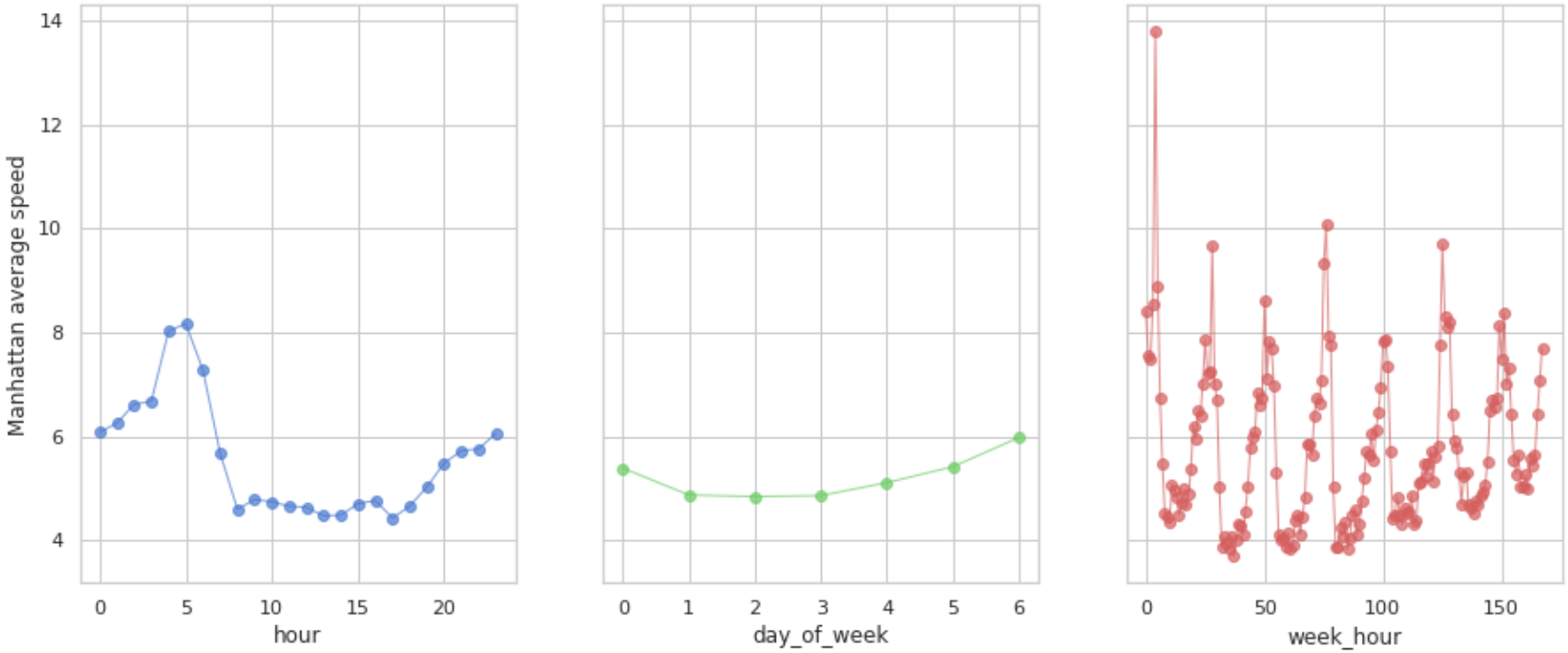
- Average of `avg_speed_h` per cluster.
- Average of `duration` per cluster.
- Average of `avg_speed_h` per cluster and hour.
- Average of `duration` per cluster and hour.
- In-cluster flow of trips for 60 min period.
- Out-cluster flow of trips for 60 min period.

In [39]:
```
# 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 [40]:
```
train_df['avg_speed_m'].describe()
```

Out[40]:
```
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 [41]:
```
# 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.

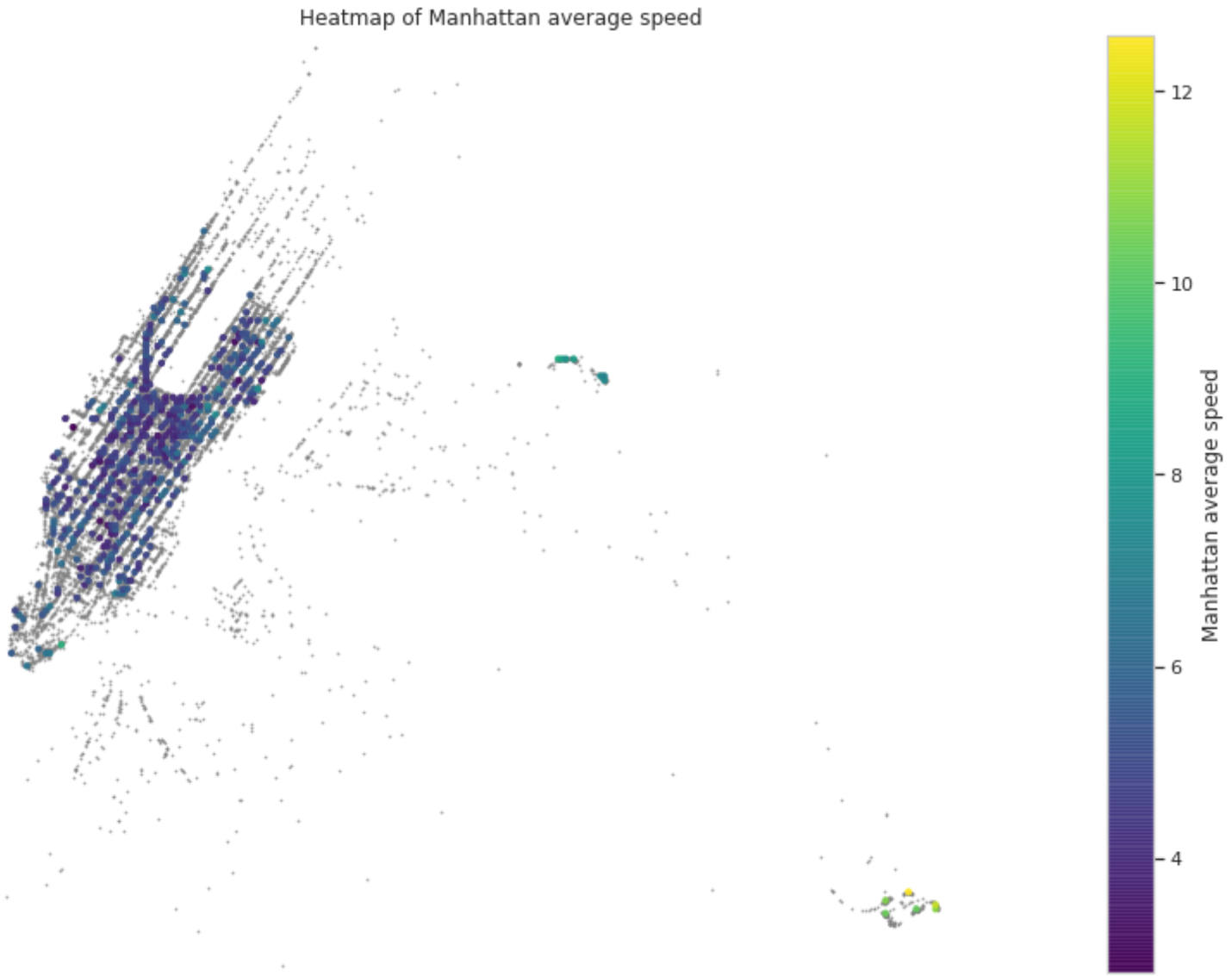
In [42]:
```
q7a_answer = r"""
The graph of Manhattan avg. speed vs. hour reflects how fast the individual is able to cover distance i

"""

# YOUR CODE HERE
#raise NotImplementedError()

print(q7a_answer)
```

```
The graph of Manhattan avg. speed vs. hour reflects how fast the individual is able to cover distance
in a taxi at specific hours and thus we can see that at 5 am one is able to get around the city the qu
ickest. Further, we can see that the hours of the day that correspond with the highest manhattan avera
ge speed are very early in the morning or very late at night. Additionally, we see that the weekends t
end to have higher avg. speeds overall which is likely due to less traffic due to people not needing t
o leave for work or other necessary obligations.
```

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 [43]:
```
# 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 [44]:
```
# 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 [45]:
```
q7b_answer = r"""
We can evidently see that the middle of Manhattan possesses very low average speeds relative to the sur
"""

# YOUR CODE HERE
#raise NotImplementedError()

print(q7b_answer)
```

```
We can evidently see that the middle of Manhattan possesses very low average speeds relative to the su
rrounding areas likely due to the attraction of tourism, the plethora of people that live there, and t
he abundance of jobs available on Wall Street. If one compares this too other regions that are not in
Manhattan, yet are still destinations for taxis,(i.e La Gaurdia) we see that there is a significantly
higher average speed there. From this we can conclude that spatial clustering would provide insight in
to the duration due to the likely slower traffic that occurs in the middle of Manhattan, yet we also m
ust be cognizant of the closer proximity between locations within Manhattan. However, I do feel as tho
ugh the benefits of this for this model would outweigh the impact of the confounding factors.
```

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 [46]:
```
Path("data/part2").mkdir(parents=True, exist_ok=True)
data_file = Path("data/part2", "data_part2.hdf") # Path of hdf file
...
```

Out[46]:
```
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**