Optimizing Ambulance Deployment in NYC: A System-Status Management Approach for Enhanced Response Times

Julia VanPutte

September 22, 2023

Abstract

In the dynamic city of New York, ensuring fast and efficient ambulance response time is critical for addressing medical emergencies, particularly in cases like out-of-hospital cardiac arrest (OHCA). This report analyses the optimal allocation of ambulances during the evening rush hour (7pm to midnight on weekdays). The aim is to provide the Fire Department of New York (FDNY) with a strategic plan to meet their contractual requirements for ambulances, which requires addressing 90% of emergency calls within 9 minutes. This problem was solved through the Maximum Expected Covering Location Problem (MEXCLP). In analysis, various sections of emergency response are modeled, including call-taking and chute times, on-scene service durations, hospital transport times, fluctuating hospital offload delays, and travel velocities of ambulances.

This investigation revealed pivotal insights. For a 15-minute hospital offload delay, a deployment of 126 ambulances is optimal. As the delay extends to 30 minutes, the requirement grows to 139 ambulances. For prolonged offload delays of 45 and 60 minutes, the demands increase to 158 and 178 ambulances, respectively.

The specific offload delay of 30 minutes was further explored, resulting in a compliance table for ambulance scheduling, utilizing system-status management. This table maps out the strategic placement of ambulances in relation to the number of available units for the FDNY.

This analysis is crucial for aiding the FDNY in meeting their requirements and enhancing emergency healthcare delivery in New York, both by staffing the optimal number of ambulances, and developing a strategic system-status management plan for ambulance placement across the NYC.

1 Introduction

As the most densely populated city in the United States, New York City faces unique demands when it comes to its healthcare and emergency response systems. Ensuring rapid and efficient ambulance services across the metropolitan area is critical for the FDNY, and the department is currently struggling to meet their contractual requirements for response time performance. This is not only detrimental to the performance of the FDNY but also a delay in response time can mean the difference between life or death, especially in urgent medical emergencies like OHCA.

In order for the FDNY to meet their contractual requirements, they must reach 90% of calls in 9 minutes in the interval of 7pm-midnight on weekdays. The key to solving this problem is optimal system-status management, which can drastically improve response-time performance.

The of emergency response requires meticulous planning, coordination, and foresight. Many factors come into play when determining ambulance response times, including call-taking time, chute time, travel time, service time at the scene, hospital transport time, and hospital offload delays.

In this study, the Maximum Expected Covering Location Problem (MEXCLP) is employed to optimize both the number of ambulances and the allocation of ambulances across the city. The goal is to firstly minimize the number of ambulances staffed during the chosen time window while ensuring FDNY's contractual requirements are met. After determining, the optimal ambulance number, the

locations of these ambulances must also be optimized using system-status management. By dissecting various scenarios, especially different hospital offload delays, the aim is to streamline the FDNY's ambulance scheduling and deployment, ultimately saving more lived in NYC.

2 Analysis

The analysis is twofold. First, the optimal number of ambulances for offload delayd of 15,30,45,and 60 minutes were calculated using MEXCLP. Then, using MEXCLP again, the optimal locations of ambulances were calculated to determine the final compliance table.

2.1 Initial Analysis

First, the analysis began from the results of HW1. Using the datafiles nyc_nodes.csv, nyc_links.csv, and nyc_population,csv, data regarding NYC locations and populations was analyzed. The data on average travel time on road segments was given without units. Therefore, the length of each road segment was calculated using the haversine function. Then, the relationship between segment length and travel time was visualized using a scatter plot and fitted to a straight line of slope 0.137. This is equated to of 7.3 m/s or approximately 26 km/hr, suggesting the time unit was in seconds. Next, road segments with speeds below 2 m/s , above 30 m/s, and above 20 m/s in New York were determined to be outliers and adjusted to 2 m/s 30 m/s, and 20 m/s respectively. Considering the fact that ambulances travel faster than regular traffic speed when on an emergency call, travel times were adjusted for ambulances on each road segment to be 0.9 times the regular traffic speed. Using the 2022 FDNY report [Fir22], it was determined that the total number of emergency calls in NYC was 1.55M. Therefore, the average number of calls per hour across NYC in 2022 was approximately 177. These calls came from all 5 boroughs of NYC, but our study area cuts off parts of the Bronx, Queens and Brooklyn and all of Staten Island. Accordingly, the call rate per hour was divided by 2 and used in subsequent analysis to estimate the hourly call arrival rate of our study area. It was then assumed that the likelihood of a call originating from a particular node, given in nyc_nodes.csv, was proportional to its population. From this assumption, the probability of a call from each region was calculated. Then, the shortest travel times between all nodes were calculated using the Dijkstra algorithm, imported in python as networkx.single_source_dijkstra(). After this pre-analysis, the analysis of the case began.

2.2 Optimizing the Minimum Number of Ambulances

To find the minimum number of ambulances needed to satisfy 90% of demand in 9 mins across NYC, various metrics were defined and sourced:

- 70% of calls result in the patient being transported to a hospital
  - on average 35 minutes at the scene and 15 minutes transport
  - offload delay of 15, 30, 45, or 60 minutes
  - call-taking and chute time of 4 minutes
- 30% of calls are completed at the scene, averaging 45 minutes
  - call-taking and chute time of 4 minutes

Based on this information, $Mean\ Service\ Time = 0.7 \times (35 + 15 + offload) + 0.3 \times 45$ We define $1/\mu$ as the mean service time. So, for the four different offload delays:

- 15 minutes: $Mean\ Service\ Time = 0.7 \times (35 + 15 + 15) + 0.3 \times 45 = 77.5\ minutes$
- 30 minutes: $Mean\ Service\ Time = 0.7 \times (35 + 15 + 30) + 0.3 \times 45 = 92.5\ minutes$
- 45 minutes: $Mean\ Service\ Time = 0.7 \times (35 + 15 + 45) + 0.3 \times 45 = 107.5\ minutes$
- 60 minutes: $Mean\ Service\ Time = 0.7 \times (35 + 15 + 60) + 0.3 \times 45 = 122.5\ minutes$

Then the arrival rate was dermined: λ, which was 177/2 calls/hr = 88.5 calls/hr. Next, the minimum number of ambulances to satisfy demand were found. The approach was to solve MEXCLP with a chosen $a$number of ambulances, such that the probability that an ambulance is free is less than 0.75 $p \leq 0.75$ and the utilization of an ambulance is ≤ 0.75, because too high utilization affects efficiency. MEXCLP's objective will give E[# of calls per hour with response time under threshold]. MEXCLP was solved iteratively to determine the smallest a such that

$$\frac{\text{E[\# of calls per hour with response time under threshold]}}{\text{Total calls per hour}} \geq 0.9$$

.

2.2.1 MEXCLP

$i = 1, ..., m$ demand locations ; $j = 1, ..., n$ ambulance stations
x_j = # of ambulances stationed at location j ; $0 \leq x_j \leq c_j$; integer
y_i = # of ambulances that can reach location i on time ; integer
Objective: Maximize the number of calls covered within 9 minutes. This involves maximizing

$$\sum_{i=1}^{m} \sum_{k=1}^{a} b_{ik} y_i(k)$$

Constraints are:

- $y_i \geq y_i(1) + y_i(2) + ... + y_i(10)$
- $b_{ij} = \lambda_i(1 - p^j) - b_{ij-1}$ and $b_{i1} = \lambda_i(1 - p)$
- $\sum_j x_j \alpha_{ij} \geq y_i$
- $\sum_j x_j = a$
- $0 \leq x_j$
- y_{ij} binary
- α_{ij} is a binary variable determined by the geographical distance and travel speeds (10% faster for emergencies), where $\alpha_{ij} = 1$ is the ambulance can reach its destination within 9 mins and 0 is not. Here, $\alpha_{ij} = 1$ if $travel\ time_{ij} + 4 \leq 9\ min$ or $travel\ time_{ij} \leq 5\ min$
- p is calculated as $\frac{\sum_i \lambda_i}{\mu a}$. Given λ and μ from above, we can adjust a.

We then solve MEXCLP for each offload delay to determine the number of ambulances needed.

2.3 Solving MEXCLP to determine optimal number of ambulances

The model assumes capacities are unbounded at each node. For our calculations, Python programming language was used, leveraging Pandas for data manipulation and Gurobi for optimization. Overall, the strategy for finding a was,

1. Calculate the Mean Service Time for each offload delay.
2. Calculate the adjusted call rate (λ).
3. Solve the MEXCLP problem iteratively for different values of a until we achieve that $E[\#\ of\ calls\ per\ hour\ with\ response\ time\ under\ threshold]/\ total\ calls\ per\ hour \geq 0.9$. This will give us the number of ambulances needed for each offload delay scenario.

Using the MEXCLP problem, with defined variables and constraints as mentioned in the previous section, the minimum a to satisfy the 0.9 threshold was found through iterative optimization. For each offload delay, of 15, 30, 45, and 60 minutes, the minimum value of a was calculated to ensure that $p \leq 0.75$, as if utilization is too high this will affect efficiency. From studies in class, utilization should most of the time be between 0.7 and 0.6, so a maximum of 0.75 is reasonable. $a_{\min} = \lceil \frac{\sum \lambda_i}{0.75\mu} \rceil$,

where $\sum \lambda_i$ represents the total demand and μ denotes the service rate of an ambulance. Then, for each offload delay, the MEXCLP problem was solved iteratively, increasing a by one each time, until $E[\#\ of\ calls\ per\ hour\ with\ response\ time\ under\ threshold]/\ total\ calls\ per\ hour \geq 0.9 * 88.5 = 79.65$. After solving this problem many times, this summary provides the relationship between the offload delay time and the respective 'p' value, optimal value for the response time, and the required number of ambulances:

Offload Delay	Probability Busy	Probability Free	E[Frac Calls/hr with time¡=9 min]	Number of Ambulances
15 min	0.69	0.31	0.90	126
30 min	0.74	0.26	0.90	139
45 min	0.75	0.25	0.93	158
60 min	0.75	0.25	0.95	178

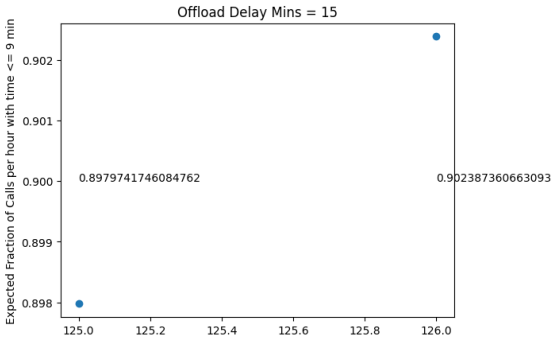


Figure 1: 15min Offload Delay Optimization

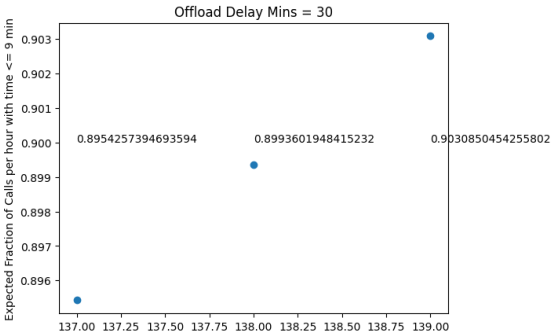


Figure 2: 30min Offload Delay Optimization

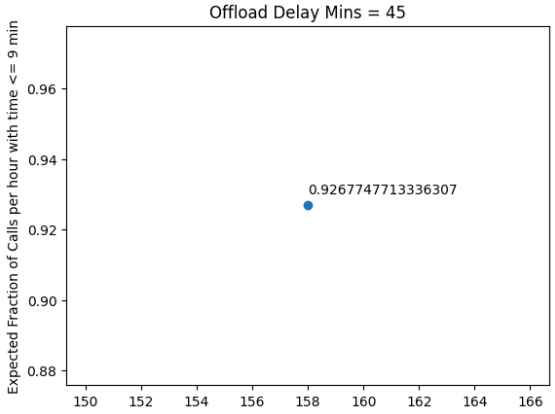


Figure 3: 45min Offload Delay Optimization

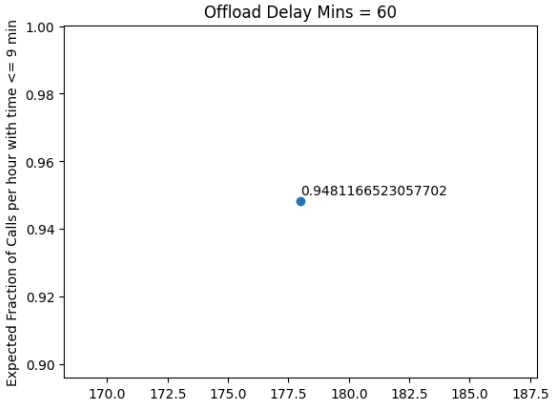


Figure 4: 60min Offload Delay Optimization

Now that the number of ambulances for each offload delay was optimized, the compliance table will be created. This model indicates the details of emergency service resource allocation. Using a mixed-integer linear programming model (MEXCLP), the model provided valuable insights into allocating resources under constraints like offload delays and compliance requirements. Further, this is optimal for FDNY because this number of ambulances is the absolute minimum number of ambulances needed to satisfy 90% of demand in 9 minutes while also keeping utilization below 0.75 for efficient work balancing. Therefore, this will reduce costs by optimally determining the number of ambulances to staff in the set time-window and also allow FDNY to meet their contractual performance requirements. However, this report will use the optimal ambulance numbers as recommendations of minimum number of ambulances to staff at each time, and build a system-status management off of this minimum number. For increasing the number of ambulances, a similar approach can be implemented to determine the

number of ambulances to meet a certain amount of demand or have a specific utilization rate. This investigation serves as a robust and cost-saving tool for the FDNY and others in emergency services, ensuring timely and optimally efficient responses to crises.

2.4 System-Status Management for Ambulance Locations

The system status management table will only be constructed for an offload delay of 30 minutes assuming the ambulances are minimally allocated to align with the MEXCLP optimization model discussed in the previous section. To determine the compliance table which guides the system-status management plan, the primary objective was to determine the optimal placement of ambulances based on the probability of them being busy and the distance and time constraints. Specifically, for a 30-minute offload delay, the aim was to find out the best positioning of ambulances depending on the number of free ambulances. Similar to the previous problem, a distance matrix was provided which represented the time taken to travel between various nodes in the NYC region. A binary matrix, α_{ij}, was derived from this to indicate if a location is reachable within 9 minutes. Here, $\alpha_{ij} = 1$ if $travel\ time_{ij} + 4 \leq 9\ min$ or $travel\ time_{ij} \leq 5\ min$. Using the same parameters as above, with 30 minute offload delay, 139 of ambulances, call rate $\lambda = 88.5$calls/hr, and mean service time $\frac{1}{\mu} = 0.7 \times (35+15+30) + 0.3 \times 45 = 92.5\ minutes$, probability busy $p = \frac{\sum_i \lambda_i}{\mu a}$. The Maximum Expected Coverage Location Problem (MEXCLP) model was implemented using the Gurobi Optimizer. The problem was first solved with a = the most likely number of ambulances $= \lceil a(1-p) \rceil = \lceil 36.4875 \rceil = 37$. The MEXCLP then gave the locations corresponding to node "name" from the nyc_nodes.csv table that would house the 37 ambulances. The optimization using MEXCLP was then iteratively executed both downwards (from 37 ambulances to 1) and upwards (from 37 to 139) to determine the optimal placement strategy across different numbers of ambulances. There was a limit placed on each node that each location could have a maximum of one ambulance. The placement was optimized from there to maximize the number of calls that could be responded to within 9 minutes. When solving iteratively with a decreasing from 37 ambulances to 1, constraints were added each time such that $x_i$for the next value of a $\leq x_i$for the previous value of a$\forall i \in \{1, .., m\}$. Similarly, when solving iteratively with a increasing from from 37 to 139, constraints were added each time to reflect that $x_i$for the next value of a $\geq x_i$for the previous value of a $\forall i \in \{1, .., m\}$.

It was found that for a 30-minute offload delay, the utilization was found to be approximately 0.7375, or 74%. Starting with 37 free ambulances, the optimization iteratively provided the best allocation of ambulances across nodes for each additional free ambulance. At each step, the previous placement was used as a basis to ensure the newly added ambulance added the maximum additional coverage. The optimal placement of ambulances at node names is described in the compliance table in the appendix, which is a list of 139 locations (nodes corresponding to ambulances), starting from the location where FDNY would place a single ambulance if only 1 ambulance is free, up to the location where FDNY would place the last ambulance if 139 ambulances are free.

For a graphical depiction of this table, we have below a map of NYC visualization, where the nodes for the first 30 optimal placements were colored red, the next 30 in orange, followed by yellow, green, and blue. If no ambulances are placed at the node, the node is shown small and in black. The visualization also has a size representation of the nodes indicating the number of ambulances placed at the node.

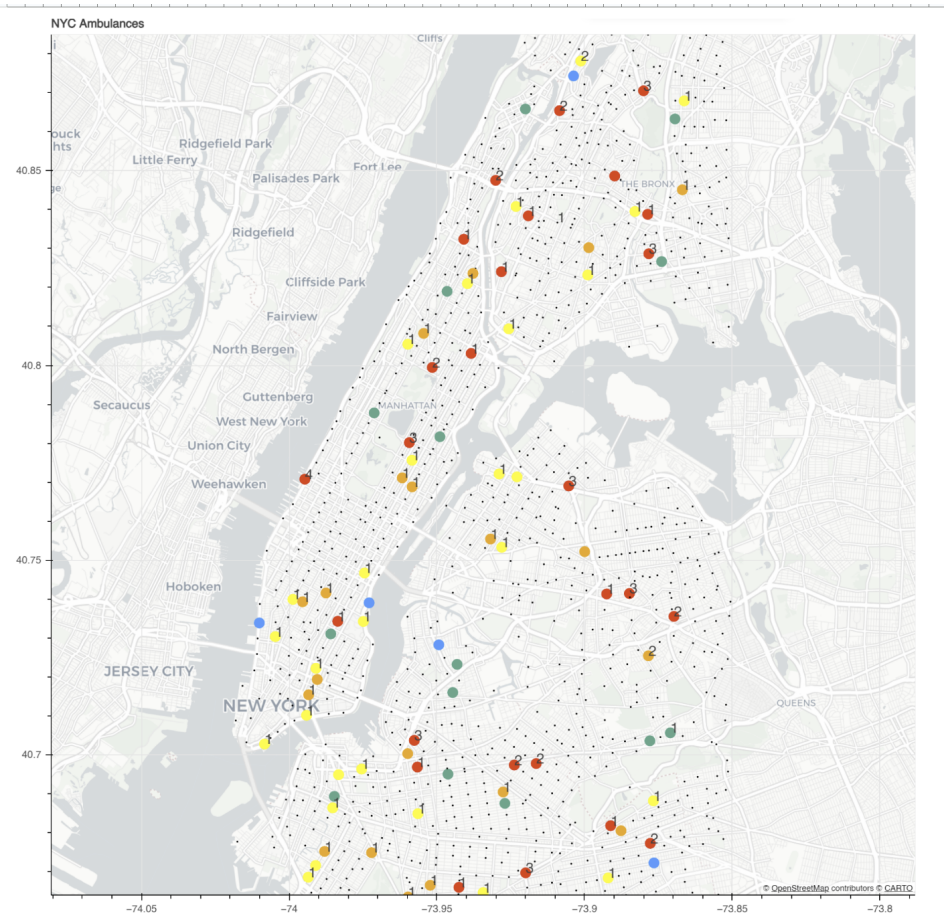


Figure 5: NYC AMBULANCES: First 30 Locations: Red, Next 30 Locations (30-60) : Orange, Third 30 Locations (60-90): Yellow , Next 30 Locations (90-120): Green , Last 30 locations (120-139): Blue , Locations with no Ambulances: Black

This visualization method gives a clear understanding of the priority and sequence of ambulance placements across NYC. The compliance table output ensures that the maximum number of calls can be responded to within the desired response time. By prioritizing locations with the highest demand and shortest response time, the model ensures that the greatest number of emergencies can be addressed efficiently. For practical application, the FDNY can refer to the compliance table when allocating ambulances. If any changes in the demand or other parameters occur, this model can be rerun to get updated placements, as code is referenced in appendix.

3 Conclusion

Through this data-driven MEXCLP model and optimization study, the FDNY can ensure they are maximizing their response efficiency. By taking into account both the likelihood of an ambulance being available, the travel time between nodes, and the location's demand, this optimization approach can save critical minutes during emergencies, potentially saving lives. The compliance table crafted underlines the shifts in optimal ambulance numbers for varied availability, and can be used for optimal system status management. This optimized approach is pivotal in ensuring timely response-times to emergencies and achieving maximum benefit from the available emergency resources.

References

[Fir22] Fire Department, City of New York (FDNY). Mayor's management report 2022: Fdny, 2022. Available: https://www.nyc.gov/assets/operations/downloads/pdf/pmmr2022/fdny.pdf.

[Hen23] Shane Henderson. Course notes in orie 4130: Service system modeling, 2023. Unpublished course notes, Cornell University.

[Ope23] OpenAI. Chatgpt, 2023. Source for code and writing help.

[Tea22a] Bokeh Development Team. Bokeh: Python visualization library, 2022. Available: http://bokeh.org/.

[Tea22b] Matplotlib Development Team. Choosing colormaps in matplotlib, 2022. Available: https://matplotlib.org/stable/users/explain/colors/colormaps.html.

[Wik22] Wikipedia. New york city, 2022. Available: https://en.wikipedia.org/wiki/New_York_City.

[Tea22a] [Ope23] [Fir22] [Hen23] [Tea22b] [Wik22]

COMPLIANCE TABLE FOR FDNY

Number of Ambulances	Node Name to Place Ambulance
1	42864760
2	42864760
3	42736037
4	42755148
5	42446036
6	42750730
7	1260335197
8	42838614
9	1858067878
10	42755148
11	42437204
12	42494213
13	596399974
14	42471841
15	42877863
16	42749785
17	42467474
18	42524972
19	42446036
20	42743114
21	42865969
22	42442877
23	61273001
24	42490957
25	42729818
26	42447207
27	1260335197
28	42432990
29	42530139
30	42486746
31	42750730
32	486868873
33	42809632
34	42481959
35	42438798
36	42877863
37	42436575
38	42466196
39	42439225
40	42755148
41	42506427
42	61273001
43	42437204
44	42485145

45	42720069
46	42824219
47	42485012
48	42429637
49	42485886
50	42743114
51	42436489
52	42430259
53	42809632
54	42815810
55	42518540
56	42756707
57	42445219
58	42530139
59	61273001
60	42490957
61	42451018
62	1858067878
63	42479643
64	42524972
65	42451409
66	42430770
67	42719583
68	42466188
69	42446036
70	42838614
71	42742419
72	42818487
73	42495185
74	42481908
75	42478553
76	42442918
77	42435362
78	42761465
79	42475209
79	42452556
79	42732863
79	42751992
79	42877863
79	42471715
79	42815829
79	42466179
79	42439910
79	42750730
79	42490957
79	42432054
80	61273001

81	1260335197
82	42486746
83	42447207
84	42864760
85	42466468
86	42719583
87	42439830
88	42759029
89	42438498
90	42833059
91	42812236
92	42467885
93	486868873
94	1858067878
95	42466179
96	42485886
97	42755148
98	42530139
99	42514354
100	42442918
101	42438043
102	42430770
103	42737338
104	42824219
105	42435981
106	42451409
107	61273001
108	42475534
109	42502404
110	42485022
111	42443048
112	42909370
113	42478553
114	42833059
115	42509198
116	42734582
117	42427870
118	42466188
119	42442870
120	42749785
121	42809632
122	42452613
123	42435362
124	42495185
125	61273001
126	42451409
127	42737338

128	42458034
129	42815810
130	42475534
131	42524972
132	42481908
133	42436489
134	42719573
135	42515129
136	42466179
137	42512083
138	42439830

APPENDIX

In [1]:
```
import pandas as pd
import numpy as np


from bokeh.plotting import figure, show
from bokeh.tile_providers import get_provider, Vendors
from bokeh.models import (GraphRenderer, Circle, MultiLine, StaticLayoutProvider,ColumnDataSource)
from bokeh.io import save, output_notebook
output_notebook()
from bokeh.models import ColumnDataSource, Label, LabelSet, Range1d
from bokeh.plotting import figure, output_file, show
import matplotlib
import matplotlib.pyplot as plt
!pip install selenium chromedriver-binary
from bokeh.io import export_png
from bokeh.models import MultiLine
```

```
BokehDeprecationWarning: 'tile_providers module' was deprecated in Bokeh 3.0.0 and will be removed, use 'add_ti
le directly' instead.
Collecting selenium
  Downloading selenium-4.12.0-py3-none-any.whl (9.4 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 9.4/9.4 MB 48.9 MB/s eta 0:00:00
Collecting chromedriver-binary
  Downloading chromedriver-binary-119.0.6022.0.0.tar.gz (5.6 kB)
  Preparing metadata (setup.py) ... done
Requirement already satisfied: urllib3[socks]<3,>=1.26 in /usr/local/lib/python3.10/dist-packages (from seleniu
m) (2.0.4)
Collecting trio~=0.17 (from selenium)
  Downloading trio-0.22.2-py3-none-any.whl (400 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 400.2/400.2 kB 35.9 MB/s eta 0:00:00
Collecting trio-websocket~=0.9 (from selenium)
  Downloading trio_websocket-0.10.4-py3-none-any.whl (17 kB)
Requirement already satisfied: certifi>=2021.10.8 in /usr/local/lib/python3.10/dist-packages (from selenium) (2
023.7.22)
Requirement already satisfied: attrs>=20.1.0 in /usr/local/lib/python3.10/dist-packages (from trio~=0.17->selen
ium) (23.1.0)
Requirement already satisfied: sortedcontainers in /usr/local/lib/python3.10/dist-packages (from trio~=0.17->se
lenium) (2.4.0)
Requirement already satisfied: idna in /usr/local/lib/python3.10/dist-packages (from trio~=0.17->selenium) (3.
4)
Collecting outcome (from trio~=0.17->selenium)
  Downloading outcome-1.2.0-py2.py3-none-any.whl (9.7 kB)
Requirement already satisfied: sniffio in /usr/local/lib/python3.10/dist-packages (from trio~=0.17->selenium)
(1.3.0)
Requirement already satisfied: exceptiongroup>=1.0.0rc9 in /usr/local/lib/python3.10/dist-packages (from trio~=
0.17->selenium) (1.1.3)
Collecting wsproto>=0.14 (from trio-websocket~=0.9->selenium)
  Downloading wsproto-1.2.0-py3-none-any.whl (24 kB)
Requirement already satisfied: pysocks!=1.5.7,<2.0,>=1.5.6 in /usr/local/lib/python3.10/dist-packages (from url
lib3[socks]<3,>=1.26->selenium) (1.7.1)
Collecting h11<1,>=0.9.0 (from wsproto>=0.14->trio-websocket~=0.9->selenium)
  Downloading h11-0.14.0-py3-none-any.whl (58 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 58.3/58.3 kB 6.5 MB/s eta 0:00:00
Building wheels for collected packages: chromedriver-binary
  Building wheel for chromedriver-binary (setup.py) ... done
  Created wheel for chromedriver-binary: filename=chromedriver_binary-119.0.6022.0.0-py3-none-any.whl size=7843
779 sha256=fae9c5fbd86f068d8e4b775c0783bfe34492a4ceb32a7e9baf5781abacc3fb08
  Stored in directory: /root/.cache/pip/wheels/5a/92/2b/9ab3aff359b926ae3538991d6459c3292c928184fc9dd81b84
Successfully built chromedriver-binary
Installing collected packages: chromedriver-binary, outcome, h11, wsproto, trio, trio-websocket, selenium
Successfully installed chromedriver-binary-119.0.6022.0.0 h11-0.14.0 outcome-1.2.0 selenium-4.12.0 trio-0.22.2
trio-websocket-0.10.4 wsproto-1.2.0
```

In [2]:
```
data = pd.read_csv('nyc_nodes.csv')
dfn = pd.DataFrame(data)
dfp = pd.read_csv('nyc_population.csv')
xs = pd.read_csv('xs_final.csv')
```

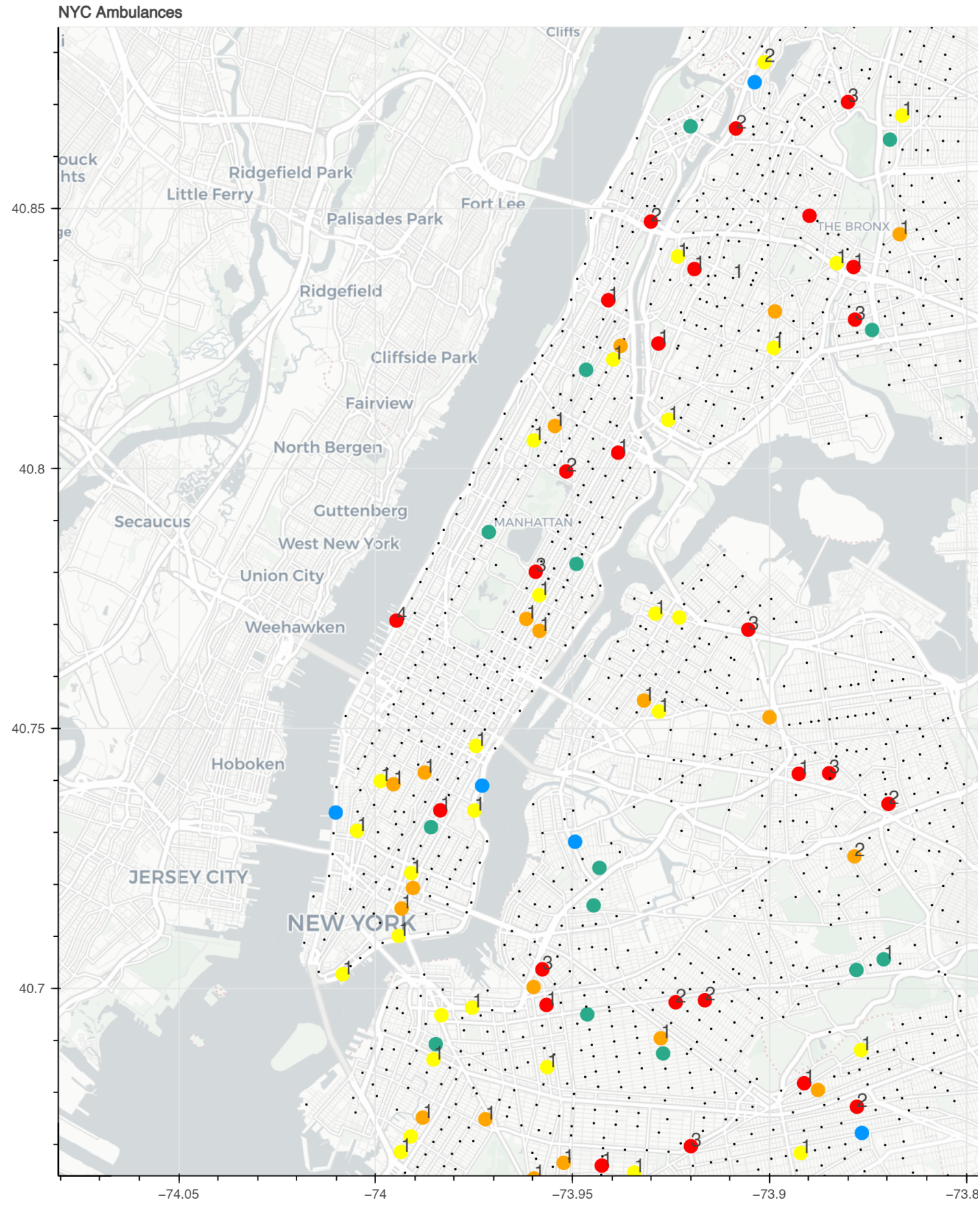
In [3]:
```
# Drop the first column of the xs DataFrame
xs = xs.drop(xs.columns[0], axis=1)

# Melt the xs DataFrame to long format with columns 'index', 'Number_of_Ambulances', and 'Node'
df_melted = xs.reset_index().melt(id_vars='index', value_name='Number_of_Ambulances', var_name='Node')

# Filter out rows where 'Number_of_Ambulances' is 0 or less
df_melted = df_melted[df_melted['Number_of_Ambulances'] > 0]

# Note: The following lines repeat the previous operations. They might be redundant.
# Melt and filter again (this seems redundant and could be removed)
df_melted = xs.reset_index().melt(id_vars='index', value_name='Number_of_Ambulances', var_name='Node')
df_melted = df_melted[df_melted['Number_of_Ambulances'] > 0]

# Rename the melted DataFrame for further processing
df = df_melted

# Convert columns to numeric data type
df['index'] = pd.to_numeric(df['index'])
df['Node'] = pd.to_numeric(df['Node'])
df['Number_of_Ambulances'] = pd.to_numeric(df['Number_of_Ambulances'])

# Pivot the DataFrame to create a table where each cell represents the 'Number_of_Ambulances' for a given 'index
pivot_df = df.pivot(index='index', columns='Node', values='Number_of_Ambulances').fillna(0)

# Calculate the difference between each row in the pivoted DataFrame to get the difference in 'Number_of_Ambular
diff_df = pivot_df.diff().fillna(0)

# Melt the difference DataFrame to long format with columns 'index', 'New_Ambulances_Added', and 'Node'
melted_df = diff_df.reset_index().melt(id_vars='index', value_name='New_Ambulances_Added', var_name='Node')

# Sort the melted DataFrame by 'index'
melted_df = melted_df.sort_values(by='index')

# Filter out rows where 'New_Ambulances_Added' is 0 or less
result = melted_df[melted_df['New_Ambulances_Added'] >0]

# Convert the result to a DataFrame
result = pd.DataFrame(result)

# Note: This line is commented out. If needed, it adds a new row to the result DataFrame.
#new_row = pd.Series({'index':0,'Node':1054,'New_Ambulances_Added':1})
#result = result.append(new_row,ignore_index=True)

# Sort the result DataFrame by 'index' and convert the columns to numeric data type
result = result.sort_values(by='index')
result['index'] = pd.to_numeric(result['index'])
result['New_Ambulances_Added'] = pd.to_numeric(result['New_Ambulances_Added'])
result['Node'] = pd.to_numeric(result['Node'])

# Group by the 'Node' column and sum the 'New_Ambulances_Added' column
grouped_df = result.groupby('Node').agg({'New_Ambulances_Added': 'sum'}).reset_index()

# Sort the grouped DataFrame by the 'Node' column
sorted_df = grouped_df.sort_values(by='Node')
```

In [4]:
```
def plotNetwork(nodes, pop, xs, result, sorted_df, title='Population Graph of NYC', on_map=False):
    """
    Plots a static map of a network = (nodes, links).
    :param nodes: pandas df with 'name', 'x', 'y' cols
                  'x' and 'y' treated as mercator coords
    :param title: str of graph title
    :param target: list of node names
    :param on_map: boolean for map background
    """
    #pre processing
    dfn = pd.merge(pop, nodes, how='inner', left_on='name', right_on='name')
```

```
dfn['label'] = np.array(xs.T[91])
dfn['label'] = dfn['label'].replace(0,None)

# get node colors
colors = []
for i in range(xs.T.shape[0]):
    node_data = result[result['Node'] == i]
    if node_data.empty:
        colors.append('#000000')
    else:
        index_val = node_data.iloc[0]['index']
        if index_val <= 30:
            colors.append('#FF0000')
        elif index_val <= 60:
            colors.append('#FFA500')
        elif index_val <= 90:
            colors.append('#FFFF00')
        elif index_val <= 120:
            colors.append('#2AAA8A')
        elif index_val <= 150:
            colors.append('#0096FF')
        else:
            colors.append('#000000')

#get node sizes
size = []
for i in range(xs.T.shape[0]):
  if sorted_df[sorted_df['Node'] == i].shape[0] == 0:
    size.append(1)
  else:
    size.append(10+int(sorted_df[sorted_df['Node'] == 5]['New_Ambulances_Added']))
graph = GraphRenderer()

# extract data
node_ids = dfn.name.values.tolist()
x = dfn.x.values.tolist()
y = dfn.y.values.tolist()
label = dfn.label.values.tolist()

# get plot boundaries
min_x, max_x = min(dfn.x)+2000, max(dfn.x)-2000
min_y, max_y = min(dfn.y)+2000, max(dfn.y)-2000

plot = figure(x_range=(min_x, max_x), y_range=(min_y, max_y),
              x_axis_type="mercator", y_axis_type="mercator",
              title=title,
              width=1000, height=1000,
              toolbar_location=None, tools=[]
             )

if on_map == True:
    # add map tile
    plot.add_tile(get_provider(Vendors.CARTODBPOSITRON_RETINA))

# define nodes
graph.node_renderer.data_source.add(node_ids, 'index')
graph.node_renderer.data_source.add(colors, 'colors')
graph.node_renderer.data_source.add(size, 'size')
graph.node_renderer.glyph = Circle(size='size', fill_color='colors', line_color='colors')

label_source = ColumnDataSource(dfn)
labels = LabelSet(x='x', y='y', text='label', level='glyph',
          x_offset=0, y_offset=0, source=label_source)

plot.add_layout(labels)

# set node locations
graph_layout = dict(zip(node_ids, zip(x, y)))
graph.layout_provider = StaticLayoutProvider(graph_layout=graph_layout)

plot.renderers.append(graph)

show(plot)
```

```
plotNetwork(dfn,dfp,xs, result, sorted_df, title="NYC Ambulances", on_map=True)
```

```
BokehDeprecationWarning: 'get_provider' was deprecated in Bokeh 3.0.0 and will be removed, use 'add_tile directly' instead.
```



First 30 Locations: Red

Next 30 Locations (30-60) : Orange

Third 30 Locations (60-90): Yellow

Next 30 Locations (90-120): Green

Last 4 locations (120-140): Blue

Locations with no Ambulances: Black

In [5]:
```
result.to_csv('final_table.csv')
```

In [ ]:
```

```

Appendix

Julia VanPutte jhv42, ORIE 4130 Case 1, Due 9/22/2023

consulted chat GPT for some coding help. Any code blocks aided by chat GPT are cited with in-line comments.

Start with HW 1 Solution

In [56]:
```
!pip install --upgrade bokeh jinja2 python-dateutil packaging matplotlib tornado pillow numpy ty
from markupsafe import Markup
import pandas as pd
!pip install haversine
from haversine import haversine

!pip install statsmodels
import statsmodels.api as sm
import matplotlib.pyplot as plt
from matplotlib import cm
import matplotlib.colors
import numpy as np
import math
from pyproj import Transformer
import copy
import itertools
import networkx as nx # tool for graphs and graph algorithms
!pip install bokeh
from bokeh.plotting import figure, show
from bokeh.io import output_notebook
from bokeh.tile_providers import get_provider, Vendors
from bokeh.models import (GraphRenderer, Circle, MultiLine, StaticLayoutProvider,
HoverTool, TapTool, EdgesAndLinkedNodes,
NodesAndLinkedEdges,
ColumnDataSource, LabelSet, NodesOnly
)
!pip install gurobipy
from gurobipy import Model, GRB, quicksum
import pandas as pd
import numpy as np
```

```
Requirement already satisfied: statsmodels in c:\users\julia\anaconda3\envs\engri_1101\lib\site-packages (0.13.5)
Requirement already satisfied: numpy>=1.17 in c:\users\julia\anaconda3\envs\engri_1101\lib\site-packages (from statsmodels) (1.21.6)
Requirement already satisfied: packaging>=21.3 in c:\users\julia\anaconda3\envs\engri_1101\lib\site-packages (from statsmodels) (23.1)
Requirement already satisfied: patsy>=0.5.2 in c:\users\julia\anaconda3\envs\engri_1101\lib\site-packages (from statsmodels) (0.5.3)
Requirement already satisfied: pandas>=0.25 in c:\users\julia\anaconda3\envs\engri_1101\lib\site-packages (from statsmodels) (1.3.5)
Requirement already satisfied: scipy<1.8,>=1.3 in c:\users\julia\anaconda3\envs\engri_1101\lib\site-packages (from statsmodels) (1.5.4)
Requirement already satisfied: python-dateutil>=2.7.3 in c:\users\julia\anaconda3\envs\engri_1101\lib\site-packages (from pandas>=0.25->statsmodels) (2.8.2)
Requirement already satisfied: pytz>=2017.3 in c:\users\julia\anaconda3\envs\engri_1101\lib\site-packages (from pandas>=0.25->statsmodels) (2022.7)
Requirement already satisfied: six in c:\users\julia\anaconda3\envs\engri_1101\lib\site-packages (from patsy>=0.5.2->statsmodels) (1.16.0)
Requirement already satisfied: bokeh in c:\users\julia\anaconda3\envs\engri_1101\lib\site-packages (2.4.3)
Requirement already satisfied: Jinja2>=2.9 in c:\users\julia\anaconda3\envs\engri_1101\lib\site-packages (from bokeh) (3.1.2)
Requirement already satisfied: PyYAML>=3.10 in c:\users\julia\anaconda3\envs\engri_1101\lib\site-packages (from bokeh) (6.0)
Requirement already satisfied: numpy>=1.11.3 in c:\users\julia\anaconda3\envs\engri_1101\lib\site-packages (from bokeh) (1.21.6)
Requirement already satisfied: typing-extensions>=3.10.0 in c:\users\julia\anaconda3\envs\engri_1101\lib\site-packages (from bokeh) (4.7.1)
Requirement already satisfied: tornado>=5.1 in c:\users\julia\anaconda3\envs\engri_1101\lib\site-packages (from bokeh) (6.2)
Requirement already satisfied: packaging>=16.8 in c:\users\julia\anaconda3\envs\engri_1101\lib\site-packages (from bokeh) (23.1)
Requirement already satisfied: pillow>=7.1.0 in c:\users\julia\anaconda3\envs\engri_1101\lib\site-packages (from bokeh) (9.5.0)
Requirement already satisfied: MarkupSafe>=2.0 in c:\users\julia\anaconda3\envs\engri_1101\lib\site-packages (from Jinja2>=2.9->bokeh) (2.1.3)
Requirement already satisfied: gurobipy in c:\users\julia\anaconda3\envs\engri_1101\lib\site-packages (10.0.3)
```

In [58]:

```
# load nodes
data = pd.read_csv('nyc_nodes.csv')
dfn = pd.DataFrame(data)
# load links
data = pd.read_csv('nyc_links.csv')
dfl = pd.DataFrame(data)
# load population
data = pd.read_csv('nyc_population.csv')
dfp = pd.DataFrame(data)
```

In [236…

```
output_notebook()
def plotNetwork(nodes, links, dis_time, criteria, population, title='Plot of Graph', target=None
    """
    Plots a static map of a network = (nodes, links).
    :param nodes: pandas df with 'name', 'x', 'y' cols
                  'x' and 'y' treated as mercator coords
    :param links: pandas df with 'start' and 'end' cols
                  with entries matching nodes' names
    :param title: str of graph title
    :param target: list of node names
    :param on_map: boolean for map background
    """

    dfn = nodes
    dfl = links
```

```
dfs = dis_time
dfp = population
speed = dfs['distance'].values/dfs['time'].values.tolist()

if len(criteria)>0:
    groups = len(criteria)

    color = cm.get_cmap('Blues', groups-2)    # PiYG
    c_map = {}
    c_map[0] = '#ff0000'
    #c_map[1] = '#99c199'
    for i in range(color.N):
        rgba = color(i)
        # rgb2hex accepts rgb or rgba
        c_map[i+1] = matplotlib.colors.rgb2hex(rgba)
    c_map[i+2] = '#FFFF00'


    e_clr = []
    for i in speed:
        for k in range(groups):
            if i>=criteria[k][0] and i<=criteria[k][1]:
                e_clr.append(c_map[k])
                break

else:
    e_clr = []
    for i in speed:
            e_clr.append('#A0A0A0')


# extract data
node_ids = dfn.name.values.tolist()
start = dfl.start.values.tolist()
end = dfl.end.values.tolist()
x = dfn.x.values.tolist()
y = dfn.y.values.tolist()


# get plot boundaries
min_x, max_x = min(dfn.x)+2000, max(dfn.x)-2000
min_y, max_y = min(dfn.y)+2000, max(dfn.y)-2000

plot = figure(x_range=(min_x, max_x), y_range=(min_y, max_y),
              x_axis_type="mercator", y_axis_type="mercator",
              title=title,
              width=600, height=470,
              toolbar_location=None, tools=[]
             )

graph = GraphRenderer()

if on_map == True:
    # add map tile
    plot.add_tile(get_provider(Vendors.CARTODBPOSITRON_RETINA))

# define nodes
graph.node_renderer.data_source.add(node_ids, 'index')
graph.node_renderer.glyph = Circle(line_color='green', line_alpha=0,
                                   fill_color='green', size=3.5,
                                   fill_alpha=0
                                  )
```

```
    # define edges
    graph.edge_renderer.data_source.data = dict(start=list(start),
                                                end=list(end),e_clr=e_clr
                                               )
    graph.edge_renderer.glyph = MultiLine(line_color = 'e_clr',
                                          line_alpha=1, line_width=.6
                                         )

    # set node locations
    graph_layout = dict(zip(node_ids, zip(x, y)))
    graph.layout_provider = StaticLayoutProvider(graph_layout=graph_layout)

    plot.renderers.append(graph)


    # add POIS
    if len(population)>0:
        color = cm.get_cmap('Reds', 7)
        for i in range(7):
            target = []
            for index,row in dfp.iterrows():
                if row['pop']>=(i*2500) and row['pop']<((i+1)*2500):
                    target.append(row['name'])

            dfpp = dfn.loc[dfn['name'].isin(target)]
            source = ColumnDataSource(dfpp)
            poi = Circle(x="x", y="y", size=3.5, line_color='blue',
                         fill_color=matplotlib.colors.rgb2hex(color(i)), line_width=0.1
                        )

            plot.add_glyph(source, poi)

    show(plot)
```

BokehJS 2.4.3 successfully loaded.

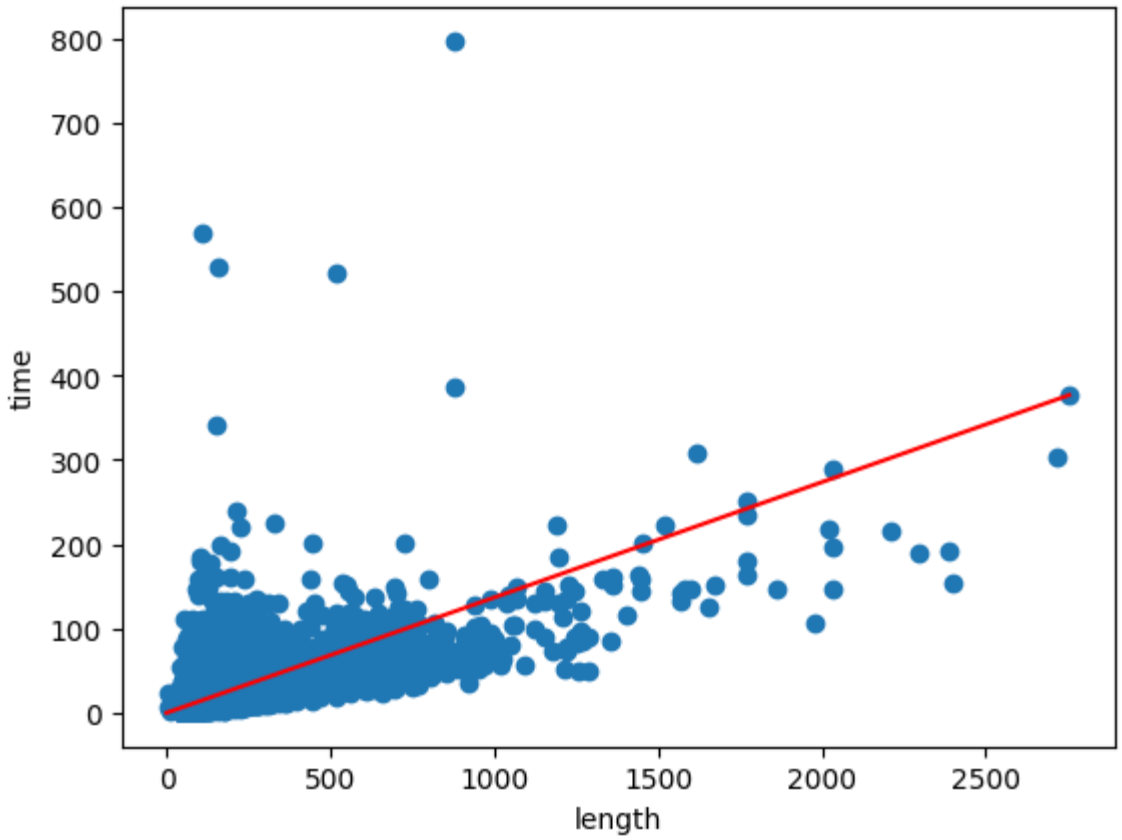
Question 1

In this question, we are going to work with travel-time data in New York City. There are two files you will work with. The first file is “nyc nodes.csv”, which tells you the name, latitude/longitude (latlong) and Mercator coordinates of the points in which we are interested. The second file is “nyc links.csv”, where each row corresponds to a road segment and tells you the start node, end node and average travel time of that road segment from 7pm-12am on a weekday. For simplicity, we assume all roads are two-way.

a) Compute the length of each road segment. (You can use the haversine() function in python to compute distances when you have latlong. There is no need to report the length.) Generate a scatter plot where the length of the segment is on one axis and the time required to travel the segment is on the other. Fit the length-time data to a straight line and report its slope. According to your observations, what is the unit of time used in the data? Justify your answer.

In [60]:
```
#For Q1 part (a)
dfs = pd.DataFrame(columns = ["distance", "time"])

for index,row in dfl.iterrows():
    start_point = row['start']
    end_point = row['end']
    #time = row['delay5pm']
```

```
    start_point_coord_row = dfn[(dfn.name==start_point)].index[0]
    end_point_coord_row = dfn[(dfn.name==end_point)].index[0]
    start_point_coord = dfn.loc[start_point_coord_row,['lat','lon']].tolist()
    end_point_coord = dfn.loc[end_point_coord_row,['lat','lon']].tolist()
    dfs.loc[index] = [haversine(start_point_coord, end_point_coord)*1000,row['time']]


plt.xlabel("length")
plt.ylabel("time")
plt.scatter(dfs['distance'], dfs['time'])
y = dfs.loc[:,['time']]
x = dfs.loc[:,['distance']]
model = sm.OLS(y, x).fit()
slope = model.params[0]
x_0 = 0
y_0 = 0
x_1 = max(x['distance'])
y_1 = slope* (x_1 - x_0) + 0_0
plt.plot([x_0, x_1], [y_0, y_1], c='r')
plt.show()
print("Fitted Speed is "+ str(round(1/slope,2)) + " m/s")
```




```
Fitted Speed is 7.31 m/s
```

The slope of the fitted line is 0.137, whose inverse is 7.3 (the fitted speed). Since the unit of the distance used here is “meter”, the unit of time could only be “second” to make this data reasonable. A speed of 7.3 m/s equates to 26 km/hr

b) Color each road segment on the map by the speed on that road segment. (You can work from the sample code named “map.py.” It is ok to not provide a legend, but be sure to explain in words which color corresponds to which speed range) and submit the graph. Also looking at the scatter plot you drew in part (a), does anything look strange? What? Should we adjust these strange data points? Justify your answer. (There is more than one correct answer to this question). If your answer is yes, adjust the strange data points by adjusting the travel time and plot the adjusted length-time scatter plot.

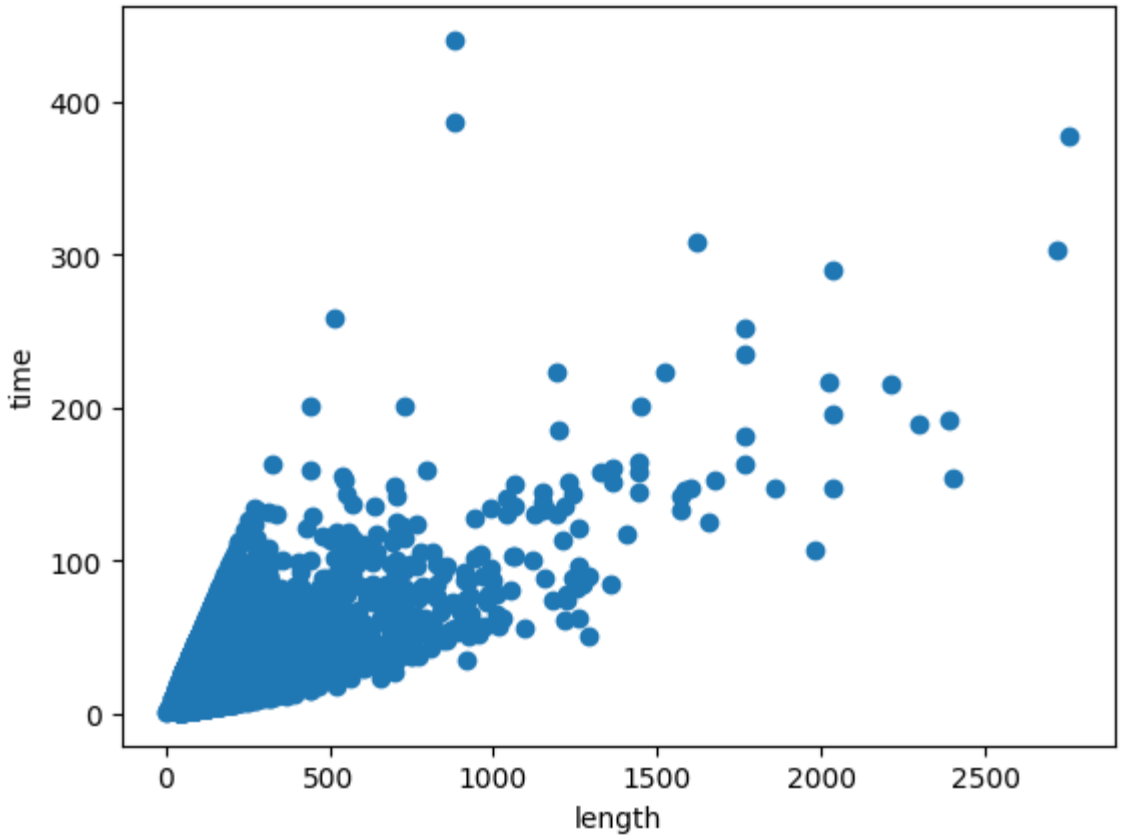
Yellow corresponds to speed being higher than 20m/s, red corresponds to speed being lower than 2m/s. For speed between 2m/s and 20m/s, we divide them into 6 intervals of length 3, the deeper the blue is, the higher the corresponding speed is.

From the scatter plot in (a), we can see some points who are far away from the fitted red line. So let us focus on road segments whose speed is too low or too high. Look at the two figures below, we can see that there are only a few short road segments whose speed are lower than 2m/s and they spread all over the city. When we focus on road segments whose speed are between 2m/s and 3m/s, many streets on Manhattan kick in. So road segments whose speed are slower than 2m/s seem to be strange data points and we will adjust them to 2m/s.

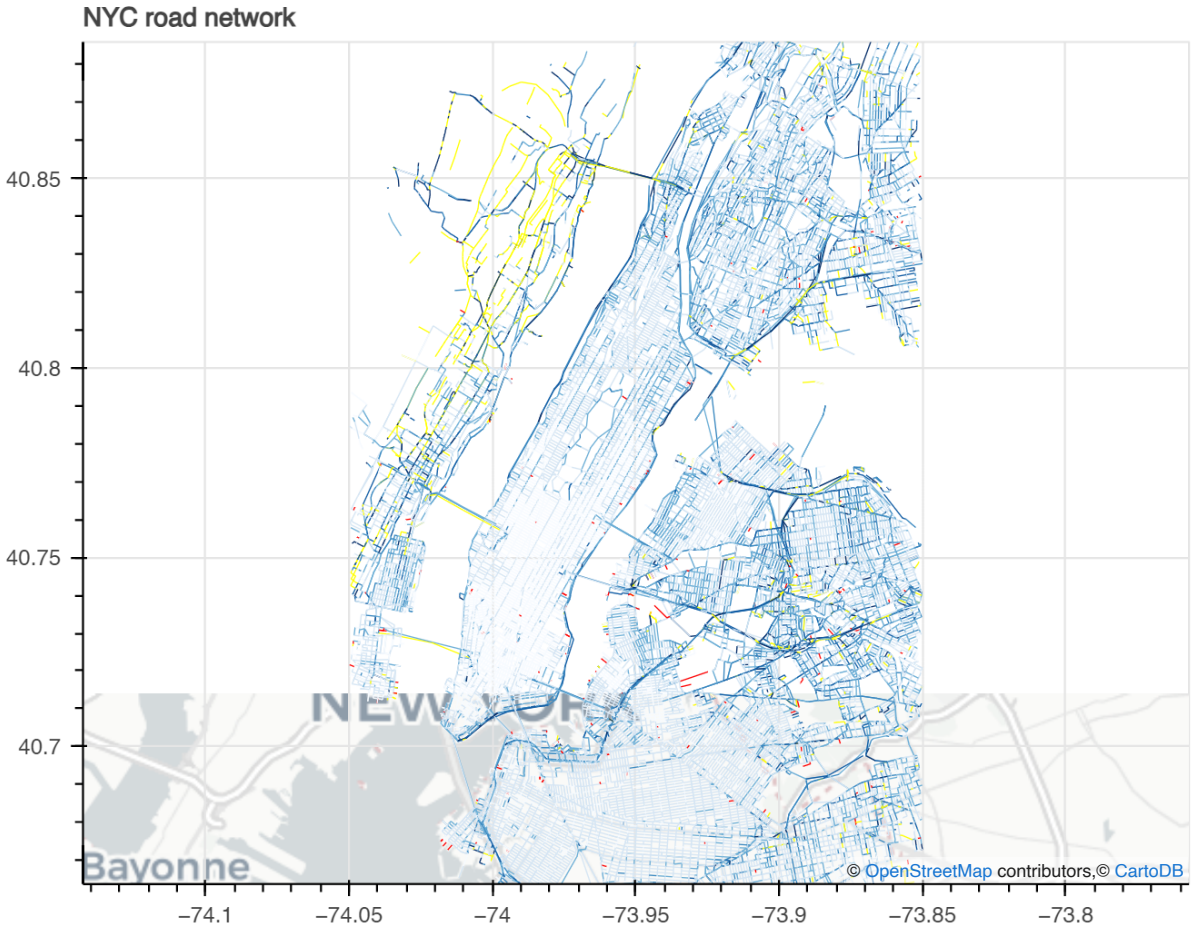
Then, let us focus on road segments whose speed are too high. We can see below that road segments with speed are higher than 30m/s are fairly short and all over the place, so they are very likely to be wrong because that corresponds to more than 108km/h. For road segments whose speed are higher than 20m/s in NYC, they seem to be all over the place and wrong as well. We may not adjust road segments whose speed is between 20m/s and 30m/s on NJ side because those road segments are clustered and they seem to be express roads. Thus, we will adjust road segments whose speed are higher than 30m/s to 30m/s, and adjust road segments in NYC whose speeds are higher than 20m/s to 20m/s.

The adjusted scatter plot is as follows:

In [61]:
```
#For Q1 part (b)


dfs_new = copy.deepcopy(dfs)    #new data with adjusted time
for index,row in dfs_new.iterrows():
    if row['distance']/row['time']<2:
        row['time'] = row['distance']/2
    else:
        if row['distance']/row['time']>30:
            row['time'] = row['distance']/30
        start = dfl.loc[index]['start']
        end = dfl.loc[index]['end']
        start_loc = dfn[dfn['name']==start][['lat','lon']].values.tolist()[0]
        end_loc = dfn[dfn['name']==end][['lat','lon']].values.tolist()[0]

        if (0<(start_loc[0]-40.66)/(start_loc[1]+74.05)<2 or 0<(end_loc[0]-40.66)/(end_loc[1]+74
            row['time'] = row['distance']/20

plt.xlabel("length")
plt.ylabel("time")
plt.scatter(dfs_new['distance'], dfs_new['time'])
plt.show()
```



In [62]:
```
plotNetwork(dfn, dfl,dfs, [[0,2],[2,5],[5,8],[6,8],[8,11],[11,14],[14,17],[17,20],[20,100]],[],
```



c) Ambulances usually travel faster than regular traffic speeds when traveling at “Lights and Sirens” speeds on their way to an incident. Let us assume these travel times are 0.9 times the travel time we have in the

data for regular traffic. Compute the travel time for the ambulances on each road segment. Don't hand in the adjusted travel times; just show us your code.

In [63]:
```
#For Q1 part (c)
ambulance_dfl = copy.deepcopy(dfl)    #travel time for ambulance
ambulance_dfl['time'] = 0.9*dfs_new['time']
```

Question 2

In this question, we want to model how emergency calls for ambulance service, as received by the Fire Department of New York (FDNY), are distributed across the city. The way we model these calls is not accurate, but should be enough for our work in this class. (True data would require us to sign a non-disclosure agreement.) We will use "nyc population.csv", which tells you the population in each region within NYC (not the whole city) and the label of a representative point matching that region

a) There were 1.55M emergency calls in total in 2022 (see https://www.nyc.gov/assets/operations/downloads/pdf/pmmr2022/fdny.pdf). Assuming that the calls arrive at a constant rate per hour (they don't; typically they are concentrated during the day with peaks in the morning and afternoon), what is the number of calls per hour across the entire city?

Assuming the calls arrive at a constant rate per hour, and having 1.55M calls in 2022, we can compute the numper of calls per hour as:

$$\frac{1,550,000 \text{ calls}}{1 \text{ year}} * \frac{1 \text{ year}}{365 \text{ days}} * \frac{1 \text{ day}}{24 \text{ hours}} = 177 \text{ calls/hour across the entire city in 2022}$$

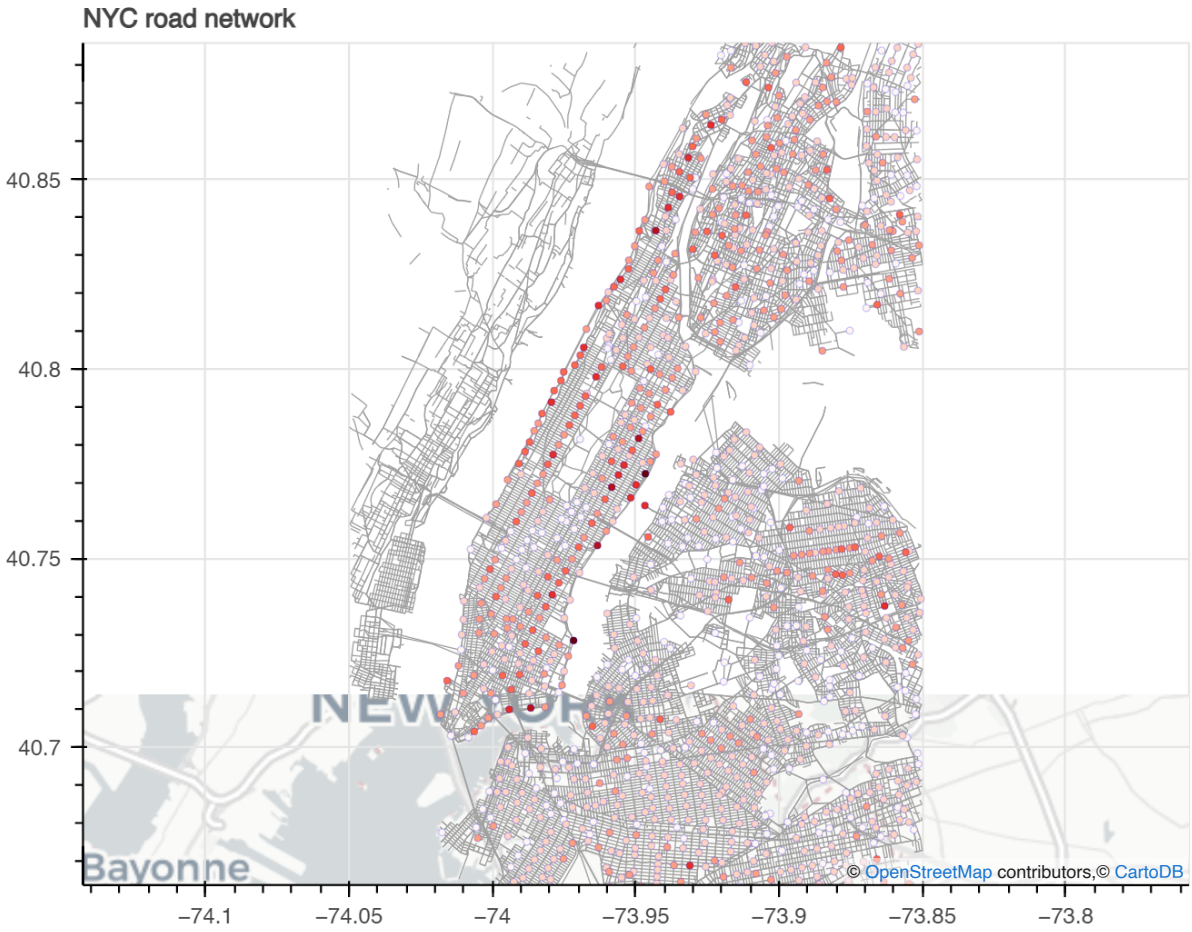
b) We divide NYC into a little more than 1000 different regions and assume that the probability that the next call comes from a particular region is proportional to the population of the region. (In reality they're more likely to come from poorer areas.) Use "nyc population.csv" to compute the probability that the next call comes from each region. (Assume the probability sums up to 1 exactly). Don't hand in your probabilities; just describe how you got them.

The population dataframe has 1161 regions each with a population. The probability that the next call comes from each region will be the population in the region divided by the total population. We can calculate this using the code below to add a new column in the data frame corresponding to the probability of a call from that region:

In [64]:
```
#For Q2 part (b)
total_population = sum(dfp['pop'].tolist())
call_prob = []
for index,row in dfp.iterrows():
    call_prob.append(row['pop']/total_population)
population_with_prob = copy.deepcopy(dfp) #a new dataframe with a new column showing the probabi
population_with_prob['prob'] = call_prob
```

c) Generate a map where each region is indicated on the map at its representative point, colored to indicate the region's population. (You can work from sample code named "map.py." It is ok to not provide a legend, but explain in words which color corresponds to which population range.)

In [65]:
```
#For Q2 part (c)
Network(dfn, dfl,dfs, [], dfp, title="NYC road network", target=[], on_map=True)
```



Since the maximum population in a region is 16538 in our data, we divide 0 to 17500 into 7 intervals of length 2500, the deeper the red is, the larger the corresponding population is.

(d) Compute the smallest ambulance travel time between each pair of representative points. (In python, you may use single source dijkstra() function from package networkx. For the first argument of this function, consider constructing the graph by from pandas edgelist() function from package networkx). There is no need to report these travel times, but do report the travel times from Pennsylvania Station to Wall St, and also the travel time from Cornell Tech to LaGuardia airport. And of course supply your code.

In [66]:
```
#For Q2 part (d)
poi = dfp['name'].tolist()  #list of representaitve points
G = nx.from_pandas_edgelist(ambulance_dfl, 'start', 'end', 'time')
"""
Please use this "distances" matrix for your case 1
"""
distances = np.zeros([len(poi),len(poi)])   #a matrix showing shortest time needed bewtween each
for i in range(len(poi)):
    out = nx.single_source_dijkstra(G, poi[i], weight='time')
    results = pd.DataFrame({'node_id':poi})
    outcome = results['node_id'].map(out[0])
    distances[i,:] = outcome


#Pennsylvania Station is 42439440 (index 148)
#Wall St is 42451409 (index 44)
#Cornell Tech is 42432875 (index 213)
#LaGuardia airport is 562073924 (index 296)

print("Smallest ambulance travel time from Pennsylvania Station to Wall St is " + str(distances[
t("Smallest ambulance travel time from Cornell Tech to LaGuardia airport is " + str(distance
```

```
Smallest ambulance travel time from Pennsylvania Station to Wall St is 883.8722051345675s.
Smallest ambulance travel time from Cornell Tech to LaGuardia airport is 852.4178312878158s.
```

Case 1

- requirements for response time performance (90% of calls reached within 9 minutes) time interval 7pm-midnight on weekdays.
- 70% transported to a hospital.
  - on average 35 minutes at scene, transport to hospital requires another 15 minutes
  - transfer the patient to the receiving hospital. anywhere from 15 minutes to an hour
  - (Recently, the hospital offload delays have mostly been closer to one hour.)
- 30% of cases the call is completed at the scene.
  - on average, 45 minutes at scene
- call-taking time plus chute time averages out at 4 minutes.

- Ambulances travel at regular traffic speeds, except when on the way to an emergency call, where they travel at "lights and sirens" speeds, which are typically 10% faster than regular travel speeds.
- use the solutions Homework 1 as an input to your modeling; refer to the time window (7pm-midnight on weekdays) that we are studying.
- Our study area cuts off parts of the Bronx, Queens and Brooklyn and all of Staten Island. divide the call rate per hour we found in Q2a of HW 1 by 2 to provide an estimate of the hourly call arrival rate to our study area.

1. make clear how many ambulances are needed to meet the response time requirements;
   - for each potential hospital offload delay (15,30,45,60 min)
2. give a compliance table showing where ambulances should be placed as a function of the number of free ambulances
   - 30 minute offload delays only
3. justify your solution.

- In your appendix you should provide your compliance table (if you recommend, say, 140 ambulances, then this is just a list of 140 locations (nodes), starting from the location where you would place a single ambulance if only 1 ambulance is free, up to the location where you would place the last ambulance if 140 ambulances are free).
- Within the 6 pages of your report you should also provide a graphical depiction of your compliance table – to do that, plot a map of NYC with the first 30 locations in your compliance table colored red, the next 30 locations colored orange, the next 30 locations colored yellow, and so on.
- Your answer to #3 should be sufficient to allow another student to recreate your results. Within the 6 pages of your report you should give the mathematical formulations (like we did in class) for any optimization methods you use and cite any other sources you use.

Mean Service Time = P(Hospital) *time_hospital* + *P(no Hospital)*time_no_hospital=
$0.7 * (35 + 15 + \text{offload}) + 0.3 * (45)$

Number of Ambulances Needed (for 15 , 30, 45 , 60 offload delay)

90% of calls reached in 9 minutes

70% Hospital : 35 scene + 15 transport + (15,30,45,60) offload

30% at scene : 45 mins at scene.

4 minutes deployment

Travel 10% faster than speed.

Call rate = call rate / 2

PART 1: Finding minimum number of ambulances to cover 90% of demand

Idea to get number of ambulances

Solve MEXCLP with a chosen, such that p < 1 (about 0.6-0.7). MEXCLP will give E[# of calls per hour with response time under threshold].

Make sure E[# of calls per hour with response time under threshold] ≥ 0.9

MEXCLP Problem

$i = 1, \ldots, m$ demand locations

$j = 1, \ldots, n$ ambulance stations

x_j = # of ambulances stationed at location j ; $0 \leq x_j \leq c_j$

y_i = # of ambulances that can reach location i on time ; integer

α_{ij} = 1 if ambulance at j can reach i in ≤ 9 mins , 0 otherwise (binary)

Ambulance is available with probability 1-p, where p=\frac{∑_iλ_i}{μa}

1/μ = mean service time; λ = calls per hour

∑_jx_j=a

∑_jx_ja_{ij}=y_i

--

Objective **\text{max} ∑_{i=1}^{m}∑_{k=1}^ab_{ik}y_{i}(k)**

st:

y_i=y_i(1)+y_i(2)+...+y_i(a)

b_{ij} = λ_i(1-p^j) - b_{i j-1} and b_{i1}=λ_i(1-p)

∑_jx_jα_{ij}\ge y_i

∑_jx_j=a

0≤x_j

x_{ij} integer

y_i integer

y_{ij} binary

α_{ij} = 1 if ambulance at j can reach i in ≤ 9 mins , 0 otherwise (binary)

assuming capacities are unbounded at each node

15 Minute Offload Delay

In [360…
```
distances = pd.DataFrame(distances)
alpha_ij = distances<(5*60)
```

In [361…
```
offload_delay = 15
lambda_i =[(177/2)*population_with_prob['prob']][0]
mu = 1/(.7*(35/60+15/60+offload_delay/60)+0.3*(45/60))
```

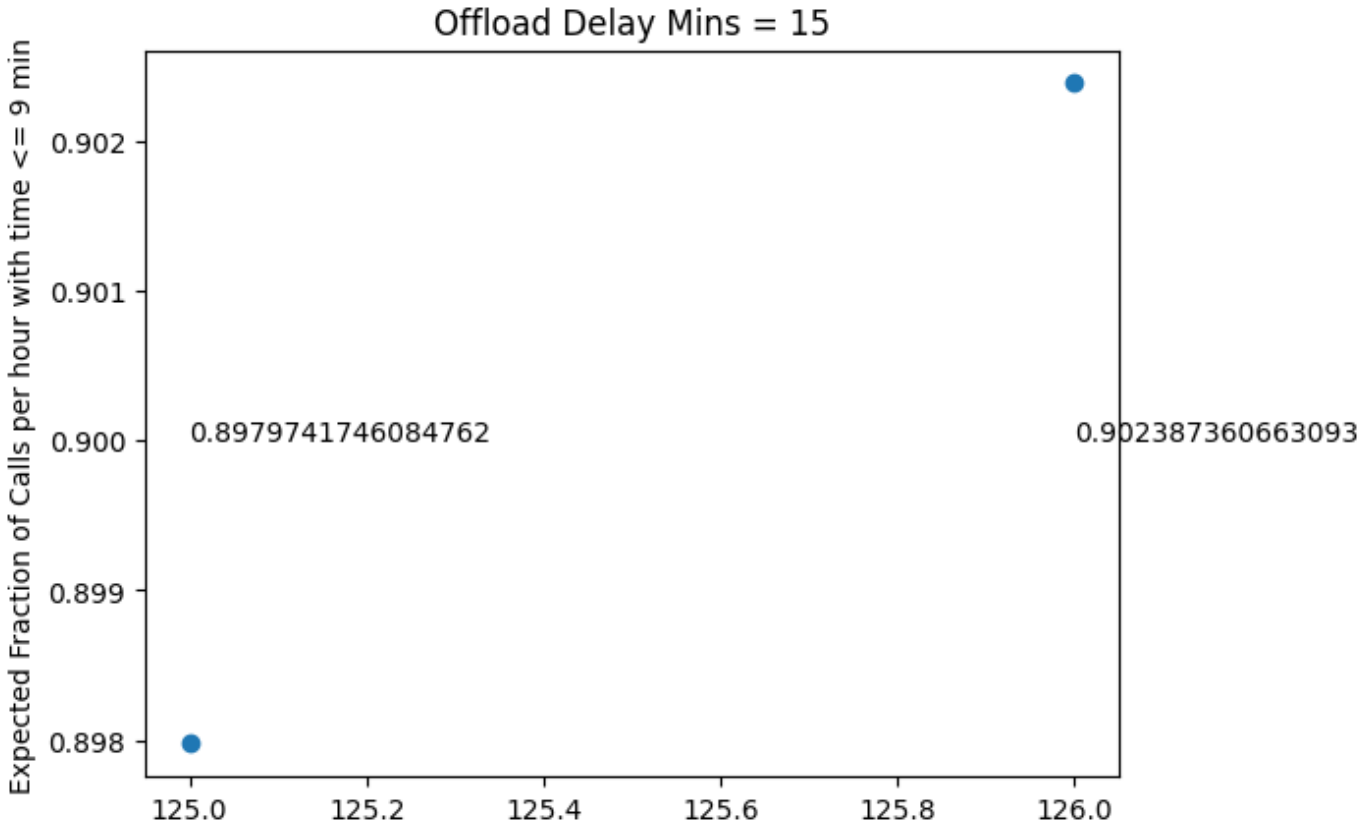
In [362…
```
#find minimum value of a
#np.sum(lambda_i)/a/mu <= 0.85
#0.85a >= np.sum(lambda_i)/mu
a_min = math.ceil(np.sum(lambda_i)/mu/0.7)
a_min
```

Out[362]:
```
125
```

In [363…
```
# WITH AID FROM CHAT GPT IN WRITING OPTIMIZATION PROBLEM
optimal_val = [0]
a_ = [a_min]
#need smallest a such that optimal_value >0.9
while optimal_val[-1] <= 0.9:

    # Your provided data and variables
    a = a_[-1]
    m_demand = len(lambda_i)
    n_stations = alpha_ij.shape[1]
    p = np.sum(lambda_i) / a / mu
    print(p)

    # Create a 2D NumPy array for b_ij
    b_ij = np.zeros((m_demand, a))  # Initialize with zeros

    # Fill in the array
    for i in range(m_demand):
        for j in range(1, a + 1):
            b_ij[i, j-1] = lambda_i[i] * (1 - p ** j) - (lambda_i[i] * (1 - p ** (j - 1)) if j >

    # Initialize the model
    m = Model("MEXCLP")
```

```
    # Add variables
    x_j = m.addVars(n_stations, lb=0, vtype=GRB.INTEGER, name="x_j")
    y_i_k = m.addVars(m_demand, a, vtype=GRB.BINARY, name="y_i_k")  # y_i(k) is now binary
    y_i = m.addVars(m_demand, lb=0, vtype=GRB.INTEGER, name="y_i")



    # Add constraints
    # Sum of x_j is equal to a
    m.addConstr(sum(x_j[j] for j in range(n_stations)) == a, "c1")

    # For each demand location i, y_i is the sum of y_i(k) for k=1,...,a
    m.addConstrs((y_i[i] >= sum(y_i_k[i, k] for k in range(10)) for i in range(m_demand)), "c2")

    # Constraint related to alpha_ij and y_i
    m.addConstrs((sum(alpha_ij.iloc[i][j] * x_j[j] for j in range(n_stations)) >= y_i[i] for i i

    # Set the objective
    m.setObjective(sum(b_ij[i, k] * y_i_k[i, k] for i in range(m_demand) for k in range(a)), GRB

    # Optimize the model
    m.optimize()

    # Extract results
    if m.status == GRB.Status.OPTIMAL:
        x_j_values = [x_j[j].X for j in range(n_stations)]
        y_i_values = [y_i[i].X for i in range(m_demand)]
        print(f"Optimal solution: x_j = {x_j_values}, y_i = {y_i_values}")
        optimal_objective_value = m.objVal
        print(f"Optimal objective value: {optimal_objective_value}")
    else:
        print("No optimal solution found.")
    print('\n')
    optimal_val.append(optimal_objective_value/(177/2))
    a_.append(a_[-1]+1)
print(optimal_val)
print(a_)
```

Out[364]: Text(0.5, 1.0, 'Offload Delay Mins = 15')



For 15 minute offload delay, we want 126 Ambulances.

Optimal Objective Value = 79.86128141868373.

30 Minute offload delay

In [365…
```
offload_delay = 30
lambda_i =[(177/2)*population_with_prob['prob']][0]
mu = 1/(.7*(35/60+15/60+offload_delay/60)+0.3*(45/60))
```

In [366…
```
#find minimum value of a
#np.sum(lambda_i)/a/mu <= 1
#a >= np.sum(lambda_i)/mu
a_min = math.ceil(np.sum(lambda_i)/mu/0.75)
```

In [367…
```
# WITH AID FROM CHAT GPT IN WRITING OPTIMIZATION PROBLEM
optimal_val = [0]
a_ = [a_min]
#need smallest a such that optimal_value >0.9
while optimal_val[-1] <= 0.9:

    # Your provided data and variables
    a = a_[-1]
    m_demand = len(lambda_i)
    n_stations = alpha_ij.shape[1]
    p = np.sum(lambda_i) / a / mu
    print(p)
```

```
    # Create a 2D NumPy array for b_ij
    b_ij = np.zeros((m_demand, a))  # Initialize with zeros

    # Fill in the array
    for i in range(m_demand):
        for j in range(1, a + 1):
            b_ij[i, j-1] = lambda_i[i] * (1 - p ** j) - (lambda_i[i] * (1 - p ** (j - 1)) if j >

    # Initialize the model
    m = Model("MEXCLP")

    # Add variables
    x_j = m.addVars(n_stations, lb=0, vtype=GRB.INTEGER, name="x_j")
    y_i_k = m.addVars(m_demand, a, vtype=GRB.BINARY, name="y_i_k")  # y_i(k) is now binary
    y_i = m.addVars(m_demand, lb=0, vtype=GRB.INTEGER, name="y_i")



    # Add constraints
    # Sum of x_j is equal to a
    m.addConstr(sum(x_j[j] for j in range(n_stations)) == a, "c1")

    # For each demand location i, y_i is the sum of y_i(k) for k=1,...,a
    m.addConstrs((y_i[i] >= sum(y_i_k[i, k] for k in range(10)) for i in range(m_demand)), "c2")

    # Constraint related to alpha_ij and y_i
    m.addConstrs((sum(alpha_ij.iloc[i][j] * x_j[j] for j in range(n_stations)) >= y_i[i] for i i

    # Set the objective
    m.setObjective(sum(b_ij[i, k] * y_i_k[i, k] for i in range(m_demand) for k in range(a)), GRB

    # Optimize the model
    m.optimize()

    # Extract results
    if m.status == GRB.Status.OPTIMAL:
        x_j_values = [x_j[j].X for j in range(n_stations)]
        y_i_values = [y_i[i].X for i in range(m_demand)]
        print(f"Optimal solution: x_j = {x_j_values}, y_i = {y_i_values}")
        optimal_objective_value = m.objVal
        print(f"Optimal objective value: {optimal_objective_value}")
    else:
        print("No optimal solution found.")
    print('\n')
    optimal_val.append(optimal_objective_value/(177/2))
    a_.append(a_[-1]+1)
print(optimal_val)
print(a_)
a_30 = a_[-2]
```

```
9.0, 6.0, 5.0, 9.0, 6.0, 10.0, 5.0, 10.0, 6.0, 5.0, 10.0, 10.0, 8.0, 9.0, 6.0, 8.0, 10.0, 9.0, 1
0.0, 6.0, 7.0, 8.0, 10.0, 8.0, 9.0, 8.0, 10.0, 10.0, 3.0, 6.0, 10.0, 8.0, 10.0, 6.0, 8.0, 10.0,
10.0, 9.0, 3.0, 10.0, 3.0, 6.0, 9.0, 8.0, 9.0, 9.0, 7.0, 10.0, 10.0, 7.0, 10.0, 6.0, 6.0, 7.0,
8.0, 9.0, 6.0, 8.0, 8.0, 8.0, 5.0, 9.0, 10.0, 7.0, 10.0, 10.0, 9.0, 6.0, 9.0, 10.0, 4.0, 10.0,
5.0, 5.0, 10.0, 5.0, 10.0, 10.0, 10.0, 7.0, 8.0, -0.0, 10.0, 9.0, 9.0, 7.0, 5.0, 10.0, 10.0, 8.
0, 10.0, 9.0, 7.0, 6.0, 10.0, 7.0, 6.0, 9.0, 10.0, 9.0, 10.0, 10.0, 7.0, 10.0, 10.0, 8.0, 3.0,
0.0, 9.0, 10.0, 10.0, 10.0, 10.0, 10.0, 6.0, 10.0, 7.0, 8.0, 9.0, 6.0, 10.0, 10.0, 10.0, 6.0, 1
0.0, 10.0, 6.0, 7.0, 7.0, 8.0, 9.0, 9.0, 5.0, 6.0, 9.0, 10.0, 9.0, 10.0, 3.0, 3.0, 4.0, 5.0, 10.
0, 9.0, 8.0, 10.0, 6.0, 7.0, 4.0, 10.0, 7.0, 9.0, 9.0, 5.0, 6.0, 8.0, 8.0, 8.0, 10.0, 8.0, 9.0,
10.0, 9.0, 9.0, 6.0, 9.0, 7.0, 5.0, 9.0, 9.0, 5.0, 6.0, 4.0, 8.0, 8.0, 10.0, 6.0, 3.0, 6.0, 4.0,
8.0, 5.0, 5.0, 10.0, 10.0, 9.0, 8.0, 10.0, 10.0, 10.0, 10.0, 10.0, 8.0, 8.0, 6.0, 7.0, 10.0, 5.
0, 6.0, 9.0, 10.0, 8.0, 2.0, 7.0, 8.0, 8.0, 6.0, 6.0, 9.0, 10.0, 10.0, 10.0, 10.0, 9.0, 10.0, 7.
0, 8.0, 6.0, 7.0, 8.0, 10.0, 9.0, 10.0, 8.0, 4.0, 9.0, 4.0, 5.0, 10.0, 10.0, 8.0, 8.0, 9.0, 9.0,
10.0, 8.0, 9.0, 10.0, 10.0, 10.0, 10.0, 9.0, 10.0, 10.0, 10.0, 10.0, 7.0, 7.0, 8.0, 7.0, 10.0,
6.0, 7.0, 7.0, 9.0, 9.0, 9.0, 10.0, 9.0, 10.0, 9.0, 10.0, 10.0, 5.0, 10.0, 10.0, 9.0, 7.0, 7.0,
10.0, 6.0, 10.0, 4.0, 3.0, 7.0, 8.0, 1.0, 6.0, 6.0, 8.0, 8.0, 10.0, 8.0, 9.0, 0.0, 4.0, 0.0, 5.
0, 6.0, 7.0, 5.0, 7.0, 3.0, 6.0, 5.0, 5.0, 4.0, 10.0, 7.0, 6.0, 5.0, 7.0, 8.0, 3.0, 7.0, 6.0, 6.
0, 7.0, 5.0, 6.0, 4.0, 7.0, 5.0, 7.0, 6.0, 7.0, 6.0, 4.0, 5.0, 6.0, 4.0, 7.0, 9.0, 7.0, 7.0, 6.
0, 9.0, 7.0, 10.0, 5.0, 6.0, 8.0, 8.0, 7.0, 5.0, 6.0, 8.0, 10.0, 7.0, 9.0, 6.0, 8.0, 1.0, 5.0,
5.0, 3.0, 5.0, 6.0, 7.0, 9.0, 7.0, 5.0, 9.0, 10.0, 8.0, 8.0, 8.0, 9.0, 3.0, 3.0, 7.0, 5.0, 7.0,
8.0, 7.0, 6.0, 5.0, 6.0, 7.0, 7.0, 4.0, 8.0, 8.0, 9.0, 5.0, 5.0, 7.0, 7.0, 6.0, 7.0, 7.0, 7.0,
5.0, 5.0, 6.0, 8.0, 3.0, 10.0, 8.0, -0.0, 6.0, 4.0, 7.0, 10.0, 8.0, 7.0, 9.0, 3.0, 7.0, 7.0, 6.
0, 6.0, 7.0, 5.0, 10.0, 5.0, 7.0, 6.0, 6.0, 4.0, 7.0, 6.0, 6.0, 6.0, 8.0, 5.0, 9.0, 3.0, 5.0, 4.
0, 10.0, 6.0, 7.0, 7.0, 10.0, 3.0, 6.0, 6.0, 4.0, 7.0, 7.0, 5.0, 7.0, 5.0, 7.0, 7.0, 10.0, 5.0,
4.0, 8.0, 8.0, 9.0, 9.0, 7.0, 7.0, 5.0, 5.0, 8.0, 5.0, 7.0, 8.0, 6.0, 6.0, 8.0, 6.0, 8.0, 8.0,
5.0, 7.0, 5.0, 8.0, 7.0, 5.0, 6.0, 10.0, 9.0, 9.0, 7.0, 5.0, 5.0, 5.0, 9.0, 9.0, 8.0, 7.0, 7.0,
7.0, 7.0, 8.0, 8.0, 7.0, 7.0, 6.0, 7.0, 10.0, 5.0, 10.0, 7.0, 10.0, 7.0, 7.0, 7.0, 7.0, 8.0, 5.
0, 10.0, 7.0, 8.0, 3.0, 4.0, 4.0, 9.0, 6.0, 8.0, 4.0, 6.0, 3.0, 8.0, 7.0, 5.0, 7.0, 8.0, 8.0, 7.
0, 6.0, 4.0, 5.0, 10.0, 7.0, 5.0, 5.0, 10.0, 10.0, 7.0, 6.0, 8.0, 8.0, 8.0, 7.0, 9.0, 7.0, 6.0,
9.0, 6.0, 6.0, 7.0, 7.0, 7.0, 7.0, 5.0, 0.0, 5.0, 4.0, 3.0, 9.0, 9.0, 7.0, 6.0, 9.0, 7.0, 4.0, 1
0.0, 10.0, 8.0, 9.0, 6.0, 9.0, 9.0, 4.0, 8.0, 5.0, 10.0, 6.0, 9.0, 9.0, 10.0, 6.0, 4.0, 6.0, 8.
0, 8.0, 9.0, 7.0, 10.0, 7.0, 5.0, 9.0, 8.0, 8.0, 8.0, 7.0, 10.0, 8.0, 7.0, 5.0, 8.0, 3.0, 5.0,
4.0, 8.0, 7.0, 8.0, 3.0, 6.0, 5.0, 5.0, 6.0, 10.0, 10.0, 10.0, 10.0, 10.0, 7.0, 3.0]
Optimal objective value: 79.92302652016384


[0, 0.8954257394693594, 0.8993601948415232, 0.9030850454255802]
[137, 138, 139, 140]
```

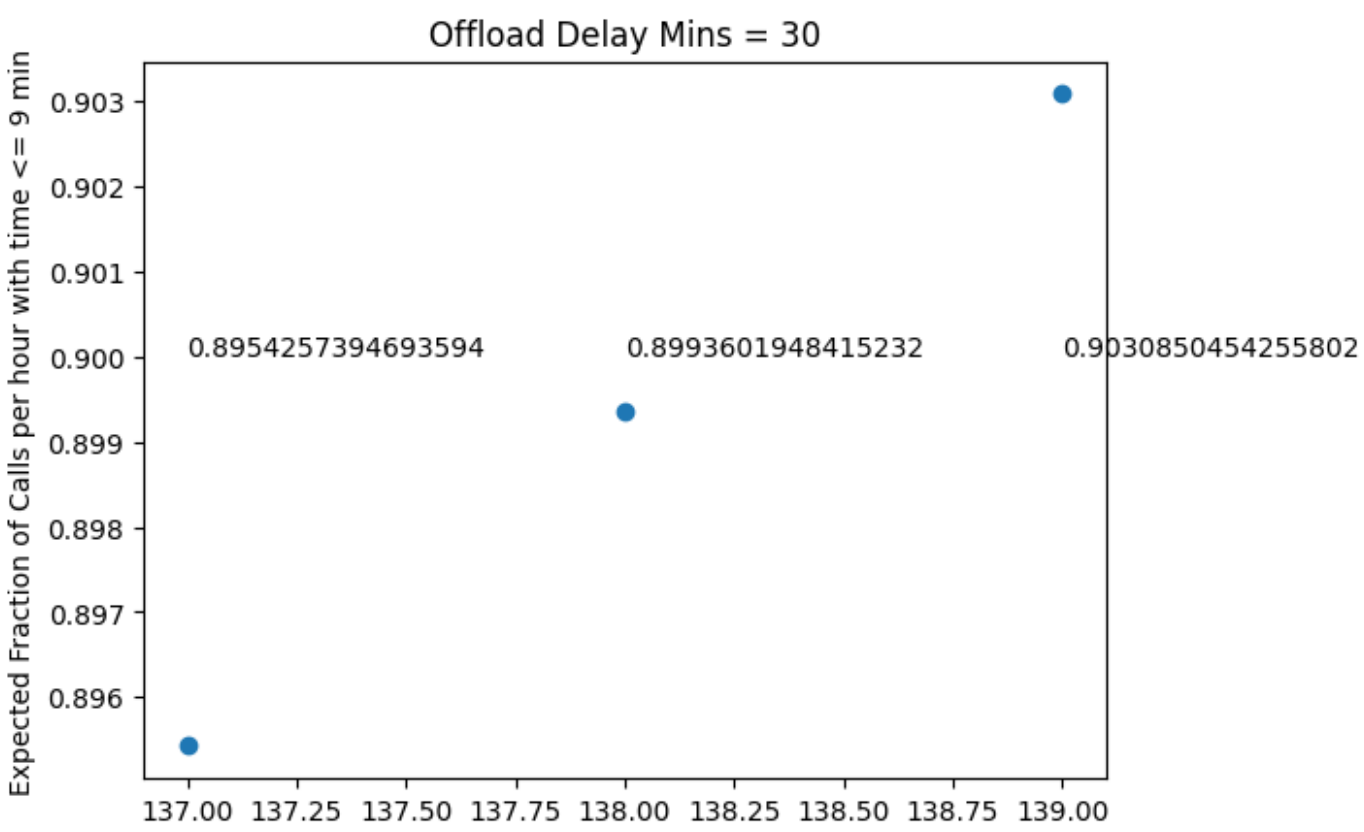
Hiding cell output because of extremely extensive optimization problem output

In [368…

```
y=optimal_val[1:]
x=a_[:-1]
plt.scatter(x,y)
plt.title(f'Offload Delay = {offload_delay}')
for i, txt in enumerate(y):
    plt.annotate(txt, (x[i], round(y[i],2)))
plt.ylabel('Expected Fraction of Calls per hour with time <= 9 min')
plt.title(f'Offload Delay Mins = {offload_delay}')
```

Out[368]:

```
Text(0.5, 1.0, 'Offload Delay Mins = 30')
```



For 30 minute offload delay, we want 129 Ambulances.

Optimal Objective Value = 79.92302652016384.

45 Minute Offload Delay

In [369…
```
offload_delay = 45
lambda_i =[(177/2)*population_with_prob['prob']][0]
mu = 1/(.7*(35/60+15/60+offload_delay/60)+0.3*(45/60))
```

In [370…
```
#find minimum value of a
#np.sum(lambda_i)/a/mu <= 1
#a >= np.sum(lambda_i)/mu
a_min = math.ceil(np.sum(lambda_i)/mu/0.75)
```

In [371…
```
# WITH AID FROM CHAT GPT IN WRITING OPTIMIZATION PROBLEM
optimal_val = [0]
a_ = [a_min]
#need smallest a such that optimal_value >0.9
while optimal_val[-1] <= 0.9:

    # Your provided data and variables
    a = a_[-1]
    m_demand = len(lambda_i)
    n_stations = alpha_ij.shape[1]
    p = np.sum(lambda_i) / a / mu
    print(p)

    # Create a 2D NumPy array for b_ij
```

```
    b_ij = np.zeros((m_demand, a))  # Initialize with zeros

    # Fill in the array
    for i in range(m_demand):
        for j in range(1, a + 1):
            b_ij[i, j-1] = lambda_i[i] * (1 - p ** j) - (lambda_i[i] * (1 - p ** (j - 1)) if j >

    # Initialize the model
    m = Model("MEXCLP")

    # Add variables
    x_j = m.addVars(n_stations, lb=0, vtype=GRB.INTEGER, name="x_j")
    y_i_k = m.addVars(m_demand, a, vtype=GRB.BINARY, name="y_i_k")  # y_i(k) is now binary
    y_i = m.addVars(m_demand, lb=0, vtype=GRB.INTEGER, name="y_i")



    # Add constraints
    # Sum of x_j is equal to a
    m.addConstr(sum(x_j[j] for j in range(n_stations)) == a, "c1")

    # For each demand location i, y_i is the sum of y_i(k) for k=1,...,a
    m.addConstrs((y_i[i] >= sum(y_i_k[i, k] for k in range(10)) for i in range(m_demand)), "c2")

    # Constraint related to alpha_ij and y_i
    m.addConstrs((sum(alpha_ij.iloc[i][j] * x_j[j] for j in range(n_stations)) >= y_i[i] for i i

    # Set the objective
    m.setObjective(sum(b_ij[i, k] * y_i_k[i, k] for i in range(m_demand) for k in range(a)), GRB

    # Optimize the model
    m.optimize()

    # Extract results
    if m.status == GRB.Status.OPTIMAL:
        x_j_values = [x_j[j].X for j in range(n_stations)]
        y_i_values = [y_i[i].X for i in range(m_demand)]
        print(f"Optimal solution: x_j = {x_j_values}, y_i = {y_i_values}")
        optimal_objective_value = m.objVal
        print(f"Optimal objective value: {optimal_objective_value}")
    else:
        print("No optimal solution found.")
    print('\n')
    optimal_val.append(optimal_objective_value/(177/2))
    a_.append(a_[-1]+1)
print(optimal_val)
print(a_)
```

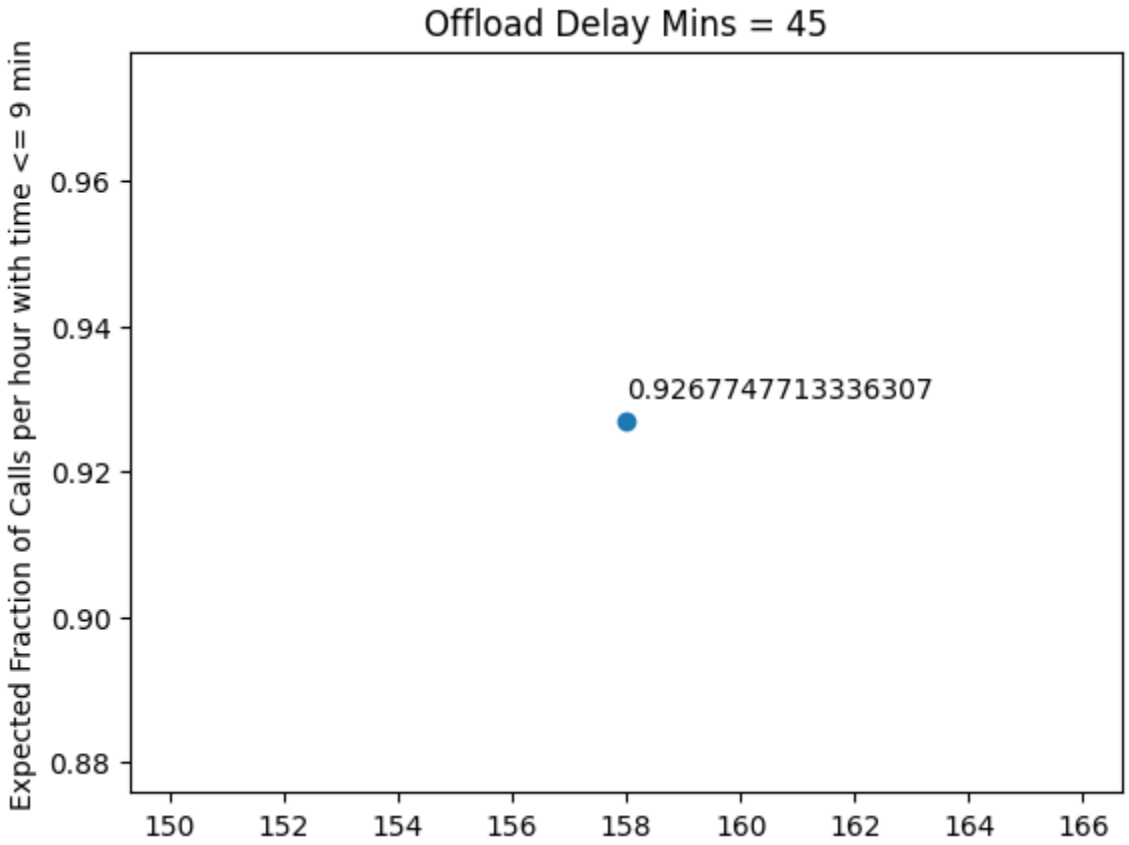
Hiding cell output because of extremely extensive optimization problem output

In [372…

```
y=optimal_val[1:]
x=a_[:-1]
plt.scatter(x,y)
plt.title(f'Offload Delay = {offload_delay}')
for i, txt in enumerate(y):
    plt.annotate(txt, (x[i], round(y[i],2)))
plt.ylabel('Expected Fraction of Calls per hour with time <= 9 min')
plt.title(f'Offload Delay Mins = {offload_delay}')
```

Out[372]:

```
Text(0.5, 1.0, 'Offload Delay Mins = 45')
```



For 45 minute offload delay, we want 158 Ambulances.

Optimal Objective Value = 82.01956726302632.

60 minute offload delay

In [373…

```
offload_delay = 60
lambda_i =[(177/2)*population_with_prob['prob']][0]
mu = 1/(.7*(35/60+15/60+offload_delay/60)+0.3*(45/60))
```

In [374…

```
#find minimum value of a
sum(lambda_i)/a/mu <= 1
```

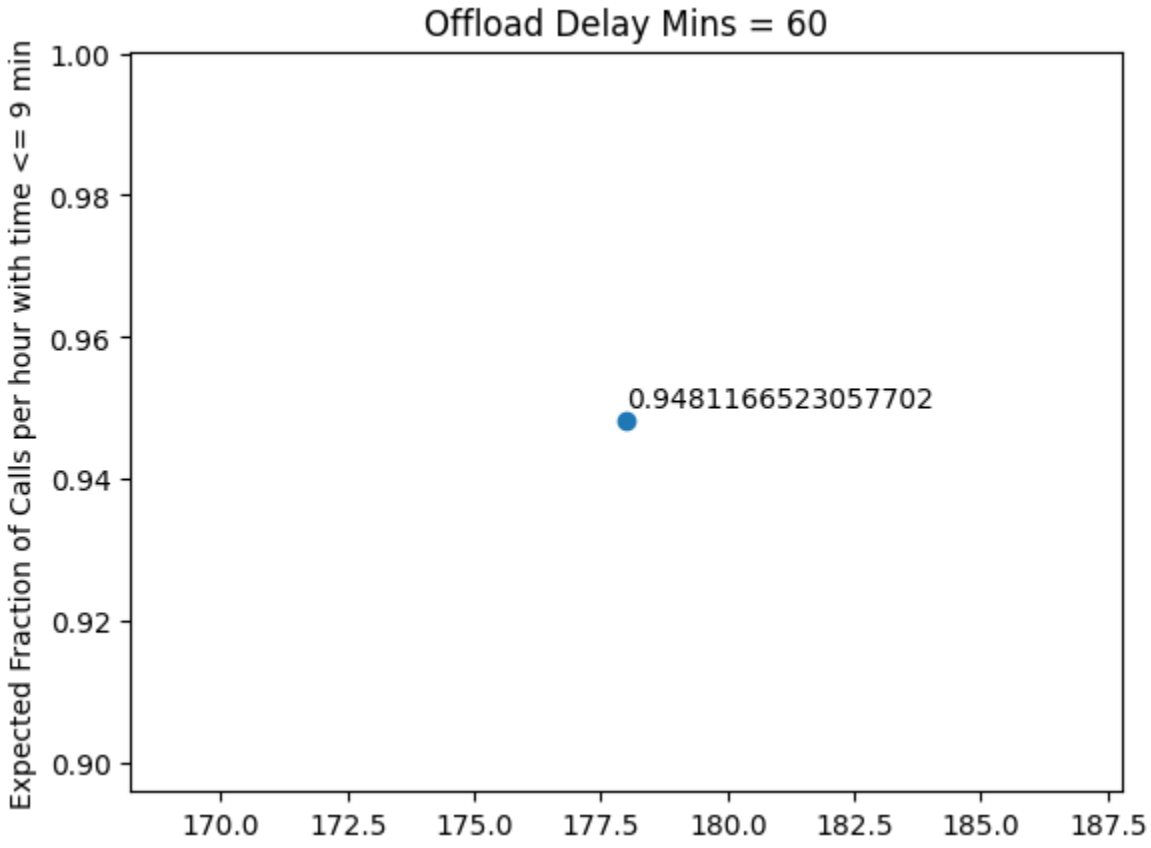
```
#a >= np.sum(lambda_i)/mu
a_min = math.ceil(np.sum(lambda_i)/mu/0.75)
```

In [375…

```
# WITH AID FROM CHAT GPT IN WRITING OPTIMIZATION PROBLEM
optimal_val = [0]
a_ = [a_min]
#need smallest a such that optimal_value >0.9
while optimal_val[-1] <= 0.9:

    # Your provided data and variables
    a = a_[-1]
    m_demand = len(lambda_i)
    n_stations = alpha_ij.shape[1]
    p = np.sum(lambda_i) / a / mu
    print(p)

    # Create a 2D NumPy array for b_ij
    b_ij = np.zeros((m_demand, a))  # Initialize with zeros

    # Fill in the array
    for i in range(m_demand):
        for j in range(1, a + 1):
            b_ij[i, j-1] = lambda_i[i] * (1 - p ** j) - (lambda_i[i] * (1 - p ** (j - 1)) if j >

    # Initialize the model
    m = Model("MEXCLP")

    # Add variables
    x_j = m.addVars(n_stations, lb=0, vtype=GRB.INTEGER, name="x_j")
    y_i_k = m.addVars(m_demand, a, vtype=GRB.BINARY, name="y_i_k")  # y_i(k) is now binary
    y_i = m.addVars(m_demand, lb=0, vtype=GRB.INTEGER, name="y_i")



    # Add constraints
    # Sum of x_j is equal to a
    m.addConstr(sum(x_j[j] for j in range(n_stations)) == a, "c1")

    # For each demand location i, y_i is the sum of y_i(k) for k=1,...,a
    m.addConstrs((y_i[i] >= sum(y_i_k[i, k] for k in range(10)) for i in range(m_demand)), "c2")

    # Constraint related to alpha_ij and y_i
    m.addConstrs((sum(alpha_ij.iloc[i][j] * x_j[j] for j in range(n_stations)) >= y_i[i] for i i

    # Set the objective
    m.setObjective(sum(b_ij[i, k] * y_i_k[i, k] for i in range(m_demand) for k in range(a)), GRB

    # Optimize the model
    m.optimize()

    # Extract results
    if m.status == GRB.Status.OPTIMAL:
        x_j_values = [x_j[j].X for j in range(n_stations)]
        y_i_values = [y_i[i].X for i in range(m_demand)]
        print(f"Optimal solution: x_j = {x_j_values}, y_i = {y_i_values}")
        optimal_objective_value = m.objVal
        print(f"Optimal objective value: {optimal_objective_value}")
    else:
        print("No optimal solution found.")
    print('\n')
    optimal_val.append(optimal_objective_value/(177/2))
    a_.append(a_[-1]+1)
    print(optimal_val)
    print(a_)
```

```
for i, txt in enumerate(y):
    plt.annotate(txt, (x[i], round(y[i],2)))
plt.ylabel('Expected Fraction of Calls per hour with time <= 9 min')
plt.title(f'Offload Delay Mins = {offload_delay}')
```

Out[376]: Text(0.5, 1.0, 'Offload Delay Mins = 60')



For 60 minute offload delay, we want 178 Ambulances.

Optimal Objective Value = 83.90832372906067.

Offload Delay | A

15 mins = 126 Ambulances

30 mins = 139 Ambulances

45 mins = 158 Ambulances

60 mins = 178 Ambulances

PART 2: Compliance Table

1. give a compliance table showing where ambulances should be placed as a function of the number of free ambulances
   - 30 minute offload delays only
2. justify your solution.

- In your appendix you should provide your compliance table (if you recommend, say, 140 ambulances, then this is just a list of 140 locations (nodes), starting from the location where you would place a single ambulance if only 1 ambulance is free, up to the location where you would place the last ambulance if 140 ambulances are free).
- Within the 6 pages of your report you should also provide a graphical depiction of your compliance table – to do that, plot a map of NYC with the first 30 locations in your compliance table colored red, the next 30 locations colored orange, the next 30 locations colored yellow, and so on.
- Your answer to #3 should be sufficient to allow another student to recreate your results. Within the 6 pages of your report you should give the mathematical formulations (like we did in class) for any optimization methods you use and cite any other sources you use.

Idea to solve

Solve MEXCLP for most probable number of ambulances. Then solve to 0 and up to a 30 min offload delay -> a=124 Ambulances

Expected number of ambulances free is a(1-p)

In [425…
```
distances = pd.DataFrame(distances)
alpha_ij = distances<(5*60)
```

In [471…
```
offload_delay = 30
a_value=a_30
lambda_i =[(177/2)*population_with_prob['prob']][0]
mu = 1/(.7*(35/60+15/60+offload_delay/60)+0.3*(45/60))
p = np.sum(lambda_i) / (a_value * mu)
print('p = Probability Busy = ',p)
print('(1-p) = Probability Free = ',1-p)
print('Number of Ambulances',a_value)
m_demand = len(lambda_i)
n_stations = alpha_ij.shape[1]
r = a_value*(1-p)
print('Start Calculations',math.ceil(r))
start = math.ceil(r)
```

```
p = Probability Busy =  0.7374999999999999
(1-p) = Probability Free =  0.26250000000000007
Number of Ambulances 139
Start Calculations 37
```

In [412…
```
ys = np.zeros((a_value,1163))
xs = np.zeros((a_value,1163))
opts = np.zeros(a_value)
```

```
TH AID FROM CHAT GPT IN WRITING OPTIMIZATION PROBLEM
```

```
prev_x_values = np.zeros(n_stations)  # Initialize array to store the last found values of x

# Loop from a = 16 to a = 124

for a in range(start, a_value+1):

    # Create a 2D NumPy array for b_ij
    b_ij = np.zeros((m_demand, a))  # Initialize with zeros

    # Fill in the array
    for i in range(m_demand):
        for j in range(1, a + 1):
            b_ij[i, j-1] = lambda_i[i] * (1 - p ** j) - (lambda_i[i] * (1 - p ** (j - 1)) if j >

    # Create new model
    m = Model("MEXCLP")

    # Add variables
    x_j = m.addVars(n_stations, lb=0, vtype=GRB.INTEGER, name="x_j")
    y_i_k = m.addVars(m_demand, a, vtype=GRB.BINARY, name="y_i_k")
    y_i = m.addVars(m_demand, lb=0, vtype=GRB.INTEGER, name="y_i")

    # Add constraints
    m.addConstr(sum(x_j[j] for j in range(n_stations)) == a, "c1")
    m.addConstrs((y_i[i] >= sum(y_i_k[i, k] for k in range(10)) for i in range(m_demand)), "c2")
    m.addConstrs((sum(alpha_ij.iloc[i][j] * x_j[j] for j in range(n_stations)) >= y_i[i] for i i

    # Add constraints based on previous x values
    for j in range(n_stations):
        m.addConstr(x_j[j] >= prev_x_values[j], f"prev_x_{j}")

    # Set the objective
    m.setObjective(sum(b_ij[i, k] * y_i_k[i, k] for i in range(m_demand) for k in range(a)), GRB

    # Optimize
    m.optimize()

    # Extract and store results
    if m.status == GRB.Status.OPTIMAL:
        x_j_values = [x_j[j].X for j in range(n_stations)]
        xs.loc[a-1] = x_j_values
        prev_x_values = np.array(x_j_values)  # Update prev_x_values for the next iteration
        print(f"For a={a}, Optimal solution: x_j = {x_j_values}")
        y_i_values = [y_i[i].X for i in range(m_demand)]
        #ys.loc[a-1] = y_i_values
        optimal_objective_value = m.objVal
        #opts.loc[a-1] = optimal_objective_value
        print(f"Optimal objective value: {optimal_objective_value}")
    else:
        print(f"For a={a}, no optimal solution found.")
```

0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.
0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.
0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, 2.0, -0.0, -0.
0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, 2.0, -0.0, -0.
0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.
0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, 1.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.
0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.
0, -0.0, -0.0, 1.0, 2.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, 1.0, -0.0,
-0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, 3.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -
0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -
0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -
0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -
0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, 1.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -
0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, 1.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -
0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, 1.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, 4.
0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, 1.
0, -0.0, 1.0, -0.0, -0.0, -0.0, -0.0, -0.0, 3.0, -0.0, -0.0, -0.0, 2.0, -0.0, -0.0, -0.0, -0.0,
-0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0,
-0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0,
-0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, 2.0, -0.0, -0.0, -0.0, 1.0, -0.0, -0.0, -
0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -
0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, 2.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -
0.0, -0.0, 1.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, 1.0, -0.
0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.
0, -0.0, -0.0, -0.0, 1.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.
0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, 2.0, 1.0, -0.0, -0.0, -0.0, -0.0,
-0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0,
-0.0, -0.0, -0.0, 1.0, 0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, 3.0, -0.0, -0.0, -0.0, -0.0, -0.
0, -0.0, -0.0, -0.0, -0.0, -0.0, 3.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.
0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, 1.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.
0, -0.0, 1.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, 1.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0,
-0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, 1.0, -0.0, -0.0, -0.0, -0.0, -0.0, -
0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, 1.0, -0.0, -0.0, -0.0, -0.0, -
0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, 1.0, -
0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -
0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, 2.0, -0.0, -0.0, -0.0, -0.0, 1.
0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.
0, -0.0, -0.0, -0.0, 3.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.
0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, 2.0, -0.0, -0.0, -0.0, -0.0, -0.0, 2.0,
-0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, 1.0, 1.0, -0.0, 1.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.
0, -0.0, -0.0, -0.0, -0.0, -0.0, 1.0, -0.0, 1.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, 1.0,
-0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, 2.0, -0.0, -0.0, -0.0, -0.0, -
0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -
0.0, 2.0, -0.0, -0.0, 1.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.
0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, 1.0, 1.0, -0.0, -0.0, -0.0,
-0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, 1.0, -0.0, -0.0, -0.0, -0.0,
1.0, -0.0, -0.0, -0.0, -0.0]
Optimal objective value: 79.8551806306155

Hiding cell output because of extremely extensive optimization problem output

In [486…

```
# Loop from a = 1 to a = 37

# prev_x_values = xs[start-1]
# for a in range(start-1, 0, -1):

prev_x_values = xs.loc[7]
for a in range(7,0,-1):

    # Create a 2D NumPy array for b_ij
    b_ij = np.zeros((m_demand, a))  # Initialize with zeros
```

```
    # Fill in the array
    for i in range(m_demand):
        for j in range(1, a + 1):
            b_ij[i, j-1] = lambda_i[i] * (1 - p ** j) - (lambda_i[i] * (1 - p ** (j - 1)) if j >

    # Create new model
    m = Model("MEXCLP")

    # Add variables
    x_j = m.addVars(n_stations, lb=0, vtype=GRB.INTEGER, name="x_j")
    y_i_k = m.addVars(m_demand, a, vtype=GRB.BINARY, name="y_i_k")
    y_i = m.addVars(m_demand, lb=0, vtype=GRB.INTEGER, name="y_i")

    # Add constraints
    m.addConstr(sum(x_j[j] for j in range(n_stations)) == a, "c1")
    m.addConstrs((y_i[i] >= sum(y_i_k[i, k] for k in range(a)) for i in range(m_demand)), "c2")
    m.addConstrs((sum(alpha_ij.iloc[i][j] * x_j[j] for j in range(n_stations)) >= y_i[i] for i i

    # Add constraints based on previous x values
    for j in range(n_stations):
        if prev_x_values[j] == 0:
            m.addConstr(x_j[j] == 0,f"prev_x_{j}")
        else:
            m.addConstr(x_j[j] <= prev_x_values[j], f"prev_x_{j}")

    # Set the objective
    m.setObjective(sum(b_ij[i, k] * y_i_k[i, k] for i in range(m_demand) for k in range(a)), GRB

    # Optimize
    m.optimize()

    # Extract and store results
    if m.status == GRB.Status.OPTIMAL:
        x_j_values = [x_j[j].X for j in range(n_stations)]
        xs.loc[a-1] = x_j_values
        prev_x_values = np.array(x_j_values)  # Update prev_x_values for the next iteration
        print(f"For a={a}, Optimal solution: x_j = {x_j_values}")
        y_i_values = [y_i[i].X for i in range(m_demand)]
        #ys.loc[a-1] = y_i_values
        optimal_objective_value = m.objVal
        #opts.loc[a-1] = optimal_objective_value
        print(f"Optimal objective value: {optimal_objective_value}")
    else:
        print(f"For a={a}, no optimal solution found.")
```

```
0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.
0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
Optimal objective value: 2.5118024451110124
```

Hiding cell output because of extremely extensive optimization problem output

In [487…

```
xs = pd.DataFrame(xs)
xs.to_csv('xs_final.csv', index=True)
```

In [483…

```
sum(xs.loc[5])
```

Out[483]:

```
0.0
```

In [ ]:

```
xs
```

In [ ]: