challenge_notebook

May 15, 2019

1 NYC Taxi Data Challenge

```
In [1]: import pandas as pd
        import numpy as np
        import geopandas as gpd
        from shapely.geometry import Point, box
        import matplotlib.pyplot as plt
        import os
        import pickle

        plt.style.use('seaborn-whitegrid')

        # Important file paths
        from external_variables import data_path,default_crs,original_csv,processed_pkl,saved_ri
```

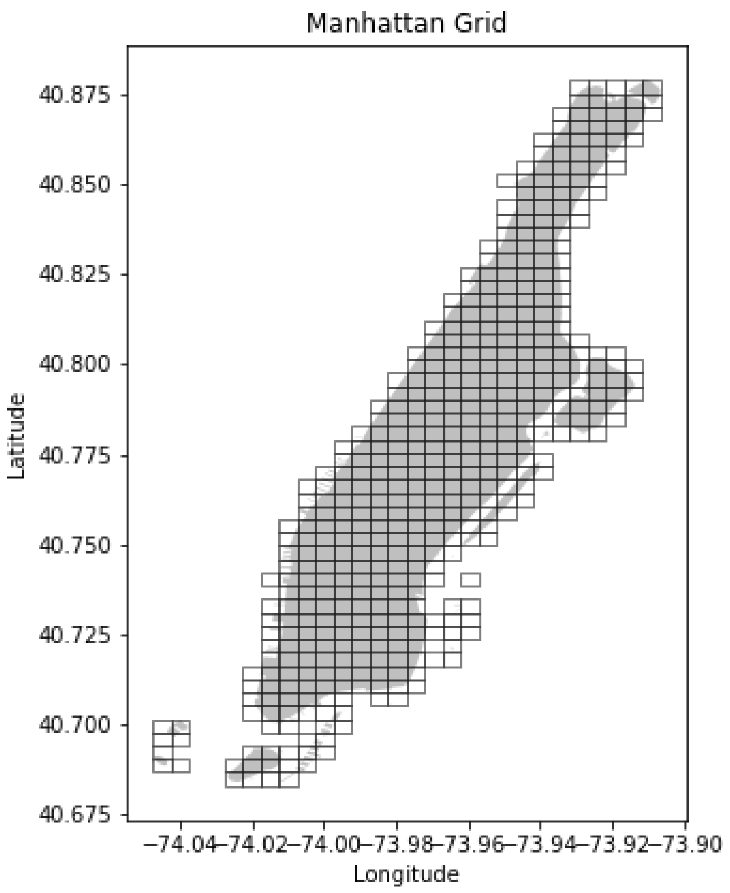
1.1 Question 1:

The question: Propose a metric or algorithm to assess the potential efficiency of aggregating rides from many vehicles into one, given the available data sources. Make realistic assumptions and necessary simplifications and state them.

My response:

The metric: * I estimate the efficiency over a given time period using the metric: $\epsilon = \frac{d_s}{d_t}$, where ϵ is the efficiency, d_t is the total number of vehicle-miles traveled by taxis over the given time period, and d_s is the number of vehicle-miles that would be saved through ridesharing. This efficiency metric represents the ***efficiency gain that could occur if taxis transitioned to ridesharing*** (began accommodating multiple independent passengers at once). * Computing d_t: I sum the distance traveled for all taxi rides over the given time period. * Computing d_s: I identify all taxi rides that would have been unnecessary with ridesharing, and then sum the distances traveled for all those unnecessary taxi rides. I explain my methodology for identifying unnecessary taxi rides below.

Identifying unnecessary rides: * Groups of one or more taxi rides are "unnecessary" if their pickups occur at nearly the same **place** and **time** as the pickup for another taxi ride, and if the total number of passengers in all of the taxis (including the original) is less than the maximum passenger capacity of a rideshare vehicle (given by `n_seats`). * Two pickups occur at nearly the same **time** if they occur within `delta_t` minutes of each other. * Two pickups occur at nearly the same **place** if they occur in the same "part of Manhattan". "Parts of Manhattan" are established by dividing Manhattan into square grid boxes with side length `delta_x`. For `delta_x = 0.25` miles, the grid boxes look like this:



manhattan_grid

NOTE: `n_seats`, `delta_t`, and `delta_x` can be changed

There are several simplifications/assumptions inherent in my efficiency metric: * Mid-route pickups/dropoffs are not taken into account. This a clear shortcoming–mid-route pickups/dropoffs are an important aspect of ridesharing.
* Pickups/dropoffs that occur in close proximity but occur on opposite sides of a grid boundary are not accommodated by a single vehicle. This is also a clear shortcoming–the grid boundaries are imaginary, so a single car could definitely accommodate these two rides in the real world. * Efficiency gains/decreases only occur while cars travel with passengers. This is also a shortcoming–a big source of inefficiency in taxi rides is that vehicles often need to travel considerable distances between a dropoff and the following pickup. Ridesharing might affect these distances traveled without passengers, and therefore affect the overall efficiency in a way that my metric does not consider.

I chose to move forward with the above assumptions/simplifcations due to computational and time constraints: * I could take mid-route pickups and dropoffs into account in two ways. (1) I could search all grid boxes along the general route from pickup->dropoff for rides that (a) have pickups at approximately the same time as the original vehicle would arrive in that grid box, and (b) go in a similar direction as the vehicle's pre-determined direction. By identifying those rides and considering them redundant, I could take them into account in my efficiency metric. Unfortunately, this would be expensive because I'd have to search through a very large subset of rides to identify rides that follow these two conditions. (2) I could implement a simplified version of the rideshare decision-making algorithm described in Santi et al. 2014. In my simplified implementation, I would consider edges from grid boxes to neighboring grid boxes as "streets". However, this would be more difficult to implement, and I'm not confident I'd be able to deliver a finished product in a week. * Combining trips with pickups/dropoffs that are on opposite sides of a grid boundary is incompatible with my grid-based approach. Abandoning the grid-based approach would make computations more expensive: for each ride that my algorithm considers, I'd have to search *all* other rides for rides that the original ride renders unnecessary. Using grid boxes allows me to greatly constrain this search, speeding up the algorithm. This is why I opted for the grid-based approach, despite its clear shortcoming. * I can't infer anything about time spent traveling without passengers because I don't model individual car movements.

Because all the simplifications described above tend to decrease the number of rides that are deemed "unnecessary", my efficiency metric represents a **lower bound** for the efficiency gain associated with ridesharing.

1.2 Question 2:

The question: Implement your proposed metric, and evaluate Manhattan's overall efficiency using yellow taxi data from the first full week (Monday-Sunday) in June 2016. Discuss the complexity of your implementation.

My response: I implement the metric, describe my algorithm, describe the computational complexity of my algorithm, and evaluate Manhattan's overall efficiency in the following cells of this notebook.

1.2.1 Prepare the data

Load borough boundaries

```
In [3]: nyc_boroughs = load_boroughs()
```

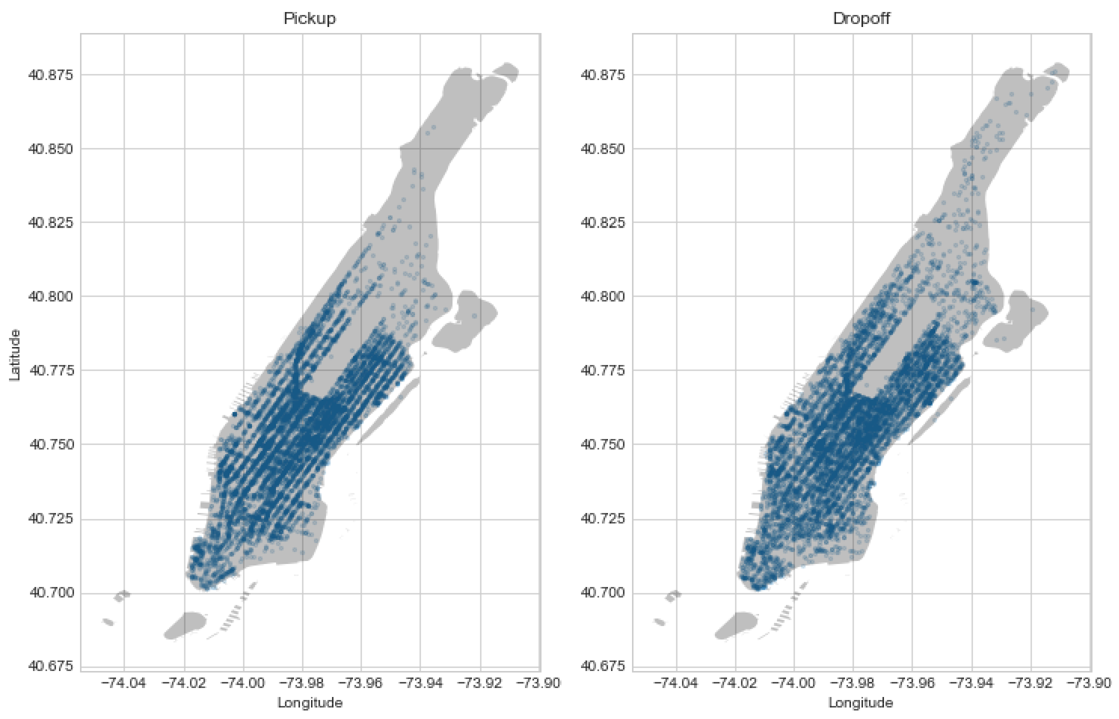
Extract rides that start and end in Manhattan using the following steps: 1. Load the dataframe in chunks–geopandas can't handle the spatial join with >100,000 points on my laptop (maybe because Manhattan's shape is really complicated?). 2. Use spatial joins to identify the start and end boroughs for each ride. 3. Select only the rides that start and end in Manhattan 4. Append the chunks of Manhattan rides to an empty dataframe, creating a single dataframe for all rides that start and end in Manhattan. 5. Save this dataframe to a pickle file so that we can load it later without needing to re-run the joins.

Steps 1-4 happen in the custom function, `prepare_ride_data`, found in `external_functions.py`. Steps 1-5 are only executed when `rerun_preparation` is set to `True`. If `rerun_preparation` is set to `False`, the code just loads the pre-processed data.

```
In [4]: rerun_preparation = False
        if rerun_preparation:
            man_rides_all = prepare_ride_data(original_csv)
            save_manhattan_rides(man_rides_all,processed_pkl)
        else:
            man_rides_all = gpd.GeoDataFrame(load_manhattan_rides(processed_pkl))
```

Below, I verify that all pickup and dropoff points in the prepared data are in Manhattan. Seems so!

```
In [5]: fig = plot_pickup_dropoff_points(man_rides_all.sample(10000),nyc_boroughs)
```



Perform some additional cleaning. In my analysis, I'll need to use the pickup times, dropoff times, passenger counts, pickup locations, dropoff locations, and trip distances. The pickup and dropoff locations were cleaned (to the best of my ability) during the process of identifying rides that start and end in Manhattan. I'll clean the other features next.

First, the pickup/dropoff times: I'll eliminate all trips that last less than 30 seconds and more than 180 minutes (three hours). Rides in the former category are unrealistically short and are therefore probably mistakes. Rides in the latter category are unreasonably long and are also probably mistakes (eg. there's a cluster of ~12000 rides with durations of about 1400 minutes, which is about 24 hours).

```
In [6]: man_rides_all['trip_duration'] = (man_rides_all['tpep_dropoff_datetime']-
                                           man_rides_all['tpep_pickup_datetime'])/pd.Timedelta('1
        #man_rides_all['trip_duration'][man_rides_all['trip_duration']>180].hist()
        #man_rides_all['trip_duration'][man_rides_all['trip_duration']<5].hist(bins=np.linspace(
        man_rides_all = man_rides_all[(man_rides_all['trip_duration']>=0.5)&(man_rides_all['trip
```

Next, the passenger counts. I eliminate rides without any passengers–these are probably errors. No passenger counts are above 9 (which seems reasonable for large taxis), so I don't have any reason to eliminate rides based on unrealistically high passenger counts.

```
In [7]: man_rides_all = man_rides_all[man_rides_all['passenger_count']>0]
```

And finally, the trip distances. I'll eliminate all rides that travel further than the approximate circumference of Manhattan (32 miles). While it's not technically impossible for a taxi to travel this far in a single ride, it seems pretty unreasonable, suggesting a possible error in the datapoint. I'll just drop those instances. I'll also eliminate all rides that travel less than one block (approximately 0.05 miles). Why would someone take a taxi for a distance shorter than a block? Distances shorter than this suggest a possible error.

```
In [8]: man_rides_all = man_rides_all[(man_rides_all['trip_distance']>=0.05)&(man_rides_all['tri
```

1.2.2 Identify unnecessary/redundant rides during first week of June

Unnecessary/redundant rides are defined in my answer to Question 1. Recap: a ride is unnecessary if it (a) starts/ends in the same grid boxes (length, width = `delta_x` miles) as another ride, (b) begins within `delta_t` minutes of another ride, and (c) has a number of passengers such that the sum of the number of passengers in both rides is less than the total passenger capacity of a single rideshare vehicle (`n_seats`). Several rides can be designated as unnecessary/redundant if they satisfy the above conditions all together. `delta_x` is a proxy for the distance passengers are required to walk to the rideshare vehicle before beginning their ride, while `delta_t` is a proxy for the maximum number of minutes passengers must wait for additional passengers to arrive before beginning their ride. Both parameters can take any specified value; I've only chosen 0.25 miles (~5 mins walking at 3 miles/hour) and 5 minutes, respectively, because they seem reasonable to me.

I identify unnecessary rides using the following methodology: 1. In preparation: * Specify `delta_x`, `delta_t`, and `n_seats`. * Create the grid boxes, assign a unique ID to each box, and then place the box geometries and IDs to a GeoDataFrame. Delete all boxes that don't intersect with landmasses that are part of Manhattan (taxi rides can't start or end in these boxes). This all happens in `create_manhattan_grid`.
* Determine the grid boxes in which the pickups and dropoffs occur for each ride using a spatial join. Add two additional columns to the taxi rides dataframe: one for the ID of the pickup box, and one for the ID of the dropoff box. All this happens in the function, `calc_pickup_dropoff_gridbox`.

2. Then, the algorithm for identifying unnecessary rides: (see `identify_unnecessary_rides` and internal functions). 1. Loop through all combinations of the pickup box (box1) and dropoff box (box2). For each combination of box1/box2, identify unnecessary rides from box1->box2 in the following steps. 1. Sort the list of rides that travel from box1->box2 by the pickup time. 2. Create zeros vector of length [number of rides that travel from box1->box2]. Call this vector `accounted_for`. When the algorithm accounts for a ride (decides if it's necessary or unnecessary), the algorithm will change the value corresponding to that ride from 0 to 1.
3. Then loop through all the individual rides that travel from box1->box2. For each ride: 1. If `accounted_for[ride_num]==1`: continue to next ride. 2. If not: change `accounted_for[ride_num]` to 1. 3. Then initialize a variable called `num_in_car`, give it a value equal to the passenger count of the ride. 4. Loop through the rides from box1->box2 that have pickups within the following `delta_t` minutes. * If `num_in_car`+[passenger count in the new ride] <= `n_seats` and `accounted_for[new_ride]==0`, then add the ride to the new car. This means changing `accounted_for[new_ride]` to 1, updating `num_in_car`, and adding the redundant ride's info to the dataframe of unnecessary rides. * If `num_in_car`+[passenger count in the new ride] > `n_seats`: move on to the next ride.
3. As unnecessary rides are identified, all their information is added to a new dataframe (called `all_saved_rides` in my implementation). This dataframe eventually contains information on all rides that are deemed unnecessary.

Complexity of my algorithm: * The process of *determining the pickup and dropoff grid boxes* involves a spatial join. While I'm not really sure what the algorithmic complexity of GeoPandas' spatial join implementation is (I didn't have any luck googling it), I'd naively guess it's $O(N * M)$, where N is the number of points and M is the number of grid boxes that the points are split between. It makes sense that to determine a single point's grid box, you probably have to check if it falls within all the boxes (in the worst case scenario) before determining which box it actually falls within. But maybe there are tricks I'm not currently aware of. * The complexity of the process of *identifying unnecessary rides* is a little more complicated. * In the worst case scenario, all rides would begin and end in the same boxes, and would begin within `delta_t` minutes of each other. Under this scenario, the initial sorting would have complexity $O(Nlog(N))$. The actual process of identifying redundant rides would operate with complexity $O(N^2)$, as the algorithm loops through all the rides, and then loops through all the rides again (within each outer loop iteration) to determine which are redundant. For large N, $O(N^2)$ would win out, leading to a total complexity of $O(N^2)$. Fortunately, this worst case scenario is very unrealistic. * In the best case scenario, all the rides would be evenly distributed between the M grid boxes such that no rides that begin/end in the same boxes have pickups within `delta_t` minutes of each other. The sorting steps for all combinations of box1/box2 would, together, have complexity $O\big(M * M * \frac{N}{M*M} log\big(\frac{N}{M*M}\big)\big)$ ~ $O\big(Nlog\big(\frac{N}{M*M}\big)\big)$. The process of identifying redundant rides would have complexity $O(N)$ because the algoritm would pass over each ride only once (the inner loop would never execute). For large N, the sorting process would win out, leading to an overall complexity of $O\big(Nlog\big(\frac{N}{M*M}\big)\big)$. Note that scenario is also very unrealistic–about 9% of the rides are redundant during the first week of June, and therefore occur within `delta_t` minutes of each other. * In a much more realistic scenario, rides would not be distributed evenly among the M grid boxes, and about P% of rides would occur within `delta_t` minutes of previous rides. The sorting step would have a complexity somewhere between $O\big(Nlog\big(\frac{N}{M*M}\big)\big)$ and $O(Nlog(N))$ (depending on how evenly the rides are distributed), while the process of finding redundant rides would have a complexity somewhere between $O(N)$ and $O(N^2)$ (depending on how evenly spaced the rides are in space/time). More precisely, if P% of rides have pickups within `delta_t` minutes of the previous pickup, the com-

plexity of the latter process would be around $O(N + PN)$, because the algorithm would have to check an additional $P * N$ rides for redundancy as it passes through all N rides. For big N, the initial sorting would probably win out, resulting in an overall complexity between $O(Nlog(\frac{N}{M*M}))$ and $O(Nlog(N))$. * The process of grouping rides by their start and end boxes has complexity $O(N)$, which is less than the complexities of all the other steps, and therefore does not dominate the complexity.
* If my assessment of the algorithmic complexities is correct (especially for the spatial join), the process of *determining the pickup and dropoff grid boxes* is probably dominant (M~464, and log(N)~20 for N~2,000,000), resulting in an overall complexity of $O(N * M)$.

Notes on the implementation below: * The process of identifying redundant rides is only executed when `recompute_unnecesary_rides` is set to `True`. When `recompute_unnecesary_rides` is set to `False`, the results are just loaded from a pkl file. * I interpreted in the "first full week of June" as including June 6-12, 2016 (Monday to Sunday). * I used "unnecessary rides", "redundant rides", and "saved rides" interchangeably in my code/comments. They all mean the same thing.

```
In [9]: # Set important parameters:
        delta_x = 0.25 # Grid box length/width in miles (related to maximum distance people are
        delta_t = 5    # Maximum permissible waiting time in minutes
        n_seats = 5    # Number of passenger seats in each rideshare vehicle

        # Create grid over Manhattan
        manhattan_grid = create_manhattan_grid(nyc_boroughs,delta_x)

        # Plot the grid, save the plot.
        #fig = plot_manhattan_grid(manhattan_grid,nyc_boroughs)
        #plt.savefig('./figs/manhattan_grid.png')
        #plt.close()

        # Select only rides in the first week of June (for the current analysis)
        man_rides_all = select_first_week(man_rides_all)

        # Identify unnecessary rides.
        recompute_unnecesary_rides=False
        if recompute_unnecesary_rides:
            man_rides_all = calc_pickup_dropoff_gridbox(man_rides_all,manhattan_grid)
            all_saved_rides = identify_unnecessary_rides(man_rides,delta_t,n_seats)
            save_manhattan_rides(all_saved_rides,saved_rides_pkl)
        else:
            all_saved_rides = load_manhattan_rides(saved_rides_pkl)
```

```
/Users/Lucien/anaconda/envs/flux-tailor/lib/python3.7/site-packages/geopandas/tools/sjoin.py:44:
  warn('CRS of frames being joined does not match!')
```

1.2.3 Estimation of Manhattan's overall efficiency

My metric suggests that transitioning to ridesharing would increase the efficiency of rides in Manhattan by about 6.6%, measured over the entire week.

```
In [10]: print('Total redundant vehicle miles: %d' % all_saved_rides['trip_distance'].sum())
         print('Total vehicle miles: %d' % man_rides_all['trip_distance'].sum())
         print('Efficiency: %.3f' % (all_saved_rides['trip_distance'].sum()/man_rides_all['trip_
```

```
Total redundant vehicle miles: 274759
Total vehicle miles: 4159291
Efficiency: 0.066
```

1.3 Question 3:

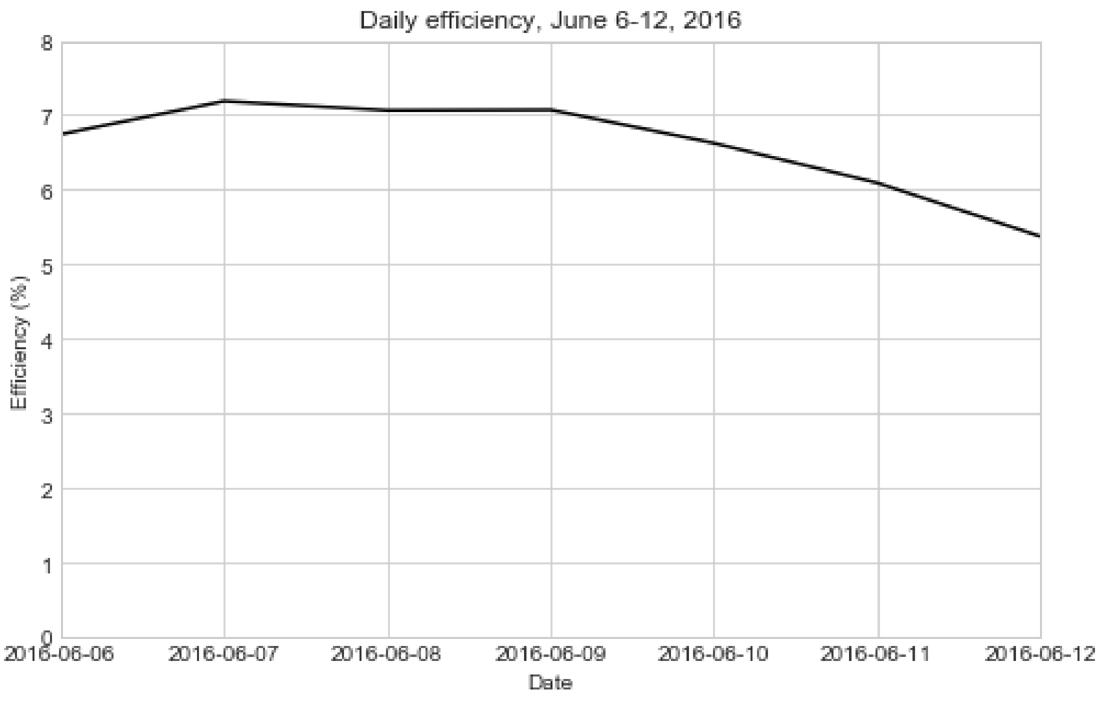
The question: Based on the implementation in Question 2, use visualizations to show how the efficiency metric varies across different time of day, and day of the week. Discuss potential business implications based on your findings.

My response: Implemented in the steps below. Recall that the efficiency metric represents a ***lower bound for the potential efficiency gain that would occur if taxis transitioned to ridesharing.***

1.3.1 How does the efficiency vary with the day of week?

The average efficiency is highest on Tuesday, Wednesday, and Thursday (just over 7%), slightly lower on Monday and Friday (between 6.5% and 7%), still lower on Saturday (6.1%), and lowest on Sunday (5.4%).

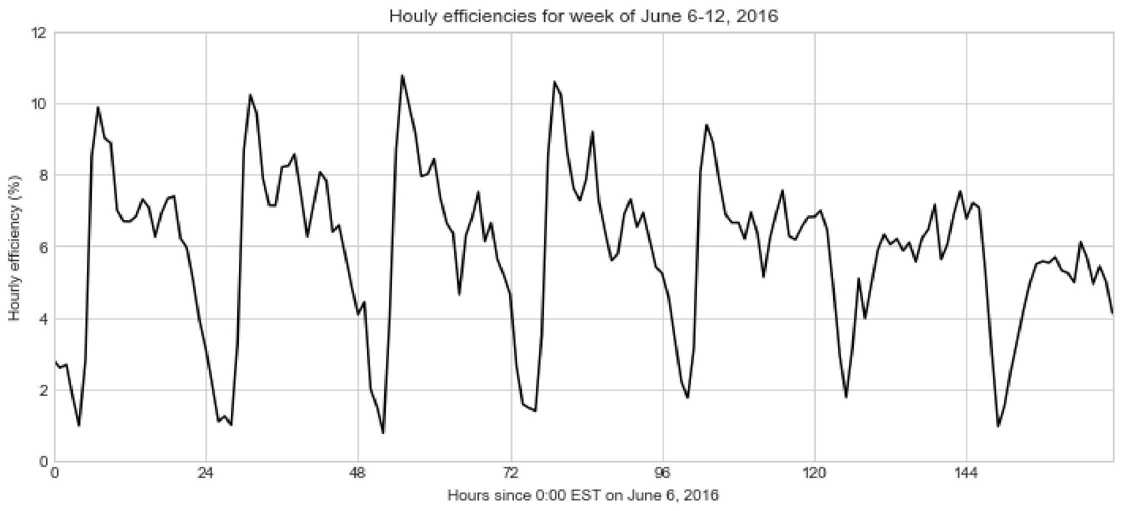
```
In [11]: daily_effs = all_saved_rides.groupby(all_saved_rides['tpep_pickup_datetime'].dt.date)['
                         man_rides_all.groupby(man_rides_all['tpep_pickup_datetime'].dt.date)['t

         fig = plot_daily_efficiency(daily_effs)
```



1.3.2 How does the efficiency vary with the hour of day, on each day of the week?

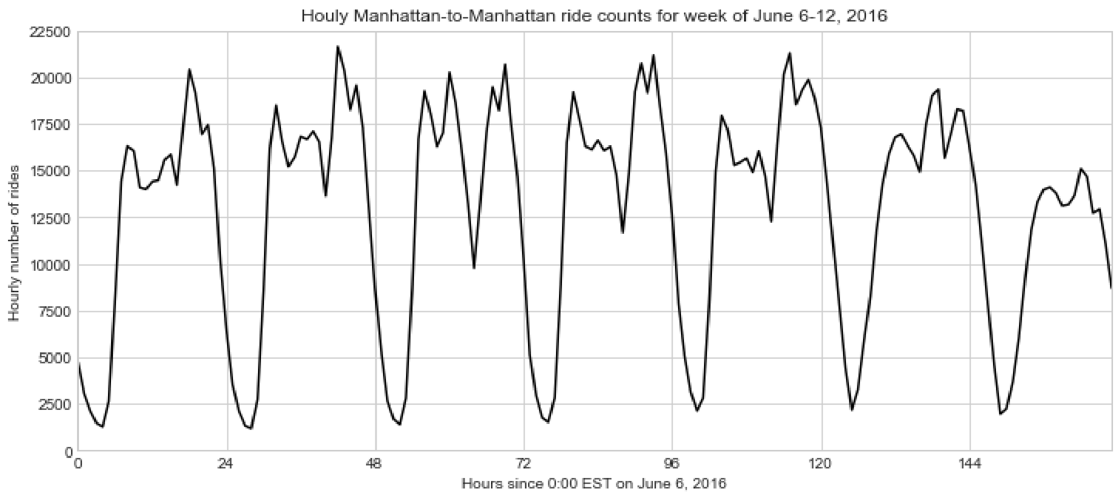
The plot below shows the average efficiency during every hour between 0:00 EST on June 6 and 23:59 EST on June 12. The numbers on the x-axis represent the number of hours since 0:00 EST on June 6, and the vertical grid lines denote the beginnings of each day from June 6-12. The graph suggests a few important things: 1. Variations in the efficiency gain with time of day are very similar from Monday-Thursday. Each day, the efficiency reaches a minimum around 4 am, then rapidly climbs to maximum at around 7 am, then falls gradually (and inconsistently) until about 7 pm, when it drops off rapidly until about 4 am on the following day. There's a local minimum in efficiency during the mid-afternoon (around 3 pm) on each of those days. 2. Variations in the efficiency gain are slightly different on Friday–rather than falling off sharply around 7 pm, the efficiency gain remains elevated until about 2 am the following day. This is probably because many people take taxis home after their night out, resulting in a higher frequency of redundant rides. 3. Variations in the efficiency gain are very different on Saturday and Sunday. The morning peak in efficiency does not occur on either day (because there's no morning commute), and the efficiency remains elevated late on Saturday night, but does not remain elevated on Sunday night. The potential efficiency gain is lower on Sunday than it is at almost any time of day from Monday-Friday.

```
In [12]: all_saved_rides['hour_of_week'] = (all_saved_rides['tpep_pickup_datetime'].dt.day-
                                            all_saved_rides['tpep_pickup_datetime'].dt.day.min()
                                            +all_saved_rides['tpep_pickup_datetime'].dt.hour
         man_rides_all['hour_of_week'] = (man_rides_all['tpep_pickup_datetime'].dt.day-
                                          man_rides_all['tpep_pickup_datetime'].dt.day.min())*24
                                          +man_rides_all['tpep_pickup_datetime'].dt.hour

         hourly_effs = all_saved_rides.groupby('hour_of_week')['trip_distance'].sum()/\
                       man_rides_all.groupby('hour_of_week')['trip_distance'].sum()*100

         fig = plot_hourly_efficiency(hourly_effs)
```



The efficiency is higher during the morning commute than it is during the evening commute on Monday-Friday. This cannot be explained by a difference in the number of rides during those two periods–more rides actually occur during the evening commute than during the morning commute on all three days (see plot below, which shows hourly Manhattan-to-Manhattan ride counts for June 6-12, 2016). The difference in efficiencies must occur because the rides tend to have more consistent pickup and dropoff locations during the morning commute.

```
In [13]: hourly_ride_counts = man_rides_all.groupby('hour_of_week')['trip_distance'].count()
         fig = plot_hourly_ride_counts(hourly_ride_counts)
```



1.3.3 Business implications

It's probably most profitable to implement ridesharing at times when the potential efficiency gain is largest. This means that it's most profitable to implement ridesharing during the morning commute on weekdays. It may also be profitable to implement ridesharing during most other daytime hours on weekdays, during the late evening on Friday and Saturday, and during the afternoon on Saturday: potential efficiency gains are also relatively high at those times (generally between 6 and 7%). Based on the efficiency metric, it is not as profitable to implement ridesharing between 3pm and 4pm on weekdays, or between 3am and 5am on any day of the week, when efficiency gains are lower because ride counts are lower.

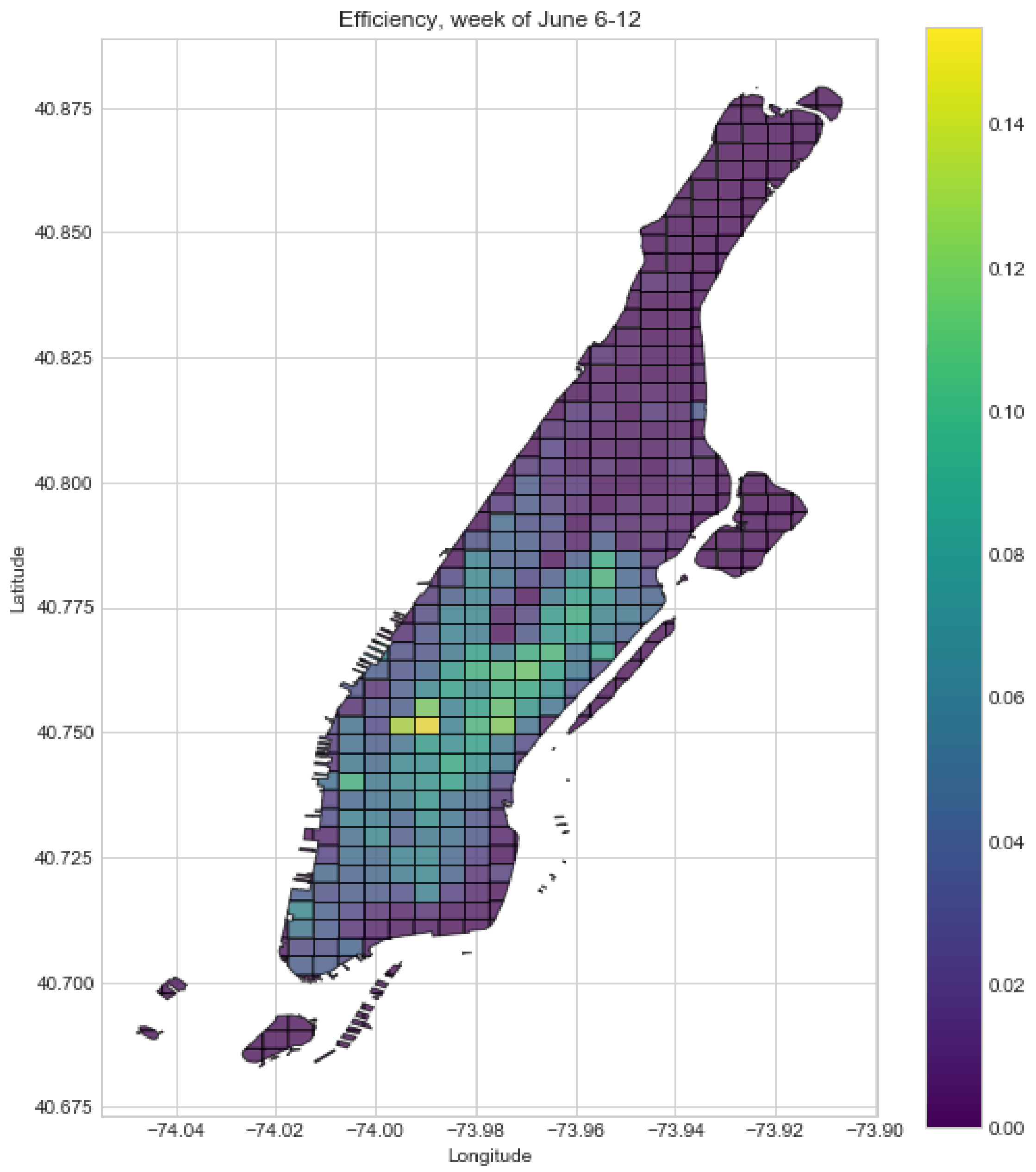
It is also important to recognize that these conclusions are based only on one week of data, and therefore might not generalize to all other weeks during the year. For example, it's possible that potential efficiency gains are larger during the evening commute in winter than they are in June. On winter evenings, it's often cold/dark when people get off of work, so people might be more willing to take a car home and more likely to travel straight home instead of first traveling somewhere else. This might make the evening commute more similar to the morning commute, resulting in a larger potential efficiency gain (note: this is just speculation–I haven't looked at the data). Therefore, I'd definitely apply my algorithm to a larger subset of weeks and analyze the all those results before drawing any conclusions or making any business recommendations.

1.4 Question 4:

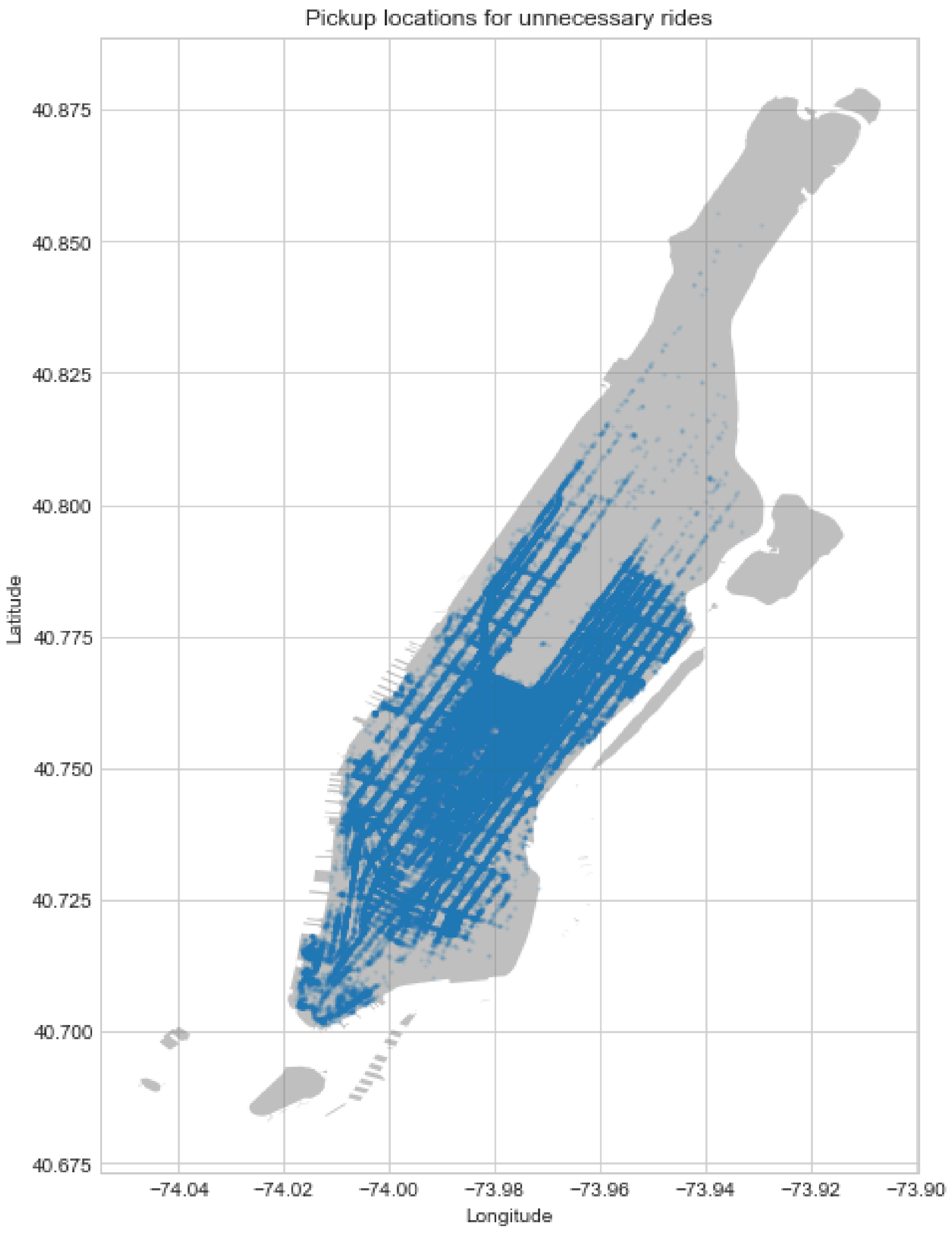
The question: Answer the following question theoretically (no need to implement). Given the metric you proposed, how would you determine whether a specific zone is more efficient than the rest of the city? How would you determine why it's more efficient?

My response: This was pretty easy to implement, so I just implemented it in the next two cells. The first plot below shows the efficiency metric, computed for each grid box over the entire week of June 6-12, 2016. The second plot shows the pickup locations for every ride that my algorithm deems unnecessary during that week. The plots suggest that it would be most beneficial to implement ridesharing in Midtown, on the Upper East Side, and in the area around Herald Square. It might also be beneficial to implement ridesharing in other parts of lower Manhattan and on the Upper West Side, where efficiencies are slightly lower. It would probably be pretty pointless to implement ridesharing above 96th Street, because very few unnecessary pickups occur in the Northern part of Manhattan.

```
In [19]: man_rides_all = calc_pickup_dropoff_gridbox(man_rides_all,manhattan_grid)
         pickup_box_effs = (all_saved_rides.groupby('pickup_box')['trip_distance'].sum()/\
                            man_rides_all.groupby('pickup_box')['trip_distance'].sum()).fillna(
         gridded_effs = manhattan_grid.merge(pickup_box_effs,left_on='grid_idx',right_index=True
                            .rename(mapper={'trip_distance':'efficiency'},axis=1).fillna(value=
         fig = efficiency_gridbox_choropleth(gridded_effs,nyc_boroughs)
```



```
In [20]: # Plot saved rides
         fig = plot_unnecessary_ride_pickup_locations(all_saved_rides,nyc_boroughs)
```


Pickup locations for unnecessary rides
Latitude
40.875
40.850
40.825
40.800
40.775
40.750
40.725
40.700
40.675
Longitude
−74.04
−74.02
−74.00
−73.98
−73.96
−73.94
−73.92
−73.90