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 = "Benjamin Liu"
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_dis
16434	8614300	2	2016-01-21 17:37:12	2016-01-21 18:37:56	2	
21929	7230200	2	2016-01-29 23:22:26	2016-01-29 23:31:23	2	
3370	9830300	2	2016-01-05 18:50:16	2016-01-05 18:56:00	2	
21975	7251500	2	2016-01-30 00:14:34	2016-01-30 00:47:13	1	
13758	6168000	1	2016-01-18 13:25:24	2016-01-18 13:38:51	1	

5 rows × 21 columns

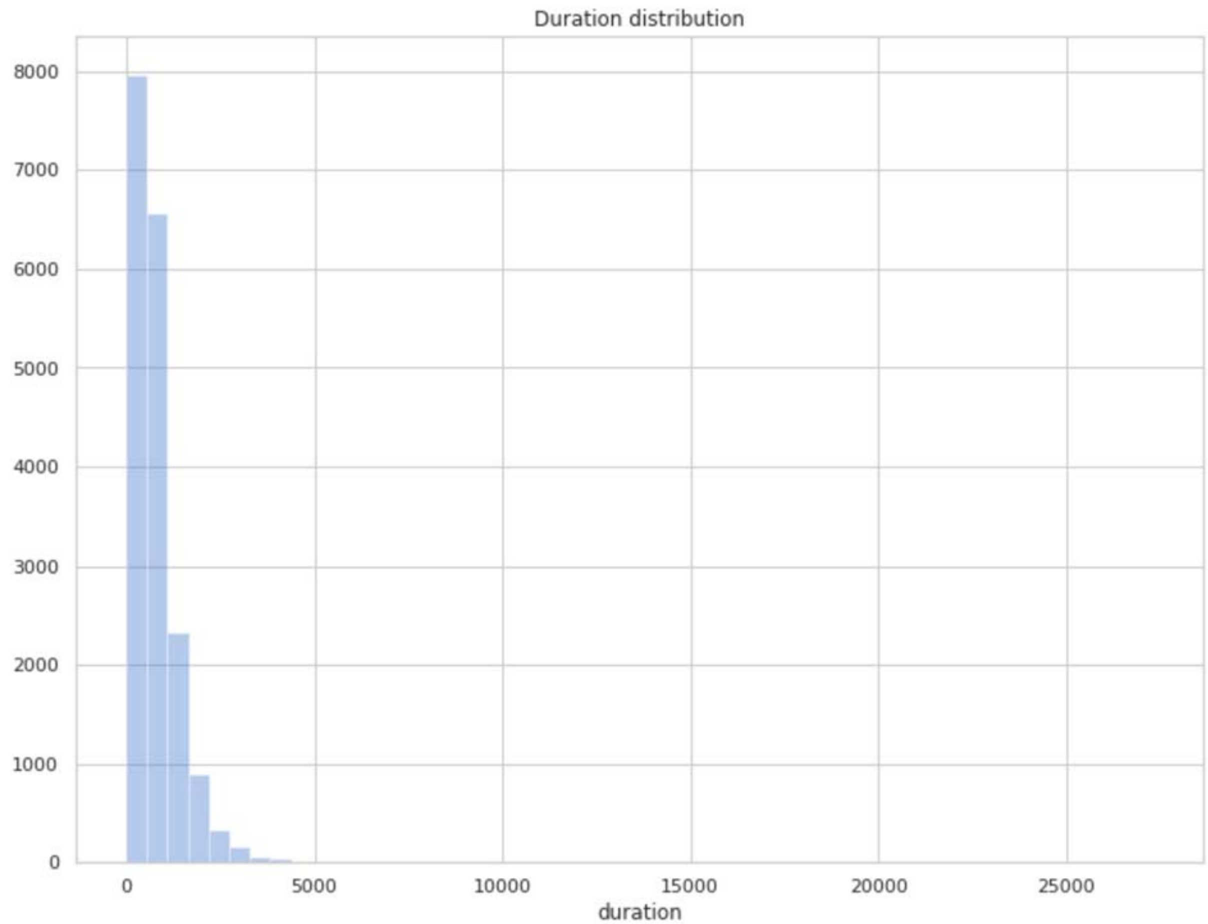
1: Data Overview

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

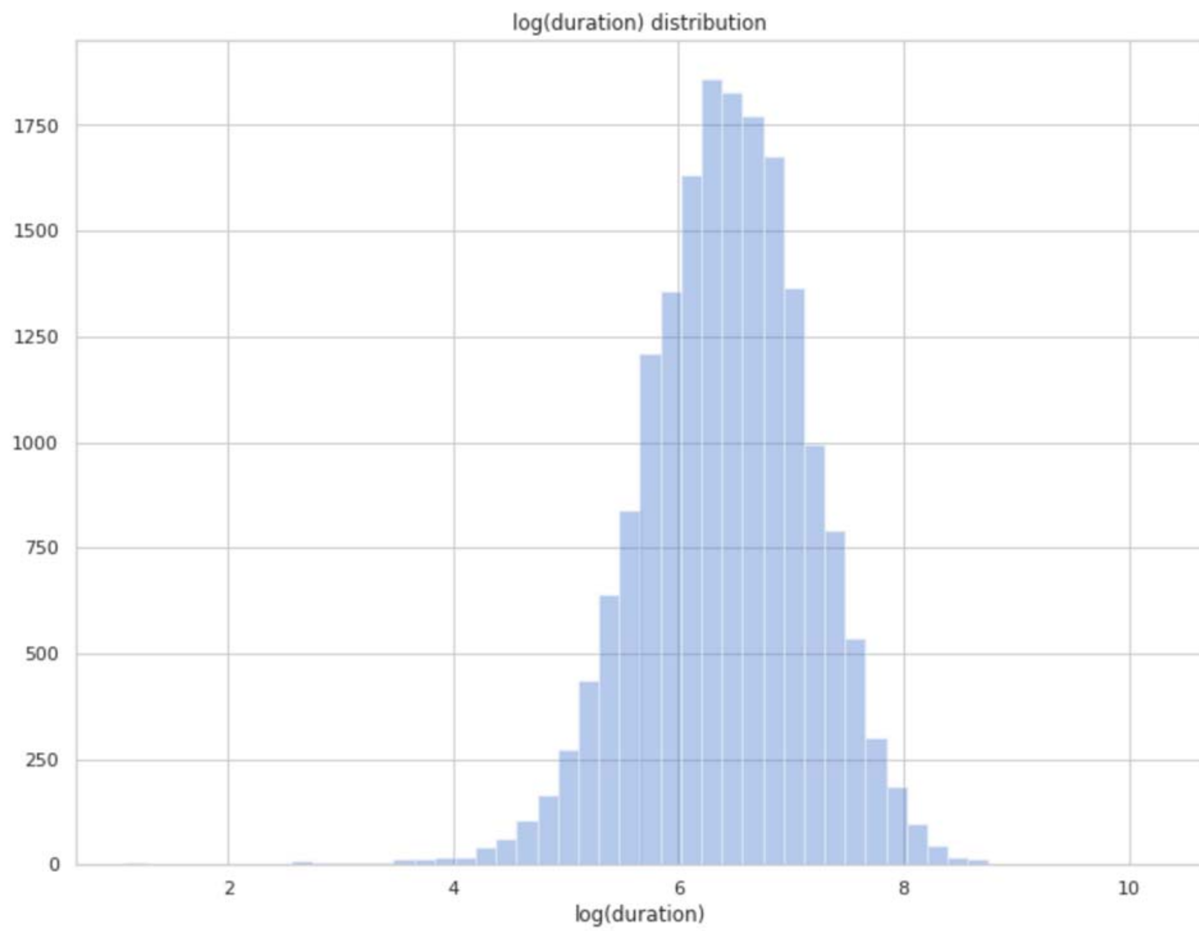
Let us take a closer look at the target `duration` variable. In the cell below, we plot its distribution using `sns.distplot` . This should give us an idea about whether we have some outliers in our data.

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

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

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



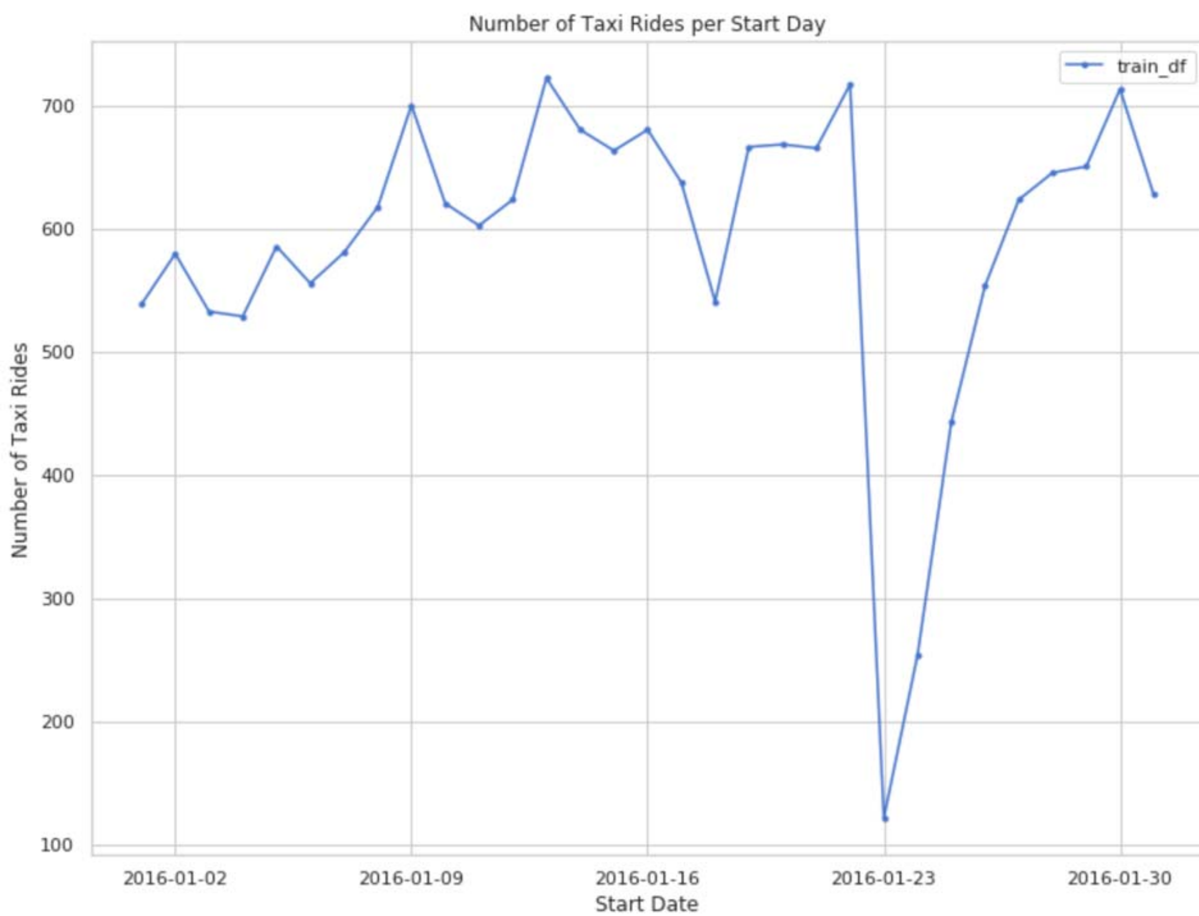
After transforming our data, we should immediately observe that we are dealing with what seems to be log-normal distribution for the target variable `duration`. We can see the behavior of shorter rides better, whereas before they were lumped in a bar near 0. This is a nice result, since it can facilitate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

# YOUR CODE HERE
# raise NotImplementedError()
### BEGIN Solution
lowest_rides_date = (tmp_train[tmp_train == min(tmp_train)]
                     .index[0]
                     .strftime("%m-%d-%Y"))
### END Solution

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"""

That is due to January 2016 United States blizzard.

"""

# YOUR CODE HERE
# raise NotImplementedError()

print(q2b_answer)
```

```
That is due to January 2016 United States blizzard.
```

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/da 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
long_rides = train_df[train_df["duration"] > 900] # rides more than 15 minutes
```

In [12]:
```
len(short_rides)
```

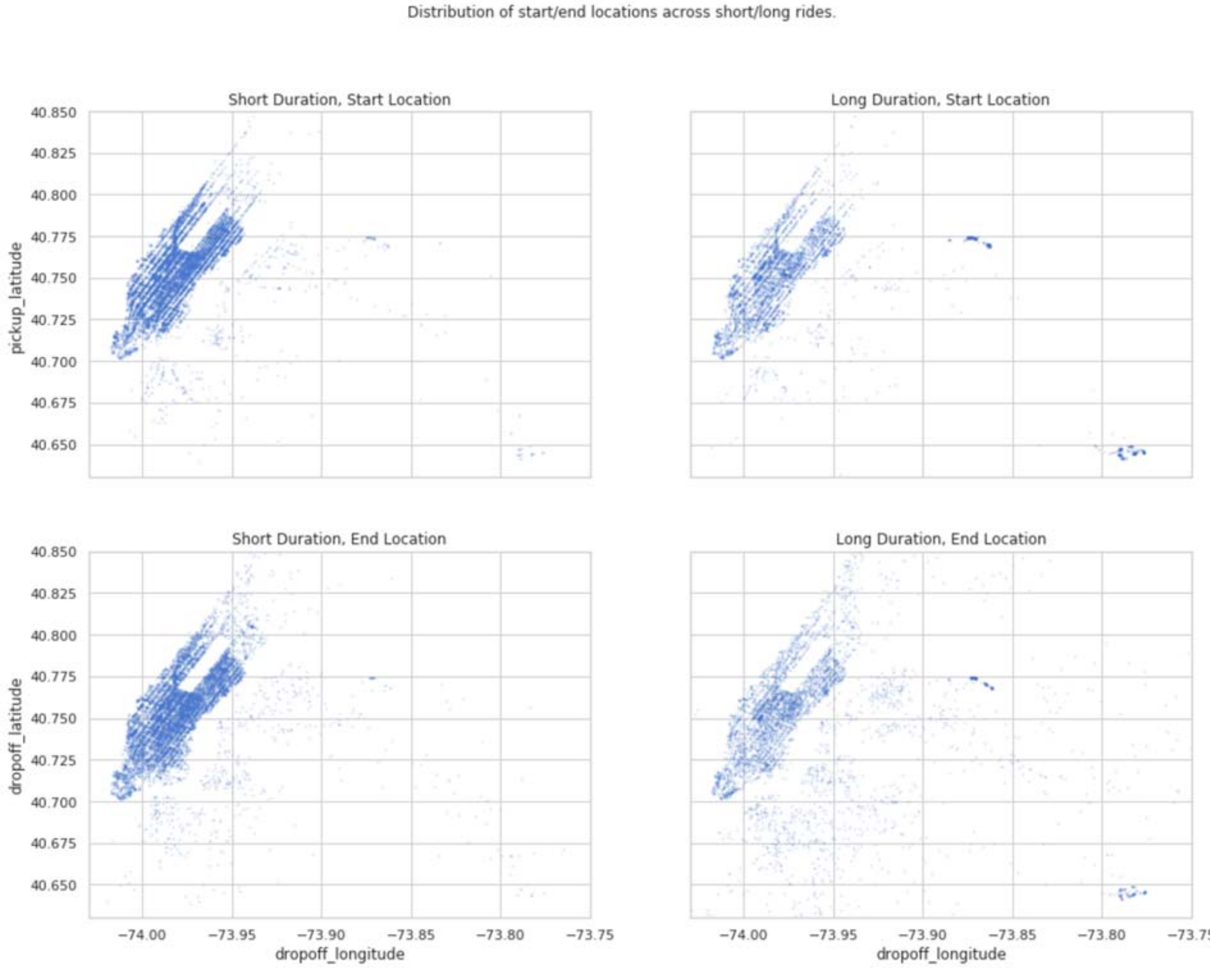
Out[12]:
```
12830
```

In [13]:
```
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 [14]:
```
# 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
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, Star
long_rides.plot(kind = "scatter", x = "pickup_longitude", y = "pickup_latitude",
                    ax = ax2, alpha = alpha, s = s, title='Long Duration, Start 
short_rides.plot(kind = "scatter", x = "dropoff_longitude", y = "dropoff_latitud
                     ax = ax3, alpha = alpha, s = s , title='Short Duration, End
long_rides.plot(kind = "scatter", x = "dropoff_longitude", y = "dropoff_latitude'
                    ax = ax4, alpha = alpha, s = s, title='Long Duration, End Lo


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


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

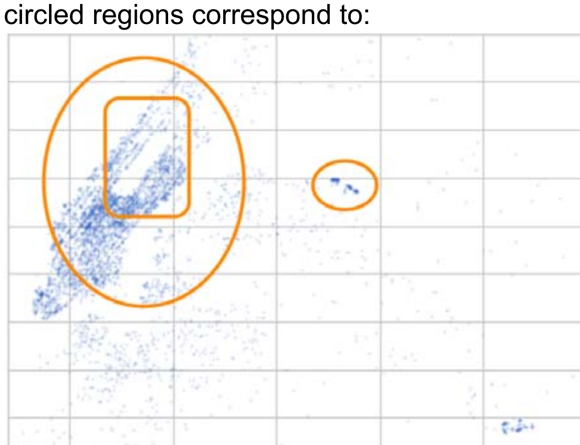


Question 3a

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,-74.0394431,12 73.9712488)] that may be useful.

In [15]:
```
q3a_answer = r"""

The circled region corresponds to Manhanttan, NYC. And the smaller circle corresp

"""

# YOUR CODE HERE
# raise NotImplementedError()

print(q3a_answer)
```

```
The circled region corresponds to Manhanttan, NYC. And the smaller circle corre
sponds to LaGuardia Airport, NYC.
```

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 [16]:
```
q3b_answer = r"""

The rectangular area corresponds to the big central park in the middle of Manhant

"""

# YOUR CODE HERE
# raise NotImplementedError()

print(q3b_answer)
```

```
The rectangular area corresponds to the big central park in the middle of Manha
nttan. Since it makes sense that there will be no taxi pickup in the middle of
a park, it is not an error/mistake in the data.
```

Question 3c

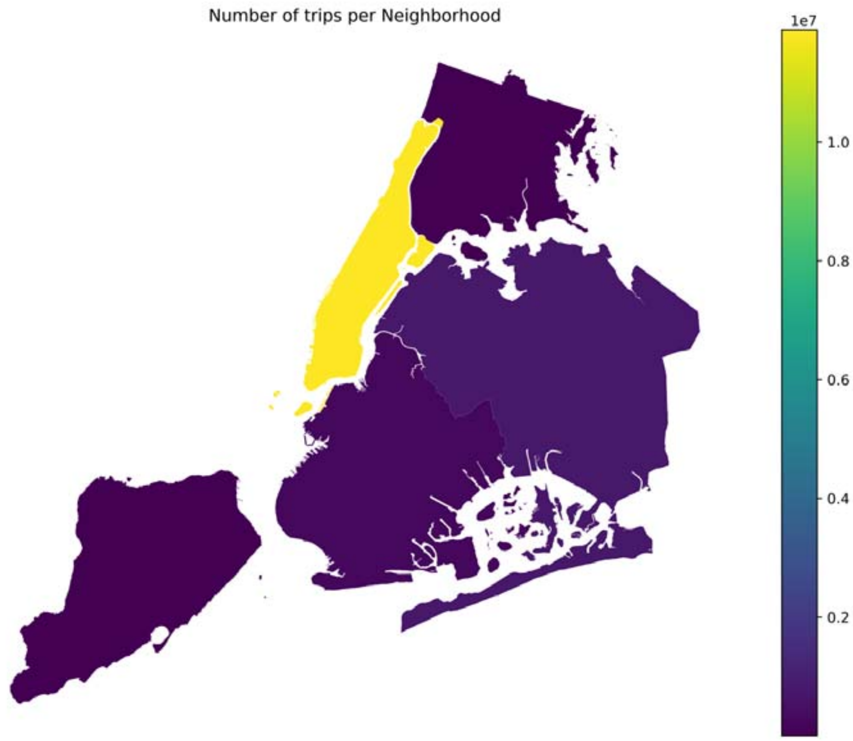
What observations/conclusions do you make based on the scatter plots above? In particular, how are trip duration and pickup/dropoff location related?

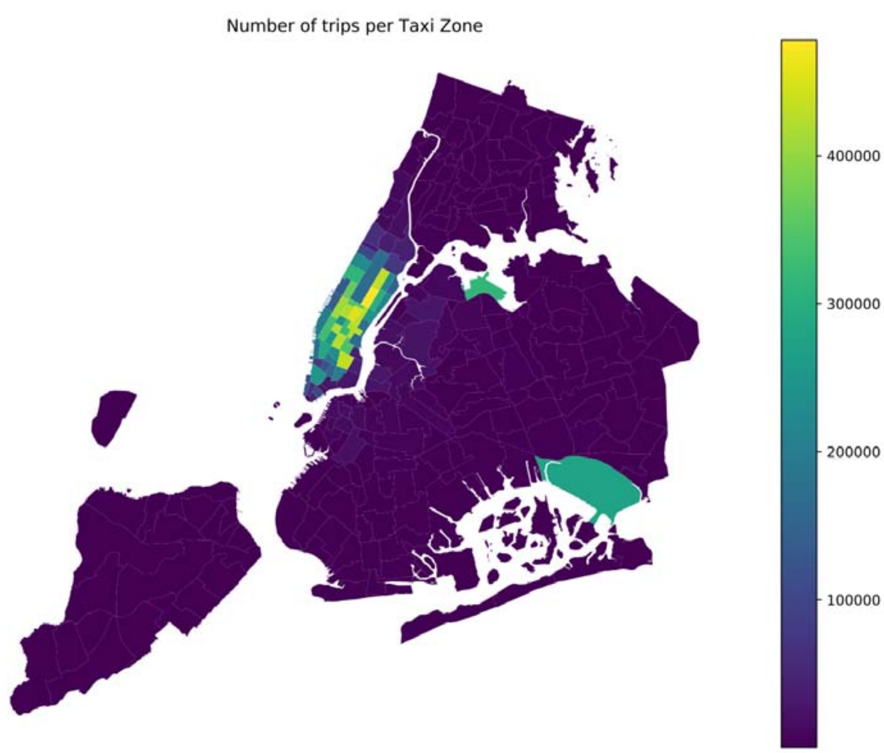
In [17]:
```
q3c_answer = r"""

The longger the trip duration, the more likely that the pickup location is outsi
The lonnger the trip duration, the more likely that the dropoff location is outs

"""

# YOUR CODE HERE
# raise NotImplementedError()

print(q3c_answer)
```

```
The longger the trip duration, the more likely that the pickup location is outs
ide Manhattan in NYC (e.g. from the LaGuardia Airport, Kennedy Airport).
The lonnger the trip duration, the more likely that the dropoff location is out
side Manhanttan in NYC (e.g. to the LaGuardia Airport, Kennedy Airport).
```

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)](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/)](http://geopandas.org/).


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


Number of trips per Taxi Zone
400000
300000
200000
100000

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 [18]:
```
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['ho

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

In [19]:
```
# Note that we are applying this transformation to train_df, short_rides and long
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:438
8: FutureWarning: Attribute 'is_copy' is deprecated and will be removed in a fu
ture version.
  object.__getattribute__(self, name)
/srv/conda/envs/data100/lib/python3.6/site-packages/pandas/core/generic.py:438
9: FutureWarning: Attribute 'is_copy' is deprecated and will be removed in a fu
ture version.
  return object.__setattr__(self, name, value)
```

In [20]:
```
train_df[['month', 'week_of_year', 'day_of_month', 'day_of_week', 'hour', 'week_
```

Out[20]:

	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 [21]:
```
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.

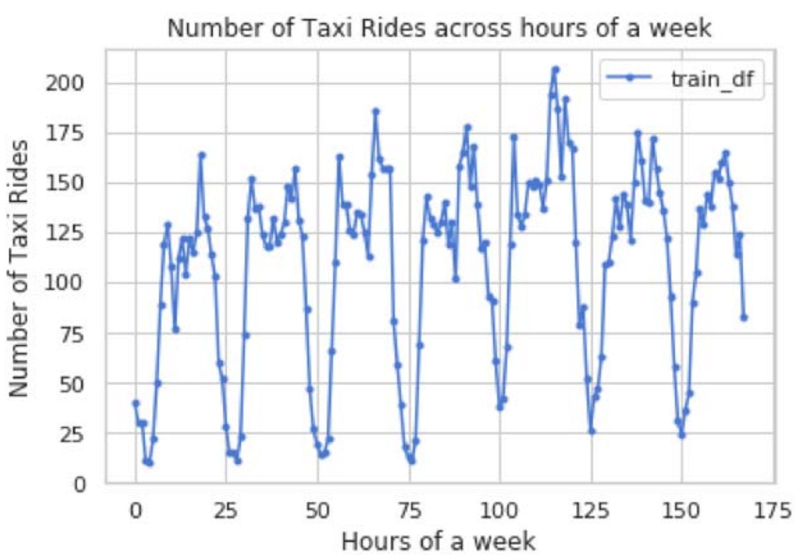
This is a open ended question. Create 2 plots that visualize temporal information from our dataset. At least one of them must visualize the hour of each day. Aside from that you can use any column from `time_columns` .

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 [22]:

```
# Visualization 1
# YOUR CODE HERE
# raise NotImplementedError()
### BEGIN Solution

plt.plot(train_df.groupby("week_hour").count()["record_id"], '.-', label='train_(

plt.title('Number of Taxi Rides across hours of a week')
plt.xlabel("Hours of a week")
plt.legend()
plt.ylabel('Number of Taxi Rides');
### END Solution
```



In [23]:
```
train_df.head()
```

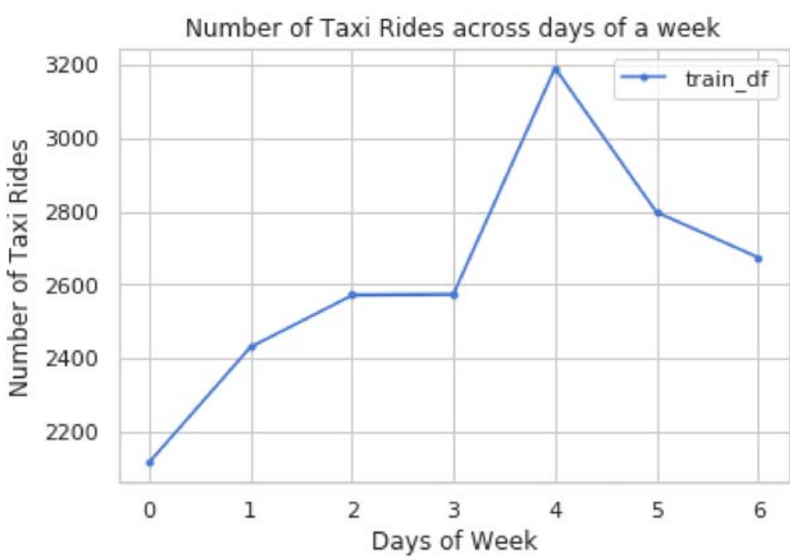
Out[23]:

	record_id	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_dis
16434	8614300	2	2016-01-21 17:37:12	2016-01-21 18:37:56	2	
21929	7230200	2	2016-01-29 23:22:26	2016-01-29 23:31:23	2	
3370	9830300	2	2016-01-05 18:50:16	2016-01-05 18:56:00	2	
21975	7251500	2	2016-01-30 00:14:34	2016-01-30 00:47:13	1	
13758	6168000	1	2016-01-18 13:25:24	2016-01-18 13:38:51	1	

5 rows × 27 columns

Visualization 2

In [24]:
```
# Visualization 2
# YOUR CODE HERE
# raise NotImplementedError()
### BEGIN Solution
plt.plot(train_df.groupby("day_of_week").count()["record_id"], '.-', label='trai

plt.title('Number of Taxi Rides across days of a week')
plt.xlabel("Days of Week")
plt.legend()
plt.ylabel('Number of Taxi Rides');
### END Solution
```



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 [25]:
```
q4b_answer = r"""

In visual 1, I choose the feature `week_hour`. The week_hour against counts plot

In visual 2, I choose the feature `day_of_week`. The day_of_week feature and the

"""

# YOUR CODE HERE
# raise NotImplementedError()

print(q4b_answer)
```

```
In visual 1, I choose the feature `week_hour`. The week_hour against counts plo
t is more informative than the hour plot and the day_of_week plot. This plot ca
n show both the pattern within a day and the pattern within a week.

In visual 2, I choose the feature `day_of_week`. The day_of_week feature and th
e lineplot method can reveal the pattern of taxi rides within a week.
```

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?

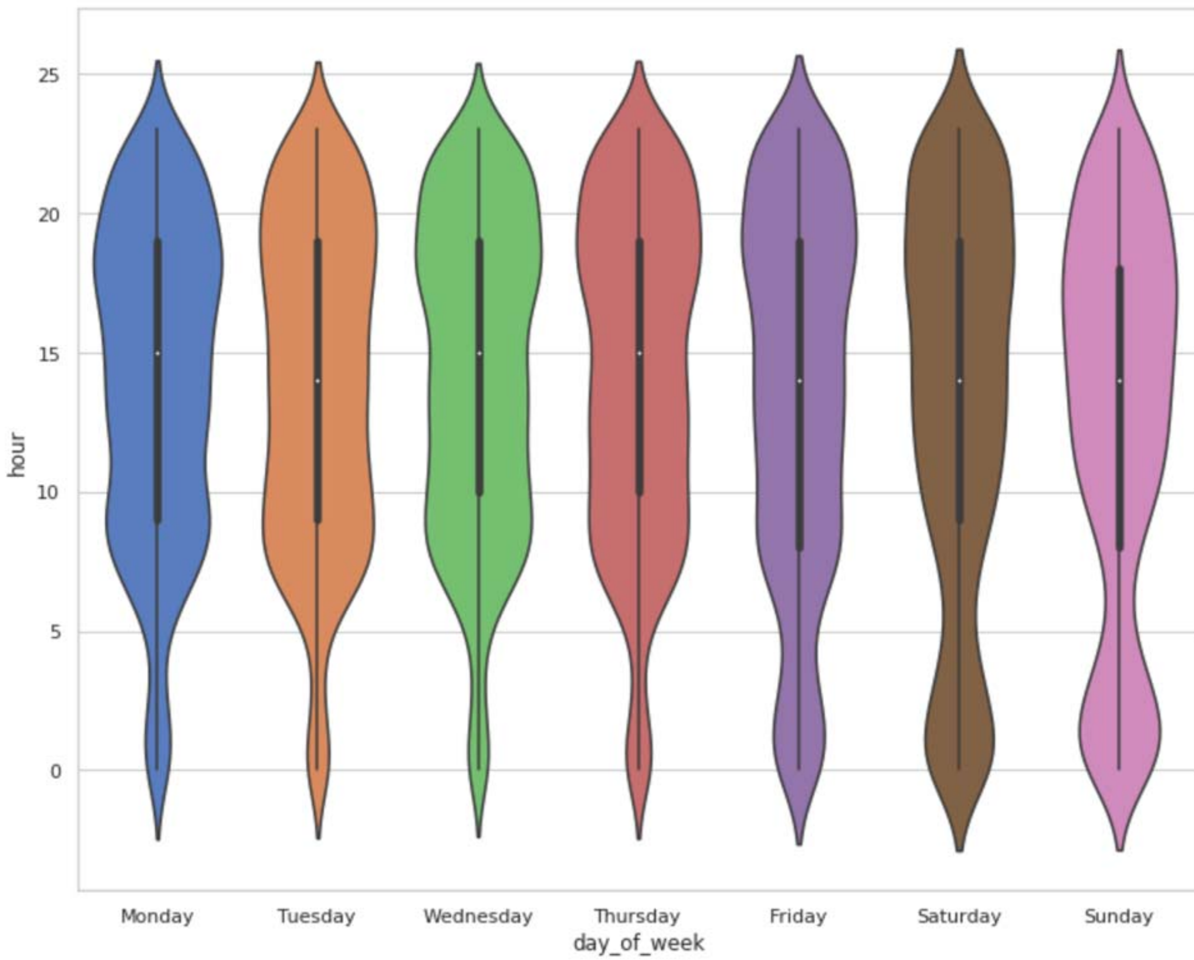
In [26]:
```
q4c_answer = r"""

In the middle of a day, the taxi rides peak. The fifth day of a week has the larg

"""

# YOUR CODE HERE
# raise NotImplementedError()

print(q4c_answer)
```

```
In the middle of a day, the taxi rides peak. The fifth day of a week has the la
rgest number of rides. For this question, according to the instruction, we only
need to consider the temporal structure of number of texi rides and thus we can
have no conclusion about the duration of each rides.
```

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 [27]:
```
fig, axes = plt.subplots(1, 1, figsize=(10, 8))
days_of_week = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday


# YOUR CODE HERE
# raise NotImplementedError()
### BEGIN Solution
sns.violinplot(x="day_of_week", y="hour", data=train_df)
plt.xticks(np.arange(7), days_of_week)
### END Solution

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 [28]:
```
q4e_answer = r"""

In the weekday, there are larger proportion of rides around 8am.
In the weekend, there are larger proportion of rides around 2am.
The feature of interest is the correlation between `hour` and `day_of_week` (aka

"""

# YOUR CODE HERE
# raise NotImplementedError()

print(q4e_answer)
```

```
In the weekday, there are larger proportion of rides around 8am.
In the weekend, there are larger proportion of rides around 2am.
The feature of interest is the correlation between `hour` and `day_of_week` (ak
a `week_hour`)
```

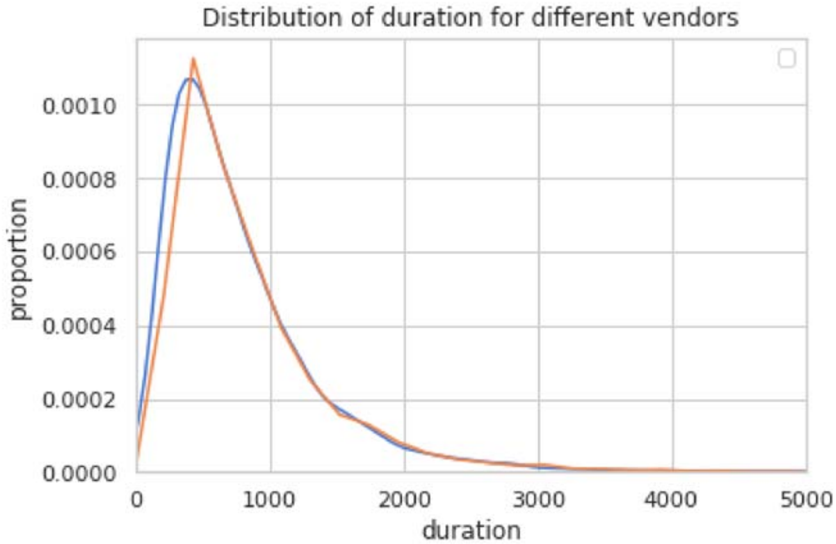
5: Vendors

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 [29]:
```
# Visualization

# YOUR CODE HERE
# raise NotImplementedError()
### BEGIN Solution
ax = sns.distplot(train_df[train_df["VendorID"]==1]["duration"], hist=False)
sns.distplot(train_df[train_df["VendorID"]==2]["duration"], hist=False)
plt.xlim(0, 5000)
plt.title("Distribution of duration for different vendors")
plt.ylabel("proportion")
ax.legend(labels=("Vendor 1", "Vendor 2"));
### NED Solution
```



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 [30]:
```
q5b_answer = r"""

The difference of two ditribution of duration can tell us whether or not `vendor_
If two distributions are closed then `vendor_id` won't have an effect on the trip

"""

# YOUR CODE HERE
# raise NotImplementedError()

print(q5b_answer)
```

```
The difference of two ditribution of duration can tell us whether or not `vendo
r_id` will have an effect on the trip duration.
If two distributions are closed then `vendor_id` won't have an effect on the tr
ip duration. Otherwise, `vendor_id` will have an effect on the trip duration an
d we need to cosider the `veondro_id` in our model.
```

Question 5c

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

In [31]:
```
q5c_answer = r"""

Since two distributions are almost identical, as mentioned above, `vendor_id` doe

"""

# YOUR CODE HERE
# raise NotImplementedError()

print(q5c_answer)
```

```
Since two distributions are almost identical, as mentioned above, `vendor_id` d
oesn't have an effect on the trip 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)](https://en.wikipedia.org/wiki/Haversine_formula) distance, the [manhattan (https://en.wikipedia.org/wiki/Taxicab_geometry)](https://en.wikipedia.org/wiki/Taxicab_geometry) distance and the [bearing (http://www.mathsteacher.com.au/year7/ch08_angles/07_bear/bearing.htm)](http://www.mathsteacher.com.au/year7/ch08_angles/07_bear/bearing.htm) angle.

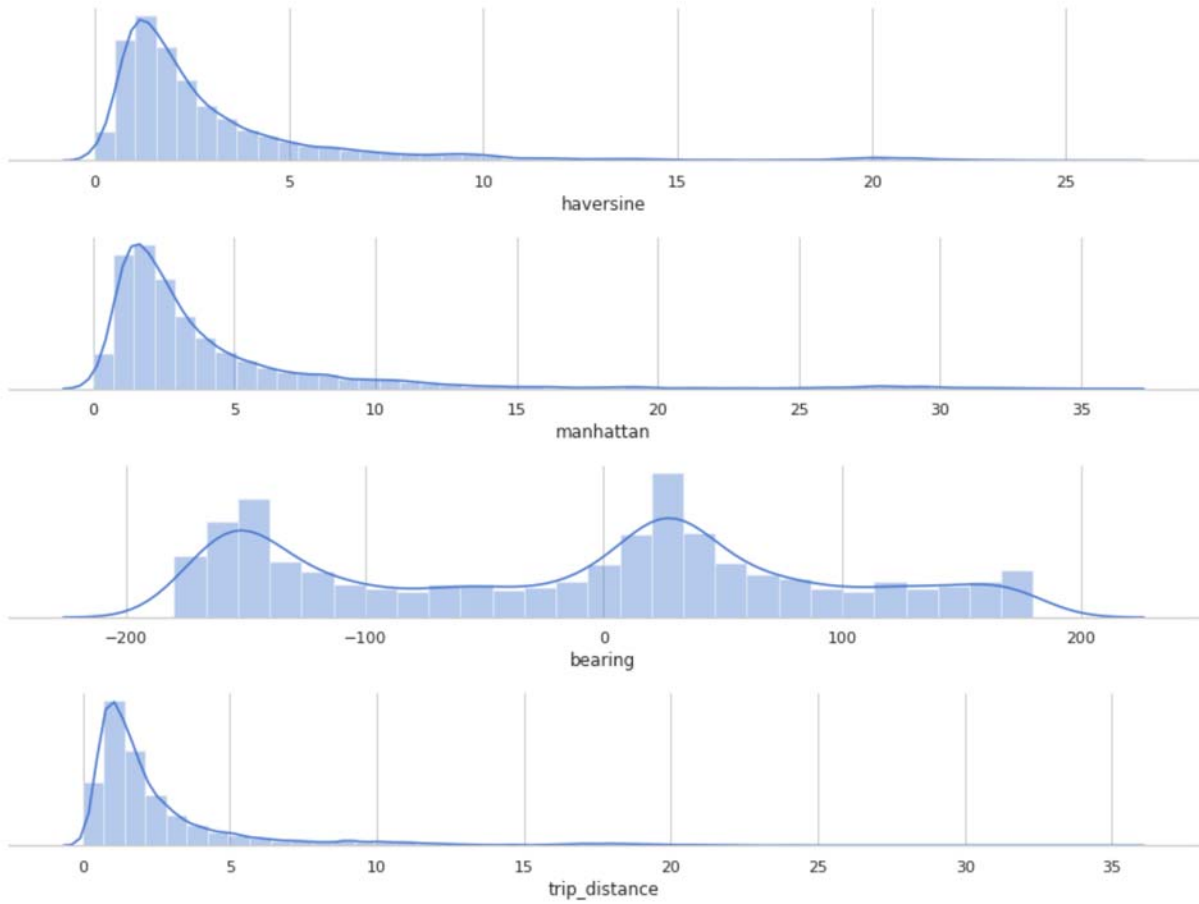
In [32]:

```
# 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 triang
    """
    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)
    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 Manhatta
    which causes the shortest path a car could take between two intersections in
    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_d
    return np.degrees(np.arctan2(y, x))
```

In [33]:
```
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 [34]:
```
train_df = add_distance_columns(train_df)
short_rides = add_distance_columns(short_rides)
long_rides = add_distance_columns(long_rides)
```

In [35]:
```
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.

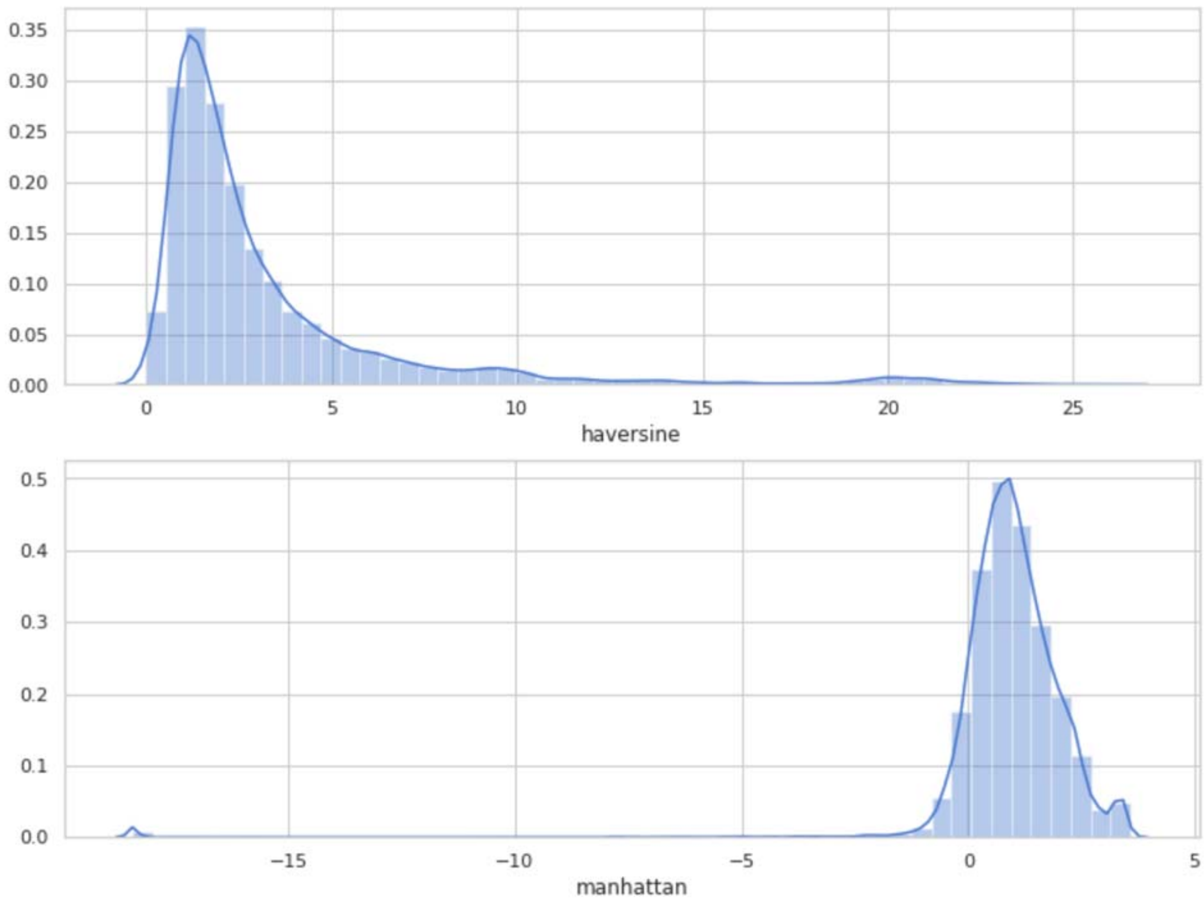
In [36]:
```
q6a_answer = r"""

Since the orientation of Manhattan is around 30 in a clockwise direction from the

"""

# YOUR CODE HERE
# raise NotImplementedError()

print(q6a_answer)
```

```
Since the orientation of Manhattan is around 30 in a clockwise direction from t
he north line, it makes sense the bearing direction peaks around 30 and -150 de
grees(these two are in the same line).
```

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 [37]:
```
# Visualization
# YOUR CODE HERE
# raise NotImplementedError()
### BEGIN Solution
fig, axes = plt.subplots(2, 1, figsize=(12, 9))
sns.distplot(train_df['haversine'], ax=axes[0], axlabel='haversine');
sns.distplot(train_df['manhattan'].map(lambda x: np.log(x+1e-8)), ax=axes[1], ax
### END Solution
```



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.

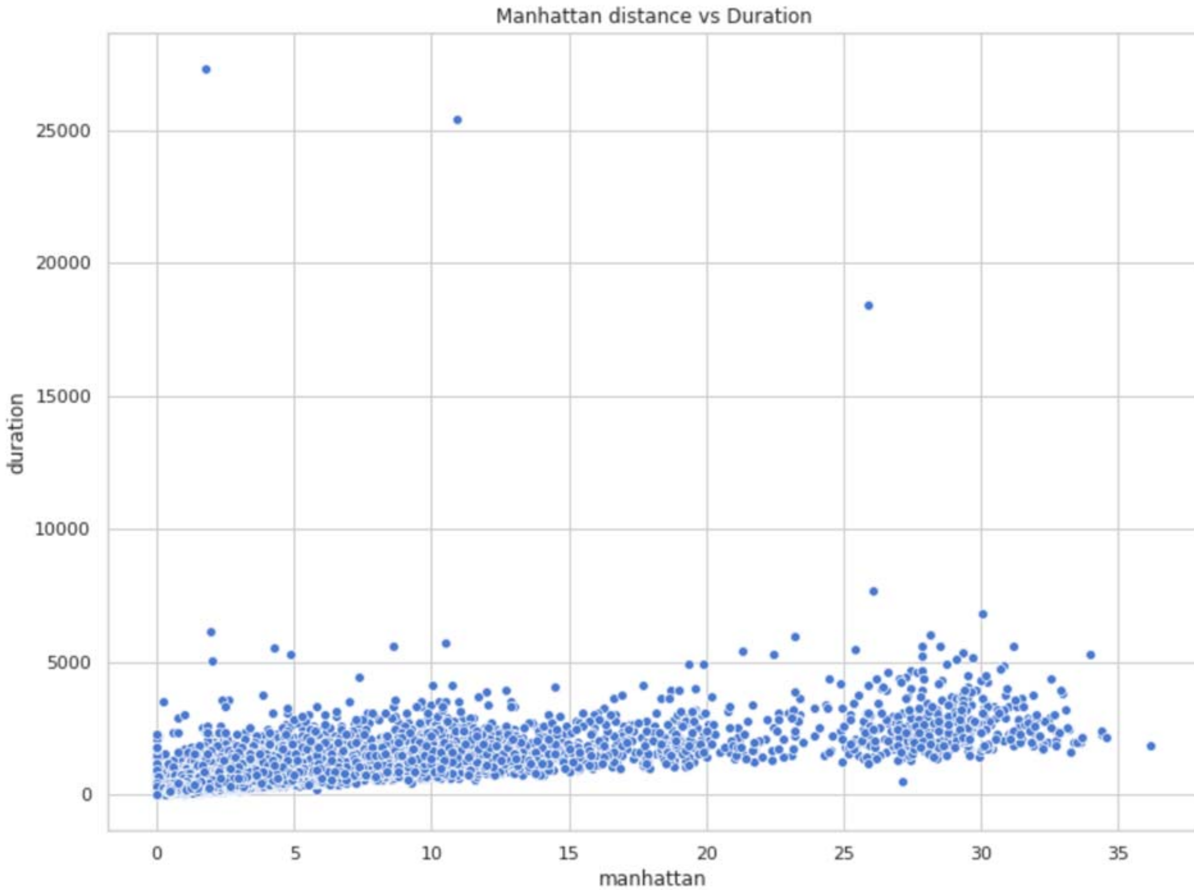
In [38]:
```
q6c_answer = r"""

The comparision of the log(distance) plot can help us to tell the difference of l

Since the log Manhattan distance has a bell shape, the Manhattan distance of our

"""

# YOUR CODE HERE
# raise NotImplementedError()

print(q6c_answer)
```

```
The comparision of the log(distance) plot can help us to tell the difference of
log distribution of Manhattan distance and haversine distance.

Since the log Manhattan distance has a bell shape, the Manhattan distance of ou
r data follows a log normal distribution and thus I would use manhattan distanc
e as a feature.
```

Question 6d

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

In [39]:
```
# YOUR CODE HERE
# raise NotImplementedError()
### BEGIN Solution
fig, axes = plt.subplots(1, 1, figsize=(12, 9))
sns.scatterplot(x="manhattan", y="duration", data=train_df)
plt.title("Manhattan distance vs Duration");
### END Solution
```



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 [40]:
```
q6e_answer = r"""

The outliers in the upper left are most likely mistakes of the data. Since it wo

"""

# YOUR CODE HERE
# raise NotImplementedError()

print(q6e_answer)
```

```
The outliers in the upper left are most likely mistakes of the data. Since it w
on't make sence to travel a short distance with a very long time.
```

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):

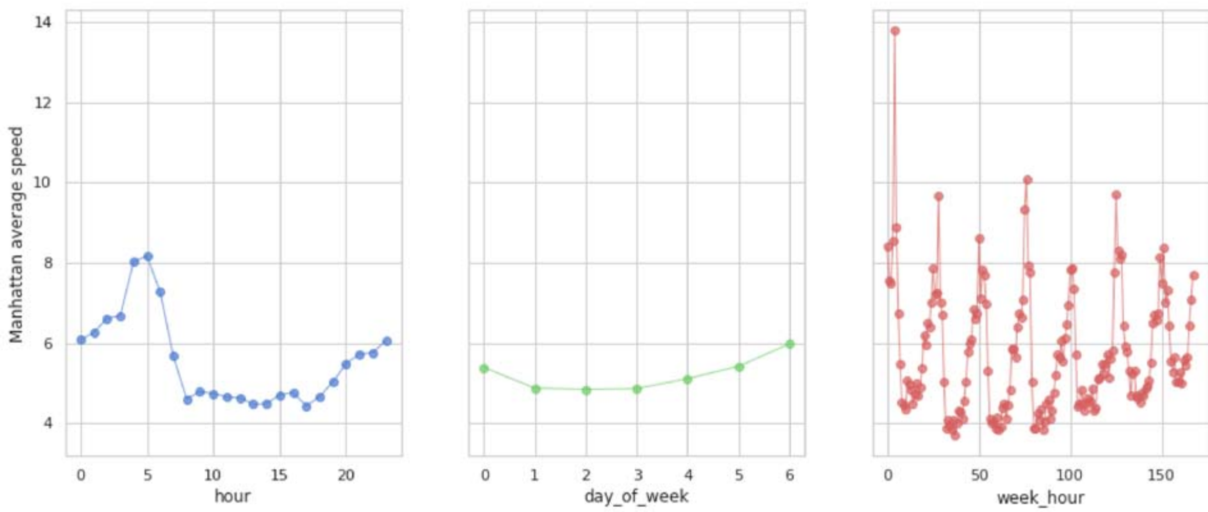
- Average of `avg_speed_h` per cluster.
- Average of `duration` per cluster.
- Average of `avg_speed_h` per cluster and hour.
- Average of `duration` per cluster and hour.
- In-cluster flow of trips for 60 min period.
- Out-cluster flow of trips for 60 min period.

In [41]:
```
# 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 [42]:
```
train_df['avg_speed_m'].describe()
```

Out[42]:
```
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 [43]:
```
# 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=(
axes[1].plot(train_df.groupby('day_of_week').mean()['avg_speed_m'], 'go-', lw=1,
axes[2].plot(train_df.groupby('week_hour').mean()['avg_speed_m'], 'ro-', lw=1, a

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.

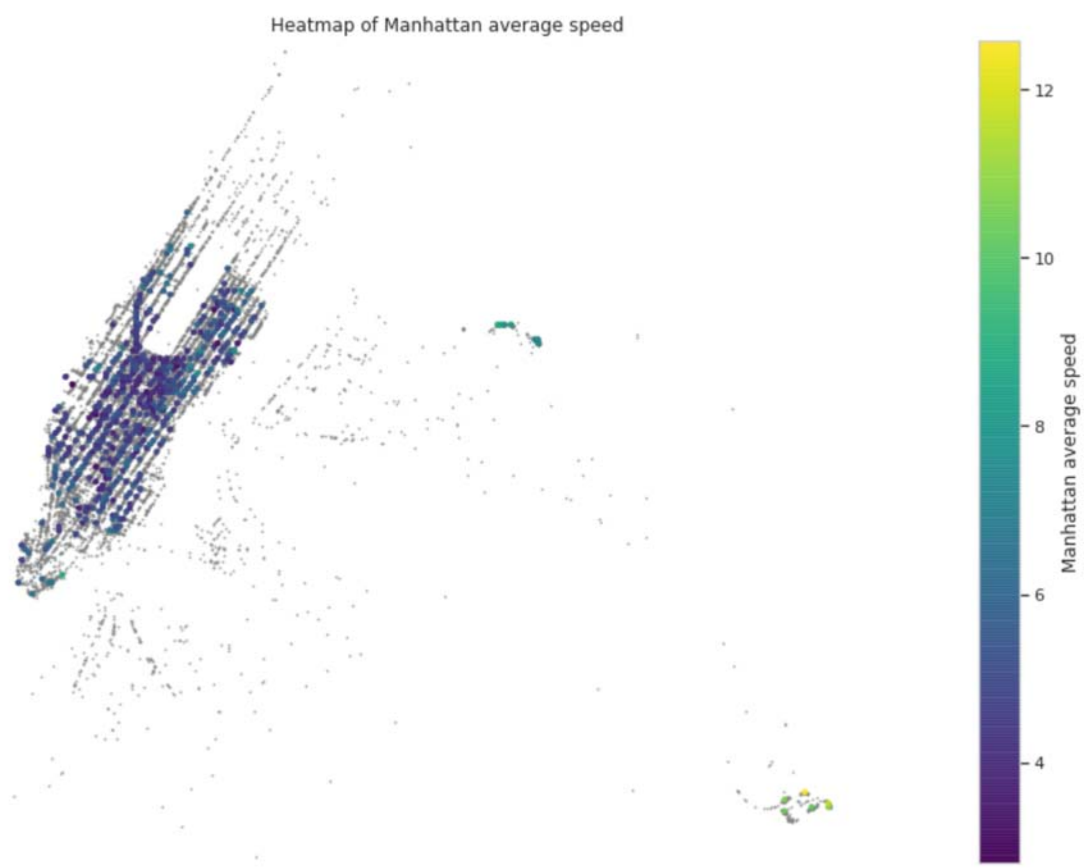
In [44]:
```
q7a_answer = r"""

1. The average speed peaks around 5am.
2. The average speed is about the same within a week.
3. The average speed is a periodic function of time with period around one day.

"""

# YOUR CODE HERE
# raise NotImplementedError()

print(q7a_answer)
```

```
1. The average speed peaks around 5am.
2. The average speed is about the same within a week.
3. The average speed is a periodic function of time with period around one 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 [45]:
```
# 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 [46]:
```
# 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 [47]:
```
q7b_answer = r"""

The taxi speed up as they move away from the center of manhattan. The farther awa

The spatial clustering would be useful since there is a speed difference among di

"""

# YOUR CODE HERE
# raise NotImplementedError()

print(q7b_answer)
```

```
The taxi speed up as they move away from the center of manhattan. The farther a
way from the city center of the pickup location, the faster the average speed i
s. Speed is highest around JFK airport, second highest is LaGuardia airport and
lowest in  Manhattan area.

The spatial clustering would be useful since there is a speed difference among
different spatial clusters. Since speed has a direct relationship with trip dur
ation, the spatial clustering would be useful in making better predictions.
```

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 [48]:
```
Path("data/part2").mkdir(parents=True, exist_ok=True)
data_file = Path("data/part2", "data_part2.hdf") # Path of hdf file
...
```

Out[48]:
```
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)](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/)](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**

In [ ]:

```
```