Trashatouille and How To Clean Them

Matthew Cowan, Tuni Le, Mary Yuan

November 8th, 2024

Abstract

The streets of New York City, particularly in Manhattan, face a persistent trash problem, with piles of garbage accumulating on sidewalks and attracting rat populations. To address this issue, we developed a mathematical model that simulates trash accumulation and rat population dynamics in Manhattan. Our model calculates the optimal number of trash collection trucks and schedules pickups for each of the twelve DSNY-designated sanitation districts (MN01-MN12), covering a total of 80 sections. Using this optimized schedule, we estimate the time trash remains on streets, aiming to minimize this duration and, as a result, curb rat activity. The outcomes of our optimizations show a collection schedule that not only reduces the required number of trucks but also ensures equitable service across neighborhoods, regardless of socioeconomic status. This approach provides a cleaner, fairer, and less rat-attractive New York City.

Contents

1 Introduction

1.1 The Problem

As population density increases, so does trash production, especially in densely populated areas like New York City. With over $800,000$ residential buildings, NYC produces an average of 24 million pounds of trash a day; Manhattan, being the most densely populated borough in NYC, generated $385,856$ tons of trash in 2023, averaging approximately 2.1 million pounds each day. Assuming typical trash bags hold 15–30 pounds, this equates to roughly $93,968$ bags of trash daily. Due to narrow streets crowded with cars and pedestrians, timely trash removal is essential to prevent potential health risks. Trash bags on the streets attract rats, which can chew through them, spreading waste and increasing health concerns. Thus, minimizing the time trash sits on the street is critical.

1.2 Our Approach

To minimize the time trash sits on the street (P), we developed a model that uses a scheduled pickup plan to calculate the total amount of time trash remains on the streets. We then use this time to model the number of rats (R) that consume the trash.

We determine P in three steps:

1. Approximating the amount of trash produced by each of the 80 sections.
2. Approximating the number of trucks needed in each section to collect all the trash during the morning or night.
3. Approximating the total time trash remains on the streets given these pickup schedules.

We chose to divide Manhattan into 80 sections instead of the traditional 12 districts to achieve a more fine-grained understanding of trash production and collection needs across different neighborhoods, accounting for local variations in trash accumulation and collection frequency. Our choice also reflects how DSNY organizes its sanitation schedules, ensuring our model is consistent with real-world practices.

With P calculated, we can the determine R by modeling rat population growth as a function that increases with P. Since the earliest trash pickup occurs during the day, we only consider rat activity during the daytime, as nighttime activity remains constant.

2 Definitions

2.1 Constants

2.1.1 Population Constants

- New York City population (2020): $N_{NYC} = 8,804,190$ people ([9])
- Number of residents in Manhattan belongs to a sanitary district (2020): $N = 1,672,628$ people ([5], [6], [9]; see processing code in Appendix A.3, result data in Appendix B.3).
  - Note that this is lower than total number of residents in Manhattan, $N_{total} = 1,694,251$, due to the fact that some areas of Manhattan are not under DSNY service (as of April 10, 2024) (e.g. Roosevelt Island).
- Total Area of Manhattan: $A = 21.34\ mi^2$ ([5], [6], [9]; see processing code in Appendix A.3, result data in Appendix B.3)
  - Note that this is smaller than the total area of Manhattan $A_{total} = 22.66\ mi^2$) because some areas are not under DSNY service (as of April 10, 2024).
- Manhattan proportion of total New York City population: $N/N_{NYC} = 1,672,628/8,804,190 \approx .1900$

- Number of districts: $d = 12$
- Total number of sections: $s = 80$ [6]
  - There are 40 main sections of DSNY-designated sections in Manhattan, each divided into two sub-sections, assigned either Schedule A or Schedule B. For brevity, we refer to these 80 sub-sections as 'sections'.
- Area of the section i: a_i (mi^2) ([5], [6], [9]; see processing code in Appendix A.3, result data in Appendix B.3)
- Population per section i: g_i (people) ([5], [6], [9]; see processing code in Appendix A.3, result data in Appendix B.3)
  - Data in Figure 1 below is represented by color intensity, with higher population areas shown in warmer colors.

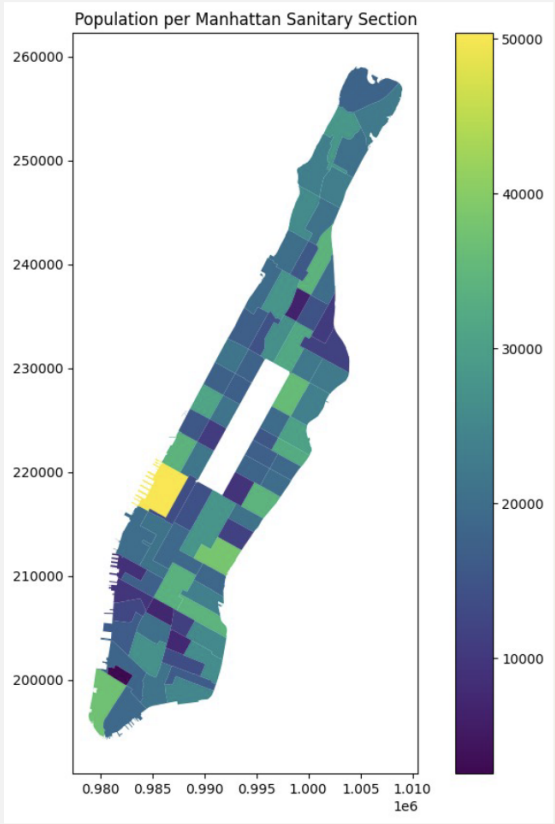


Figure 1: Population per Manhattan Sanitary Section.

- Population ratio for section i: $s_i = g_i/N$
- Area ratio for section i: $ar_i = a_i/A$

2.1.2 Trash Constants

- Total trash per year in Manhattan: 385856 tonnes ([7]; see processing code in Appendix A.1, result data in Appendix B.1)
- Total trash per day: $W = \frac{385856 \text{ tonnes} * 2000 \text{ lbs/ton}}{365 \text{ days}} \approx 2,114,280$ lbs/day
- Trash bag weight: $b_w = \frac{15+30}{2} = 22.5$ lbs

  - Based on the median of the $15 - 30$ lb range for typical trash bags.

- Pounds of trash for section i: $w_i = s_i \times W$
- Trash produced per person: $l = W/N = 1.2640$ lb/person
- Enum for morning pickup: Mn
- Enum for night pickup: Ng
- Trash curb time for morning pickup (8 pm - 7 am): $p_m = 11$ hours
  - Assume a person sets out all their trash at 8 pm.
- Trash curb time for night pickup (8 pm - 6 pm next day): $p_n = 22$ hours
  - Assume a person sets out all their trash at 8 pm.

2.1.3 Truck and Road Constants

- Average speed in New York City: $speed_{NYC} = 16.87$ mph ([8])
  - Calculated from the Speed tab in the "Driving Patterns in New York" table on TomTom by averaging all entries.
- Average speed in New York City from 7 am to 8 am: $speed_{7am} = 17.5$ mph ([8])
  - Calculated from the Speed tab in the "Driving Patterns in New York" table on TomTom by averaging all entries from 7 am to 8 am, Monday to Saturday.
- Average speed in New York City from 6 pm to 7 pm: $speed_{6pm} = 12$ mph ([8])
  - Calculated from the Speed tab in the "Driving Patterns in New York" table on TomTom by averaging all entries from 6 pm to 7 pm, Monday to Saturday.
- Average speed in Manhattan: $speed_{manhattan} = \frac{6.9+4.8}{2} = 5.85$ mph ([3])
  - Estimated as the average speed between Midtown Manhattan and the Central Business District in Fiscal Year 2024
- Trash truck speed in Manhattan (morning): $t_m = 6.07$ mph
  - Estimated as $t_m = speed_{manhattan} \times \frac{speed_{7am}}{speed_{NYC}}$.
- Trash truck speed in Manhattan (night): $t_n = 4.16$ mph
  - Estimated as $t_n = speed_{manhattan} \times \frac{speed_{6pm}}{speed_{NYC}}$.
- Total number of trucks assigned to Manhattan: $t_{total} = \lfloor N/N_{NYC} \times 2230 \rfloor = 423 \pm 50$ trucks
  - Approximating by multiplying the proportion of Manhattan's population out of the entire population of NYC by the total available number of trucks. We also decided to add ± 50 to this final number to account for possible hazards on the road and other unforeseen circumstances.
- Capacity of a truck: $t_c = 12$ tons ≈ 1050 trash bags
  - Assume typical trash bags hold $15 - 30$ pounds.
- Total Manhattan road length: $r_{total} = 640$ miles ([4])
- Road length of section i: r_i miles
  - Estimated as $r_{total} \times ar_i$ ([4], [5], [6], [9]; see processing code in Appendix A.4, result data in Appendix B.4).

- Truck trash pickup rate: $t_p \approx 341$ in lbs/mi
  - Calculated using Manhattan's population density ($N_{total}/A_{total} = 72,918$ people/sq mi). Taking the square root gave an estimated population per mile (270 people/mi), which we multiplied by the trash generated per person to yield a pickup rate of approximately 341 lbs/mi.

2.1.4 Rat Constants

- Total rat count in 2023: $R = 230$ units ([2]; see processing code in Appendix A.5, output graph below, see Figure 2)
  - R is estimated from incident reports by summing Manhattan street rat reports recorded at 7 am throughout the year to approximate the initial count when morning trucks arrive.

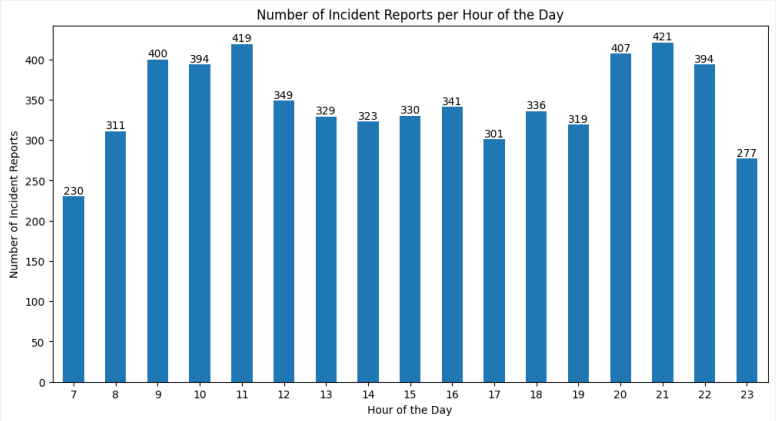


Figure 2: Accumulated Number of Incident Reports per Hour in 2023

- Initial amount of rats on the street in section i: $\lambda_i = s_i \times R$
- Rat growth constant: $\gamma = 0.38$
  - Calculated as $\ln\left(\frac{336}{230}\right)$, where 336 and 230 are the rat sightings at 6 pm and 7 am, respectively. Since we are using an exponential model with base e, we take the natural log of this ratio to determine the growth rate.

2.2 Variables

We define variables as parameters to test for minimizing T and P. Specifically:

- Pickup time threshold p_t, based on population density per section.
  - From Appendix A.6, we used the mean (5708.549 people/mi^2) and standard deviation (2044.205) of population densities across 80 sections. We tested thresholds within the range 5708.549 ± 2044.205, dividing it into 100 points to evaluate truck count, trash exposure time, and rat population.

3 Assumptions

Our model relies on the following key assumptions, organized by category:

3.1 Population and Trash Production

- Trash production and the number of trucks needed are proportional to the population.
- People produce a constant amount of trash each day.
- All residents place their trash on the side of the road for collection.

3.2 Truck Operations and Movement

- Trucks operate at a constant average speed.
- Trucks always adhere to exact scheduled start and end times, either ending upon reaching capacity or completing the hour.
- Trucks are stationed in their assigned sections prior to their scheduled pickup times.
- Trucks slow down on streets with higher trash density, speed of the trucks will balance out the density of trash.
- All trash in a section is collected within the hour of scheduled pickup.
- Trucks traverse streets uniformly, picking up all trash along their route.
- Each truck services all Manhattan sections three times a week, following current DSNY practices ([6]; see processing code in Appendix A.2, result data in Appendix B.2).

3.3 Trash and Rat Behavior

- Trash bags remain on the street until collected.
- The likelihood of rats approaching trash increases the longer it sits on the street.
- Rats are uniformly distributed across Manhattan and are equally likely to approach any single trash bag.
- After trash is collected, rats vacate the streets and return to the sewers.

3.4 Road and Street Layout

- Roads are uniformly distributed throughout Manhattan.

4 Model

4.1 Motivation

In our model, we account for reduced traffic from 7 am to 8 am, allowing trucks to work faster and collect more trash. Therefore, we prioritize assigning morning pickups to sections with the highest trash volumes.

Our primary goal was to implement a function T that determines the number of trucks required for each section to complete trash collection within an hour. To achieve this, we approximated the total road miles per section and used truck speed to calculate the necessary truck count to cover the entire section within the time frame.

However, we also need to ensure that all trash gets picked up, which is what motivated us to come up with a trash pickup rate. We chose a constant pickup rate, t_p, assuming that trucks slow down on streets with more trash, balancing speed with trash density. We calculated the pounds of trash each truck collects per hour as $t_p \times [t_m \text{ or } t_n]$, and the required number of trucks as $\frac{w_i}{t_p \times [t_m \text{ or } t_n]}$. If this amount exceeds truck capacity, we adjusted the truck count by multiplying by $\left\lceil \frac{t_p \times [t_m \text{ or } t_n]}{t_c} \right\rceil$.

Lastly, we rounded all values up, as trucks count cannot be a fraction.

To calculate the total trash exposure time, we assigned pickup times to sections and then used p_m and p_n to determine how long trash sat on sidewalks, which in turn allowed us to estimate rat presence.

We assume that rat population correlates with human population density: the more dense the human population, the more dense the trash, and the more rats congregate on the sidewalks. Furthermore, each rat increases the likelihood of others congregating, suggesting an exponential growth pattern. Following established literature, we model this growth using base e.

Combining these ideas together, we developed a set of modeling functions:

4.2 Model Functions

- $D(p)$: Function to determine what time of day trash gets picked up. p is the population density.

$$D(p) = \begin{cases} Mn & p \geq p_t \\ Ng & else \end{cases}$$

- $T(t, s)$: Function to determine how many trucks to send to each section. t is the time of pickup (morning/night) and s is the section.

$$T(t,s) = \begin{cases} max(\left\lceil \frac{m_s}{t_m} \right\rceil, \left\lceil \frac{w_s}{t_p * t_m} * \left\lceil \frac{t_p * t_m}{t_c} \right\rceil \right\rceil) & t = Mn \\ \\ max(\left\lceil \frac{m_s}{t_n} \right\rceil, \left\lceil \frac{w_s}{t_p * t_n} * \left\lceil \frac{t_p * t_n}{t_c} \right\rceil \right\rceil) & t = Ng \end{cases}$$

- $P(t, s)$: Function to determine how long the trash bags sat on the side of the road each day. t is the time of pickup (morning/night) and s is the section.

$$P(t,s) = \begin{cases} \frac{w_s}{b_w} * p_m & t = Mn \\ \\ \frac{w_s}{b_w} * p_n & t = Ng \end{cases}$$

- $R(s, t)$: Function that returns the number of rats. s is the section number and t is the time that the garbage has been sitting on the sidewalk.

$$R(s,t) = \lambda_s e^{\gamma t}$$

4.3 Days of Pickup

In our resource allocation strategy, we must discuss the weekly schedule for sanitation truck visits. Due to a limited number of trucks, it is not feasible to visit every section in one day. All Manhattan sections are currently serviced three times a week, split into 2 blocks: Monday-Wednesday-Friday (MWF) and Tuesday-Thursday-Saturday (TTS) ([6]; see processing code in Appendix A.2, result data in Appendix B.2). We will retain this schedule for our model.

Assuming trucks are stationed in their designated sections before pickup times, we don't need to consider garage locations. To optimize truck sharing across sections, we divided Manhattan into an upper and lower half, with 40 sections in each.

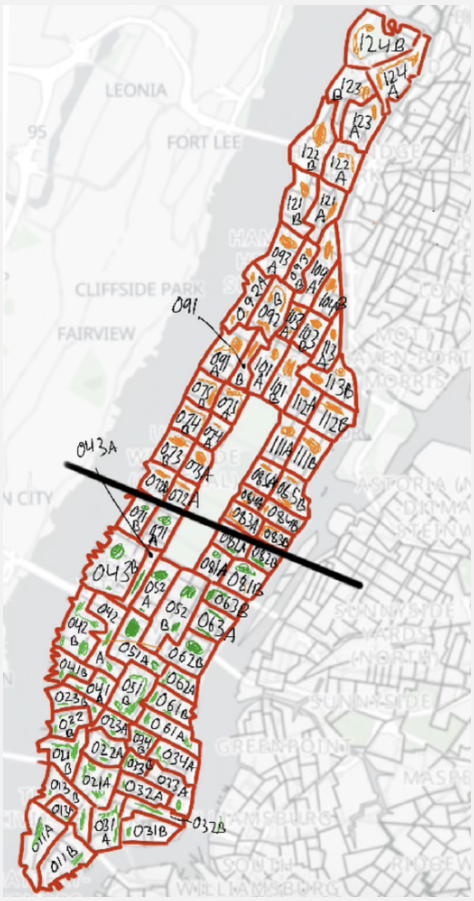


Figure 3: Manhattan Division

We believe that the assumption that trucks are ready in their designated areas before pickup time is reasonable because truck drivers work for longer shifts than just the hour of pickup (a typical shift for a sanitation worker lasts around 8 hours [1]). Therefore, the workers will have enough time to get to the sections before they start picking up garbage.

This division also encourages equity, which we will mention more in our implications.

5 Optimization

5.1 Threshold to Determine Pickup Time

One of the main areas we wanted to examine for optimization was the threshold – measured in population density – used to determine if a section should be cleaned in the morning or at night. To reiterate the importance of this in the model, this has a direct effect on how long trash is sitting out in the street, disturbing the public space, and holds high priority in the amelioration of Manhattan's trash problem. Using some simple Python code over the population density of each section, the mean section population density was computed and used as the initial threshold. Specifically, we found $\mu = 81553.194$ and $\sigma = 28956.875$. To see how the model reacted to this, we sampled 100 evenly spaced thresholds within the range $[\mu - \sigma, \mu + \sigma]$. The goal was to find a threshold that is, of course, realistic and maximizes how many sections can be serviced in the morning so the fewest trash is sitting. We obtained these graphs:

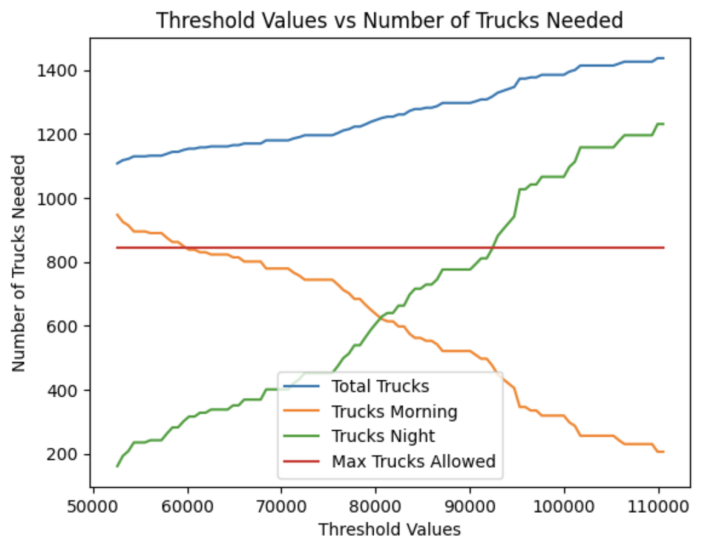


Figure 4: Threshold Values vs Number of Trucks Needed

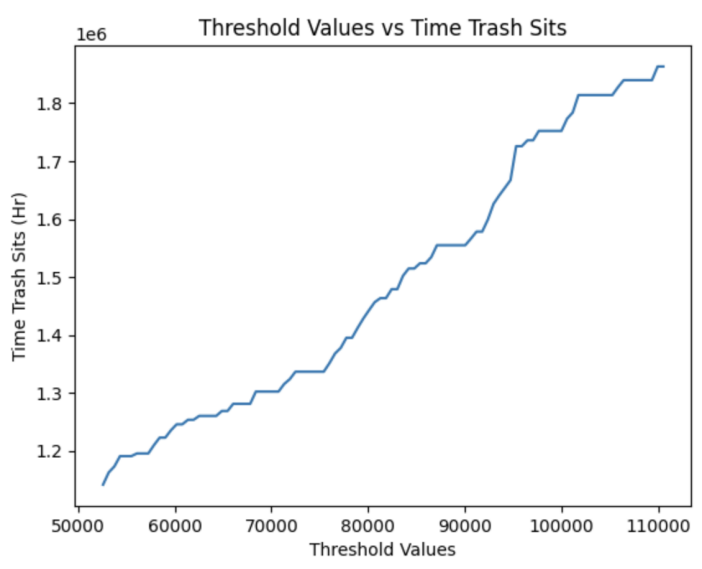


Figure 5: Threshold Values vs Time Trash Sits

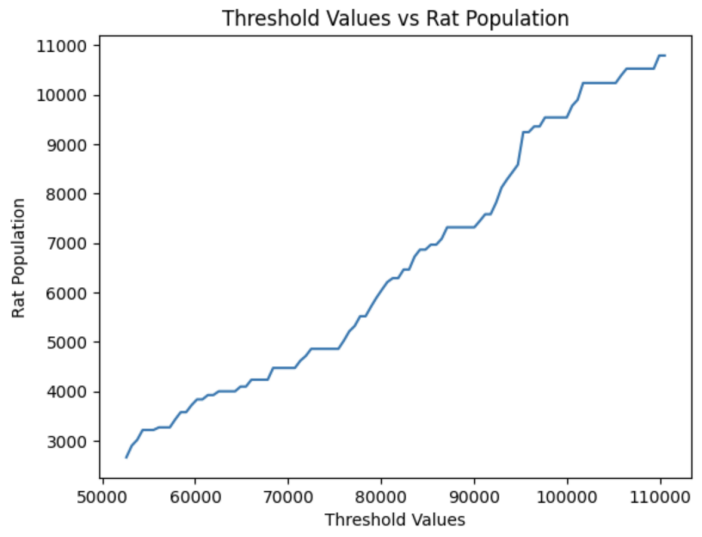


Figure 6: Threshold Values vs Rat Population

After plotting the resulting trucks needed (see code in Appendix C), time trash sits, and rat population, we realized that the threshold must be between 60201.155 to 92375.460, as any higher or lower would cause us to exceed the number of trucks we have. Since the time trash sits on the sidewalks and the rat population both has the lowest value at 60201.155, we found that the optimal threshold was 60201.155.

With this threshold, we can see that more trash sits for 11 hours compared to 22 hours. This in turn reduces the rat population to a relatively sparse distribution across Manhattan.

5.2 Adjacency Matrix

In order to make best use of the function, T, that assigns trucks based on miles of sections, we want to optimize the areas the function receives as input. For instance, feeding each section one at a time into f could generate x trucks needed for section x and y trucks needed for section y, but it's possible that considering the total area at once of section $x + y$ could only need $z < x, y$ trucks. We would like to restrict areas fed into f by considering up to 4 neighboring sections of a given section. To do this, we created an $80x80$ adjacency matrix to represent an undirected graph between sections. We then apply f to each section individually, all s_i, s_j where s_i and s_j are neighbors, as well as for all s_i, s_j, s_k, and all 4-tuples as well. Finally, we do take the minimum of these so we use the fewest trucks to cover the most area.

We decided to create two adjacency matrices: one for the sections on the MWF schedule, and another for the sections on the TTS schedule. Use a dictionary representation, we coded these matrices as such:

```
mwf_neighbors = {
    '072A' : ['072B', '073A', '073B'],
    '072B' : ['072A', '073A','073B'],
    '073A' : ['073B','074A', '074B', '072A', '072B'],
    '073B' : ['073A', '072A', '072B', '074A','074B'],
    '074A' : ['074B',' 073A', '073B', '075A', '075B'],
    '074B' : ['074A', '073A', '073B', '075A', '075B'],
    '075A' : ['075B', '091A', '091B', '074A', '074B','101A'],
    '075B' : ['075A','074A', '074B', '091A', '091B'],
    '083A' : ['083B', '084A', '084B'],
    '083B' : ['083A', '084B'],
    '084A' : [ '084B', '083A', '085A','085B'],
    '084B' : ['084A', '085B', '085A', '083A', '083B'],
    '085A' : ['085B', '084A', '084B', '111A', '111B'],
    '085B' : ['085A', '084A', '084B', '111A', '111B'],
    '111A' : ['111B', '085A', '085B', '112A', '112B'],
    '111B' : ['111A', '085A', '085B', '112A', '112B'],
    '112A' : ['112B', '111A', '111B', '113B', '101B'],
    '112B' : ['112A', '111A', '111B', '113B'],
    '091A' : ['091B','075A', '075B', '092A', '092B'],
    '091B' : ['091A', '075A', '075B', '092B', '101A'],
    '101A' : ['101B', '091B', '092B', '103A', '103B', '075A'],
    '101B' : ['101A', '103A', '103B', '113A', '113B', '112A'],
    '113B' : ['113A', '112A', '112B', '101B'],
    '113A' : ['113B', '103B', '104B', '101B'],
    '103A' : ['103B', '101A', '101B', '092B','104A', '104B'],
    '103B' : ['103A', '113A', '101A', '101B', '104A', '104B'],
    '092A' : ['092B', '091A', '093A'],
    '092B' : ['092A', '091B', '101A', '103A', '093A', '093B', '104A'],
    '093A' : ['093B', '092A', '092B', '121A', '121B'],
    '093B' : ['093A', '092B', '104A', '121A', '121B'],
    '104A' : ['104B', '103A', '103B', '092B', '093B', '121A'],
    '104B' : ['104A', '103A', '103B', '113A'],
    '121A' : ['121B', '093B', '104A', '122A', '122B', '093A'],
    '121B' : ['121A', '093A', '093B', '122A', '122B'],
    '122A' : ['122B', '121A', '121B', '123A', '123B'],
    '122B' : ['122A', '121A', '121B', '123A', '123B'],
    '123A' : ['123B', '122A','122B', '124A'],
    '123B' : ['123A', '122A', '122B', '124A', '124B'],
    '124A' : ['124B', '123A', '123B'],
    '124B' : ['124A', '123B']
}
```

Figure 7: Adjacency Matrix for MWF Sections

```
tts_neighbors = {
    '011A' : ['011B', '0131A', '013B'],
    '011B' : ['011A', '013A', '013B', '031A'],
    '013A' : ['013B', '011A', '011B'],
    '013B' : ['013A', '011A', '011B', '031A', '021A', '021B'],
    '031A' : ['031B', '013B', '021A', '032B', '032A', '011B'],
    '031B' : ['031A', '032B'],
    '032B' : ['033A', '031B', '031A', '032A'],
    '032A' : ['032B', '031A', '021A', '022A', '033B', '033A'],
    '021A' : ['021B', '013B', '031A', '032A', '033B', '022A'],
    '021B' : ['021A', '013B', '022A', '022B', '023A'],
    '022A' : ['022B', '021B', '021A', '032A', '033B', '034B', '023A', '023B'],
    '022B' : ['022A', '023B', '023A', '021B'],
    '033A' : ['033B', '032A', '032B', '034A'],
    '033B' : ['033A', '032A', '022A', '021A', '034B', ',034A'],
    '034A' : ['034B', '033A', '033B', '061A'],
    '034B' : ['034A', '033B', '022A', '023A', '061A', '051B'],
    '023A' : ['023B', '034B', '022A', '022B', '041A', '051B', '061A'],
    '023B' : ['023A', '022B', '041A', '041B'],
    '041A' : ['041B','023B', '051B', '023A', '042A'],
    '041B' : ['041A', '042A', '042B', '023B'],
    '051B' : ['051A', '023A', '034B', '023B', '041A', '042B', '061A', '061B', '062A'],
    '061A' : ['061B', '034A', '034B', '051B', '023A'],
    '061B' : ['061A', '051B', '062A'],
    '062A' : ['062B', '061B', '051B', '051A'],
    '042A' : ['042B', '051A', '043B', '043A', '052A', '041B', '041A', '051B'],
    '042B' : ['042A', '041B', '043B'],
    '051A' : ['051B', '062A', '062B', '042A', '043A', '052A', '052B'],
    '062B' : ['062A', '051A','052B','063A'],
    '052A' : ['052B', '043A', '051A', '042A'],
    '052B' : ['052A', '051A', '062B', '063A', '063B', '081A'],
    '063A' : ['063B', '062B', '052B'],
    '063B' : ['063A', '081A', '081B', '052B'],
    '043A' : ['043B', '052A', '051A', '042A', '071A'],
    '043B' : ['043A', '071A', '071B', '042A', '042B'],
    '071A' : ['071B', '043A', '043B'],
    '071B' : ['071A', '043B'],
    '081A' : ['081B', '063B', '052B', '082A'],
    '081B' : ['081A', '063B', '082B'],
    '082A' : ['082B', '081A', '081B'],
    '082B' : ['082A', '081B']
}
```

Figure 8: Adjacency Matrix for TTS Sections

5.3 Sharing Sections

To approach sharing sections, we iterated through every possible combination of neighbors, up to 4 neighbors, for each section and calculated the number of trucks needed when applying T to their joint area and population. The optimal shared sections were then found by repeatedly taking the minimum of trucks needed for a shared region.

However, when we merged the sections, the minimum number of trucks that we needed was 479 for TTS and 452 for MWF (Appendix C). Since this was higher than the number of trucks we got before merging sections, we determined that under our model, it is not optimal to merge sections.

6 Sensitivity Testing

6.1 Truck Speed in the Morning

Although we reached a value of 6.07 mph through research, we wanted to test how sensitive our model was against different truck speeds in the morning. To do so, we decided to pick a range of different truck speeds in the morning and see how our model reacts (See code in Appendix C).

After running the model with a variety of truck speeds, we obtained this graph:

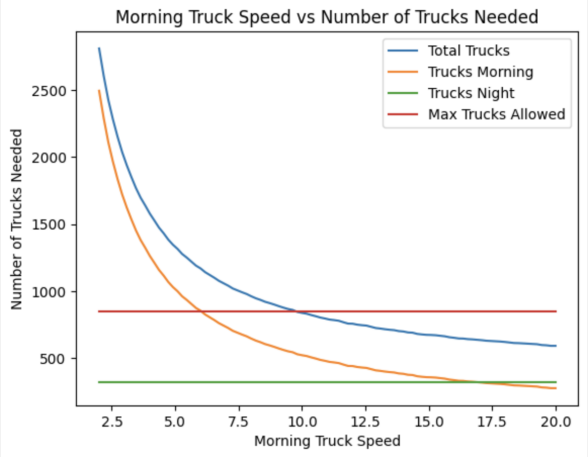


Figure 9: Morning Truck Speed vs Number of Trucks Needed

As we can see from the graph, the model is more sensitive to lower truck speeds, as a small change can lead to a large increase or decrease in the number of trucks needed.

6.2 Truck Speed At Night

Alongside testing how sensitive our model was to truck speed in the morning, we also wanted to test how sensitive it was to changing truck speeds at night. Since our model favors more morning pickups, as it reduces the number of rats and the amount of time trash sits on sidewalks, we were interested in seeing how it would react if the truck speed at night exceeds the speed in the morning.

We obtained this graph (code in Appendix C):

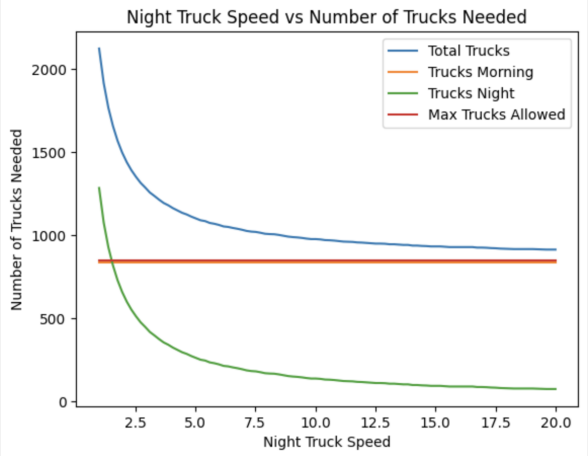


Figure 10: Night Truck Speed vs Number of Trucks Needed

From the graph, truck speed at night appeared even more sensitive to smaller values than truck speed in the morning. It also seemed to ”flatten” out much faster than truck speed in the morning.

7 Results

7.1 Proposed Schedule

After running our model, we were able to produce a schedule that detailed time of pickup, days of pickup, and the number of trucks to send to each section. Here is a sample of the schedule for a few sections (the full schedule is in Appendix D):

Section	Time	Trucks for one cleaning	Cycle
MN124B	Night	17	MonWedFri
MN123A	Night	19	MonWedFri
MN111A	Morning	22	MonWedFri
MN052A	Night	13	TueThuSAt
MN093B	Morning	10	MonWedFri
MN042B	Night	18	TueThuSAt
MN074B	Morning	11	MonWedFri
MN043A	Morning	9	TueThuSAt
MN083B	Morning	12	MonWedFri
MN072B	Morning	10	MonWedFri
MN062B	Morning	24	TueThuSAt
MN061A	Morning	21	TueThuSAt
MN042A	Night	19	TueThuSAt
MN121B	Morning	16	MonWedFri
MN063B	Morning	13	TueThuSAt

Figure 11: A Sample of Our Optimized Schedule

We found that this schedule minimized the amount of time trash gets left on the streets and subsequently minimizes the rat population .

We also derived the following table to show the minimum amount of trash sitting time. The “11-hour sit” and

"22-hour sit" columns represent the number of bags that remain on the street for 11 and 22 hours before collection, respectively.

Day	11 hour sit	22 hour sit
Monday	40,120.570501907700	7,314.387292376870
Tuesday	34,583.25438971630	11,949.787815999100
Wednesday	40,120.570501907700	7,314.387292376870
Thursday	34,583.25438971630	11,949.787815999100
Friday	40,120.570501907700	7,314.387292376870
Saturday	34,583.25438971630	11,949.787815999100
Total	224,111.47467487200	57,792.52532512790

Figure 12: Trash sitting times.

7.2 Evaluating Effectiveness

We can use $P(t, s)$ to evaluate effectiveness directly: the lower the number, the more effective our strategy is. We can also include the number of trucks used and claim that lower numbers are more effective than higher ones. This is because fewer trucks create less harmful emissions and also cost less to operate. Lastly, we can add the number of rats, as, similar to $P(t, s)$, the lower $R(s, t)$ is, the better. Therefore, we can quantitatively evaluate effectiveness by adding $P(t, s)$, $T(t, s)$, and $R(s, t)$. By doing this, the advantages of our model become apparent. As we can see in figure 12, by modeling and optimizing around pickup times, we achieved our goal in significantly picking up more trash in the morning, addressing the problem of streets being piled with garbage. Moreover, the total amount of trucks (Figure 13) falls within our range of allowed values, meaning no excess of new equipment is needed to meet this schedule.

```
{'morning': 450, 'night': 120}
{'morning': 388, 'night': 196}
```

Figure 13: Number of Trucks for MWF (First line) and TTS

8 Implications

8.1 Evaluating Equity

To evaluate equity, we first consider the socioeconomic spread of Manhattan:

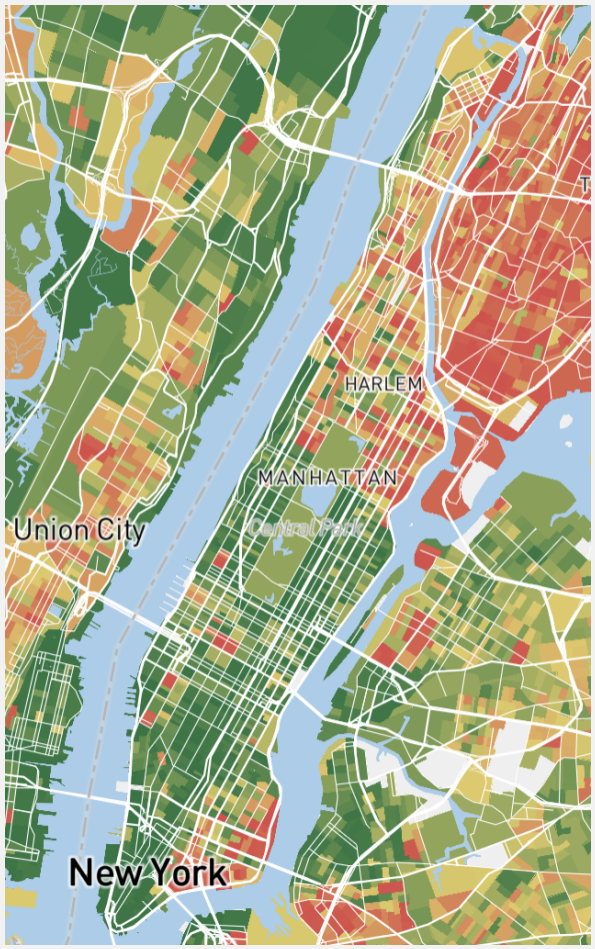


Figure 14: Socioeconomic Spread of Manhanttan

We analyze our model's equity through the lens of multicalibration, a technique in responsible machine learning that ensures model predictions are accurate and unbiased across various subgroups within a population, promoting fairness by aligning model performance consistently for each group.

Generally, the upper half of Manhattan is less affluent than the lower half. By dividing Manhattan horizontally for Monday-Wednesday-Friday and Tuesday-Thursday-Saturday schedules, we address the trash generated by each area proportionally. This ensures that areas with lower socioeconomic status, which may produce less trash relative to affluent neighborhoods, are not left with trash sitting out until a nighttime pickup.

In line with multi-calibration principles, our model aims to ensure that service levels (such as trash collection frequency and timing) are calibrated across socioeconomic groups. This approach minimizes biases that might otherwise lead to unequal trash sitting times in lower-income areas. Thus, our model seeks not only operational efficiency but also fairness, ensuring that no socioeconomic group consistently experiences longer trash sitting times due to inequitable scheduling.

9 Conclusion

In this paper, we optimized the trash collection schedule of Manhattan by proposing and optimizing a model that simulates trash collection in Manhattan. We broke the model into 4 main parts: determining the time and day of trash collection, assigning trucks to each section, finding the total time that trash sits on the sidewalks, and finally modeling the rat population.

While modeling the number of trucks that gets sent to each section, we considered if merging any sections would reduce that number. We discovered that merging sections will not decrease the number of trucks needed to fully pick up all the trash - the amount of trucks needed actually increased. Therefore, we decided that the schedule we come up with should concern only individual sections.

Alongside the number of trucks, we also looked at a function to determine the time of day of pickup, and optimized that function to return the best threshold. We discovered that there is an optimal threshold of 60201.155. With that threshold, the time that garbage sits on the sidewalk and the rat population is at its lowest in our model.

Moreover, since it is logical to assume that the longer trash sits out, the more rats swarm to it, we modeled the rat population with an exponential function. We then used that function to see what schedules minimized the number of rats.

Finally, we used all the optimizations to propose a schedule that reduces the number of rats and improves the overall appearance of the streets, and analyzed its effects on different socioeconomic classes. We ultimately discovered that our schedule will not prioritize more affluent areas.

10 Limitations

The rate of picking up trash, or t_p, was difficult to estimate without concrete data on how long it takes a for a team of 2 garbage men to process a bag of trash. Therefore, we estimated how many pounds of trash there are per mile, and used that to calculate the total pounds of trash a truck can pick up each mile. However, this is not the most accurate estimate we can make, as we made many simplifying assumptions. For example, we assumed that the population and streets in Manhattan are uniformly distributed when it is clear that they are not. Nevertheless, we believe that the distribution of people and streets in Manhattan, in expectation, averages out to be approximately uniformly distributed.

In addition to t_p, our model of the rat population was not the most accurate. Because there are very little literature on the attraction rate of rats towards garbage, we decided to use an exponential function to measure the number of rats as time passes. Although our function did return a higher number as time passed, it was not very heavily based in research. The best way to model the rat population would be to observe them *in vivo*, but we lack the resources to do so.

11 Future Considerations

We have a few ideas to how we could refine and enhance the practicality of our models in the future.

First, our models would benefit from data and model refinement. We believe that improving data accuracy for trash processing times (t_p) and rat attraction rates could strengthen our model. Both of these metrics can be more accurately estimated through partnering with local agencies and wildlife experts.

Second, better socioeconomic and geographic adaption could help provide us a more robust understanding of equity. Right now, we divide our Manhattan boundaries into two halves, but we believe such simple line can not encapsulate the vast differences of the diverse people of New York City. Instead of simple horizontal divisions, future models could use clusters based on socioeconomic and demographic data for fairer trash collection schedules. Perhaps incorporating population density and land-use data could also improve trash output predictions.

Finally, as we are addressing trash collection and rat populations, examining its environmental impacts seem to be a logical next step. Expand our metrics to include fuel/energy consumption, emissions, and seasonal variations could enhance our model's environmental relevance. This would help us not only optimize for efficiency and equity but also contribute to a cleaner, more sustainable urban environment.

12 References

[1] Business Insider. (2020, March 26). A day in the life of a Department of New York City Sanitation worker. Business Insider. Retrieved November 11, 2024, from https://www.businessinsider.com/department-of-new-york-city-sanitation-worker-day-in-the-life.

[2] City of New York. (2024, November 10). Rats Heat Map. NYC Open Data. Retrieved November 10, 2024, from https://data.cityofnewyork.us/Social-Services/Rats-Heat-Map/g642-4e55.

[3] Hoylman-Sigal, B., Schwartz, S. (2024, September). *Analysis of NYC Traffic Congestion and Emergency Response Times*. Retrieved from https://www.nysenate.gov/sites/default/files/admin/structure/media/manage/filefile/a/2024-09/speed-kills-report-9-20-24final.pdf.

[4] NBC New York. (2024, January). *Urban hiker walked every street in Manhattan, covering 640 miles*. NBC New York. Retrieved from https://www.nbcnewyork.com/news/local/urban-hiker-walked-every-street-in-manhattan-covering-640-miles/5093280/.

[5] New York City Department of City Planning. (2020). *2020 Census Tracts (Clipped to Shoreline)*. New York City Planning. Retrieved November 10, 2024, from https://www.nyc.gov/site/planning/data-maps/open-data/census-download-metadata.page.

[6] New York City Department of Sanitation. (2024, April 10). *DSNY frequencies map* [Data set]. NYC Open Data. Retrieved November 10, 2024, from https://data.cityofnewyork.us/City-Government/DSNY-Frequencies-Map-/5qcq-w32p.

[7] New York City Department of Sanitation. (2024, November 8). *DSNY monthly tonnage data*. NYC Open Data. Retrieved November 10, 2024, from https://data.cityofnewyork.us/City-Government/DSNY-Monthly-Tonnage-Data/ebb7-mvp5/aboutdata.

[8] TomTom. (2024). *New York Traffic*. Retrieved November 11, 2024, from https://www.tomtom.com/traffic-index/new-york-traffic/.

[9] U.S. Census Bureau. (2020). *Profile of general population and housing characteristics: Decennial census, DEC demographic profile, Table DP1*. Retrieved November 10, 2024, from https://data.census.gov/table/DECENNIALDP2020.DP1?g=050XX00US36061$1400000060XX00US3606144919160XX00US3651000&d=DEC%20Demographic%20Profile.

A Appendix A: Code to Generate Constants and Variables

A.1 Code to Generate Monthly Refuse Tonnage Collected per DSNY Manhattan District in 2023

The input CSV file, DSNYMonthlyTonnageData20241108.csv, was imported from the NYC Open Data site ([7]), contains monthly tonnage data for refuse collected across New York City.

```
import csv
import matplotlib.pyplot as plt

def filtermanhattanrows(inputfile, outputfile):
    # Open the input CSV file
    with open(inputfile, mode='r', newline='', encoding='utf-8') as csvfile:
        # Use DictReader to access rows as dictionaries
        reader = csv.DictReader(csvfile)

        # Check if 'borough' column is present
        if 'BOROUGH' not in reader.fieldnames:
            raise ValueError("Input file does not have a 'borough' column")

        # Filter rows where borough is "Manhattan"
        filteredrows = [row for row in reader if row['BOROUGH'].strip() == "Manhattan"]

    # Write filtered rows to a new CSV file
    with open(outputfile, mode='w', newline='', encoding='utf-8') as csvfile:
        # Initialize the writer with the same fieldnames
        writer = csv.DictWriter(csvfile, fieldnames=reader.fieldnames)
        writer.writeheader()
        writer.writerows(filteredrows)

def plotrefusevsdistrict(csvfile):
    communitydistricts = []
    refusetonscollected = []
    tonsofrefuse = 0
    # Read the CSV file and extract relevant columns
    with open(csvfile, mode='r', newline='', encoding='utf-8') as file:
        reader = csv.DictReader(file)

        # Ensure required columns exist
        if 'COMMUNITYDISTRICT' not in reader.fieldnames or 'REFUSETONSCOLLECTED' not in reader.
    fieldnames:
            raise ValueError("CSV file must contain 'community district' and 'refusetonscollected
    ' columns")

        for row in reader:
            # Collect data for the plot
            try:
                district = int(row['COMMUNITYDISTRICT'].strip())
                refusetons = float(row['REFUSETONSCOLLECTED'].strip())
                if row['MONTH'][0:4] == '2023':
                    tonsofrefuse += refusetons
                communitydistricts.append(district)
                refusetonscollected.append(refusetons)
            except ValueError:
                # Skip rows with invalid data in either column
                print(f"Skipping invalid row: -row ")
                continue
    print(f'Refuse from 2023: -tonsofrefuse ')

    # Plotting
    plt.figure(figsize=(10, 6))
    plt.scatter(communitydistricts, refusetonscollected, color='blue', alpha=0.6)
    plt.title("Refuse Tons Collected by Community District in Manhattan")
    plt.xlabel("Community District")
    plt.ylabel("Refuse Tons Collected")
    plt.grid(True)
```

```
    plt.show()
# Usage
# inputcsv = 'DSNYMonthlyTonnageData20241108.csv'
outputcsv = 'manhattandata.csv'
# filtermanhattanrows(inputcsv, outputcsv)
plotrefusevsdistrict(outputcsv)
```

A.2 Code to Identify the Current DSNY Schedule for Manhattan

The input CSV file, DSNYFrequencies20241109.csv, was imported from the NYC Open Data site ([6]), contains frequency data for sanitation services across New York City.

```
import csv
import sys
def genschedule(csvfile):
    districts = -
        '01': [],
        '02' : [],
        '03' : [],
        '04' : [],
        '05' : [],
        '06' : [],
        '07': [],
        '08' : [],
        '09' : [],
        '10' : [],
        '11' : [],
        '12' : []

    with open(csvfile, mode='r', newline='', encoding='utf-8') as file:
        reader = csv.DictReader(file)

        # Ensure required columns exist
        # if 'COMMUNITYDISTRICT' not in reader.fieldnames or 'REFUSETONSCOLLECTED' not in reader.
    fieldnames:
        #     raise ValueError("CSV file must contain 'community district' and '
    refusetonscollected' columns")
        printlist = []
        count = 1
        csv.fieldsizelimit(sys.maxsize)
        for row in reader:
            # Collect data for the plot
            try:
                section = row['SECTION'].strip()
                freqrefuse = row['FREQREFUSE']
                if section[0:2] == 'MN':
                    district = section[2:4]
                    districts[district].append(freqrefuse)
                    printlist.append(f'-section  : -freqrefuse ')
                    count += 1

            except ValueError:
                # Skip rows with invalid data in either column
                print(f"Skipping invalid row: -row ")
                continue
        for item in sorted(printlist, key= lambda string : int(string[2:4])):
            print(item)
        #print(districts)
genschedule('DSNYFrequencies20241109.csv')
```

A.3 Notebooks Code to Calculate Population, Area, and Population Density for DSNY Manhattan Sections.

Step 1: Install Required Libraries

We install the necessary Python libraries for geospatial data processing: geopandas, fiona, and shapely. These libraries enable us to work with geospatial data, read shapefiles, and perform geometric operations in our analysis.

```
!pip install geopandas
!pip install fiona
!pip install shapely
```

```
Requirement already satisfied: geopandas in
/usr/local/lib/python3.10/dist-packages (1.0.1)
Requirement already satisfied: numpy>=1.22 in
/usr/local/lib/python3.10/dist-packages (from geopandas) (1.26.4)
Requirement already satisfied: pyogrio>=0.7.2 in
/usr/local/lib/python3.10/dist-packages (from geopandas) (0.10.0)
Requirement already satisfied: packaging in
/usr/local/lib/python3.10/dist-packages (from geopandas) (24.1)
Requirement already satisfied: pandas>=1.4.0 in
/usr/local/lib/python3.10/dist-packages (from geopandas) (2.2.2)
Requirement already satisfied: pyproj>=3.3.0 in
/usr/local/lib/python3.10/dist-packages (from geopandas) (3.7.0)
Requirement already satisfied: shapely>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from geopandas) (2.0.6)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.10/dist-packages (from pandas>=1.4.0-
>geopandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in
/usr/local/lib/python3.10/dist-packages (from pandas>=1.4.0-
>geopandas) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in
/usr/local/lib/python3.10/dist-packages (from pandas>=1.4.0-
>geopandas) (2024.2)
Requirement already satisfied: certifi in
/usr/local/lib/python3.10/dist-packages (from pyogrio>=0.7.2-
>geopandas) (2024.8.30)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.2-
>pandas>=1.4.0->geopandas) (1.16.0)
Collecting fiona
  Downloading fiona-1.10.1-cp310-cp310-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (56 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 56.6/56.6 kB 2.8 MB/s eta
0:00:00
ent already satisfied: attrs>=19.2.0 in
/usr/local/lib/python3.10/dist-packages (from fiona) (24.2.0)
Requirement already satisfied: certifi in
```

```
/usr/local/lib/python3.10/dist-packages (from fiona) (2024.8.30)
Requirement already satisfied: click~=8.0 in
/usr/local/lib/python3.10/dist-packages (from fiona) (8.1.7)
Collecting click-plugins>=1.0 (from fiona)
  Downloading click_plugins-1.1.1-py2.py3-none-any.whl.metadata (6.4
kB)
Collecting cligj>=0.5 (from fiona)
  Downloading cligj-0.7.2-py3-none-any.whl.metadata (5.0 kB)
Downloading fiona-1.10.1-cp310-cp310-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl (17.3 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 17.3/17.3 MB 66.3 MB/s eta
0:00:00
ent already satisfied: shapely in /usr/local/lib/python3.10/dist-
packages (2.0.6)
Requirement already satisfied: numpy<3,>=1.14 in
/usr/local/lib/python3.10/dist-packages (from shapely) (1.26.4)
```

Step 2: Importing Related Files

We import the necessary files for our analysis: the contents of New York City census tracts ZIP file(`nyct2020_24c.zip`; New York City Department of City Planning, 2020) after unzipping, the Manhattan demographic data by census tracts (`DECENNIALDP2020.DP1-Data.csv`; U.S. Census Bureau, 2020), and the schedules and boundaries of the sections in the DSNY Frequencies map, filtered to include only the sections of Manhattan sanitary districts (`DSNY - Manhattan Frequencies.csv`; New York City Department of Sanitation, 2020). These files provide the essential data for understanding population distribution, geographic boundaries, and sanitation schedules.

```
from google.colab import files
uploaded = files.upload()

<IPython.core.display.HTML object>

Saving DECENNIALDP2020.DP1-Data.csv to DECENNIALDP2020.DP1-Data.csv
Saving DSNY - Manhattan Frequencies.csv to DSNY - Manhattan
Frequencies.csv
Saving nyct2020.dbf to nyct2020.dbf
Saving nyct2020.prj to nyct2020.prj
Saving nyct2020.shp to nyct2020.shp
Saving nyct2020.shp.xml to nyct2020.shp.xml
Saving nyct2020.shx to nyct2020.shx
```

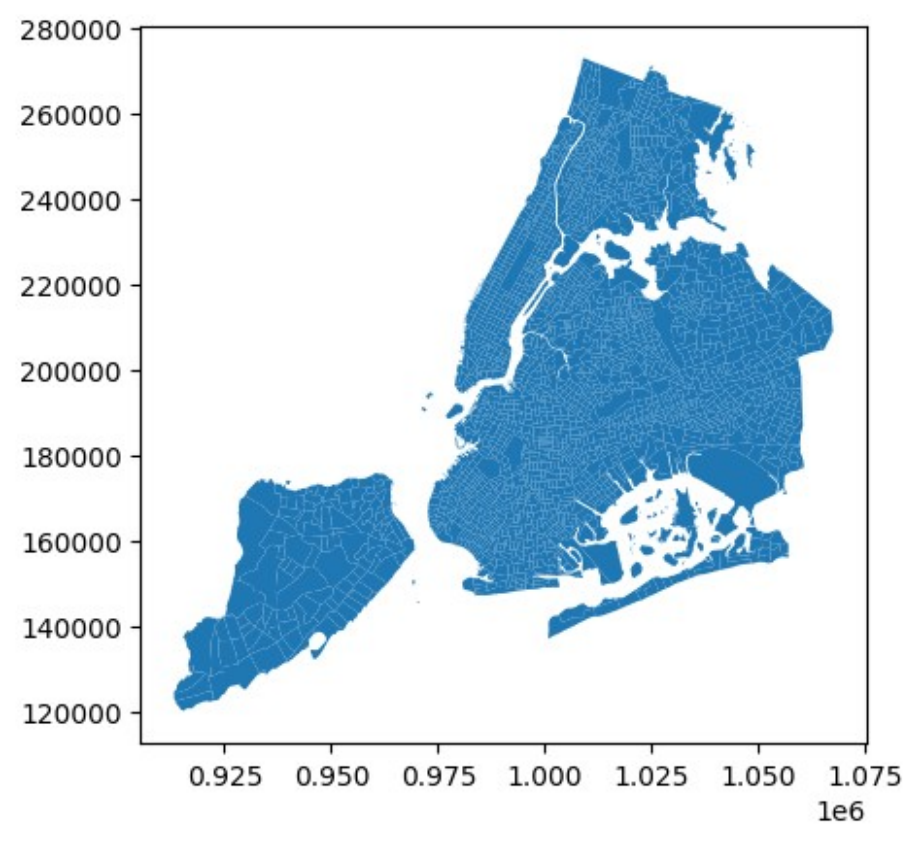
Step 3: Reading Census Tracts Boundaries

We load the `nyct2020.shp` shapefile using GeoPandas, inspect its contents, and verify the Coordinate Reference System (CRS). We then display the first few rows of the data and generate a plot to visualize the geographic boundaries of the census tracts.

```
import geopandas as gpd
import pandas as pd

shapefile_path = 'nyct2020.shp'

gdf_tract = gpd.read_file(shapefile_path)
print("Shapefile Data (First Few Rows):")
print(gdf_tract.head())

print("CRS of GDF")
print(gdf_tract.crs)
print(gdf_tract.plot())
```

```
Shapefile Data (First Few Rows):
  CTLabel BoroCode   BoroName  CT2020 BoroCT2020 CDEligibil  \
0       1        1  Manhattan  000100    1000100       None
1    2.01        1  Manhattan  000201    1000201       None
2       6        1  Manhattan  000600    1000600       None
3   14.01        1  Manhattan  001401    1001401       None
4   14.02        1  Manhattan  001402    1001402       None

                                              NTAName NTA2020 CDTA2020
\
0  The Battery-Governors Island-Ellis Island-Libe...  MN0191     MN01

1                              Chinatown-Two Bridges  MN0301     MN03

2                              Chinatown-Two Bridges  MN0301     MN03

3                                   Lower East Side  MN0302     MN03

4                                   Lower East Side  MN0302     MN03


                                          CDTANAME        GEOID
PUMA  \
0  MN01 Financial District-Tribeca (CD 1 Equivalent)  36061000100
4121
1   MN03 Lower East Side-Chinatown (CD 3 Equivalent)  36061000201
4103
2   MN03 Lower East Side-Chinatown (CD 3 Equivalent)  36061000600
4103
3   MN03 Lower East Side-Chinatown (CD 3 Equivalent)  36061001401
4103
```

```
4   MN03 Lower East Side-Chinatown (CD 3 Equivalent)  36061001402
4103

     Shape_Leng    Shape_Area  \
0  10833.043929  1.843005e+06
1   4754.495247  9.723121e+05
2   6976.286215  2.582705e+06
3   5075.332000  1.006117e+06
4   4459.156019  1.226206e+06

                                                geometry
0  MULTIPOLYGON (((972081.788 190733.467, 972184....
1  POLYGON ((988548.218 197770.375, 987978.808 19...
2  POLYGON ((986961.185 199553.643, 987206.139 19...
3  POLYGON ((987475.016 200297.218, 987705.443 20...
4  POLYGON ((988387.669 201258.312, 988621.002 20...
CRS of GDF
EPSG:2263
Axes(0.22093,0.11;0.58314x0.77)
```



Step 4: Load Population Data from Census Data

In this step, we import the `DECENNIALDP2020.DP1-Data.csv` file and filter it to retain only the relevant columns: GEO_ID, NAME (the name of each Census Tract), and DP1_0001C (population data for each tract). The GEO_ID values are cleaned by removing the prefix "1400000US" to ensure they match the format of the GEOID in the Census Tract Boundaries. The population data is then converted to numeric values to ensure it is ready for further analysis.

```
import geopandas as gpd
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import colors

df = pd.read_csv('DECENNIALDP2020.DP1-Data.csv')
df['GEO_ID'] = df['GEO_ID'].astype(str)
df['GEO_ID'] = df['GEO_ID'].str.replace('1400000US', '', regex=False)
df = df[['GEO_ID', 'NAME', 'DP1_0001C']]
df['DP1_0001C'] = pd.to_numeric(df['DP1_0001C'], errors='coerce')
print(df.head())
```

```
                GEO_ID
NAME  \
0            Geography                            Geographic Area Name

1  0600000US3606144919  Manhattan borough, New York County, New York

2          36061000100     Census Tract 1; New York County; New York

3          36061000201  Census Tract 2.01; New York County; New York

4          36061000202  Census Tract 2.02; New York County; New York


   DP1_0001C
0        NaN
1  1694251.0
2        0.0
3     2012.0
4     7266.0
```

Step 5: Match GEO_ID between Census Tract and Population of Census Tract

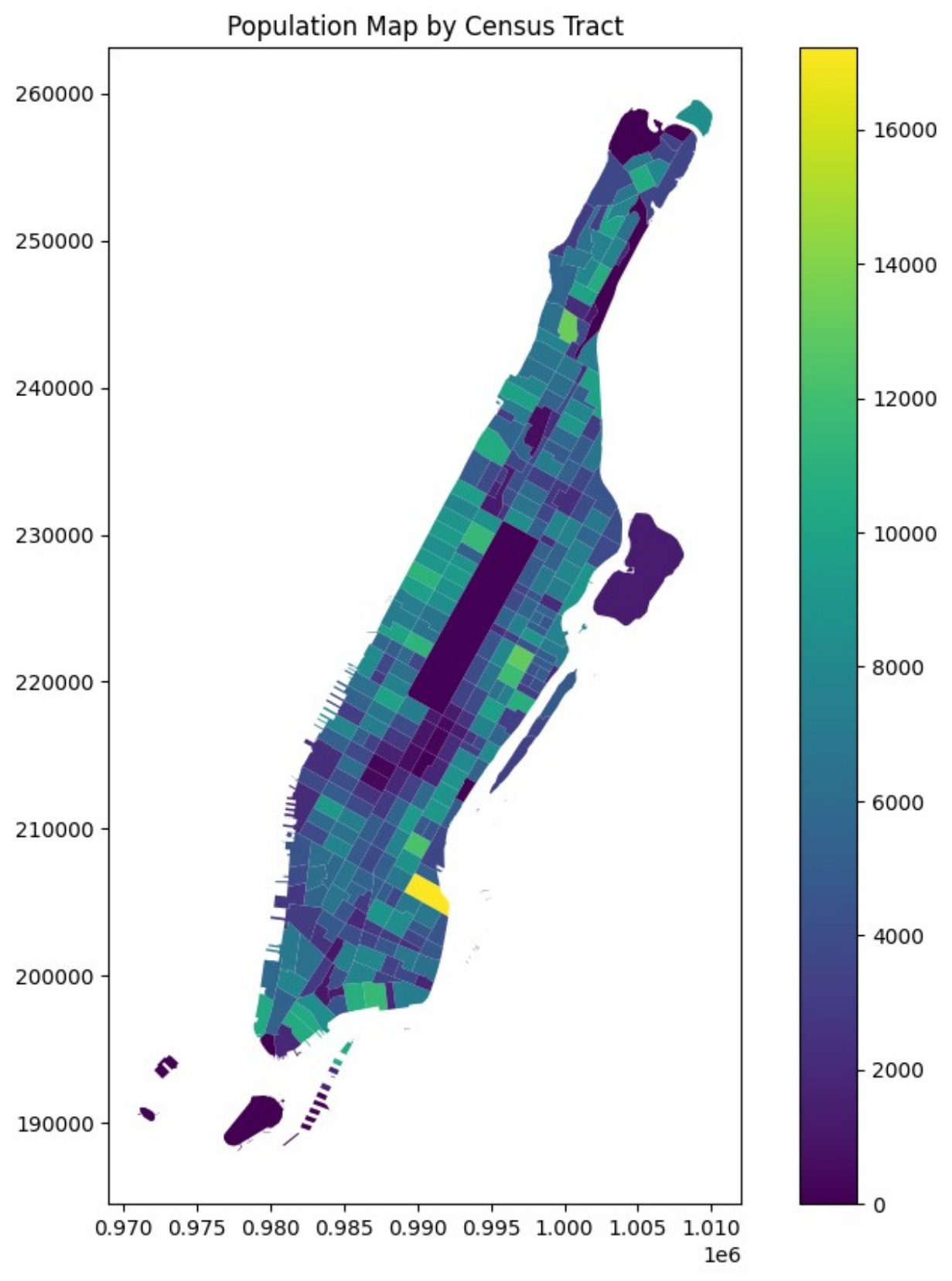
We match the GEO_ID values from the census tracts data with the corresponding population data for each tract. This ensures that population information is correctly linked to its respective census tract.

The data is then sorted by population in descending order, and the relevant columns—NAME, DP1_0001C (population), and geometry—are selected.

We then plot the population data on a map, color-coded by population density, with warmer colors representing higher population densities.

```
gdf_tract.rename(columns={'GEOID': 'GEO_ID'}, inplace=True)
gdf_tract['GEO_ID'] = gdf_tract['GEO_ID'].astype(str)

gdf_tract_pop = gdf_tract.merge(df, on='GEO_ID')
gdf_tract_pop = gdf_tract_pop[[col for col in gdf_tract_pop if col !=
'geometry'] + ['geometry']]
gdf_tract_pop = gdf_tract_pop[['NAME', 'DP1_0001C', 'geometry']]
gdf_tract_pop = gdf_tract_pop.sort_values(by='DP1_0001C',
ascending=False)

print(gdf_tract_pop.head())

norm = colors.Normalize(vmin=gdf_tract_pop['DP1_0001C'].min(),
vmax=gdf_tract_pop['DP1_0001C'].max())
ax = gdf_tract_pop.plot(column='DP1_0001C', figsize=(10, 10),
legend=True,
                        cmap='viridis', norm=norm)
plt.title('Population Map by Census Tract')
plt.show()
```

```
                                            NAME  DP1_0001C  \
157   Census Tract 44; New York County; New York    17222.0
273  Census Tract 245; New York County; New York    13385.0
207  Census Tract 138; New York County; New York    13109.0
225   Census Tract 66; New York County; New York    12341.0
205  Census Tract 134; New York County; New York    11882.0

                                              geometry
157  MULTIPOLYGON (((994681.406 203127.675, 994780....
273  POLYGON ((1000900.245 244770.33, 1000854.156 2...
207  POLYGON ((998004.556 221839.304, 997877.341 22...
225  POLYGON ((990299.771 209477.741, 990580.025 20...
205  POLYGON ((997354.606 220665.721, 997216.263 22...
```

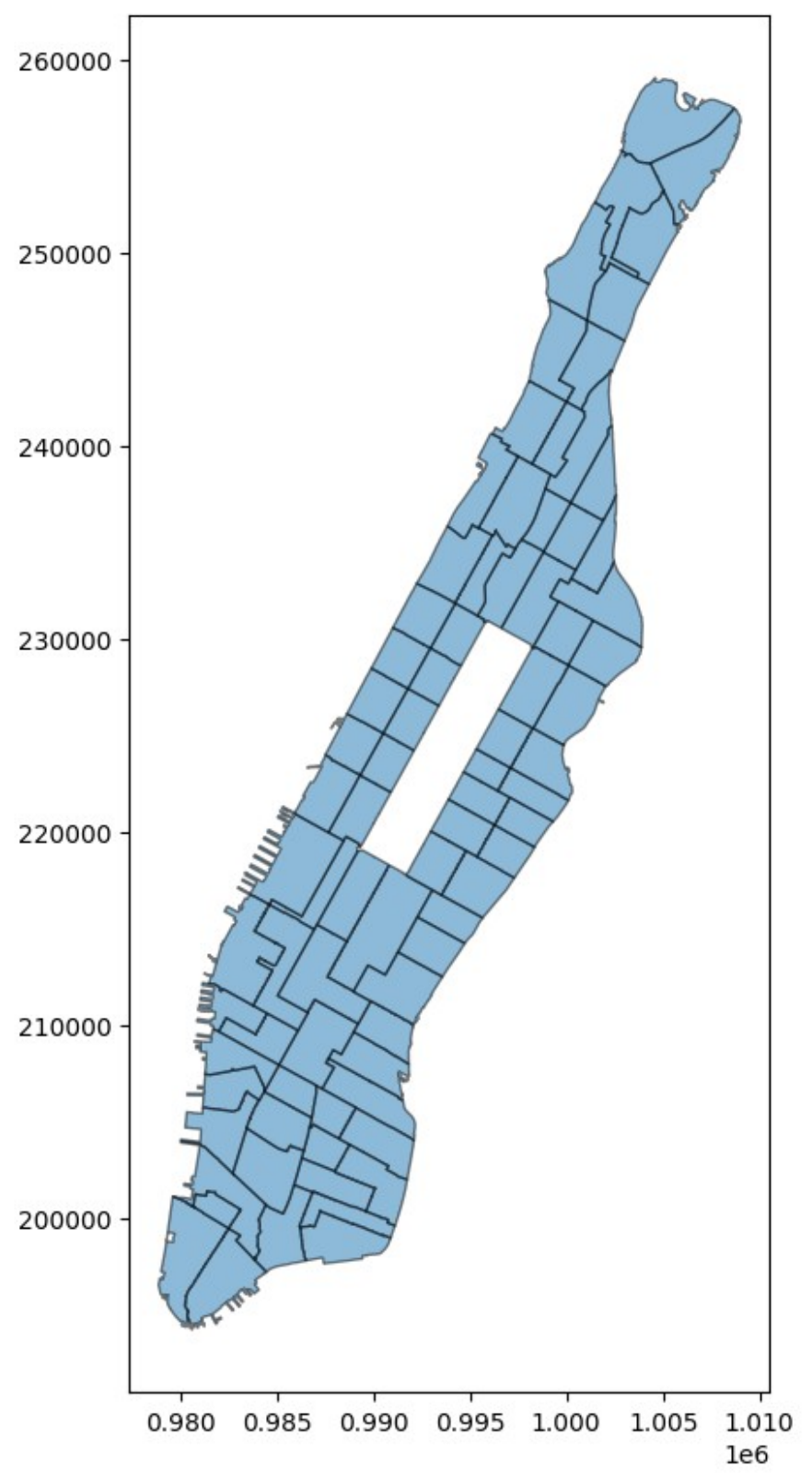

Population Map by Census Tract
260000
250000
240000
230000
220000
210000
200000
190000
0.970
0.975
0.980
0.985
0.990
0.995
1.000
1.005
1.010
1e6
16000
14000
12000
10000
8000
6000
4000
2000
0

Step 6: Process Sub-Sections of the Sanitary District

We load `DSNY - Manhattan Frequencies.csv` and filter it to retain unique SECTIONs. If a SECTION (for example, MN01) has both SCHEDULE A and SCHEDULE B, we create separate entries, MN01A and MN01B. Then we plot the GeoDataFrame matching with the census tracts' CRS.

```
import pandas as pd
import geopandas as gpd
from shapely import wkt

df = pd.read_csv('DSNY - Manhattan Frequencies.csv')
df['geometry'] = df['multipolygon'].apply(wkt.loads)

gdf_section = gpd.GeoDataFrame(df, geometry='geometry')
gdf_section['SECTION'] = gdf_section['SECTION'] +
gdf_section['FREQUENCY']
gdf_section = gdf_section[['SECTION', 'geometry']]

# Set the initial CRS to WGS84 (EPSG:4326).
gdf_section.set_crs('EPSG:4326', inplace=True)

# Reproject the GeoDataFrame to match the existing map's CRS
(EPSG:2263)
gdf_section = gdf_section.to_crs('EPSG:2263')

gdf_section.plot(figsize=(10, 10), edgecolor='black', alpha=0.5)
```

```
<Axes: >
```


260000
250000
240000
230000
220000
210000
200000
0.980
0.985
0.990
0.995
1.000
1.005
1.010
1e6

Step 7: Calculate Population for Each Section

We perform a spatial join between the section boundaries (gdf_section) and the census tract population data (gdf_tract_pop) using the intersects predicate. This allows us to associate each sub-section with the relevant census tracts.

We then calculate the intersection area of each sub-section and tract and normalize the population data based on the intersection area. The population for each sub-section is approximated by multiplying the proportion of the intersection area by the total population of the corresponding census tract.

```
gdf_section_pop = gpd.sjoin(gdf_section, gdf_tract_pop, how="left",
predicate='intersects')
gdf_section_pop = gdf_section_pop.sort_values(by='NAME',
ascending=True)
gdf_section_pop['intersection_area'] = gdf_section_pop.apply(lambda
row: row.geometry.intersection(row.geometry).area, axis=1)
intersection_area_sum = gdf_section_pop.groupby('NAME')
['intersection_area'].transform('sum')
gdf_section_pop['population'] = (gdf_section_pop['intersection_area']
/ intersection_area_sum) * gdf_section_pop['DP1_0001C']
print(gdf_section_pop.head())
```

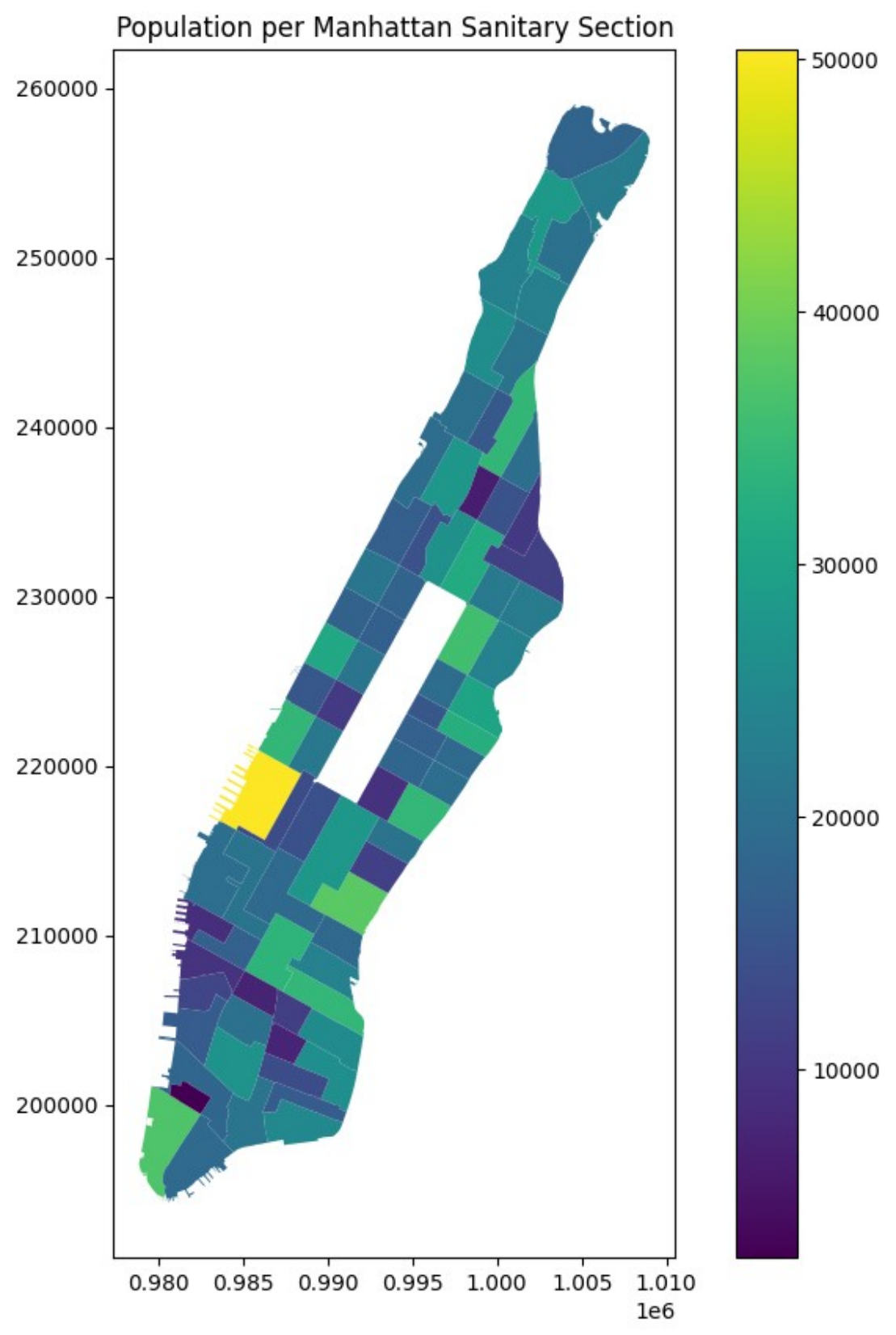
```
   SECTION                                           geometry
index_right  \
34  MN033A  MULTIPOLYGON (((988031.855 202415.936, 988038....
154
58  MN032B  MULTIPOLYGON (((989634.202 200301.194, 989687....
154
2   MN031B  MULTIPOLYGON (((986519.228 197782.935, 986518....
154
15  MN032A  MULTIPOLYGON (((987914.012 202457.438, 988031....
178
34  MN033A  MULTIPOLYGON (((988031.855 202415.936, 988038....
178

                                          NAME  DP1_0001C  \
34  Census Tract 10.01; New York County; New York     1767.0
58  Census Tract 10.01; New York County; New York     1767.0
2   Census Tract 10.01; New York County; New York     1767.0
15  Census Tract 10.02; New York County; New York     6300.0
34  Census Tract 10.02; New York County; New York     6300.0

    intersection_area   population
34       6.964273e+06   637.569691
58       4.455337e+06   407.880007
2        7.881607e+06   721.550302
15       5.504890e+06  2049.148178
34       6.964273e+06  2592.390900
```

Step 8: Aggregate Population and Plot Final Results

We aggregate the population data for each unique SECTION by merging the geometries of the sections within each district using the unary_union function to. The population for each district is then summed.

We generate a color-coded map using the viridis colormap, where the color intensity represents the population size, with warmer colors indicating higher populations.

```
from shapely.ops import unary_union

gdf_final = gdf_section_pop.groupby(['SECTION'], as_index=False).agg({
    'geometry': lambda x: unary_union(x),
    'population': 'sum'
})
gdf_final = gpd.GeoDataFrame(gdf_final, geometry='geometry')
gdf_final = gdf_final.sort_values(by='population', ascending=False)
norm = colors.Normalize(vmin=gdf_final['population'].min(),
vmax=gdf_final['population'].max())
ax = gdf_final.plot(column='population', figsize=(10, 10),
legend=True,
                    cmap='viridis', norm=norm)

plt.title('Population per Manhattan Sanitary Section')
plt.show()
```


Population per Manhattan Sanitary Section
260000
250000
240000
230000
220000
210000
200000
0.980
0.985
0.990
0.995
1.000
1.005
1.010
1e6
50000
40000
30000
20000
10000

Step 9: Reproject to Local CRS and Calculate Population Density

We first set the coordinate reference system (CRS) of the gdf_final GeoDataFrame to EPSG:2236, then reproject it to EPSG:2263 for consistency with local spatial data standards.

We calculate the area of each section by dividing the geometry area by a constant (27,878,400) to convert square foot to miles square. Using the calculated area, we compute the population density for each section by dividing the population by the area.

```
gdf_final = gdf_final.set_crs('EPSG:2236', allow_override=True)
gdf_final = gdf_final.to_crs(epsg=2263)
gdf_final['AREA'] = gdf_final.geometry.area / 27878400
gdf_final['population_density'] = gdf_final['population'] /
gdf_final['AREA']
geometry_column = gdf_final.pop('geometry')
gdf_final['geometry'] = geometry_column

print(gdf_final.head())
```

```
   SECTION    population      AREA  population_density  \
23  MN043B  50378.061365  0.699221         72048.834281
31  MN062B  38204.797955  0.458590         83309.225962
0   MN011A  37262.861770  0.588056         63366.201610
66  MN111A  35990.692268  0.391768         91867.360610
45  MN081B  34493.552647  0.356352         96796.280293

                                             geometry
23  POLYGON ((-3898265.126 -11559497.202, -3898395...
31  POLYGON ((-3895735.293 -11570633.436, -3895734...
0   POLYGON ((-3911867.821 -11583481.959, -3911868...
66  POLYGON ((-3881957.938 -11550914.048, -3881958...
45  POLYGON ((-3892324.466 -11563145.791, -3892310...
```

Step 10: Save and Download the CSV File

We save the gdf_final GeoDataFrame as a CSV file and download the file directly to our local machine.

```
from google.colab import files

csv_file_path = '/content/gdf_final_population_data.csv'
gdf_final.to_csv(csv_file_path, index=False)

files.download(csv_file_path)
```

A.4 Code for Road Length per Section of Each DSNY Manhattan District

```
# -*- coding: utf-8 -*-
"""Road Length per Section.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1StJIleNeHk5Bjb-N7gpzYMYDIz3Xb6a0
"""

from google.colab import files
uploaded = files.upload()

import pandas as pd

data = pd.readcsv('gdffinalpopulationdata.csv')
data['roadlength'] = 640 * data['AREA'] / 21.34
data.tocsv('gdffinalpopulationdatawithroadlength.csv', index=False)
files.download('gdffinalpopulationdatawithroadlength.csv')
```

A.5 Code for Rat Incident Reports in Manhattan in 2023

The input CSV file, Rats__Heat_Map.csv, was imported from the NYC Open Data ([2]), contains all 311 Service Requests from 2010 to present.

```
# -*- coding: utf-8 -*-
"""RathattanOverTime.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1SOWBtO7fyxQ9JlW47n4sPttdBqPmHB7w
"""

from google.colab import files
uploaded = files.upload()

import pandas as pd

df = pd.readcsv('Rats Heat Map.csv')
df = df[['Created Date', 'Community Board']]
df = df[df['Community Board'].str.contains('MANHATTAN', case=False, na=False)]
df['Created Date'] = pd.todatetime(df['Created Date'], format='%m/%d/%Y %I:%M:%S %p')
df = df.sortvalues(by='Created Date', ascending=False).resetindex(drop=True)

print(df.head())

import pandas as pd
import matplotlib.pyplot as plt

startdate = '2023-01-01'
enddate = '2024-01-01'
filtereddf = df[(df['Created Date']>=startdate) & (df['Created Date'] < enddate)].copy()

filtereddf['Hour'] = filtereddf['Created Date'].dt.hour
filtereddf = filtereddf[filtereddf['Hour']>=7]
hourlycounts = filtereddf.groupby('Hour').size()

plt.figure(figsize=(12, 6))
hourlycounts.plot(kind='bar')
plt.xlabel('Hour of the Day')
plt.ylabel('Number of Incident Reports')
plt.title('Number of Incident Reports per Hour of the Day')
plt.xticks(rotation=0)

for index, value in enumerate(hourlycounts):
    plt.text(index, value, f'-value ', ha='center', va='bottom')
```

```
plt.show()
```

A.6 Code for Calculating Statistical Measures of Population Density in Manhattan

The input CSV file, gdffinalpopulationdata-2.csv, is the .csv output from Appendix A.3.

```
import numpy as np
import csv

def stats(csvfile):
    numlist = []
    with open(csvfile, mode='r', encoding='utf-8') as file:
        reader = csv.DictReader(file)
        for row in reader:
            numlist.append(float(row['populationdensity']))

    sort = np.sort(numlist)
    median = np.median(sort)
    std = np.std(sort)
    mean = np.mean(sort)
    q1 = np.percentile(sort, 25)
    q3 = np.percentile(sort, 75)
    print(f'Median: {median} ')
    print(f'Std: {std} ')
    print(f'Mean: {mean} ')
    print(f'Q1: {q1} ')
    print(f'Q3: {q3} ')

stats('gdffinalpopulationdata-2.csv')
```

B Appendix B: Output Data from Code in Appendix A

B.1 Monthly Refuse Tonnage Collected per DSNY Manhattan District in 2023 (.csv output from Appendix A.1)

The REFUSETONSCOLLECTED column represents the total amount of refuse collected in tonnes each month (indicated by the MONTH column) for each district (COMMUNITYDISTRICT) of Manhattan borough. Note that this .csv have been sorted by BOROUGH, MONTH, then COMMUNITY DISTRICT for better visual representation.

MONTH	BOROUGH	COMMUNITYDISTRICT	REFUSETONSCOLLECTED
2023 / 01	Manhattan	1	1341.9
2023 / 01	Manhattan	2	1783.7
2023 / 01	Manhattan	3	2652.4
2023 / 01	Manhattan	4	2272.7
2023 / 01	Manhattan	5	1247.2
2023 / 01	Manhattan	6	2736.8
2023 / 01	Manhattan	7	4305.5
2023 / 01	Manhattan	8	4853.8
2023 / 01	Manhattan	9	2097.4
2023 / 01	Manhattan	10	2554.2
2023 / 01	Manhattan	11	2387.0
2023 / 01	Manhattan	12	4485.8
2023 / 02	Manhattan	1	1192.3
2023 / 02	Manhattan	2	1610.8
2023 / 02	Manhattan	3	2325.0
2023 / 02	Manhattan	4	2074.2
2023 / 02	Manhattan	5	1098.7
2023 / 02	Manhattan	6	2434.2
2023 / 02	Manhattan	7	3785.3
2023 / 02	Manhattan	8	4238.0
2023 / 02	Manhattan	9	1830.7
2023 / 02	Manhattan	10	2185.8
2023 / 02	Manhattan	11	2068.8
2023 / 02	Manhattan	12	3869.4
2023 / 03	Manhattan	1	1319.5
2023 / 03	Manhattan	2	1762.9
2023 / 03	Manhattan	3	2679.6
2023 / 03	Manhattan	4	2276.8
2023 / 03	Manhattan	5	1221.7
2023 / 03	Manhattan	6	2711.6
2023 / 03	Manhattan	7	4234.8
2023 / 03	Manhattan	8	4661.6
2023 / 03	Manhattan	9	2084.6
2023 / 03	Manhattan	10	2505.8
2023 / 03	Manhattan	11	2383.2
2023 / 03	Manhattan	12	4471.7
2023 / 04	Manhattan	1	1257.4
2023 / 04	Manhattan	2	1707.7
2023 / 04	Manhattan	3	2381.7

Figure 15: Monthly Tonnage of Refuse Collected by Manhattan District in 2023 (1 of 4)

2023 / 04	Manhattan	4	2193.7
2023 / 04	Manhattan	5	1190.9
2023 / 04	Manhattan	6	2587.7
2023 / 04	Manhattan	7	3992.7
2023 / 04	Manhattan	8	4492.2
2023 / 04	Manhattan	9	1937.3
2023 / 04	Manhattan	10	2329.5
2023 / 04	Manhattan	11	2200.7
2023 / 04	Manhattan	12	4068.8
2023 / 05	Manhattan	1	1437.0
2023 / 05	Manhattan	2	1963.8
2023 / 05	Manhattan	3	2804.1
2023 / 05	Manhattan	4	2505.2
2023 / 05	Manhattan	5	1353.7
2023 / 05	Manhattan	6	2977.9
2023 / 05	Manhattan	7	4593.2
2023 / 05	Manhattan	8	5012.4
2023 / 05	Manhattan	9	2205.1
2023 / 05	Manhattan	10	2598.0
2023 / 05	Manhattan	11	2481.8
2023 / 05	Manhattan	12	4710.9
2023 / 06	Manhattan	1	1355.9
2023 / 06	Manhattan	2	1812.8
2023 / 06	Manhattan	3	2583.5
2023 / 06	Manhattan	4	2419.7
2023 / 06	Manhattan	5	1281.2
2023 / 06	Manhattan	6	2785.8
2023 / 06	Manhattan	7	4408.2
2023 / 06	Manhattan	8	4820.6
2023 / 06	Manhattan	9	2100.8
2023 / 06	Manhattan	10	2517.3
2023 / 06	Manhattan	11	2397.8
2023 / 06	Manhattan	12	4483.9
2023 / 07	Manhattan	1	1238.0
2023 / 07	Manhattan	2	1629.8
2023 / 07	Manhattan	3	2546.3
2023 / 07	Manhattan	4	2288.2
2023 / 07	Manhattan	5	1187.1
2023 / 07	Manhattan	6	2656.0
2023 / 07	Manhattan	7	4018.4

Figure 16: Monthly Tonnage of Refuse Collected by Manhattan District in 2023 (2 of 4)

2023 / 07	Manhattan	8	4284.6
2023 / 07	Manhattan	9	2003.3
2023 / 07	Manhattan	10	2475.0
2023 / 07	Manhattan	11	2242.5
2023 / 07	Manhattan	12	4447.7
2023 / 08	Manhattan	1	1300.9
2023 / 08	Manhattan	2	1734.2
2023 / 08	Manhattan	3	2694.1
2023 / 08	Manhattan	4	2389.0
2023 / 08	Manhattan	5	1245.9
2023 / 08	Manhattan	6	2803.3
2023 / 08	Manhattan	7	4133.0
2023 / 08	Manhattan	8	4460.2
2023 / 08	Manhattan	9	2083.6
2023 / 08	Manhattan	10	2520.1
2023 / 08	Manhattan	11	2316.6
2023 / 08	Manhattan	12	4505.1
2023 / 09	Manhattan	1	1398.6
2023 / 09	Manhattan	2	1816.3
2023 / 09	Manhattan	3	2548.9
2023 / 09	Manhattan	4	2420.0
2023 / 09	Manhattan	5	1296.0
2023 / 09	Manhattan	6	2782.8
2023 / 09	Manhattan	7	4367.6
2023 / 09	Manhattan	8	4900.2
2023 / 09	Manhattan	9	2093.2
2023 / 09	Manhattan	10	2522.0
2023 / 09	Manhattan	11	2349.5
2023 / 09	Manhattan	12	4530.3
2023 / 10	Manhattan	1	1417.4
2023 / 10	Manhattan	2	1884.7
2023 / 10	Manhattan	3	2655.0
2023 / 10	Manhattan	4	2446.6
2023 / 10	Manhattan	5	1339.5
2023 / 10	Manhattan	6	2835.2
2023 / 10	Manhattan	7	4453.7
2023 / 10	Manhattan	8	4999.9
2023 / 10	Manhattan	9	2144.7
2023 / 10	Manhattan	10	2534.6
2023 / 10	Manhattan	11	2356.1

Figure 17: Monthly Tonnage of Refuse Collected by Manhattan District in 2023 (3 of 4)

2023 / 10	Manhattan	12	4602.8
2023 / 11	Manhattan	1	1323.5
2023 / 11	Manhattan	2	1824.9
2023 / 11	Manhattan	3	2525.4
2023 / 11	Manhattan	4	2324.9
2023 / 11	Manhattan	5	1270.8
2023 / 11	Manhattan	6	2684.3
2023 / 11	Manhattan	7	4446.2
2023 / 11	Manhattan	8	4799.5
2023 / 11	Manhattan	9	2096.1
2023 / 11	Manhattan	10	2549.1
2023 / 11	Manhattan	11	2328.6
2023 / 11	Manhattan	12	4553.0
2023 / 12	Manhattan	1	1345.5
2023 / 12	Manhattan	2	1825.8
2023 / 12	Manhattan	3	2452.2
2023 / 12	Manhattan	4	2316.5
2023 / 12	Manhattan	5	1238.4
2023 / 12	Manhattan	6	2695.6
2023 / 12	Manhattan	7	4388.6
2023 / 12	Manhattan	8	4785.7
2023 / 12	Manhattan	9	2097.2
2023 / 12	Manhattan	10	2535.7
2023 / 12	Manhattan	11	2284.8
2023 / 12	Manhattan	12	4568.9
Total			386856.0

Figure 18: Monthly Tonnage of Refuse Collected by Manhattan District in 2023 (4 of 4)

B.2 Current DSNY Schedule for Manhattan (output from Appendix A.2)

The Code column indicates the DSNY section, while Days shows the corresponding DSNY collection schedule. Note that there are two schedules, A (Mon, Wed, Fri) and B (Tue, Thu, Sat).

Code	Days
MN124	Tue, Thu, Sat
MN121	Tue, Thu, Sat
MN031	Tue, Thu, Sat
MN061	Mon, Wed, Fri
MN043	Mon, Wed, Fri
MN052	Tue, Thu, Sat
MN061	Tue, Thu, Sat
MN083	Mon, Wed, Fri
MN093	Mon, Wed, Fri
MN041	Mon, Wed, Fri
MN084	Mon, Wed, Fri
MN063	Tue, Thu, Sat
MN033	Tue, Thu, Sat
MN051	Tue, Thu, Sat
MN022	Tue, Thu, Sat
MN032	Mon, Wed, Fri
MN011	Mon, Wed, Fri
MN075	Mon, Wed, Fri
MN084	Tue, Thu, Sat
MN082	Mon, Wed, Fri
MN011	Tue, Thu, Sat
MN013	Tue, Thu, Sat
MN121	Mon, Wed, Fri
MN021	Tue, Thu, Sat
MN023	Mon, Wed, Fri
MN071	Mon, Wed, Fri
MN122	Mon, Wed, Fri
MN072	Mon, Wed, Fri
MN034	Mon, Wed, Fri
MN072	Tue, Thu, Sat
MN103	Tue, Thu, Sat
MN123	Mon, Wed, Fri
MN052	Mon, Wed, Fri
MN062	Mon, Wed, Fri
MN033	Mon, Wed, Fri
MN101	Tue, Thu, Sat
MN093	Tue, Thu, Sat
MN101	Mon, Wed, Fri
MN062	Tue, Thu, Sat
MN073	Tue, Thu, Sat

Table 1: Trash Pickup Schedule by Code and Days (1 of 2)

Code	Days
MN043	Tue, Thu, Sat
MN092	Tue, Thu, Sat
MN104	Mon, Wed, Fri
MN041	Tue, Thu, Sat
MN111	Tue, Thu, Sat
MN112	Tue, Thu, Sat
MN034	Tue, Thu, Sat
MN112	Mon, Wed, Fri
MN091	Tue, Thu, Sat
MN085	Tue, Thu, Sat
MN081	Tue, Thu, Sat
MN085	Mon, Wed, Fri
MN113	Tue, Thu, Sat
MN081	Mon, Wed, Fri
MN042	Mon, Wed, Fri
MN082	Tue, Thu, Sat
MN031	Mon, Wed, Fri
MN083	Tue, Thu, Sat
MN032	Tue, Thu, Sat
MN023	Tue, Thu, Sat
MN071	Tue, Thu, Sat
MN103	Mon, Wed, Fri
MN013	Mon, Wed, Fri
MN091	Mon, Wed, Fri
MN111	Mon, Wed, Fri
MN074	Tue, Thu, Sat
MN075	Tue, Thu, Sat
MN074	Mon, Wed, Fri
MN124	Mon, Wed, Fri
MN122	Tue, Thu, Sat
MN123	Tue, Thu, Sat
MN104	Tue, Thu, Sat
MN021	Mon, Wed, Fri
MN113	Mon, Wed, Fri
MN022	Mon, Wed, Fri
MN042	Tue, Thu, Sat
MN092	Mon, Wed, Fri
MN051	Mon, Wed, Fri
MN063	Mon, Wed, Fri
MN073	Mon, Wed, Fri

Table 2: Trash Pickup Schedule by Code and Days (2 of 2)

B.3 Population, Area, and Population Density for DSNY Manhattan Sections (.csv output from Appendix A.3)

The SECTION column represents the 80 sections designated by the DSNY. The population column provides the population estimation, AREA indicates the area of the section in square miles (mi^2), and population_density estimates population density estimation people per square mile (people/mi^2).

SECTION	population	AREA	population_density
MN011A	37262.861770370800	0.4464206763347780	83470.28653849090
MN011B	18925.276847659600	0.39253268503221000	48213.250945221600
MN013A	2558.276749021240	0.09575040741344250	26718.181343864200
MN013B	18281.22198813680	0.33964645773151900	53824.26806461070
MN021A	27150.711135239200	0.35458465871734600	76570.46199757360
MN021B	16549.05626795810	0.2771125957153080	59719.610453794500
MN022A	20493.2317051634	0.24462222256008500	83775.0204813455
MN022B	12631.70289738040	0.2067515269800890	61096.05613020180
MN023A	7364.7271927098900	0.15473515538424300	47595.694555781900
MN023B	9715.20177106199	0.21689179261231700	44792.85109892290
MN031A	21100.939183622500	0.2982801735215480	70742.01055504650
MN031B	25051.791374366100	0.30403939362077400	82396.53117323690
MN032A	13740.171669645200	0.21234080183826400	64708.10861923230
MN032B	16517.03339374590	0.17186263509053600	96106.01737279760
MN033A	26185.700871048100	0.26863216065292200	97477.90736374450
MN033B	7568.805293942220	0.13538748147388500	55904.76469127750
MN034A	25393.753254984200	0.2741596118274800	92623.97581363520
MN034B	11380.071042558500	0.14020473460059000	81167.52315802790
MN041A	17214.90670028890	0.19915859219234000	86438.18230881740
MN041B	9013.976262453380	0.174339758681552	51703.50315167210
MN042A	20882.842659515700	0.3594009886487920	58104.577669714000
MN042B	19618.35901629160	0.44365202099802300	44220.15022530250
MN043A	13684.483958050800	0.19161121774411600	71417.96873461490
MN043B	50378.0613647638	0.531301070306461	94820.17669511770
MN051A	18932.648279314000	0.4202595952853920	45049.88938195960
MN051B	33659.11969499500	0.366048651459326	91952.58488402090
MN052A	14422.618550261800	0.3079212529587250	46838.65894828320
MN052B	27899.210818540700	0.5949440209083260	46893.8418373308
MN061A	34353.76322187520	0.29472739929991400	116561.14532777800
MN061B	23501.16614361980	0.2514624532838260	93457.95301334310
MN062A	18450.068145544600	0.20423098114524200	90339.2229821561
MN062B	38204.7979551308	0.34832883720405000	109680.2615074630
MN063A	11551.799069828000	0.21755057280979600	53099.37326585510
MN063B	20723.613906892500	0.1807784233072230	114635.43894103900
MN071A	21650.396469064600	0.22972694398492800	94244.04509766590
MN071B	34249.75470153310	0.2639700995750120	129748.61454640000
MN072A	10906.497248978600	0.17478664868374000	62398.91508368510
MN072B	15510.351351606400	0.2019025788250320	76820.9670320638
MN073A	21227.35311710930	0.1863375075077650	113918.842218198
MN073B	31250.391355279600	0.221109302296488	141334.5844372280

Figure 19: Population, Area, and Population Density per Section of DSNY Manhattan District (1 of 2)

MN074A	17001.237677813500	0.16779280601312500	101322.80448593000
MN074B	17546.081798320000	0.20273739792094500	86545.85674993210
MN075A	17832.92336195990	0.1877729371981360	94970.67909814260
MN075B	21330.46125522260	0.22773674402917900	93662.80064357790
MN081A	9421.897153223710	0.21115376287531800	44621.024152845100
MN081B	34493.55264700420	0.27073063009375	127409.12483770200
MN082A	18419.508429285600	0.19888539905968100	92613.67861276900
MN082B	19689.883223379400	0.16812025253833500	117117.85419124100
MN083A	17311.997409336400	0.17187334809707100	100725.31664164000
MN083B	18856.37683799410	0.15279561491473000	123409.14919919100
MN084A	15616.395937627900	0.10623564101517400	146997.7098871880
MN084B	32726.769788575500	0.17685615210512400	185047.39246572900
MN085A	19535.571866697000	0.18453982803130100	105861.00613133500
MN085B	30495.49410305340	0.24612240760442600	123903.76967247300
MN091A	16755.25574833570	0.31368930965892600	53413.53763873450
MN091B	14559.386468742700	0.1710877903041300	85098.92168729040
MN092A	19388.03158676470	0.2852493413478130	67968.71640493730
MN092B	27954.88528109240	0.35523787224637400	78693.426194447
MN093A	19992.5899096369	0.3044165432558410	65675.1098209355
MN093B	15719.98852038550	0.18138490343443200	86666.46574624090
MN101A	26568.137507222100	0.2803724753434220	94760.14888651040
MN101B	31738.74047223520	0.31359182003232500	101210.3583217300
MN103A	6612.961901426470	0.1458144152783200	45351.907688990300
MN103B	14818.024374613100	0.21690467685745000	68315.83619725960
MN104A	34137.829600471500	0.34084091739925100	100157.66258627800
MN104B	19607.36396475890	0.21563394461439200	90928.93050684470
MN111A	35990.692268422700	0.29777653140958600	120864.77096785800
MN111B	23798.90000058340	0.3140230012486800	75787.12357359040
MN112A	22136.853762359700	0.20957520616300200	105627.25509210400
MN112B	22278.583610238400	0.2764431539384960	80590.10792213380
MN113A	10549.544305717300	0.19595886201158400	53835.50505153340
MN113B	11845.211949705800	0.35393079003182700	33467.59390060600
MN121A	20804.572991920700	0.2880174096790980	72233.73411732510
MN121B	26014.862537468300	0.3275235402615600	79428.98552175170
MN122A	23160.97927841640	0.29081419472973900	79641.84588699510
MN122B	23307.49646900830	0.4065601189735730	57328.53612856030
MN123A	20610.27662840940	0.34769332775569700	59277.17037725540
MN123B	28326.804985223800	0.36490928277130500	77626.97832758820
MN124A	22537.103810241	0.42710705276145200	52766.87346773560
MN124B	17978.05617755420	0.548974572143818	32748.431511768500
TOTAL	1672628	21.3443880633618	

Figure 20: Population, Area, and Population Density per Section of DSNY Manhattan District (2 of 2)

Note that the population is represented as a non-integer because the boundaries of census tracts do not align perfectly with DSNY Manhattan section boundaries. To address this, we approximate the population distribution across overlapping sections by calculating the intersection area between each census tract and DSNY section. We then normalize the population data based on these intersection areas. Specifically, the population for each subsection is estimated by multiplying the proportion of the intersection area by the total population of the respective census tract.

B.4 Road Length per Section of Each DSNY Manhattan District (.csv output from Appendix A.4)

An extension of the .csv output from Appendix A.3, with an extra column road_length represents the road length of each DSNY section.

SECTION	population	AREA	population_density	road_length
MN043B	50378.0613647638	0.531301070306461	94820.17669511770	15.934052717719500
MN062B	38204.7979551308	0.3483288372040500	109680.2615074630	10.446600553448500
MN011A	37262.861770370800	0.4464206763347780	83470.28653849090	13.38843640366720
MN111A	35990.692268422700	0.2977765314095860	120864.77096785800	8.930505159425250
MN081B	34493.55264700420	0.27073063009375	127409.12483770200	8.119381596063730
MN061A	34353.76322187520	0.2947273992999140	116561.14532777800	8.839059772818420
MN071B	34249.75470153310	0.2639700995750120	129748.61454640000	7.916629040675160
MN104A	34137.82960047150	0.340840917399251	100157.66258627800	10.222033136622300
MN051B	33659.11969499500	0.366048651459326	91952.58488402080	10.978028909745500
MN084B	32726.769788575500	0.1768561521051240	185047.3924657290	5.304027054699120
MN101B	31738.74047223520	0.3135918200323250	101210.3583217300	9.404815596095980
MN073B	31250.3913552796	0.221109302296488	141334.5844372280	6.631206816764400
MN085B	30495.49410305340	0.2461224076044260	123903.76967247300	7.381365551397960
MN123B	28326.804985223800	0.3649092827713050	77626.97832758820	10.94385852734930
MN092B	27954.88528109240	0.355237872246374	78693.426194447	10.653806852749700
MN052B	27899.21081854070	0.5949440209083260	46893.8418373308	17.8427447695093
MN021A	27150.71113523920	0.3545846587173460	76570.46199757360	10.634216568842600
MN101A	26568.137507222100	0.2803724753434220	94760.14888651040	8.408546589493430
MN033A	26185.700871048100	0.2686321606529220	97477.90736374450	8.056447179843960
MN121B	26014.862537468300	0.3275235402615600	79428.98552175170	9.82263663389871
MN034A	25393.75325498420	0.2741596118274800	92623.97581363530	8.222218911414590
MN031B	25051.79137436610	0.3040393936207740	82396.53117323690	9.118332329770160
MN111B	23798.90000058340	0.3140230012486800	75787.12357359040	9.41774699152554
MN061B	23501.16614361980	0.2514624532838260	93457.95301334300	7.541516874491510
MN122B	23307.49646900830	0.4065601189735730	57328.53612856030	12.192993258813800
MN122A	23160.97927841640	0.2908141947297390	79641.84588699510	8.721700310545110
MN124A	22537.103810241	0.427107052761452	52766.87346773560	12.80920870512320
MN112B	22278.58361023840	0.2764431539384960	80590.10792213380	8.290703773225740
MN112A	22136.853762359700	0.2095752061630020	105627.25509210400	6.285292031130340
MN071A	21650.396469064600	0.2297269439849280	94244.04509766590	6.889655302265890
MN075B	21330.46125522260	0.2277367440291790	93662.80064357780	6.829967955889170

Figure 21: Road Length per Section of Each DSNY Manhattan District (1 of 3)

MN073A	21227.35311710930	0.1863375075077650	113918.842218198	5.588378856840180
MN031A	21100.939183622500	0.2982801735215480	70742.01055504650	8.945609702614380
MN042A	20882.842659515700	0.3594009886487920	58104.577669714000	10.778661327798800
MN121A	20804.572991920700	0.2880174096790980	72233.73411732510	8.63782297069459
MN063B	20723.613906892500	0.1807784233072230	114635.43894103900	5.421658430957030
MN123A	20610.27662840940	0.3476933277556970	59277.17037725540	10.427541226037800
MN022A	20493.2317051634	0.2446222225600850	83775.0204813455	7.3363740599088300
MN093A	19992.5899096369	0.3044165432558410	65675.1098209355	9.129643284148930
MN082B	19689.883223379400	0.1681202525383350	117117.85419124100	5.042031941168440
MN042B	19618.35901629160	0.4436520209980230	44220.15022530250	13.305402691599600
MN104B	19607.36396475890	0.2156339446143920	90928.93050684470	6.466997401743710
MN085A	19535.571866697000	0.1845398280313010	105861.00613133500	5.534465320526350
MN092A	19388.03158676470	0.2852493413478130	67968.71640493730	8.554806863289620
MN051A	18932.648279314000	0.4202595952853920	45049.88938195960	12.603849155700600
MN011B	18925.276847659600	0.3925326850322100	48213.250945221600	11.772301706683000
MN083B	18856.37683799410	0.1527956149147300	123409.14919919100	4.582436436055640
MN062A	18450.068145544600	0.2042309811452420	90339.2229821561	6.125015367055060
MN082A	18419.508429285600	0.1988853990596810	92613.67861276900	5.964698003664270
MN013B	18281.22198813680	0.3396464577315190	53824.26806461070	10.186210541151500
MN124B	17978.05617755420	0.548974572143818	32748.431511768500	16.464092135522200
MN075A	17832.92336195990	0.1877729371981360	94970.67909814260	5.631428294602030
MN074B	17546.081798320000	0.2027373979209450	86545.85674993210	6.080221868294490
MN083A	17311.997409336400	0.1718733480970710	100725.31664164000	5.154589633651620
MN041A	17214.90670028890	0.1991585921923400	86438.18230881740	5.97289123725856
MN074A	17001.237677813500	0.1677928060131250	101322.80448593000	5.032211614264300
MN091A	16755.25574833570	0.3136893096589260	53413.53763873450	9.407739371214270
MN021B	16549.05626795810	0.2771125957153080	59719.610453794500	8.310780752474100
MN032B	16517.03339374590	0.1718626350905360	96106.01737279760	5.154268343858640
MN093B	15719.98852038550	0.1813849034344320	86666.46574624090	5.4398471507983500
MN084A	15616.395937627900	0.1062356410151740	146997.7098871880	3.186073582460700
MN072B	15510.351351606400	0.2019025788250320	76820.9670320638	6.055185119401140
MN103B	14818.024374613100	0.2169046768574500	68315.83619725960	6.505107459642360

Figure 22: Road Length per Section of Each DSNY Manhattan District (2 of 3)

MN091B	14559.386468742700	0.1710877903041300	85098.92168729040	5.131030262166980
MN052A	14422.618550261800	0.3079212529587250	46838.65894828320	9.234751728846480
MN032A	13740.171669645200	0.2123408018382640	64708.10861923230	6.368233982028520
MN043A	13684.483958050800	0.1916112177441160	71417.96873461490	5.746540738342750
MN022B	12631.70289738040	0.2067515269800890	61096.05613020180	6.200608119365360
MN113B	11845.211949705800	0.3539307900318270	33467.59390060600	10.614606636380900
MN063A	11551.799069828000	0.2175505728097960	53099.37326585510	6.524478284829880
MN034B	11380.071042558500	0.1402047346005900	81167.52315802790	4.204828029258540
MN072A	10906.497248978600	0.17478664868374	62398.91508368510	5.241961347591080
MN113A	10549.544305717300	0.1959588620115840	53835.50505153340	5.876929319935030
MN023B	9715.20177106199	0.2168917926123170	44792.85109892290	6.504721053040440
MN081A	9421.897153223710	0.2111537628753180	44621.02415284510	6.3326339381538800
MN041B	9013.976262453380	0.174339758681552	51703.50315167210	5.22855883581037
MN033B	7568.805293942220	0.1353874814738850	55904.76469127750	4.060355583096820
MN023A	7364.727192709890	0.154735155384243	47595.69455578190	4.640604472629590
MN103A	6612.961901426470	0.1458144152783200	45351.90768899030	4.3730658752635900
MN013A	2558.276749021240	0.0957504074134424	26718.1813438642	2.871614842764910

Figure 23: Road Length per Section of Each DSNY Manhattan District (3 of 3)

C Appendix C: Code for Optimizations and Sensitivity Testing

Math Modeling Code

This is the place to put our code

```
#Defining important constants
class Section:
  POPULATION_UNITS ='People'
  AREA_UNITS = 'Miles squared'
  TRASH_UNITS = 'Pounds'
  def __init__(self, name : str, total_population: int, area: float):
        self.name = name
        self.total_population = total_population
        self.area = area
        self.pop_density = total_population / area
        self.trash = (total_population / POPULATION_OF_MANHATTAN) *
TOTAL_AMOUNT_OF_TRASH_PER_DAY
        self.init_rat = (total_population / POPULATION_OF_MANHATTAN) *
INITIAL_RAT


MANHATTAN_AREA = 21.34 #miles squared
POPULATION_OF_MANHATTAN = 1672628 #people
TRUCK_SPEED_MORN = 6.07
TRUCK_SPEED_NIGHT = 4.16
TRUCK_CAPACITY = 23625 #pounds of trash
TOTAL_AMOUNT_OF_TRASH_PER_DAY = 2114280 #pounds of trash
PICKUP_RATE = 341
INITIAL_RAT = 230

#Generate all 80 districts of Manhattan
import csv
sections = set ()
with open('/content/gdf_final_population_data-2.csv', mode='r',
encoding='utf-8', ) as csv_file:
  csv_reader = csv.DictReader(csv_file)
  for row in csv_reader:
    sections.add(Section(row['SECTION'], float(row['population']),
float(row['AREA'])))


#Some constants for sections
MEAN_POPULATION_DENSITY = 81553.19397429019
POP_STD = 28956.875170245734
POP_LOWER_BOUND = MEAN_POPULATION_DENSITY - POP_STD
POP_UPPER_BOUND = MEAN_POPULATION_DENSITY + POP_STD
# Q1_POPULATION_DENSITY = 4088.6482984484473
# Q3_POPULATION_DENSITY = 6802.399865108731

def pickup_Time(section, threshold=60201.155):
```

```
  """
  Returns true for morning, false for night
  """
  return section.pop_density >= threshold


def miles(section):
  return section.area / MANHATTAN_AREA * 640

from math import ceil
import math

def trucks_needed(section, is_morning):
  speed = TRUCK_SPEED_MORN if is_morning else TRUCK_SPEED_NIGHT
  pr = PICKUP_RATE
  trucks_needed_per_miles = ceil((miles(section) / speed ))
  # print(trucks_needed_per_miles)
  trucks_needed_per_pickup = ceil((section.trash) / (pr * speed) *
ceil((pr * speed / TRUCK_CAPACITY)))
  # print(trucks_needed_per_pickup)
  return max(trucks_needed_per_miles, trucks_needed_per_pickup)

def time_sitting(is_morning, section):
  """
  Returns quantatative metric that when divided by pm yeilds how many
bags
  sat for pm hours
  """
  ws = 2114280 * section.total_population / POPULATION_OF_MANHATTAN
  bw = 22.5
  pm = 11 if is_morning else 22
  return (ws / bw) * pm

def rat_ppo(r, t, section):
  """
  returns the number of rats after a certain amount of time
  """

  lam = section.init_rat
  return lam * math.e ** (r * t)
```

Now we need to run these functions in a series to generate our simulation. The simulation will also track how much trash sits on the street for how long.

```
import csv
MON_WED_FRI = [
    'MN072A', 'MN072B','MN073A', 'MN073B','MN074A','MN074B',
'MN075A','MN075B',

'MN091A','MN091B','MN092A','MN092B','MN101A','MN101B','MN112A','MN112B
```

```
',

'MN111A','MN111B','MN085A','MN085B','MN084A','MN084B','MN083A','MN083B
',

'MN113A','MN113B','MN103A','MN103B','MN093A','MN093B','MN104A','MN104B
',

'MN121A','MN121B','MN122A','MN122B','MN123A','MN123B','MN124A','MN124B
'
    ]
TUES_THURS_SAT = [
    'MN071A', 'MN071B', 'MN043A', 'MN043B', 'MN042A', 'MN042B',
'MN082A', 'MN082B',
    'MN081A', 'MN081B', 'MN063A', 'MN063B', 'MN052A', 'MN052B',
'MN062A', 'MN062B',
    'MN051A', 'MN051B', 'MN041A', 'MN041B', 'MN061A', 'MN061B',
'MN023A', 'MN023B',
    'MN022A', 'MN022B', 'MN034A', 'MN034B', 'MN033A', 'MN033B',
'MN021A', 'MN021B',
    'MN032A', 'MN032B', 'MN013A', 'MN013B', 'MN031A', 'MN031B',
'MN011A', 'MN011B'
]

with open('naive.csv', mode='w', encoding='utf-8') as output:
  field_names = ['Day', '11 hour sit', '22 hour sit', 'Trucks used']
  days = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday',
'Saturday']
  writer = csv.DictWriter(output, fieldnames=field_names)
  writer.writeheader()
  th_total = 0
  tth_total = 0
  trucks_total = -1
  for i in range(6):
    twelve_hour_count = 0
    twenty_two_hour_count = 0
    trucks_used = 0
    if i % 2 == 0:
      for section in sections:
        if section.name in MON_WED_FRI:
          is_morning = pickup_Time(section)
          print(is_morning)
          trucks = trucks_needed(section, is_morning)
          time = time_sitting(pickup_Time(section), section)
          if is_morning:
            twelve_hour_count += time
          else:
            twenty_two_hour_count += time
          trucks_used += trucks
```

```
    else:
        for section in sections:
          if section.name in TUES_THURS_SAT:
            is_morning = pickup_Time(section)
            print(is_morning)
            trucks = trucks_needed(section, is_morning)
            time = time_sitting(pickup_Time(section), section)
            if is_morning:
              twelve_hour_count += time
            else:
              twenty_two_hour_count += time
            trucks_used += trucks

    #After iterating through a day, write to file
    th_total += twelve_hour_count
    tth_total += twenty_two_hour_count
    trucks_total = max(trucks_total, trucks_used)
    writer.writerow({
        'Day': days[i],
        '11 hour sit': f"{twelve_hour_count / 11:,}",
        '22 hour sit': f"{twenty_two_hour_count / 22:,}",
        'Trucks used': trucks_used
    })
  writer.writerow({
      'Day': 'Total',
      '11 hour sit': f"{th_total / 11:,}",
      '22 hour sit': f"{tth_total / 22:,}",
      'Trucks used': trucks_total
  })

mwf = {'morning' : 0,
       'night' : 0}
tts = {'morning' : 0,
       'night' : 0}
for section in sections:
  if section.name in MON_WED_FRI:
    pt = pickup_Time(section)
    if pt:
      mwf['morning'] += trucks_needed(section, pt)
    else:
      mwf['night'] += trucks_needed(section, pt)
  else:
    pt = pickup_Time(section)
    if pt:
      tts['morning'] += trucks_needed(section, pt)
    else:
      tts['night'] += trucks_needed(section, pt)
```

```
print(mwf)
print(tts)

#Sensitivity Testing over threshold for assiging morning or night
import numpy as np
from matplotlib import pyplot as plt

values = np.linspace(POP_LOWER_BOUND, POP_UPPER_BOUND, 100)
count = 1
for value in values:
  with open(f'sens{count}.csv', mode='w', encoding='utf-8') as output:
    field_names = ['Day', '11 hour sit', '22 hour sit', 'Trucks used']
    days = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday',
'Saturday']
    writer = csv.DictWriter(output, fieldnames=field_names)
    writer.writeheader()
    th_total = 0
    tth_total = 0
    trucks_total = -1
    for i in range(6):
      twelve_hour_count = 0
      twenty_two_hour_count = 0
      trucks_used = 0
      if i % 2 == 0:
        for section in sections:
          if section.name in MON_WED_FRI:
            is_morning = pickup_Time(section, threshold=value)
            print(is_morning)
            trucks = trucks_needed(section, is_morning)
            time = time_sitting(pickup_Time(section), section)
            if is_morning:
              twelve_hour_count += time
            else:
              twenty_two_hour_count += time
            trucks_used += trucks

      else:
          for section in sections:
            if section.name in TUES_THURS_SAT:
              is_morning = pickup_Time(section, threshold=value)
              print(is_morning)
              trucks = trucks_needed(section, is_morning)
              time = time_sitting(pickup_Time(section), section)
              if is_morning:
                twelve_hour_count += time
              else:
                twenty_two_hour_count += time
              trucks_used += trucks

      #After iterating through a day, write to file
```

```
      th_total += twelve_hour_count
      tth_total += twenty_two_hour_count
      trucks_total = max(trucks_total, trucks_used)
      writer.writerow({
          'Day': days[i],
          '11 hour sit': f"{twelve_hour_count / 11:,}",
          '22 hour sit': f"{twenty_two_hour_count / 22:,}",
          'Trucks used': trucks_used
      })
      count += 1
    writer.writerow({
        'Day': 'Total',
        '11 hour sit': f"{th_total / 11:,}",
        '22 hour sit': f"{tth_total / 22:,}",
        'Trucks used': trucks_total
    })

#Sensitivity Testing over threshold for assiging morning or night plot
import numpy as np
from matplotlib import pyplot as plt

values = np.linspace(POP_LOWER_BOUND, POP_UPPER_BOUND, 100)
num_trucks = []
num_trucks_m = []
num_trucks_n = []
trash_sit = []
num_rats = []

for v in values:
  num_truck_section = 0
  num_truck_morning = 0
  num_truck_night = 0
  trash_sit_section = 0
  num_rat_section = 0

  for s in sections:
    num_truck_section += trucks_needed(s, pickup_Time(s, v))

    trash_sit_section += time_sitting(pickup_Time(s, v), s)

    if (pickup_Time(s, v)):
      num_rat_section += rat_ppo(0.38, 0, s)
      num_truck_morning += trucks_needed(s, pickup_Time(s, v))
    else:
      num_rat_section += rat_ppo(0.38, 11, s)
      num_truck_night += trucks_needed(s, pickup_Time(s, v))

  num_trucks.append(num_truck_section)
  num_trucks_m.append(num_truck_morning)
  num_trucks_n.append(num_truck_night)
```

```
  trash_sit.append(trash_sit_section)
  num_rats.append(num_rat_section)


plt.plot(values, num_trucks)
plt.plot(values, num_trucks_m)
plt.plot(values, num_trucks_n)
plt.plot(values, np.ones(100) * 423 * 2)
plt.legend(["Total Trucks", "Trucks Morning", "Trucks Night", "Max
Trucks Allowed"])
plt.xlabel("Threshold Values")
plt.ylabel("Number of Trucks Needed")
plt.title("Threshold Values vs Number of Trucks Needed")
plt.show()

plt.plot(values, trash_sit)
plt.xlabel("Threshold Values")
plt.ylabel("Time Trash Sits (Hr)")
plt.title("Threshold Values vs Time Trash Sits")
plt.show()

plt.plot(values, num_rats)
plt.xlabel("Threshold Values")
plt.ylabel("Rat Population")
plt.title("Threshold Values vs Rat Population")

#Sensitivity Testing over truck speed (morning)
import numpy as np
from matplotlib import pyplot as plt

TRUCK_SPEED_MORN = 6.07
TRUCK_SPEED_NIGHT = 4.16
thres = 60201.1547

values = np.linspace(2, 20, 100)
num_trucks = []
num_trucks_m = []
num_trucks_n = []
trash_sit = []
num_rats = []

for v in values:
  TRUCK_SPEED_MORN = v
  TRUCK_SPEED_NIGHT = 4.16
  num_truck_section = 0
  num_truck_morning = 0
  num_truck_night = 0
  trash_sit_section = 0
  num_rat_section = 0
```

```
  for s in sections:
    num_truck_section += trucks_needed(s, pickup_Time(s, 60201.1547))

    trash_sit_section += time_sitting(pickup_Time(s, 60201.1547), s)

    if (pickup_Time(s, 60201.1547)):
      num_rat_section += rat_ppo(0.2, 0, s)
      num_truck_morning += trucks_needed(s, pickup_Time(s,
60201.1547))
    else:
      num_rat_section += rat_ppo(0.2, 11, s)
      num_truck_night += trucks_needed(s, pickup_Time(s, 60201.1547))

  num_trucks.append(num_truck_section)
  num_trucks_m.append(num_truck_morning)
  num_trucks_n.append(num_truck_night)
  trash_sit.append(trash_sit_section)
  num_rats.append(num_rat_section)

  if num_truck_morning <= 423 * 2 and num_truck_night <= 423 * 2:
    print(v)

plt.plot(values, num_trucks)
plt.plot(values, num_trucks_m)
plt.plot(values, num_trucks_n)
plt.plot(values, np.ones(100) * 423 * 2)
plt.legend(["Total Trucks", "Trucks Morning", "Trucks Night", "Max
Trucks Allowed"])
plt.xlabel("Morning Truck Speed")
plt.ylabel("Number of Trucks Needed")
plt.title("Morning Truck Speed vs Number of Trucks Needed")
plt.show()

#Sensitivity Testing over truck speed (night)
import numpy as np
from matplotlib import pyplot as plt

TRUCK_SPEED_MORN = 6.07
TRUCK_SPEED_NIGHT = 4.16
thres = 60201.1547

values = np.linspace(1, 20, 100)
num_trucks = []
num_trucks_m = []
num_trucks_n = []
trash_sit = []
num_rats = []

for v in values:
  TRUCK_SPEED_MORN = 6.07
```

```
  TRUCK_SPEED_NIGHT = v
  num_truck_section = 0
  num_truck_morning = 0
  num_truck_night = 0
  trash_sit_section = 0
  num_rat_section = 0

  for s in sections:
    num_truck_section += trucks_needed(s, pickup_Time(s, 60201.1547))

    trash_sit_section += time_sitting(pickup_Time(s, 60201.1547), s)

    if (pickup_Time(s, 60201.1547)):
      num_rat_section += rat_ppo(0.2, 0, s)
      num_truck_morning += trucks_needed(s, pickup_Time(s,
60201.1547))
    else:
      num_rat_section += rat_ppo(0.2, 11, s)
      num_truck_night += trucks_needed(s, pickup_Time(s, 60201.1547))

  num_trucks.append(num_truck_section)
  num_trucks_m.append(num_truck_morning)
  num_trucks_n.append(num_truck_night)
  trash_sit.append(trash_sit_section)
  num_rats.append(num_rat_section)

  # if num_truck_morning <= 423 * 2 and num_truck_night <= 423 * 2:
  #   print(v)
print(num_trucks_m)
plt.plot(values, num_trucks)
plt.plot(values, num_trucks_m)
plt.plot(values, num_trucks_n)
plt.plot(values, np.ones(100) * 423 * 2)
plt.legend(["Total Trucks", "Trucks Morning", "Trucks Night", "Max
Trucks Allowed"])
plt.xlabel("Night Truck Speed")
plt.ylabel("Number of Trucks Needed")
plt.title("Night Truck Speed vs Number of Trucks Needed")
plt.show()
```

We will now optimize by considering shared areas of neighboring sections using an adjacency matrix.

```
#Adjacency matrix optimization
from heapq import heappush, heappop
from itertools import combinations
import numpy as np
tts_neighbors = {
    '011A' : ['011B', '013A', '013B'],
```

```
    '011B' : ['011A', '013A', '013B', '031A'],
    '013A' : ['013B', '011A', '011B'],
    '013B' : ['013A', '011A', '011B', '031A', '021A', '021B'],
    '031A' : ['031B', '013B', '021A', '032B', '032A', '011B'],
    '031B' : ['031A', '032B'],
    '032B' : ['033A', '031B', '031A', '032A'],
    '032A' : ['032B', '031A', '021A', '022A', '033B', '033A'],
    '021A' : ['021B', '013B', '031A', '032A', '033B', '022A'],
    '021B' : ['021A', '013B', '022A', '022B', '023A'],
    '022A' : ['022B', '021B', '021A', '032A', '033B', '034B', '023A',
'023B'],
    '022B' : ['022A', '023B', '023A', '021B'],
    '033A' : ['033B', '032A', '032B', '034A'],
    '033B' : ['033A', '032A', '022A', '021A', '034B', '034A'],
    '034A' : ['034B', '033A', '033B', '061A'],
    '034B' : ['034A', '033B', '022A', '023A', '061A', '051B'],
    '023A' : ['023B', '034B', '022A', '022B', '041A', '051B', '061A'],
    '023B' : ['023A', '022B', '041A', '041B'],
    '041A' : ['041B','023B', '051B', '023A', '042A'],
    '041B' : ['041A', '042A', '042B', '023B'],
    '051B' : ['051A', '023A', '034B', '023B', '041A', '042B', '061A',
'061B', '062A'],
    '061A' : ['061B', '034A', '034B', '051B', '023A'],
    '061B' : ['061A', '051B', '062A'],
    '062A' : ['062B', '061B', '051B', '051A'],
    '042A' : ['042B', '051A', '043B', '043A', '052A', '041B', '041A',
'051B'],
    '042B' : ['042A', '041B', '043B'],
    '051A' : ['051B', '062A', '062B', '042A', '043A', '052A', '052B'],
    '062B' : ['062A', '051A','052B','063A'],
    '052A' : ['052B', '043A', '051A', '042A'],
    '052B' : ['052A', '051A', '062B', '063A', '063B', '081A'],
    '063A' : ['063B', '062B', '052B'],
    '063B' : ['063A', '081A', '081B', '052B'],
    '043A' : ['043B', '052A', '051A', '042A', '071A'],
    '043B' : ['043A', '071A', '071B', '042A', '042B'],
    '071A' : ['071B', '043A', '043B'],
    '071B' : ['071A', '043B'],
    '081A' : ['081B', '063B', '052B', '082A'],
    '081B' : ['081A', '063B', '082B'],
    '082A' : ['082B', '081A', '081B'],
    '082B' : ['082A', '081B']
}

mwf_neighbors = {
    '072A' : ['072B', '073A', '073B'],
    '072B' : ['072A', '073A','073B'],
    '073A' : ['073B','074A', '074B', '072A', '072B'],
    '073B' : ['073A', '072A', '072B', '074A','074B'],
```

```
    '074A' : ['074B','073A', '073B', '075A', '075B'],
    '074B' : ['074A', '073A', '073B', '075A', '075B'],
    '075A' : ['075B', '091A', '091B', '074A', '074B','101A'],
    '075B' : ['075A','074A', '074B', '091A', '091B'],
    '083A' : ['083B', '084A', '084B'],
    '083B' : ['083A', '084B'],
    '084A' : [ '084B', '083A', '085A','085B'],
    '084B' : ['084A', '085B', '085A', '083A', '083B'],
    '085A' : ['085B', '084A', '084B', '111A', '111B'],
    '085B' : ['085A', '084A', '084B', '111A', '111B'],
    '111A' : ['111B', '085A', '085B', '112A', '112B'],
    '111B' : ['111A', '085A', '085B', '112A', '112B'],
    '112A' : ['112B', '111A', '111B', '113B', '101B'],
    '112B' : ['112A', '111A', '111B', '113B'],
    '091A' : ['091B','075A', '075B', '092A', '092B'],
    '091B' : ['091A', '075A', '075B', '092B', '101A'],
    '101A' : ['101B', '091B', '092B', '103A', '103B', '075A'],
    '101B' : ['101A', '103A', '103B', '113A', '113B', '112A'],
    '113B' : ['113A', '112A', '112B', '101B'],
    '113A' : ['113B', '103B', '104B', '101B'],
    '103A' : ['103B', '101A', '101B', '092B','104A', '104B'],
    '103B' : ['103A', '113A', '101A', '101B', '104A', '104B'],
    '092A' : ['092B', '091A', '093A'],
    '092B' : ['092A', '091B', '101A', '103A', '093A', '093B', '104A'],
    '093A' : ['093B', '092A', '092B', '121A', '121B'],
    '093B' : ['093A', '092B', '104A', '121A', '121B'],
    '104A' : ['104B', '103A', '103B', '092B', '093B', '121A'],
    '104B' : ['104A', '103A', '103B', '113A'],
    '121A' : ['121B', '093B', '104A', '122A', '122B', '093A'],
    '121B' : ['121A', '093A', '093B', '122A', '122B'],
    '122A' : ['122B', '121A', '121B', '123A', '123B'],
    '122B' : ['122A', '121A', '121B', '123A', '123B'],
    '123A' : ['123B', '122A','122B', '124A'],
    '123B' : ['123A', '122A', '122B', '124A', '124B'],
    '124A' : ['124B', '123A', '123B'],
    '124B' : ['124A', '123B']
}



def neighbors(neighbors, section, n):
    """Return all subsets of size n of the neighbors for a given
section.

    Args:
        section (str): The section name.
        n (int): The subset size.

    Returns:
        List of tuples: Each tuple contains a subset of the section's
```

```
neighbors.
    """
    # Get the list of neighbors for the given section
    section_neighbors = neighbors.get(section, [])

    # Generate all possible subsets of size n
    subsets = [set(comb) for comb in combinations(section_neighbors,
n)]

    return subsets

def intersect_list(lst, st):
  for item in lst:
    if len(st.intersection(item)) > 0:
      return True
  return False

def num_trucks_merged(section_set):
  area = 0
  population = 0
  for section in section_set:
    secc = next((s for s in sections if s.name == ('MN' + section)),
None)
    area += secc.area
    population += secc.total_population

  pseudo_section = Section('pseudo', population, area)
  is_morning = pickup_Time(pseudo_section, 60201.1547)
  return trucks_needed(pseudo_section, is_morning)

def mergeSections(neighborhood):

  #Step2: Fill the min-queue
  minq = []
  for i in range(2, 4):
      for section in neighborhood:
          for combo in neighbors(neighborhood, section, i):
            trucks = num_trucks_merged(combo)
            heappush(minq, (trucks, combo))



  #Step3: Find optimized regions
  trucks = 0
  total_len = 0
  merged_regions = []

  while True:
    if len(minq) == 0 or total_len == len(neighborhood):
      break
```

```
    small_trucks, region = heappop(minq)

    if intersect_list(merged_regions, region):
      continue
    total_len += len(region)
    trucks += small_trucks
    merged_regions.append(region)

  print(f'Trucks needed: {trucks}')

  return trucks


mergeSections(tts_neighbors)

mergeSections(mwf_neighbors)
```

D Appendix D: Full Optimized Schedule

Section	Time	Trucks for one cleaning	Cycle
MN124B	Night	17	MonWedFri
MN123A	Night	19	MonWedFri
MN111A	Morning	22	MonWedFri
MN052A	Night	13	TueThuSAt
MN093B	Morning	10	MonWedFri
MN042B	Night	18	TueThuSAt
MN074B	Morning	11	MonWedFri
MN043A	Morning	9	TueThuSAt
MN083B	Morning	12	MonWedFri
MN072B	Morning	10	MonWedFri
MN062B	Morning	24	TueThuSAt
MN061A	Morning	21	TueThuSAt
MN042A	Night	19	TueThuSAt
MN121B	Morning	16	MonWedFri
MN063B	Morning	13	TueThuSAt
MN122A	Morning	15	MonWedFri
MN032A	Morning	9	TueThuSAt
MN032B	Morning	11	TueThuSAt
MN084B	Morning	20	MonWedFri
MN034A	Morning	16	TueThuSAt
MN085B	Morning	19	MonWedFri
MN104A	Morning	21	MonWedFri
MN013B	Night	17	TueThuSAt
MN112B	Morning	14	MonWedFri
MN091A	Night	15	MonWedFri
MN052B	Night	25	TueThuSAt
MN122B	Night	21	MonWedFri
MN083A	Morning	11	MonWedFri
MN082B	Morning	13	TueThuSAt
MN062A	Morning	12	TueThuSAt
MN022A	Morning	13	TueThuSAt
MN101B	Morning	20	MonWedFri
MN021B	Night	15	TueThuSAt
MN074A	Morning	11	MonWedFri
MN041B	Night	9	TueThuSAt
MN031B	Morning	16	TueThuSAt
MN124A	Night	21	MonWedFri
MN073B	Morning	20	MonWedFri
MN103B	Morning	10	MonWedFri
MN011A	Morning	23	TueThuSAt

Figure 24: Fully Optimized Schedule (1 of 2)

MN084A	Morning	10	MonWedFri
MN022B	Morning	8	TueThuSAt
MN033B	Night	7	TueThuSAt
MN104B	Morning	12	MonWedFri
MN075A	Morning	11	MonWedFri
MN041A	Morning	11	TueThuSAt
MN071B	Morning	21	TueThuSAt
MN051A	Night	17	TueThuSAt
MN051B	Morning	21	TueThuSAt
MN075B	Morning	14	MonWedFri
MN031A	Morning	13	TueThuSAt
MN082A	Morning	12	TueThuSAt
MN034B	Morning	7	TueThuSAt
MN023A	Night	7	TueThuSAt
MN021A	Morning	17	TueThuSAt
MN112A	Morning	14	MonWedFri
MN081A	Night	9	TueThuSAt
MN033A	Morning	16	TueThuSAt
MN123B	Morning	18	MonWedFri
MN091B	Morning	9	MonWedFri
MN072A	Morning	7	MonWedFri
MN103A	Night	6	MonWedFri
MN085A	Morning	12	MonWedFri
MN071A	Morning	14	TueThuSAt
MN011B	Night	17	TueThuSAt
MN043B	Morning	31	TueThuSAt
MN092A	Morning	12	MonWedFri
MN013A	Night	3	TueThuSAt
MN061B	Morning	15	TueThuSAt
MN121A	Morning	13	MonWedFri
MN093A	Morning	13	MonWedFri
MN081B	Morning	22	TueThuSAt
MN113A	Night	10	MonWedFri
MN092B	Morning	18	MonWedFri
MN023B	Night	9	TueThuSAt
MN073A	Morning	13	MonWedFri
MN113B	Night	11	MonWedFri
MN063A	Night	11	TueThuSAt
MN101A	Morning	17	MonWedFri
MN111B	Morning	15	MonWedFri

Figure 25: Fully Optimized Schedule (2 of 2)

This optimized schedule divides Manhattan into 80 sections, assigning each a specific pickup time (morning or night) and cycle (Mon-Wed-Fri or Tue-Thu-Sat). The "Trucks for one cleaning" column indicates the required trucks per collection.

E Appendix E: Executive Summary

Dear Head of Waste Management Division at DSNY,

First, we would like to express our gratefulness to New York's Strongest for your fight to keep our streets clean, and we are excited to support you on this journey. At Cornell University, we have developed a model aimed at optimizing Manhattan's trash collection schedule to reduce street trash and curb the rat population.

The model consists of four main components: determining the time and day for trash collection, assigning trucks to each DSNY sanitation section, calculating the time trash remains on sidewalks, and estimating the impact on rat populations. This approach aims to optimize the existing collection schedule.

- **Recommendations**: We recommend you consult Appendix D for our full schedule. However, in general, we recommend that you prioritize areas with high population density, as they typically generate more trash. It is imperative to clean these areas as early as possible to reduce garbage and rats on busy streets.
- **On efficiency and equity**: Not only will our model increase efficiency, it also remains equitable to the diverse people of Manhattan. We can assure you that under our schedule, no one will have to live in dirty streets for longer than a day.
- **Limitations**: As with all mathematical models, ours has inherent limitations. We recognize that the accuracy of our trash pickup rates and rat population dynamics may be constrained by resource limitations and a lack of extensive literature on these topics. However, we believe that our model offers a valuable foundation for advancing a cleaner New York City.
- **Future Considerations**: Our model can be refined by improving data accuracy for trash processing times (t_p) and rat attraction rates through partnerships with local agencies and integrating in vivo data. Additionally, using socioeconomic clusters instead of simple boundaries would better capture equity, and including environmental metrics like fuel consumption and emissions could enhance sustainability.

In the future, we would love to work with you on ways to improve our schedule. With your help, we will have the capabilities to make New York cleaner, fairer, and less rat-friendly for all residents.

We appreciate your time and consideration of our project. If you have any questions, we welcome the opportunity to discuss and further refine our findings.

Sincerely,
Matthew Cowan, Tuni Le, Mary Yuan