The Dynamics of Bike Sharing: A Data-Driven Approach to Citi Bike Fleet Management in NYC

Julia VanPutte (JHV42)

November 5, 2023

Abstract

Bike-sharing systems are very common amongst large cities in the United States. Optimizing the distribution and availability of bicycles in such cities is critical for the efficiency of bike-sharing systems. This study will determine the optimal fleet size and re-balancing strategy for Citi Bike, New York City's largest bike-sharing service, by analyzing usage data from July 2023. First, the network's spatial dynamics were analyzed to understand active stations and their geographic location. Utilizing Poisson processes, the hourly arrival rates of riders initiating and ending bike rides at each station on weekdays were calculated. Transition probabilities between stations were computed to understand riders' flow within the network, forming the basis for calculating the return rates of bikes to each station.

To minimize service outages, the system was modeled using a fluid dynamic model, capturing the hourly fluctuations in bike demand and supply at individual stations. By leveraging the calculated arrival and transition rates, the optimal number of bikes required at each station by 5 am were found. These numbers were optimized to reduce service disruptions, or "unhappiness". Additionally, I computed the net flow of bikes across stations to determine the overnight rebalancing needs.

The findings predict the optimal Citi Bike fleet size and provide a strategic framework for nightly bike redistribution to maintain system equilibrium. Further, the potential impact of adding or removing a bike at 2 pm on service outage reduction was quantified, to guide a point allocation system, Bike Angels.

Finally, the model's limitations are discussed, including assumptions of uniform demand patterns, lack of rerouting consideration, and potential data inaccuracies, proposing future improvements to refine the system's operational efficiency.

1 Introduction

As the most densely populated city in the United States, New York City presents a unique set of challenges for urban mobility. Citi Bike, NYC's largest bike-sharing program, serves as a crucial component of this urban transit network. However, the operational efficiency of such a system depends on the strategic distribution and availability of bikes throughout the city. Inadequate bike supply at high-demand stations or a surplus where demand is low can lead to decreased usage and rider satisfaction.

In this report, by developing a model to optimize the fleet size and the nightly redistribution of bikes, the reliability of the bike-sharing system is improved. A critical aspect of analysis is understanding the flow of bikes between stations which influences the availability of bikes and docking points citywide. Further, an analysis of Bike Angels, Citi Bike's crowd-sourced approach to balance bike distribution across the city, was conducted. The model was extended to Bike Angels to determine optimal point allocations at each station. In this way, Citi Bike can optimally leverage community engagement to enhance the availability of bikes and decrease the amount of overnight re-balancing needed.

The goal was to determine the optimal number of bikes in the fleet and the most efficient redistribution strategy to minimize the occurrence of bike shortages, resulting in an unsuccessful pickup, and dock shortages, resulting in an unsuccessful return. The analysis offers insights into enhancing the functionality of Citi Bike, supporting the urban mobility of New York City.

2 Analysis

The overarching goal of the analysis was to determine the optimal fleet size, the optimal bike allocation at 5am, the necessitated overnight re-balancing, and optimal point values for Bike Angels community re-balancing.

2.1 Preliminary Data Cleaning

The initial phase involved reparation of the dataset acquired from Citi Bike, specifically the July 2023 usage data for New York City [Cit23]. This dataset contained all rides taken in July 2023, noting station locations of start and end. In order to ensure the accuracy of further analysis, the dataset was filtered to exclude entries with missing start or end station identifiers. Further, data was acquired for the capacity of each station from the ORIE 4130 Canvas site. Then, the information regarding station capacity was combined with the respective start and end points of each ride.

During the data cleaning process, approximately 1.8% of rides were excluded due to lack of capacity information. A plot of the missing stations is shown below, with all stations included in blue and the missing stations in red. A hypothesis for the lack

of capacity for these stations is these stations were newly added to the Citi bike network and not yet reflected in the capacity data sourced from ORIE 4130 Canvas Site.

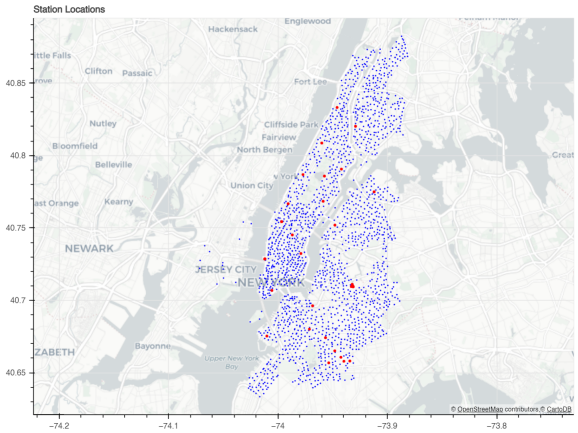


Figure 1: Plot of NYC CitiBike Stations. Missing stations are shown in red

Further, null values in the end station latitude and longitude were replaced with the stations' corresponding start latitude and longitude. A complete list of the 1921 stations and their locations was then generated from the usage data.

2.2 Estimating Hourly Bike Departure Rates for Citi Bike Stations

The arrival rates of Citi Bike users at each station throughout a typical weekday were quantified. This is equal to the rate of bike departure from the station. The underlying assumption here is that bike rentals at each station follow a Poisson process, with a constant rate of bike pickups during each hour from 5 am to midnight, excluding weekends. It is assumed that the number of rides from midnight to 5 am is negligible, and these times were ignored.

Utilizing the July 2023 Citi Bike trip data, the average hourly arrival rates were calculated, denoted as $\mu_t(i)$, for each station i, where t ranges from 5 to 23, corresponding to the hours from 5 am to 11 pm. This resulted in 19 distinct values for each station, representing the expected number of rides initiated per hour. To get these values, I found the number of trips starting at station i for each hour and then divided by 20, which is the number of weekdays in a month.

The computation involved grouping the dataset by week number, hour, and station ID, followed by aggregating the counts of rides initiated during each hour for each station. These counts were then averaged to estimate the constant hourly rates.

The estimated arrival rates are depicted graphically below, showcasing the distribution of bike demand across the city as the day progresses.

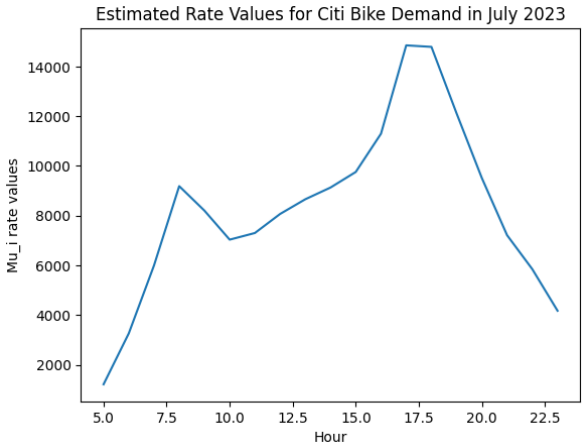


Figure 2: Estimated Bike Departure Rates per Hour, Averaged over All Stations

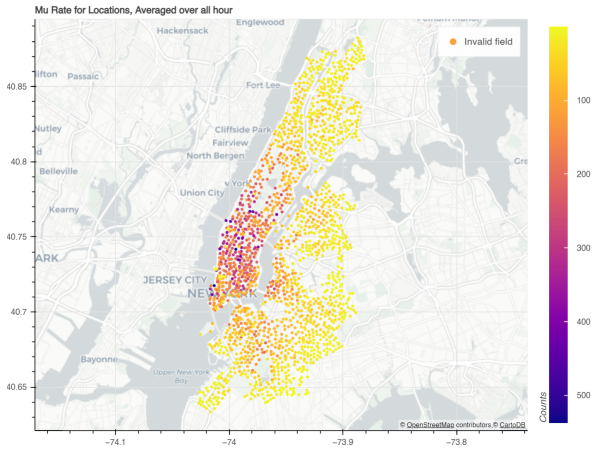


Figure 3: Estimated Bike Departure Rates for Each Station, Averaged over Hours from 5 am to 11 pm

2.3 Transition Probabilities and Bike Arrival Rates for Citi Bike Stations

Then, I estimated transition probabilities for rides between Citi Bike stations and the subsequent arrival rates of riders returning bikes to each station. Transition probabilities (Probability that the destination station is j given a ride leaves Station i in hour

t), denoted as $p_t(i, j)$, represent the likelihood of a bike being rented from station i and returned to station j during hour t on weekdays. From the data, this is just the fraction of rides beginning in Station i in time t that go to Station j.

To estimate these probabilities, I analyzed the July 2023 trip data to count transitions from each station i to station j for each hour t. I then divided these counts by the total number of departures from station i during hour t to obtain the transition probabilities. For hours where no rides originated from a particular station, I adhered to the convention of setting $p_t(i, i) = 1$ and $p_t(i, j) = 0$ for all $j \neq i$.

Subsequently, I computed the hourly arrival rates of riders returning bikes to each station. It was again assumed that these rates were constant for each hour and a Poisson Process. These rates, $\lambda_t(i)$, were calculated for each station i from 5 am to midnight on weekdays. To ensure an accurate reflection of actual bike returns, I derived these values from the previously calculated transition probabilities combined with the average hourly departure rates, $\mu_t(i)$, for each station. This integration allowed me to estimate the influx of bikes to each station throughout a typical weekday.

The resulting transition matrices and arrival rates provide a dynamic view of bike flow throughout the city. Graphical representations, including plots and network visualizations, are shown below to illustrate the distribution of these rates across different times and locations.

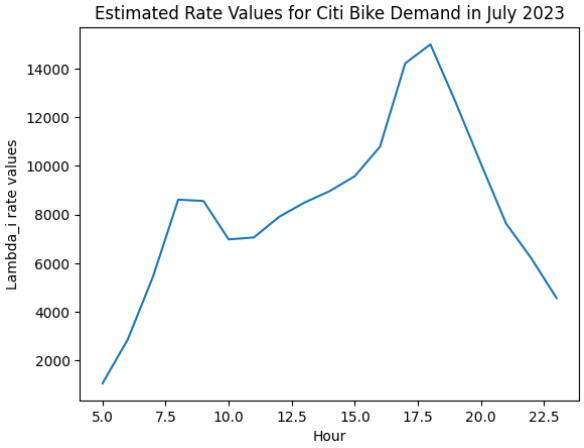


Figure 4: Estimated Bike Arrival Rates per Hour, Averaged over All Stations

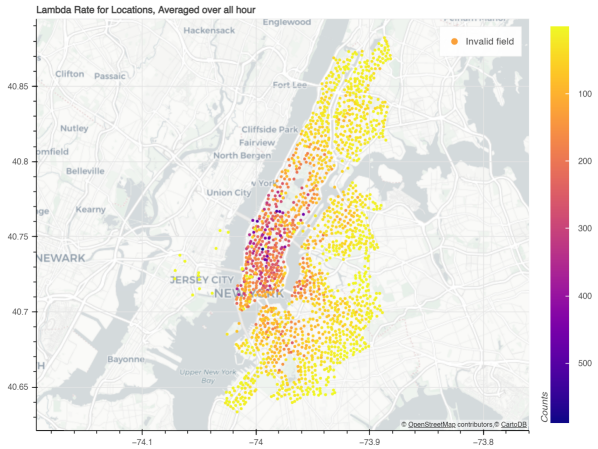


Figure 5: Estimated Bike Arrival Rates for Each Station, Averaged over Hours from 5 am to 11 pm

It was also necessary to normalize the Mu and Lambda rates to maintain a balanced state. The total number of bike takeaways (departures) and the number of bike returns (arrivals) differed by about 6,000. This was due to the data cleaning at the beginning, where I dropped NaNs and missing stations. This meant more bikes were taken away than returned, leading to a deficit of bikes. To correct this, the arrival rates were normalized so that their sum was equal to the departure rates. This ensures that the net flow is zero. This normalization process is critical for modeling a bike-sharing system where the number of bikes available for users is maintained throughout the day.

2.4 Modeling the Optimal Fleet Size and Locations at 5am

In order to determine the optimal fleet size, I modeled the system as a continuous fluid model at each station, with fixed departure and arrival rates each hour. I focused on optimizing the number of bikes available at each station by 5 AM to minimize the possibility of both failed bike pickups (demand exceeds supply) and failed returns (capacity is exceeded by returns). This optimization takes into account the fluid dynamics of bike movement between stations throughout the day.

The process began by establishing a predictive model that uses the known rates of bike departures ($\mu_i(t)$) and arrivals ($\lambda_i(t)$) at each station i per hour t. The station's capacity (c_i) is considered the upper bound for the number of bikes it can hold at any given time. The model aims to minimize the cumulative unhappiness ($y_i(t)$), which arises from failed pickups and returns at every station during each hour of operation, from 5 AM to 11 PM.

The cumulative unhappiness is calculated using the following formula:

angry bikers(t) = failed returns + failed pickups = $y_i(t)$ = unhappy people at the end of period t. $x_i(t)$ = number of bikes at time t. So, $x_i(t+1) = x_i(t) + (\lambda_i(t) - \mu_i(t)$ (pinned to $[0, c_i]$

$y_i(t)$ = failed returns + failed pickups = $\max(x_i(t) + (\lambda_i(t) - \mu_i(t)) - c_i, 0) + \max(-(x_i(t) + (\lambda_i(t) - \mu_i(t))), 0)$

This formula calculates the number of failed returns and pickups for each station i and each hour t based on the difference between the number of bikes at a station after considering arrivals and departures ($x_i(t+1) = x_i(t) + (\lambda_i(t) - \mu_i(t))$) and the station's capacity, constrained between 0 and c_i.

The optimization problem is then: $\min_x \sum_{i \in \text{stations}} \sum_{t=5\text{AM}}^{11\text{PM}} y_i(t)$

This problem was solved for each station independently by iterating over all possible numbers of bikes that could be present at the station at 5 AM and calculating the cumulative unhappiness throughout the day. The optimal number of bikes for each

station is the number that results in the least cumulative unhappiness. The optimal initial count for each station was determined by finding the count that minimizes the cumulative unhappiness. Finally, the fleet size was calculated as the sum of the optimal number of bikes for all stations.

The recommendation for the fleet size is 30,720 bikes. This represents the total count of bikes that should be distributed across the network by 5 AM to minimize service outages during a typical weekday. Below, is a graph of bike location at 5 am for optimality. With this allocation, the number of failed pickups and failed returns, which determines the total user satisfaction with the Citi Bike system, is minimized.

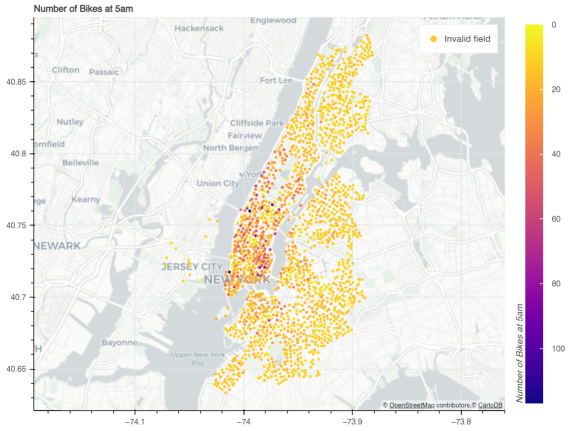


Figure 6: Plot of NYC CitiBike Stations and Number of Bikes for Optimal System at 5am. Darker colors indicate more bikes.

2.5 Dynamic Distribution of Bike Fleet Across NYC Citi Bike Stations

To find the distribution of bikes throughout the day, I calculated the dynamic number of bikes at each station over the course of the day, starting with the optimal number of bikes at 5 AM. The net change in the number of bikes (x_t) at each station for every subsequent hour is computed by accounting for the rate of bikes taken (μ_i) and returned (λ_i) at each station. These values are clamped between zero and the station's capacity (c_i), representing the physical limit of the station's docks.

For each station, the calculated bike numbers and the total unhappiness for the station—defined as the sum of failed returns and pickups—are stored. This unhappiness score is a measure of service quality, with a lower score indicating fewer service disruptions.

I also acknowledge a model limitation: due to assumptions that bikes can disappear or suddenly appear (ignoring potential rerouting to other stations when service disruptions occur), the day can end with a different number of bikes than it started with. To address this, normalization is applied to the bike numbers for each hour so that the total number of bikes across all stations at the end of the day matches the total at the start of the day (30,720), ensuring conservation of flow. Detailed visualization of this data throughout the day is included in the appendix. These processes underscore the importance of considering not just the initial conditions but also the flow and distribution of bikes throughout the entire day to ensure optimal service and user satisfaction.

2.6 Predicted Bike Fleet Outages and System Performance Evaluation

In a well-calibrated bike-sharing system, ensuring that bikes are available when and where they are needed is crucial for customer satisfaction and operational success. Here I evaluate the predicted number of outages on a typical weekday within the Citi Bike system, presuming an ideal bike distribution each morning. Outages, in this context, refer to instances where either bikes or docking points are unavailable, leading to user dissatisfaction.

The analysis employs the fleet optimization model to estimate the expected daily outages and contrasts this with the average daily ride count. By deploying a color-coded visualization in Figure 7, I illustrate spatial unhappiness levels across the network's nodes, highlighting stations with higher predicted outages. This visual tool not only identifies problem areas but also aids in strategizing redistribution efforts.

The total predicted outages daily is 2928. Contextualizing this against the average number of rides taken in a typical weekday, 141,054, I see that approximately 2% of rides result in unhappiness. This percentage gives a clear indication of the efficiency of the proposed bike distribution strategy and the reliability of the service from a user's perspective.

2.7 Optimizing Overnight Bike Redistribution

The logistical challenge of bike-sharing systems often lies in the redistribution of bikes to match the dynamic demand patterns of users. The model outputs the number of bikes at each station at the end of a typical weekday (11 pm), which is then compared to

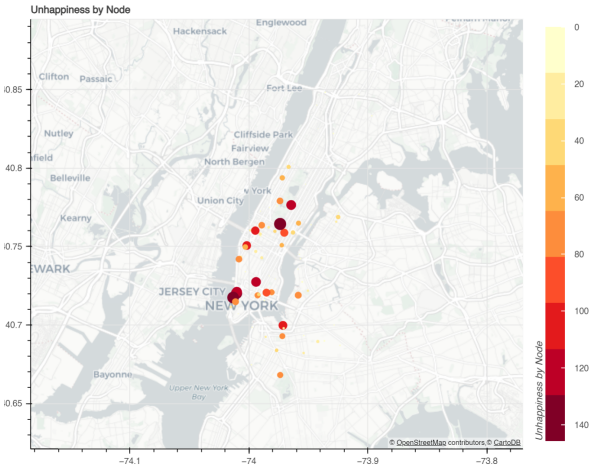


Figure 7: Plot of NYC CitiBike Stations and Number of Daily Outages for Optimal System at 5 am. Darker colors and Larger Sizes indicate more unhappiness.

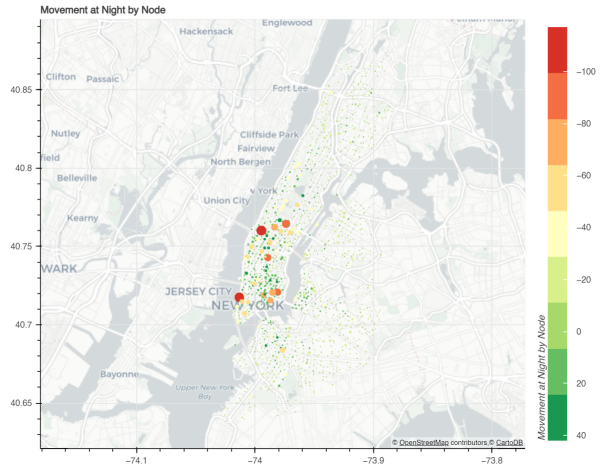


Figure 8: Plot of NYC CitiBike Stations and Number of Overnight Bikes Moved for Optimal System at 5 am. Darker Colors and Larger Sizes indicate more movement. Stations with surplus bikes are red and those with deficits are green

the optimal number of bikes needed at 5 am. The difference between these two figures represents the net movement required per station to reset the system for the next day's operations. This calculation is crucial for the planning of redistribution operations, involving trucks and personnel to relocate bikes physically.

A visual representation maps out these movements across the network in Figure 8, using a color gradient to indicate stations with surplus bikes (red) and those with deficits (green). The intensity of the color reflects the magnitude of movement: brighter colors signify larger numbers of bikes to be moved. This heatmap serves as a strategic tool for efficient routing and allocation of redistribution resources.

The aggregate of absolute movements across all stations gives the total number of bikes that need to be moved overnight, which is 10,438. This helps to assess the operational demands on the redistribution team and can be used to forecast logistical requirements such as vehicle capacity and staffing levels.

In conclusion, the total movement calculated provides a quantifiable measure of the operational workload, enabling the management to ensure that each morning starts with the ideal bike placement, thus minimizing potential service outages and maintaining high customer satisfaction.

2.8 Setting Bike Angels Points at 2pm

Bike-sharing systems often use incentive programs like Bike Angels to balance the distribution of their fleet. Bike Angels is an incentive program created by Citi Bike to help balance the distribution of bikes across their network of docking stations. By doing so, Bike Angels contributes to a more dynamic and responsive bike-share system, especially during peak hours when the demand for bikes can cause significant imbalances in bike availability. In this scenario, the Bike Angels program awards points based on the impact of adding or removing a bike from a station at 2pm on a weekday, with the goal of reducing outages until the end of the day.

The analysis begins by assuming all stations are at half capacity at 2pm. The model then simulates the effect of adding or removing a bike at this time and measures the potential reduction in outages for the rest of the day. The resulting data shows whether adding or removing a bike would yield a better improvement in system performance for each station.

Graphically, each station is represented by a colored dot on a map:

This color-coded map provides a clear visual guide to optimizing bike placement for the Bike Angels program. By assigning point values to actions based on their color, Bike Angels can incentivize users to move bikes from red to blue stations, thereby enhancing system reliability and user experience. Here, if a station is red, the user would get a point for removing a bike. If the station is blue, the user would get a point for adding a bike.

This approach could be directly translated into a dynamic point allocation system for Bike Angels, where points vary in real time based on the current distribution of bikes and predicted demand. This would encourage user participation in re-balancing efforts and minimize the manual overnight re-balancing needed by Citi Bike, ultimately leading to a more efficient and reliable bike-sharing service.

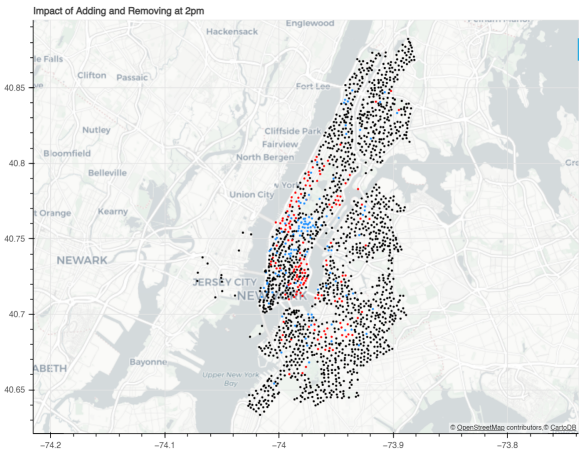


Figure 9: Bike Angels Points. Black indicates that changing the number of bikes has no significant impact on reducing outages. Blue suggests that adding a bike will most effectively reduce outages. Red signifies that removing a bike is the preferable action.

2.9 Limitations of the Model and Computation

While the proposed model provides valuable insights into the optimization of the Citi Bike fleet, there are limitations stemming from modeling assumptions, data constraints, and computational simplifications. Understanding these limitations is essential for interpreting the results and for guiding future improvements.

Elimination of Data One significant limitation of the model is the elimination of data, particularly the exclusion of weekends and the time frame from midnight to 5 am. This was predicated on the assumption that bike usage during these periods is negligible. However, this may not accurately reflect the true demand Ignoring these times could lead to an underestimation of the fleet size needed to accommodate late-night or early-morning riders, as well as weekend usage patterns. A more robust approach would involve incorporating 24/7 data to capture the complete demand cycle.

Uniform Demand Patterns The model assumes uniform demand patterns across different weekdays, which fails to account for variations in ridership due to factors such as weather, special events, or seasonal changes. This assumption could be relaxed by including more granular data and statistical methods to account for these variables, thus providing a more accurate representation of the stochastic nature of demand.

Lack of Rerouting Consideration Another limitation is the model's disregard for potential rerouting behaviors by users in response to bike or dock shortages. In reality, users may choose to go to a different station if their initial choice is unavailable, which can affect the overall system dynamics. Future models could incorporate user behavior models to simulate and predict such decisions, which would enable a more dynamic response to varying system states.

Demand Censoring Our model has a crucial limitation due to demand censoring present in the dataset. Demand censoring refers to situations where people intended to use the service, but due to the unavailability of bikes or docks, they resorted to other means of transportation. Unfortunately, the Citi Bike data used for July does not capture such instances, leading to an under-representation of demand and an underestimation of service usage. This type of censoring obscures the true demand levels and can result in inaccurate estimations of parameters such as station capacity needs and arrival rates. However, due to Citi Bike's data privacy, overcoming this censoring is not possible here. To enhance the accuracy of our model, it would be useful to have access to comprehensive data, including unsuccessful bike rental attempts. Nonetheless, obtaining this data while also adhering to privacy standards is subject to availability.

3 Conclusion

In conclusion, this study represents a comprehensive approach to optimizing the Citi Bike system in New York City, synthesizing complex data into actionable insights for fleet management and operational efficiency. By leveraging historical usage patterns and spatial dynamics, the research delineates an ideal fleet size and a strategic redistribution model that minimizes service outages and enhances user satisfaction. The proposed model, while robust, acknowledges intrinsic limitations such as demand censoring, static demand assumptions, and potential data inaccuracies. Future work could incorporate real-time data analytics and more nuanced user behavior models to refine the system further. Nonetheless, the findings lay a solid foundation for Citi Bike to operate a more reliable and user-centric bike-sharing program, contributing positively to the fabric of urban mobility in NYC.

References

[Cit23] CitiBike. 202307-citibike-tripdata.csv.zip. Jul 2023.

[FHOS19] Daniel Freund, Shane G. Henderson, Eoin O'Mahony, and David B. Shmoys. Analytics and bikes: Riding tandem with motivate to improve mobility. *INFORMS J. Appl. Anal.*, 49:310–323, 2019.

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

[JFWH16] Nanjing Jian, Daniel Freund, Holly M. Wiberg, and Shane G. Henderson. Simulation optimization for a large-scale bike-sharing system. In *2016 Winter Simulation Conference (WSC)*, pages 602–613, 2016.

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

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

Ope23 Hen23 Wik22 JFWH16 FHOS19 Cit23

In [1]:

```
!pip install --upgrade bokeh jinja2 python-dateutil packaging matplotlib tornado pillow numpy typing-ext
!pip install haversine
!pip install statsmodels
!pip install bokeh
!pip install gurobipy



import numpy as np
!pip install scipy
import scipy as scs
import scipy.linalg as scl
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d
import scipy
import pandas as pd
from shapely.geometry import Point
import geopandas as gpd
from geopandas import GeoDataFrame
from markupsafe import Markup
from haversine import haversine
from matplotlib import cm
import matplotlib.colors
import math
from pyproj import Transformer
import copy
import itertools
import networkx as nx
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)
from gurobipy import Model, GRB, quicksum
import numpy as np
import pandas as pd
!pip install rtree
import rtree
from scipy.spatial.distance import cdist
import numpy as np
import pandas as pd
from scipy.spatial.distance import cdist
```

```
Requirement already satisfied: bokeh in c:\users\julia\anaconda3\envs\engri_1101\lib\site-packages (2.
4.3)
Requirement already satisfied: jinja2 in c:\users\julia\anaconda3\envs\engri_1101\lib\site-packages (3.
1.2)
Requirement already satisfied: python-dateutil in c:\users\julia\anaconda3\envs\engri_1101\lib\site-pac
kages (2.8.2)
Requirement already satisfied: packaging in c:\users\julia\anaconda3\envs\engri_1101\lib\site-packages
(23.2)
Requirement already satisfied: matplotlib in c:\users\julia\anaconda3\envs\engri_1101\lib\site-packages
(3.5.3)
Requirement already satisfied: tornado in c:\users\julia\anaconda3\envs\engri_1101\lib\site-packages
(6.2)
Requirement already satisfied: pillow in c:\users\julia\anaconda3\envs\engri_1101\lib\site-packages (9.
5.0)
Requirement already satisfied: numpy in c:\users\julia\anaconda3\envs\engri_1101\lib\site-packages (1.2
1.6)
Requirement already satisfied: typing-extensions in c:\users\julia\anaconda3\envs\engri_1101\lib\site-p
ackages (4.7.1)
Requirement already satisfied: MarkupSafe in c:\users\julia\anaconda3\envs\engri_1101\lib\site-packages
(2.1.3)
Requirement already satisfied: PyYAML>=3.10 in c:\users\julia\anaconda3\envs\engri_1101\lib\site-packag
es (from bokeh) (6.0)
Requirement already satisfied: six>=1.5 in c:\users\julia\anaconda3\envs\engri_1101\lib\site-packages
(from python-dateutil) (1.16.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\julia\anaconda3\envs\engri_1101\lib\site-p
ackages (from matplotlib) (1.4.4)
Requirement already satisfied: pyparsing>=2.2.1 in c:\users\julia\anaconda3\envs\engri_1101\lib\site-pa
ckages (from matplotlib) (3.0.9)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\julia\anaconda3\envs\engri_1101\lib\site-p
ackages (from matplotlib) (4.38.0)
Requirement already satisfied: cycler>=0.10 in c:\users\julia\anaconda3\envs\engri_1101\lib\site-packag
es (from matplotlib) (0.11.0)
Requirement already satisfied: haversine in c:\users\julia\anaconda3\envs\engri_1101\lib\site-packages
(2.8.0)
Requirement already satisfied: statsmodels in c:\users\julia\anaconda3\envs\engri_1101\lib\site-package
s (0.13.5)
Requirement already satisfied: packaging>=21.3 in c:\users\julia\anaconda3\envs\engri_1101\lib\site-pac
kages (from statsmodels) (23.2)
Requirement already satisfied: numpy>=1.17 in c:\users\julia\anaconda3\envs\engri_1101\lib\site-package
s (from statsmodels) (1.21.6)
Requirement already satisfied: patsy>=0.5.2 in c:\users\julia\anaconda3\envs\engri_1101\lib\site-packag
es (from statsmodels) (0.5.3)
Requirement already satisfied: scipy>=1.3 in c:\users\julia\anaconda3\envs\engri_1101\lib\site-packages
(from statsmodels) (1.5.4)
Requirement already satisfied: pandas>=0.25 in c:\users\julia\anaconda3\envs\engri_1101\lib\site-packag
es (from statsmodels) (1.3.5)
Requirement already satisfied: pytz>=2017.3 in c:\users\julia\anaconda3\envs\engri_1101\lib\site-packag
es (from pandas>=0.25->statsmodels) (2022.7)
Requirement already satisfied: python-dateutil>=2.7.3 in c:\users\julia\anaconda3\envs\engri_1101\lib\s
ite-packages (from pandas>=0.25->statsmodels) (2.8.2)
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: pillow>=7.1.0 in c:\users\julia\anaconda3\envs\engri_1101\lib\site-packa
ges (from bokeh) (9.5.0)
Requirement already satisfied: typing-extensions>=3.10.0 in c:\users\julia\anaconda3\envs\engri_1101\li
b\site-packages (from bokeh) (4.7.1)
Requirement already satisfied: PyYAML>=3.10 in c:\users\julia\anaconda3\envs\engri_1101\lib\site-packag
es (from bokeh) (6.0)
Requirement already satisfied: numpy>=1.11.3 in c:\users\julia\anaconda3\envs\engri_1101\lib\site-packa
ges (from bokeh) (1.21.6)
Requirement already satisfied: tornado>=5.1 in c:\users\julia\anaconda3\envs\engri_1101\lib\site-packag
es (from bokeh) (6.2)
Requirement already satisfied: packaging>=16.8 in c:\users\julia\anaconda3\envs\engri_1101\lib\site-pac
kages (from bokeh) (23.2)
Requirement already satisfied: Jinja2>=2.9 in c:\users\julia\anaconda3\envs\engri_1101\lib\site-package
s (from bokeh) (3.1.2)
Requirement already satisfied: MarkupSafe>=2.0 in c:\users\julia\anaconda3\envs\engri_1101\lib\site-pac
```

```
kages (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)
C:\Users\julia\anaconda3\envs\engri_1101\lib\site-packages\numpy\_distributor_init.py:32: UserWarning:
loaded more than 1 DLL from .libs:
C:\Users\julia\anaconda3\envs\engri_1101\lib\site-packages\numpy\.libs\libopenblas.QVLO2T66WEPI7JZ63PS3
HMOHFEY472BC.gfortran-win_amd64.dll
C:\Users\julia\anaconda3\envs\engri_1101\lib\site-packages\numpy\.libs\libopenblas.XWYDX2IKJW2NMTWSFYNG
FUWKQU3LYTCZ.gfortran-win_amd64.dll
  stacklevel=1)
Requirement already satisfied: scipy in c:\users\julia\anaconda3\envs\engri_1101\lib\site-packages (1.
5.4)
Requirement already satisfied: numpy>=1.14.5 in c:\users\julia\anaconda3\envs\engri_1101\lib\site-packa
ges (from scipy) (1.21.6)
Requirement already satisfied: rtree in c:\users\julia\anaconda3\envs\engri_1101\lib\site-packages (1.
0.1)
Requirement already satisfied: typing-extensions>=3.7 in c:\users\julia\anaconda3\envs\engri_1101\lib\s
ite-packages (from rtree) (4.7.1)
```

In [230…

```
from bokeh.plotting import figure, show, output_notebook
from bokeh.tile_providers import get_provider, Vendors
from bokeh.models import ColumnDataSource, Circle, StaticLayoutProvider, GraphRenderer
from bokeh.transform import linear_cmap
from bokeh.palettes import Blues8 as palette
```

In [120…

```
import json
from pandas import json_normalize

# Replace 'filename.json' with your actual JSON file name
with open('citibike_station_information.json', 'r') as file:
    capacity = json.load(file)



capacity = pd.json_normalize(capacity['data'], record_path='stations')
data = pd.read_csv('202307-citibike-tripdata.csv',dtype={'start_station_id':str, 'end_station_id':str})
```

drop NANs

In [121…

```
data.dropna(subset=['start_station_id', 'end_station_id'], inplace=True)
```

Generate a list of the stations and their locations from the usage data. This will be needed in the following step, which is very similar to the work you performed in HW 2.

In the first citibike dataset (data), there are slightly over 2000 total stations in total, but the available capacity data (capacity) only covers approximately 1800 stations.

I will delete any rides that start or end at the missing stations.

BTW, please plot those missing station locations on a map, so you can potentially get some idea of why they're missing. You should also report the percentage of rides that you had to delete because of this problem. (These are prudent steps when you encounter missing data.)

In [121…

```
# CHAT GPT
# First, we'll rename the column in stations_df to match the key in rides_df for joining
capacity.rename(columns={'short_name': 'start_station_id'}, inplace=True)
```

```
# Merge to get the start station capacity
rides_with_start_capacity = pd.merge(
    data,
    capacity[['start_station_id', 'capacity']],
    on='start_station_id',
    how='left'
)

# Rename the capacity column to reflect it's the start station capacity
rides_with_start_capacity.rename(columns={'capacity': 'start_station_capacity'}, inplace=True)

# Next, rename the column in stations_df to match the end_station_id key in rides_df
capacity.rename(columns={'start_station_id': 'end_station_id'}, inplace=True)

# Merge to get the end station capacity
rides_with_all_capacity = pd.merge(
    rides_with_start_capacity,
    capacity[['end_station_id', 'capacity']],
    on='end_station_id',
    how='left'
)

# Rename the capacity column to reflect it's the end station capacity
rides_with_all_capacity.rename(columns={'capacity': 'end_station_capacity'}, inplace=True)

# Now rides_with_all_capacity has both the start and end station capacities

# Filter out rides where the start or end station capacity is NaN (missing)
filtered_rides = rides_with_all_capacity.dropna(subset=['start_station_capacity', 'end_station_capacity'

# Now `filtered_rides` will have only the rides where both start and end station capacities are known
```

In [121…

```
dropped_rows = rides_with_all_capacity[rides_with_all_capacity['start_station_capacity'].isnull() | ride
```

In [121…

```
print("Due to data mismatch, we need to delete rides that do not have capacities in the capacity dataset
print("This deleted ", round(100*(1-(len(filtered_rides)/len(rides_with_all_capacity))),3),"% of rides")
```

```
Due to data mismatch, we need to delete rides that do not have capacities in the capacity dataset
This deleted  1.755 % of rides
```

In [121…

```
print("Due to data mismatch, we need to delete rides that do not have capacities in the capacity dataset
print("This deleted ", round(100*(len(dropped_rows)/len(rides_with_all_capacity)),3),"% of rides")
```

```
Due to data mismatch, we need to delete rides that do not have capacities in the capacity dataset
This deleted  1.755 % of rides
```

In [121…

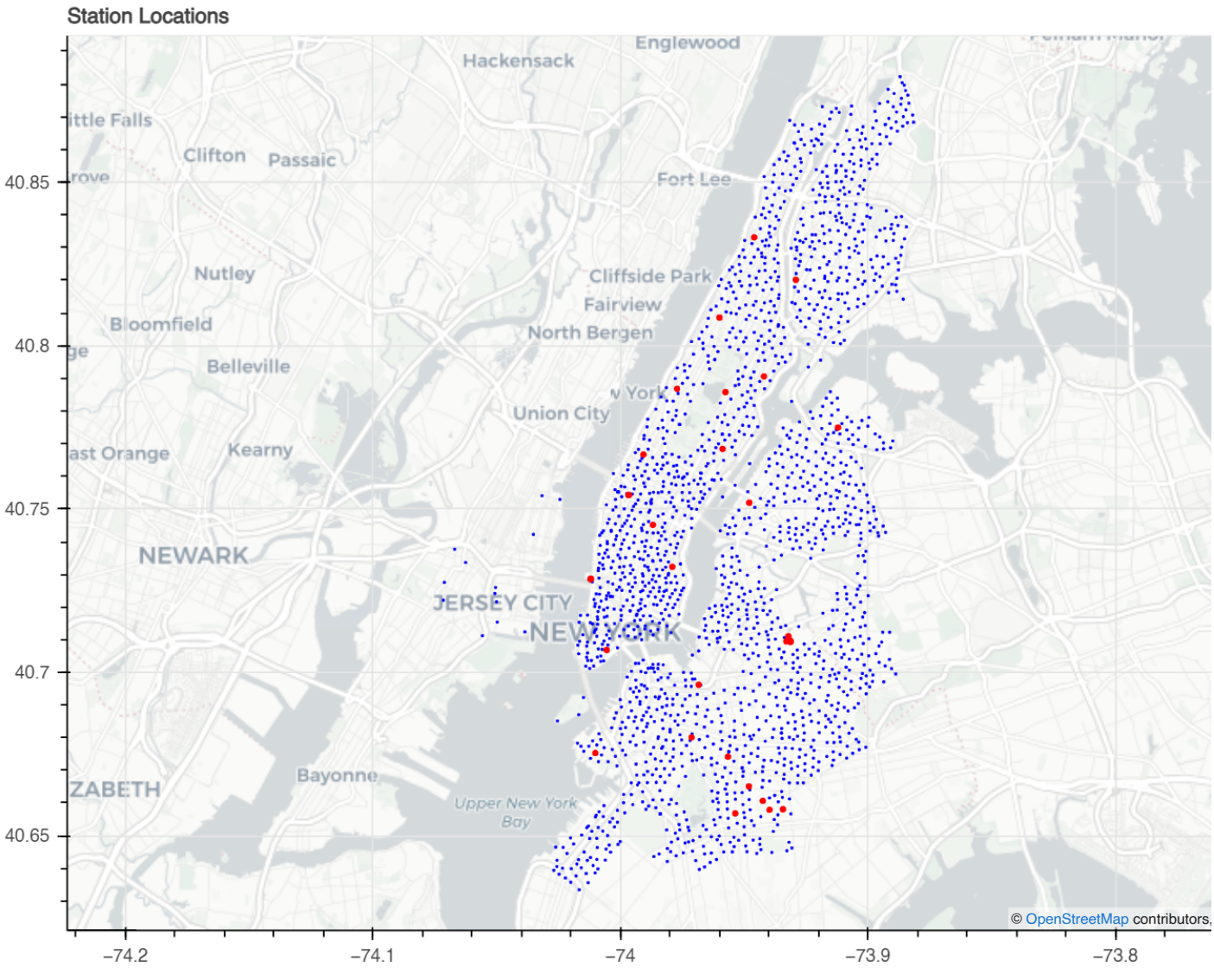
```
# Vectorized function for coordinate conversion
def lat_lon_to_mercator(lat, lon):
    k = 6378137
    x = lon * (k * np.pi / 180.0)
    y = np.log(np.tan((90 + lat) * np.pi / 360.0)) * k
    return x, y

# Identify missing and non-missing stations in one go
starts = dropped_rows[['start_lat','start_lng','start_station_id']].rename(columns={'start_lat':'lat','s
ends = dropped_rows[['end_lat','end_lng','end_station_id']].rename(columns={'end_lat':'lat','end_lng':'l
missing_stations = pd.concat([starts,ends]).drop_duplicates('id')
missing_stations['x'], missing_stations['y'] = lat_lon_to_mercator(missing_stations['lat'], missing_stat

starts_ = filtered_rides[['start_lat','start_lng','start_station_id']].rename(columns={'start_lat':'lat'
ends_ = filtered_rides[['end_lat','end_lng','end_station_id']].rename(columns={'end_lat':'lat','end_lng'
filtered_rides_not_missing = pd.concat([starts_,ends_]).drop_duplicates('id')
filtered_rides_not_missing['x'], filtered_rides_not_missing['y'] = lat_lon_to_mercator(filtered_rides_nc
missing_stations = missing_stations[~missing_stations['id'].isin(filtered_rides_not_missing['id'])]
```

In [128…

```
# chat GPT
# This function is a modified version of the one provided to fit our data
def plotNetwork(nodes, missing, title='Plot of Graph', on_map=True):
```

```
    output_notebook()

    # Prepare the data for plotting
    nodes_source = ColumnDataSource(nodes)
    missing_source = ColumnDataSource(missing)

    # Define the plot
    plot = figure(title=title,
                  x_axis_type="mercator", y_axis_type="mercator",
                  width=800, height=600,
                  toolbar_location=None, tools="pan,wheel_zoom,reset")

    # Add map tile if requested
    if on_map:
        plot.add_tile(get_provider(Vendors.CARTODBPOSITRON))

    # Add the nodes to the plot

    plot.circle(x='x', y='y', size=1, color='blue',source=nodes_source)
    plot.circle(x='x', y='y', size=3, color='red',source=missing_source)


    # Show the plot
    show(plot)

# Call the function with our prepared data
plotNetwork(filtered_rides_not_missing, missing_stations, title='Station Locations')
```

Loading BokehJS ...



Red Dots are points missing from the dataset.

In [121…
```
station_list = filtered_rides_not_missing
```

correct null values

In [121…
```
for i in station_list.columns:
    print(station_list[pd.isna(station_list[i])])
```

```
Empty DataFrame
Columns: [lat, lng, id, x, y]
Index: []
Empty DataFrame
Columns: [lat, lng, id, x, y]
Index: []
Empty DataFrame
Columns: [lat, lng, id, x, y]
Index: []
Empty DataFrame
Columns: [lat, lng, id, x, y]
Index: []
Empty DataFrame
Columns: [lat, lng, id, x, y]
Index: []
```

In [121…
```
for i in filtered_rides.columns:
    print(i)
    print(filtered_rides[pd.isna(filtered_rides[i])])
    print(' ')
```

```
ride_id
Empty DataFrame
Columns: [ride_id, rideable_type, started_at, ended_at, start_station_name, start_station_id, end_stati
on_name, end_station_id, start_lat, start_lng, end_lat, end_lng, member_casual, start_station_capacity,
end_station_capacity]
Index: []

rideable_type
Empty DataFrame
Columns: [ride_id, rideable_type, started_at, ended_at, start_station_name, start_station_id, end_stati
on_name, end_station_id, start_lat, start_lng, end_lat, end_lng, member_casual, start_station_capacity,
end_station_capacity]
Index: []

started_at
Empty DataFrame
Columns: [ride_id, rideable_type, started_at, ended_at, start_station_name, start_station_id, end_stati
on_name, end_station_id, start_lat, start_lng, end_lat, end_lng, member_casual, start_station_capacity,
end_station_capacity]
Index: []

ended_at
Empty DataFrame
Columns: [ride_id, rideable_type, started_at, ended_at, start_station_name, start_station_id, end_stati
on_name, end_station_id, start_lat, start_lng, end_lat, end_lng, member_casual, start_station_capacity,
end_station_capacity]
Index: []

start_station_name
Empty DataFrame
Columns: [ride_id, rideable_type, started_at, ended_at, start_station_name, start_station_id, end_stati
on_name, end_station_id, start_lat, start_lng, end_lat, end_lng, member_casual, start_station_capacity,
end_station_capacity]
Index: []

start_station_id
Empty DataFrame
Columns: [ride_id, rideable_type, started_at, ended_at, start_station_name, start_station_id, end_stati
on_name, end_station_id, start_lat, start_lng, end_lat, end_lng, member_casual, start_station_capacity,
end_station_capacity]
Index: []

end_station_name
Empty DataFrame
Columns: [ride_id, rideable_type, started_at, ended_at, start_station_name, start_station_id, end_stati
on_name, end_station_id, start_lat, start_lng, end_lat, end_lng, member_casual, start_station_capacity,
end_station_capacity]
Index: []

end_station_id
Empty DataFrame
Columns: [ride_id, rideable_type, started_at, ended_at, start_station_name, start_station_id, end_stati
on_name, end_station_id, start_lat, start_lng, end_lat, end_lng, member_casual, start_station_capacity,
end_station_capacity]
Index: []

start_lat
Empty DataFrame
Columns: [ride_id, rideable_type, started_at, ended_at, start_station_name, start_station_id, end_stati
on_name, end_station_id, start_lat, start_lng, end_lat, end_lng, member_casual, start_station_capacity,
end_station_capacity]
Index: []

start_lng
Empty DataFrame
Columns: [ride_id, rideable_type, started_at, ended_at, start_station_name, start_station_id, end_stati
on_name, end_station_id, start_lat, start_lng, end_lat, end_lng, member_casual, start_station_capacity,
end_station_capacity]
Index: []
```

```
end_lat
                 ride_id rideable_type           started_at  \
3090     380E35EA418B51E3   docked_bike  2023-07-06 18:45:31
6729     06049F65B806C6B5   docked_bike  2023-07-03 11:32:36
33357    010F59801CC7CB0E   docked_bike  2023-07-17 16:06:28
38030    045AE1802D2AF3FE   docked_bike  2023-07-01 11:10:37
38042    E9FFA3EE748C4826   docked_bike  2023-07-01 11:11:49
...                   ...           ...                  ...
3672285  125F1F0B751BE9A0   docked_bike  2023-07-02 16:58:33
3679448  36DBA75E4699073E   docked_bike  2023-07-06 20:25:42
3681860  CCBC8BC222C6235F   docked_bike  2023-07-04 19:53:39
3712269  A989B5C23B40C23F   docked_bike  2023-07-01 17:32:44
3749907  E5BCC1BDE27628C7   docked_bike  2023-07-03 12:36:05

                   ended_at            start_station_name start_station_id  \
3090     2023-07-06 18:59:54    Rivington St & Chrystie St          5453.01
6729     2023-07-03 11:54:02       Lafayette St & Grand St          5422.09
33357    2023-07-17 16:49:26      Division Ave & Hooper St          5045.05
38030    2023-07-01 11:38:48       W 16 St & The High Line          6233.05
38042    2023-07-01 11:39:09       W 16 St & The High Line          6233.05
...                      ...                           ...              ...
3672285  2023-07-02 17:12:48        Union Ave & Jackson St          5300.06
3679448  2023-07-06 20:42:55                  Sands St Gate          4812.04
3681860  2023-07-04 20:38:46                49 Ave & 21 St          6128.04
3712269  2023-07-01 18:06:04         Myrtle Ave & Fleet Pl          4628.07
3749907  2023-07-03 12:53:05  Plaza St East & Flatbush Ave          4010.01

                 end_station_name end_station_id  start_lat  start_lng  \
3090          S 4 St & Roebling St        5195.06  40.721101 -73.991925
6729     South End Ave & Albany St        5114.08  40.720280 -73.998790
33357          Kent Ave & Grand St        5388.01  40.706842 -73.954435
38030    South End Ave & Albany St        5114.08  40.743349 -74.006818
38042    South End Ave & Albany St        5114.08  40.743349 -74.006818
...                           ...            ...        ...        ...
3672285         N 9 St & Wythe Ave        5489.06  40.716075 -73.952029
3679448        Kent Ave & Grand St        5388.01  40.699569 -73.979827
3681860        Kent Ave & Grand St        5388.01  40.742520 -73.948852
3712269   Manhattan Ave & Devoe St        5219.05  40.693534 -73.981909
3749907   Parkside Ave & Parade Pl        3376.04  40.673134 -73.969106

         end_lat  end_lng member_casual  start_station_capacity  \
3090         NaN      NaN        casual                    64.0
6729         NaN      NaN        casual                    78.0
33357        NaN      NaN        casual                    61.0
38030        NaN      NaN        casual                    60.0
38042        NaN      NaN        casual                    60.0
...          ...      ...           ...                     ...
3672285      NaN      NaN        casual                    19.0
3679448      NaN      NaN        casual                    19.0
3681860      NaN      NaN        casual                    21.0
3712269      NaN      NaN        casual                    47.0
3749907      NaN      NaN        casual                    63.0

         end_station_capacity
3090                     41.0
6729                     45.0
33357                    45.0
38030                    45.0
38042                    45.0
...                       ...
3672285                  31.0
3679448                  45.0
3681860                  45.0
3712269                  24.0
3749907                  22.0

[134 rows x 15 columns]
```

```
end_lng
                 ride_id rideable_type           started_at  \
3090     380E35EA418B51E3   docked_bike  2023-07-06 18:45:31
6729     06049F65B806C6B5   docked_bike  2023-07-03 11:32:36
33357    010F59801CC7CB0E   docked_bike  2023-07-17 16:06:28
38030    045AE1802D2AF3FE   docked_bike  2023-07-01 11:10:37
38042    E9FFA3EE748C4826   docked_bike  2023-07-01 11:11:49
...                   ...           ...                  ...
3672285  125F1F0B751BE9A0   docked_bike  2023-07-02 16:58:33
3679448  36DBA75E4699073E   docked_bike  2023-07-06 20:25:42
3681860  CCBC8BC222C6235F   docked_bike  2023-07-04 19:53:39
3712269  A989B5C23B40C23F   docked_bike  2023-07-01 17:32:44
3749907  E5BCC1BDE27628C7   docked_bike  2023-07-03 12:36:05

                   ended_at            start_station_name start_station_id  \
3090     2023-07-06 18:59:54    Rivington St & Chrystie St          5453.01
6729     2023-07-03 11:54:02       Lafayette St & Grand St          5422.09
33357    2023-07-17 16:49:26      Division Ave & Hooper St          5045.05
38030    2023-07-01 11:38:48       W 16 St & The High Line          6233.05
38042    2023-07-01 11:39:09       W 16 St & The High Line          6233.05
...                      ...                           ...              ...
3672285  2023-07-02 17:12:48        Union Ave & Jackson St          5300.06
3679448  2023-07-06 20:42:55                  Sands St Gate          4812.04
3681860  2023-07-04 20:38:46                49 Ave & 21 St          6128.04
3712269  2023-07-01 18:06:04         Myrtle Ave & Fleet Pl          4628.07
3749907  2023-07-03 12:53:05  Plaza St East & Flatbush Ave          4010.01

                 end_station_name end_station_id  start_lat  start_lng  \
3090          S 4 St & Roebling St        5195.06  40.721101 -73.991925
6729     South End Ave & Albany St        5114.08  40.720280 -73.998790
33357          Kent Ave & Grand St        5388.01  40.706842 -73.954435
38030    South End Ave & Albany St        5114.08  40.743349 -74.006818
38042    South End Ave & Albany St        5114.08  40.743349 -74.006818
...                            ...            ...        ...        ...
3672285         N 9 St & Wythe Ave        5489.06  40.716075 -73.952029
3679448        Kent Ave & Grand St        5388.01  40.699569 -73.979827
3681860        Kent Ave & Grand St        5388.01  40.742520 -73.948852
3712269   Manhattan Ave & Devoe St        5219.05  40.693534 -73.981909
3749907   Parkside Ave & Parade Pl        3376.04  40.673134 -73.969106

         end_lat  end_lng member_casual  start_station_capacity  \
3090         NaN      NaN        casual                    64.0
6729         NaN      NaN        casual                    78.0
33357        NaN      NaN        casual                    61.0
38030        NaN      NaN        casual                    60.0
38042        NaN      NaN        casual                    60.0
...          ...      ...           ...                     ...
3672285      NaN      NaN        casual                    19.0
3679448      NaN      NaN        casual                    19.0
3681860      NaN      NaN        casual                    21.0
3712269      NaN      NaN        casual                    47.0
3749907      NaN      NaN        casual                    63.0

         end_station_capacity
3090                     41.0
6729                     45.0
33357                    45.0
38030                    45.0
38042                    45.0
...                       ...
3672285                  31.0
3679448                  45.0
3681860                  45.0
3712269                  24.0
3749907                  22.0

[134 rows x 15 columns]

member_casual
```

```
Empty DataFrame
Columns: [ride_id, rideable_type, started_at, ended_at, start_station_name, start_station_id, end_stati
on_name, end_station_id, start_lat, start_lng, end_lat, end_lng, member_casual, start_station_capacity,
end_station_capacity]
Index: []

start_station_capacity
Empty DataFrame
Columns: [ride_id, rideable_type, started_at, ended_at, start_station_name, start_station_id, end_stati
on_name, end_station_id, start_lat, start_lng, end_lat, end_lng, member_casual, start_station_capacity,
end_station_capacity]
Index: []

end_station_capacity
Empty DataFrame
Columns: [ride_id, rideable_type, started_at, ended_at, start_station_name, start_station_id, end_stati
on_name, end_station_id, start_lat, start_lng, end_lat, end_lng, member_casual, start_station_capacity,
end_station_capacity]
Index: []
```

there are null values in end_lng and end_lat

In [122…

```
filtered_rides['end_lat'] = np.where(pd.isna(filtered_rides['end_lat']), filtered_rides['start_lat'], fi
filtered_rides['end_lng'] = np.where(pd.isna(filtered_rides['end_lng']), filtered_rides['start_lng'], fi
```

```
C:\Users\julia\anaconda3\envs\engri_1101\lib\site-packages\ipykernel_launcher.py:1: SettingWithCopyWarn
ing:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.
html#returning-a-view-versus-a-copy
  """Entry point for launching an IPython kernel.
C:\Users\julia\anaconda3\envs\engri_1101\lib\site-packages\ipykernel_launcher.py:2: SettingWithCopyWarn
ing:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.
html#returning-a-view-versus-a-copy
```

In [122…

```
for i in filtered_rides.columns:
    print(i)
    print(filtered_rides[pd.isna(filtered_rides[i])])
    print(' ')
```

```
ride_id
Empty DataFrame
Columns: [ride_id, rideable_type, started_at, ended_at, start_station_name, start_station_id, end_stati
on_name, end_station_id, start_lat, start_lng, end_lat, end_lng, member_casual, start_station_capacity,
end_station_capacity]
Index: []

rideable_type
Empty DataFrame
Columns: [ride_id, rideable_type, started_at, ended_at, start_station_name, start_station_id, end_stati
on_name, end_station_id, start_lat, start_lng, end_lat, end_lng, member_casual, start_station_capacity,
end_station_capacity]
Index: []

started_at
Empty DataFrame
Columns: [ride_id, rideable_type, started_at, ended_at, start_station_name, start_station_id, end_stati
on_name, end_station_id, start_lat, start_lng, end_lat, end_lng, member_casual, start_station_capacity,
end_station_capacity]
Index: []

ended_at
Empty DataFrame
Columns: [ride_id, rideable_type, started_at, ended_at, start_station_name, start_station_id, end_stati
on_name, end_station_id, start_lat, start_lng, end_lat, end_lng, member_casual, start_station_capacity,
end_station_capacity]
Index: []

start_station_name
Empty DataFrame
Columns: [ride_id, rideable_type, started_at, ended_at, start_station_name, start_station_id, end_stati
on_name, end_station_id, start_lat, start_lng, end_lat, end_lng, member_casual, start_station_capacity,
end_station_capacity]
Index: []

start_station_id
Empty DataFrame
Columns: [ride_id, rideable_type, started_at, ended_at, start_station_name, start_station_id, end_stati
on_name, end_station_id, start_lat, start_lng, end_lat, end_lng, member_casual, start_station_capacity,
end_station_capacity]
Index: []

end_station_name
Empty DataFrame
Columns: [ride_id, rideable_type, started_at, ended_at, start_station_name, start_station_id, end_stati
on_name, end_station_id, start_lat, start_lng, end_lat, end_lng, member_casual, start_station_capacity,
end_station_capacity]
Index: []

end_station_id
Empty DataFrame
Columns: [ride_id, rideable_type, started_at, ended_at, start_station_name, start_station_id, end_stati
on_name, end_station_id, start_lat, start_lng, end_lat, end_lng, member_casual, start_station_capacity,
end_station_capacity]
Index: []

start_lat
Empty DataFrame
Columns: [ride_id, rideable_type, started_at, ended_at, start_station_name, start_station_id, end_stati
on_name, end_station_id, start_lat, start_lng, end_lat, end_lng, member_casual, start_station_capacity,
end_station_capacity]
Index: []

start_lng
Empty DataFrame
Columns: [ride_id, rideable_type, started_at, ended_at, start_station_name, start_station_id, end_stati
on_name, end_station_id, start_lat, start_lng, end_lat, end_lng, member_casual, start_station_capacity,
end_station_capacity]
Index: []
```

```
end_lat
Empty DataFrame
Columns: [ride_id, rideable_type, started_at, ended_at, start_station_name, start_station_id, end_stati
on_name, end_station_id, start_lat, start_lng, end_lat, end_lng, member_casual, start_station_capacity,
end_station_capacity]
Index: []

end_lng
Empty DataFrame
Columns: [ride_id, rideable_type, started_at, ended_at, start_station_name, start_station_id, end_stati
on_name, end_station_id, start_lat, start_lng, end_lat, end_lng, member_casual, start_station_capacity,
end_station_capacity]
Index: []

member_casual
Empty DataFrame
Columns: [ride_id, rideable_type, started_at, ended_at, start_station_name, start_station_id, end_stati
on_name, end_station_id, start_lat, start_lng, end_lat, end_lng, member_casual, start_station_capacity,
end_station_capacity]
Index: []

start_station_capacity
Empty DataFrame
Columns: [ride_id, rideable_type, started_at, ended_at, start_station_name, start_station_id, end_stati
on_name, end_station_id, start_lat, start_lng, end_lat, end_lng, member_casual, start_station_capacity,
end_station_capacity]
Index: []

end_station_capacity
Empty DataFrame
Columns: [ride_id, rideable_type, started_at, ended_at, start_station_name, start_station_id, end_stati
on_name, end_station_id, start_lat, start_lng, end_lat, end_lng, member_casual, start_station_capacity,
end_station_capacity]
Index: []
```

Suppose that the rides that are initiated at each Station i follow a Poisson process in time, with a rate function that is constant in each hour of the day on weekdays. (Don’t use the data for weekends - just ignore it.) In other words, at Station i, there is a constant arrival rate μ0(i) in place from 5am to 6am on weekdays (Monday, Tuesday, Wednesday, Thursday and Friday), a potentially different constant value μ1(i) from 6am to 7am on weekdays, . . ., a potentially different constant value μ18(i) from 11pm to midnight on weekdays. Using the Citibike data you obtain from Step 1, estimate these 19 numbers for each station i, where μt(i) represents the average number of rides initiated per hour in the tth hour of a weekday from Station i. Ignore the variation between weekdays that you saw in HW 2. Also ignore censoring. We’ll assume

that the number of rides from midnight to 5am is negligible and ignore them

In [122…

```
citi_bike = pd.read_csv('202307-citibike-tripdata.csv')
citi_bike['started_at'] = pd.to_datetime(citi_bike['started_at'])
citi_bike['day_of_week'] = citi_bike['started_at'].dt.dayofweek
citi_bike['hour'] = citi_bike['started_at'].dt.hour
citi_bike['week_number'] = citi_bike['started_at'].dt.isocalendar().week

citi_bike = citi_bike[citi_bike['day_of_week']<6]
citi_bike = citi_bike[citi_bike['hour']>4]
citi_bike_grouped = citi_bike.groupby([
    citi_bike['week_number'],    # Group by week
    citi_bike['hour'],           # Group by day of the week
    citi_bike['start_station_id'] # Group by stTION
]).size().reset_index(name='counts')

citi_bike_grouped.columns = ['week_number', 'hour','station', 'counts']

citi_bike_grouped
```

```
C:\Users\julia\anaconda3\envs\engri_1101\lib\site-packages\IPython\core\interactiveshell.py:3457: DtypeWarning: Columns (5,7) have mixed types.Specify dtype option on import or set low_memory=False.
  exec(code_obj, self.user_global_ns, self.user_ns)
```

Out[1222]:

	week_number	hour	station	counts
0	26	5	2912.08	1
1	26	5	3520.01	1
2	26	5	3625.07	1
3	26	5	3731.11	1
4	26	5	3791.02	1
...	...	...	...	...
272524	31	23	8734.02	2
272525	31	23	8752.01	1
272526	31	23	8778.01	3
272527	31	23	8795.01	3
272528	31	23	8841.03	1

272529 rows × 4 columns

In [122…

```
sum(citi_bike_grouped['counts'])
```

Out[1223]:

```
3152658
```

In [122…

```
all_weeks = range(26, 32)  # 26 to 31 inclusive
all_hours = range(5, 24)   # 5 to 23 inclusive
all_stations = citi_bike_grouped['station'].unique()

from itertools import product

all_combinations = pd.DataFrame(product(all_weeks, all_hours, all_stations), columns=['week_number', 'ho
complete_df = all_combinations.merge(citi_bike_grouped, on=['week_number', 'hour', 'station'], how='left
complete_df
```

Out[1224]:

	week_number	hour	station	counts
0	26	5	2912.08	1.0
1	26	5	3520.01	1.0
2	26	5	3625.07	1.0
3	26	5	3731.11	1.0
4	26	5	3791.02	1.0
...	...	...	...	...
432739	31	23	3256.04	0.0
432740	31	23	3155.01	0.0
432741	31	23	3157.08	0.0
432742	31	23	3187.01	0.0
432743	31	23	3157.08	1.0

432744 rows × 4 columns

In [122…

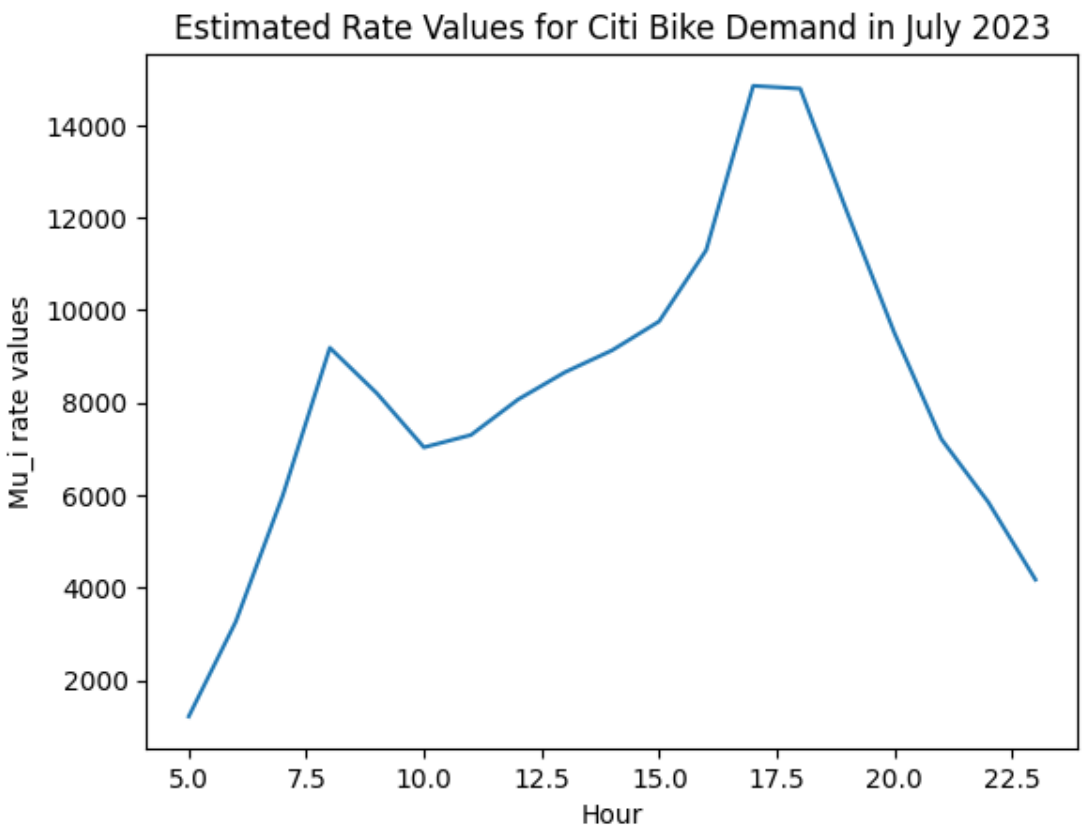
```
mus = complete_df.groupby(['hour','station']).sum()['counts'].reset_index()
mus = mus.reset_index()
mus['counts'] = mus['counts']/20
```

In [122…

```
group = mus.groupby(['hour']).sum()['counts'].reset_index()

plt.plot(group['hour'],group['counts'])

plt.xlabel('Hour')
plt.ylabel('Mu_i rate values')
plt.title('Estimated Rate Values for Citi Bike Demand in July 2023')
```

Out[1226]:

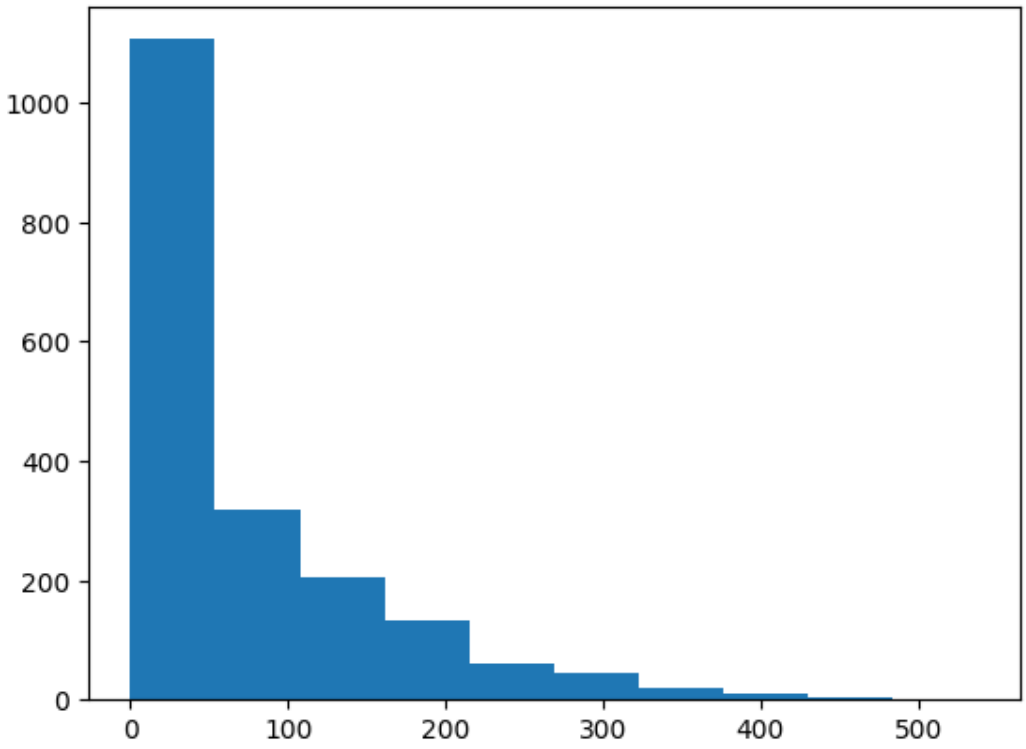
```
Text(0.5, 1.0, 'Estimated Rate Values for Citi Bike Demand in July 2023')
```



In [122…
```
mus = mus.rename(columns={'station':"id"})
mus_group = mus[['id','counts']].groupby(['id']).sum()['counts'].reset_index()
plot_nodes = pd.merge(station_list,mus_group,on='id',how='left')
plot_nodes.dropna(inplace=True)
```

In [122…
```
plt.hist(plot_nodes['counts'])
```

Out[1228]:
```
(array([1107.,  318.,  205.,  132.,   61.,   44.,   19.,   11.,    3.,
           2.]),
 array([5.00000e-02, 5.38250e+01, 1.07600e+02, 1.61375e+02, 2.15150e+02,
        2.68925e+02, 3.22700e+02, 3.76475e+02, 4.30250e+02, 4.84025e+02,
        5.37800e+02]),
 <BarContainer object of 10 artists>)
```

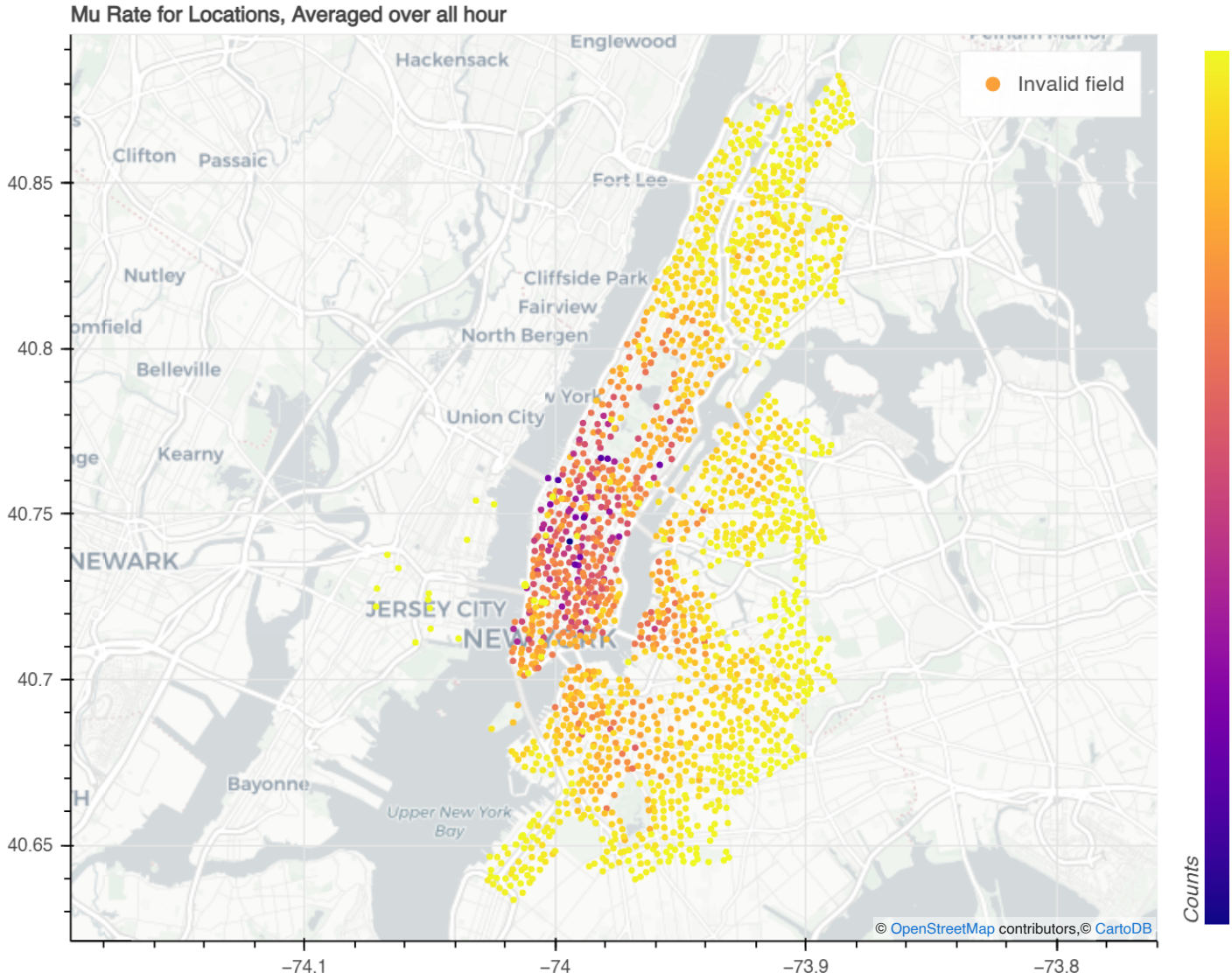


In [128…
```
from bokeh.models import ColumnDataSource, LinearColorMapper, ColorBar
from bokeh.transform import transform
from bokeh.layouts import layout
def plotNetwork(nodes, title='Plot of Graph', on_map=True):
    output_notebook()

    # Prepare the data for plotting
    nodes_source = ColumnDataSource(nodes)

    # Create a color mapper for 'counts' column with a high and low bound
    color_mapper = LinearColorMapper(palette='Plasma256', low=max(nodes['counts']), high=min(nodes['cour

    # Define the plot
    plot = figure(title=title,
                  x_axis_type="mercator", y_axis_type="mercator",
                  width=800, height=600,
                  toolbar_location=None, tools="pan,wheel_zoom,reset")

    # Add map tile if requested
    if on_map:
        plot.add_tile(get_provider(Vendors.CARTODBPOSITRON))

    # Add the nodes to the plot with color mapped by 'counts'
    plot.circle(x='x', y='y', size=3, color=transform('counts', color_mapper),
                legend_field='criteria', source=nodes_source)

    # Add a color bar to the side of the plot to show the color mapping
    color_bar = ColorBar(color_mapper=color_mapper, label_standoff=12, location=(0,0), title='Counts')
```

```
    plot.add_layout(color_bar, 'right')

    # Show the plot
    show(layout(plot))

# Call the function with the prepared data
plotNetwork(nodes=plot_nodes,  title='Mu Rate for Locations, Averaged over all hour')
```

Loading BokehJS ...

```
ERROR:bokeh.core.validation.check:E-1001 (BAD_COLUMN_NAME): Glyph refers to nonexistent column name. Th
is could either be due to a misspelling or typo, or due to an expected column being missing. : LegendIt
em(id='1587121', ...)
```



Estimate, for each hour t the transition probabilities pt(i, j) = Pr(destination station is j | a ride leaves Station i in hour t). (This is just the fraction of rides beginning in Station i in time t that go to Station j). Effectively, you are generating 19 matrices, one for each hour, where each matrix is a transition matrix so is nonnegative and has row sums equal to 1. If no rides

originate from Station i in hour t then just set pt(i, i) = 1 and pt(i, j)=0 for j not equal to i

In [330…

```
def calculate_transition_probabilities(df):
    transition_matrices = {}

    # Extract the hour from 'started_at'
    df['hour'] = pd.to_datetime(df['started_at']).dt.hour

    # Get all unique station IDs for initializing matrices
    all_stations = set(df['start_station_id']).union(set(df['end_station_id']))

    # Loop through each hour to calculate transition probabilities
    for t in range(5,24):  # Assuming 24 hours in a day, 0 to 23
        # Filter rides that start in hour t
        df_hour = df[df['hour'] == t]

        # Count the number of rides from each start station to each end station
        transition_counts = df_hour.groupby(['start_station_id', 'end_station_id']).size().unstack(fill_

        # Ensure all stations are present in the matrix
        transition_counts = transition_counts.reindex(index=all_stations, columns=all_stations, fill_val

        # Calculate probabilities by dividing each row by the row sum
        row_sums = transition_counts.sum(axis=1)
        transition_probs = transition_counts.div(row_sums, axis=0)

        # Fill diagonal with 1 where row sums are 0 (no rides originating from the station)
        for station in all_stations:
            if row_sums.get(station, 0) == 0:
                transition_probs.at[station, station] = 1

        # Fill remaining NaNs with 0 (should not be needed, but just to be safe)
        transition_probs.fillna(0, inplace=True)

        # Store the transition matrix for hour t
        transition_matrices[t] = transition_probs

    return transition_matrices

# Sample usage:
transition_matrices = calculate_transition_probabilities(filtered_rides)
```

```
C:\Users\julia\anaconda3\envs\engri_1101\lib\site-packages\ipykernel_launcher.py:5: SettingWithCopyWarn
ing:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.
html#returning-a-view-versus-a-copy
  """
```

Compute, for each hour t, the arrival rate of riders who are returning bikes to Station i for all stations i. In other words, at Station i, there is a constant arrival rate of bikers returning bikes λ0(i) in place from 5am to 6am on weekdays (Monday, Tuesday, Wednesday, Thursday and Friday), a potentially different constant value λ1(i) from 6am to 7am on weekdays, . . ., a potentially different constant value λ18(i) from 11pm to

midnight on weekdays. Compute these 19 numbers for each station i, where λt(i) represents the average number of riders returning bikes per hour in the tth hour of the week from Station i. You can get these values from Parts 2 and 3 above; don't estimate them from the data.

In [331…

```
# Chat GPT
def calculate_arrival_rates(mus, transition_matrices):
    # Dictionary to hold the arrival rates for each station
    arrival_rates = {station_id: pd.Series(0, index=range(5, 24)) for station_id in station_list['id'].u

    # Iterate through each hour t
    for t in range(5, 24):
        # Get the transition matrix for hour t
        transition_matrix_t = transition_matrices.get(t)
        # Skip hours where transition matrix is not available
        if transition_matrix_t is None:
            continue

        # Calculate the arrival rates for each station
        for station_id in arrival_rates.keys():
            # Get the lambda value for the current station and hour
            mu_value = mus[(mus['id'] == station_id) & (mus['hour'] == t)]['counts'].sum()
            # Calculate the arrival rates using transition probabilities
            arrival_rate = (transition_matrix_t.loc[:, station_id] * mu_value).sum() if station_id in tr
            # Store the arrival rate in the dictionary under the station id
            arrival_rates[station_id][t] = arrival_rate

    return arrival_rates

# Sample usage:
# Assuming transition_matrices is already computed and is a dictionary
# where each key is an hour and each value is a DataFrame representing the transition matrix for that ho
# Also, assuming lambdas is a DataFrame with columns 'id', 'hour', and 'counts'.
arrival_rates = calculate_arrival_rates(mus, transition_matrices)

# arrival_rates is a dictionary where each key is a station id and each value is a Series
# with index as hours and the values are the arrival rates for that station across hours.
```

In [339…

```
lambdas = pd.DataFrame(arrival_rates)
lambdas = lambdas.reset_index()
lambdas = lambdas.rename(columns={'index':'hour'})
lambdas = lambdas.melt(id_vars=['hour'],
                       var_name='id',
                       value_name='counts')
lambdas = lambdas.reset_index()
```

Out[339]:

	index	hour	id	counts
0	0	5	7756.10	16
1	1	6	7756.10	28
2	2	7	7756.10	30
3	3	8	7756.10	37
4	4	9	7756.10	69
...	...	...	...	...
36494	36494	19	JC052	0
36495	36495	20	JC052	0
36496	36496	21	JC052	0
36497	36497	22	JC052	0
36498	36498	23	JC052	0

36499 rows × 4 columns

In [123…

```
lambdas_filtered = lambdas[lambdas['id'].isin(station_list['id'])]
lambdas_filtered
```

Out[1239]:

	index	hour	id	counts
241	241	5	2733.03	0.15
242	242	5	2782.02	0.05
243	243	5	2821.05	0.10
244	244	5	2832.03	0.05
245	245	5	2861.02	0.00
...	...	...	...	...
41597	41597	23	JC105	0.00
41598	41598	23	JC109	0.00
41599	41599	23	JC110	0.00
41600	41600	23	JC115	0.00
41601	41601	23	JC116	0.00

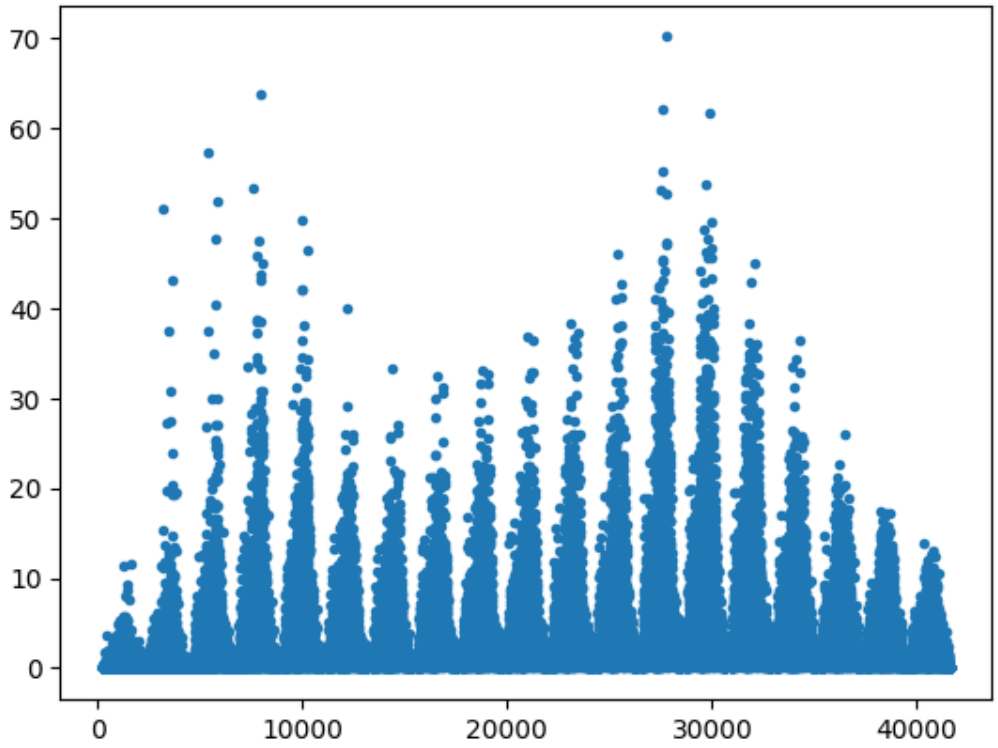
36461 rows × 4 columns

In [124…

```
plt.plot(lambdas_filtered['index'],lambdas_filtered['counts'],'.')
```

Out[1240]:

```
[<matplotlib.lines.Line2D at 0x28e96fd3288>]
```



In [124…
```
mus_filtered = mus[mus['id'].isin(station_list['id'])]
mus_filtered = mus_filtered.reset_index()
mus_filtered['index_'] = mus_filtered.index
mus_filtered
```

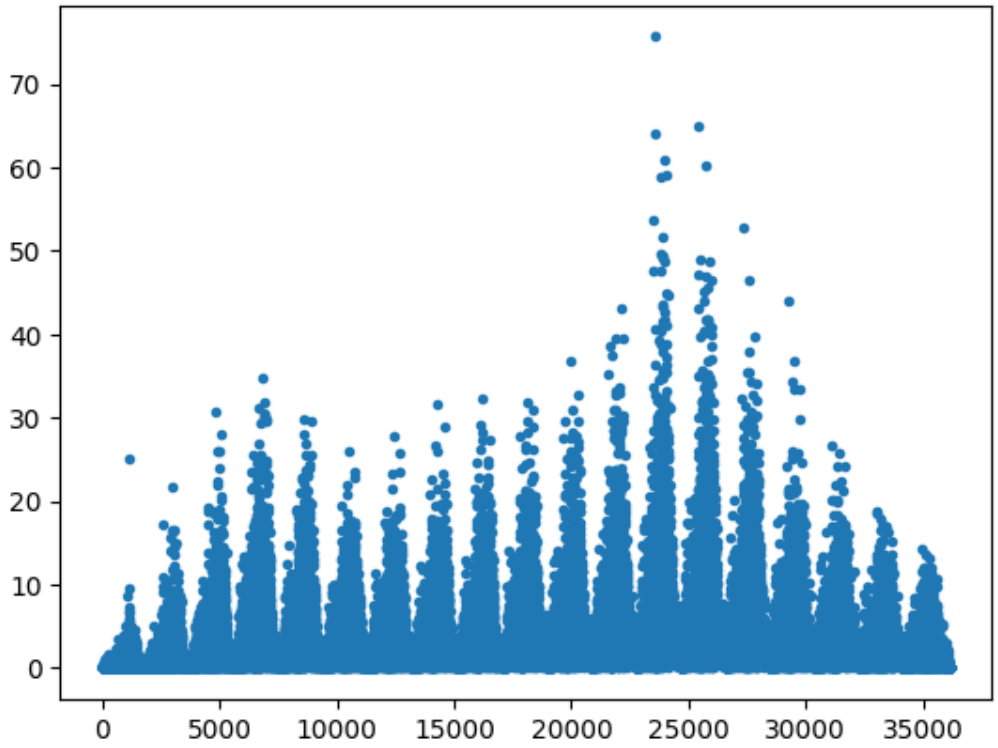
Out[1241]:

	level_0	index	hour	id	counts	index_
0	1866	1866	5	2733.03	0.1	0
1	1867	1867	5	2782.02	0.0	1
2	1868	1868	5	2821.05	0.1	2
3	1869	1869	5	2832.03	0.0	3
4	1870	1870	5	2861.02	0.1	4
...	...	...	...	...	...	...
36133	72113	72113	23	JC102	0.0	36133
36134	72114	72114	23	JC104	0.0	36134
36135	72115	72115	23	JC105	0.0	36135
36136	72116	72116	23	JC115	0.0	36136
36137	72117	72117	23	JC116	0.0	36137

36138 rows × 6 columns

In [124…
```
plt.plot(mus_filtered['index_'],mus_filtered['counts'],'.')
```

Out[1242]:
```
[<matplotlib.lines.Line2D at 0x28e9bdca2c8>]
```

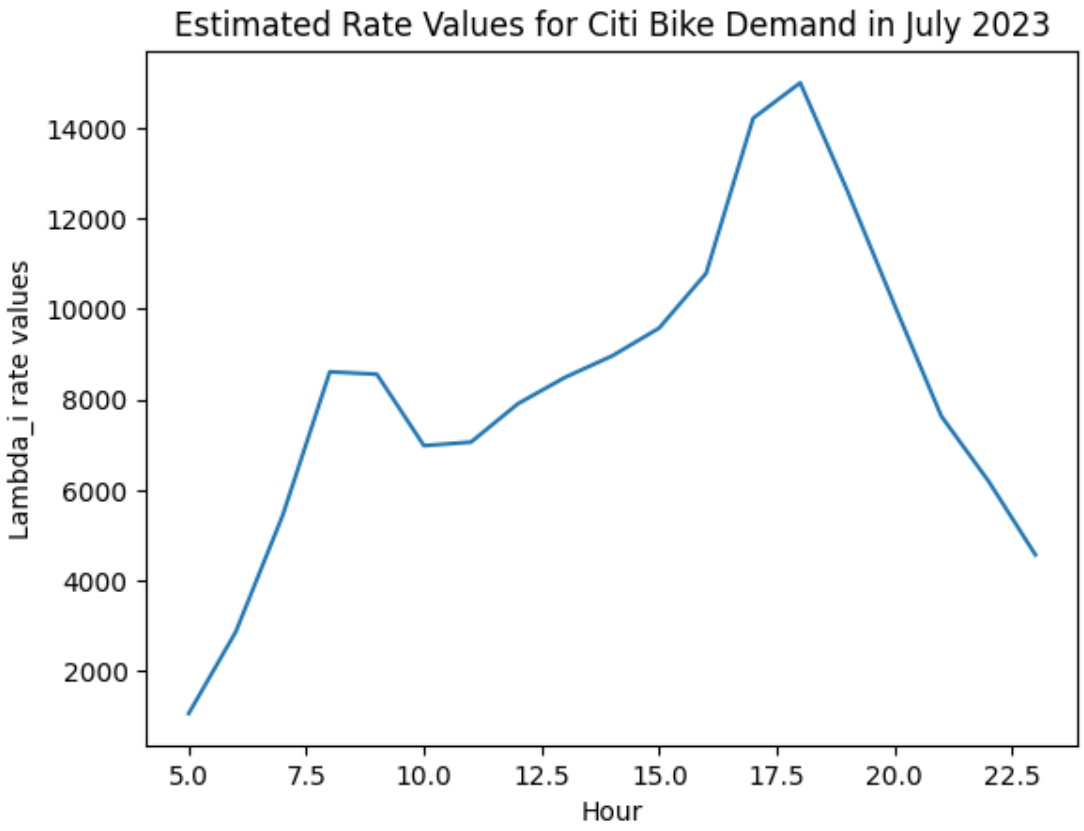


In [124…

```
group = lambdas.groupby(['hour']).sum()['counts'].reset_index()

plt.plot(group['hour'],group['counts'])

plt.xlabel('Hour')
plt.ylabel('Lambda_i rate values')
plt.title('Estimated Rate Values for Citi Bike Demand in July 2023')
```

Out[1243]:

```
Text(0.5, 1.0, 'Estimated Rate Values for Citi Bike Demand in July 2023')
```



In [124…

```
lambdas = lambdas.rename(columns={'station':"id"})
lambdas_group = lambdas[['id','counts']].groupby(['id']).sum()['counts'].reset_index()
```

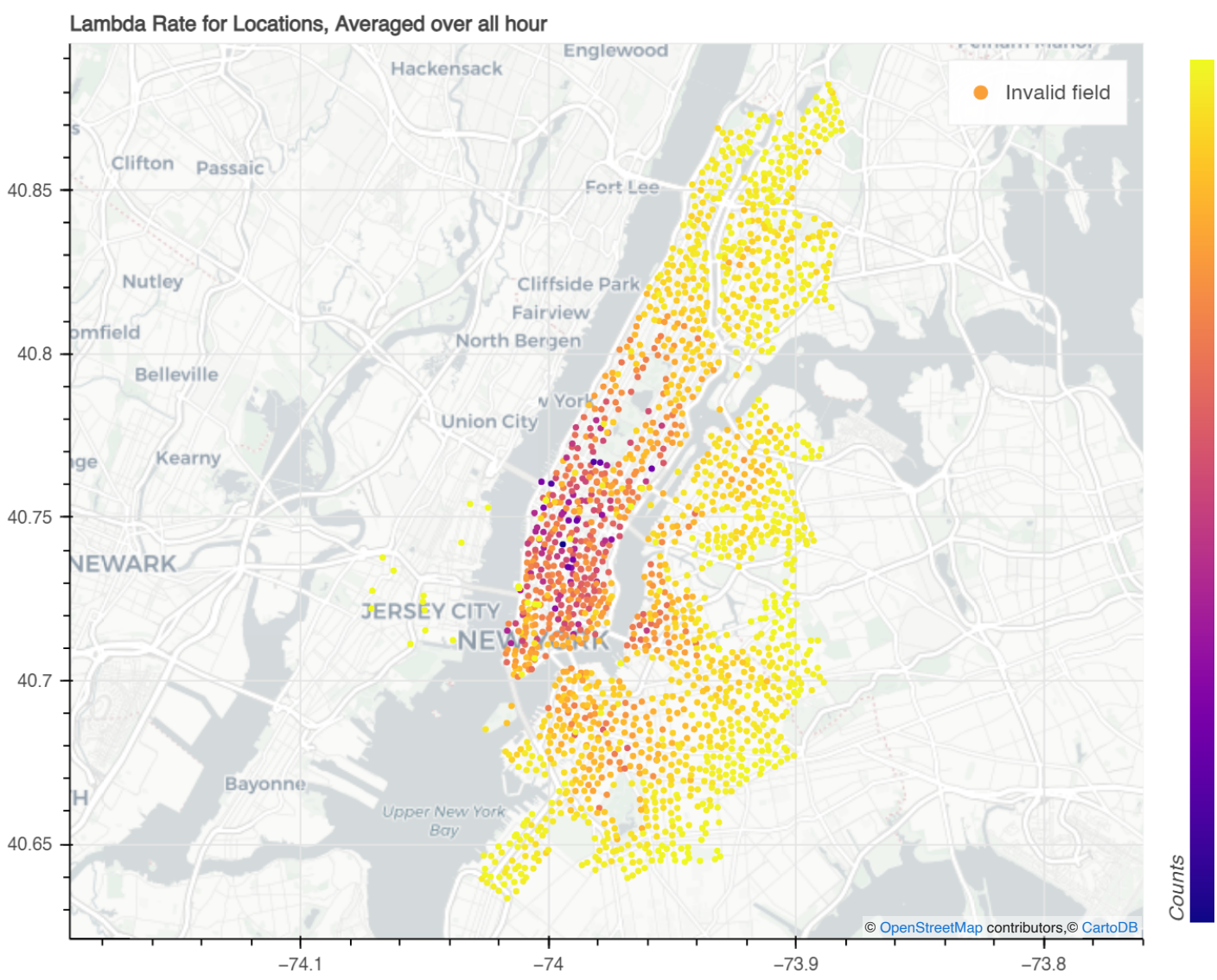
```
plot_nodes = pd.merge(station_list,lambdas_group,on='id',how='left')
plot_nodes.dropna(inplace=True)
```

In [128…

```
from bokeh.models import ColumnDataSource, LinearColorMapper, ColorBar
from bokeh.transform import transform
from bokeh.layouts import layout
def plotNetwork(nodes, title='Plot of Graph', on_map=True):
    output_notebook()

    # Prepare the data for plotting
    nodes_source = ColumnDataSource(nodes)

    # Create a color mapper for 'counts' column with a high and low bound
    color_mapper = LinearColorMapper(palette='Plasma256', low=max(nodes['counts']), high=min(nodes['cour

    # Define the plot
    plot = figure(title=title,
                  x_axis_type="mercator", y_axis_type="mercator",
                  width=800, height=600,
                  toolbar_location=None, tools="pan,wheel_zoom,reset")

    # Add map tile if requested
    if on_map:
        plot.add_tile(get_provider(Vendors.CARTODBPOSITRON))

    # Add the nodes to the plot with color mapped by 'counts'
    plot.circle(x='x', y='y', size=3, color=transform('counts', color_mapper),
                legend_field='criteria', source=nodes_source)

    # Add a color bar to the side of the plot to show the color mapping
    color_bar = ColorBar(color_mapper=color_mapper, label_standoff=12, location=(0,0), title='Counts')
    plot.add_layout(color_bar, 'right')

    # Show the plot
    show(layout(plot))

# Call the function with the prepared data
plotNetwork(nodes=plot_nodes,  title='Lambda Rate for Locations, Averaged over all hour')
```


Loading BokehJS ...

```
ERROR:bokeh.core.validation.check:E-1001 (BAD_COLUMN_NAME): Glyph refers to nonexistent column name. Th
is could either be due to a misspelling or typo, or due to an expected column being missing. : LegendIt
em(id='1575841', ...)
```



In [124…

```
join = pd.merge(lambdas_filtered.groupby('id').sum().reset_index(),mus_filtered.groupby('id').sum().rese
join = join.fillna(0)
join
```

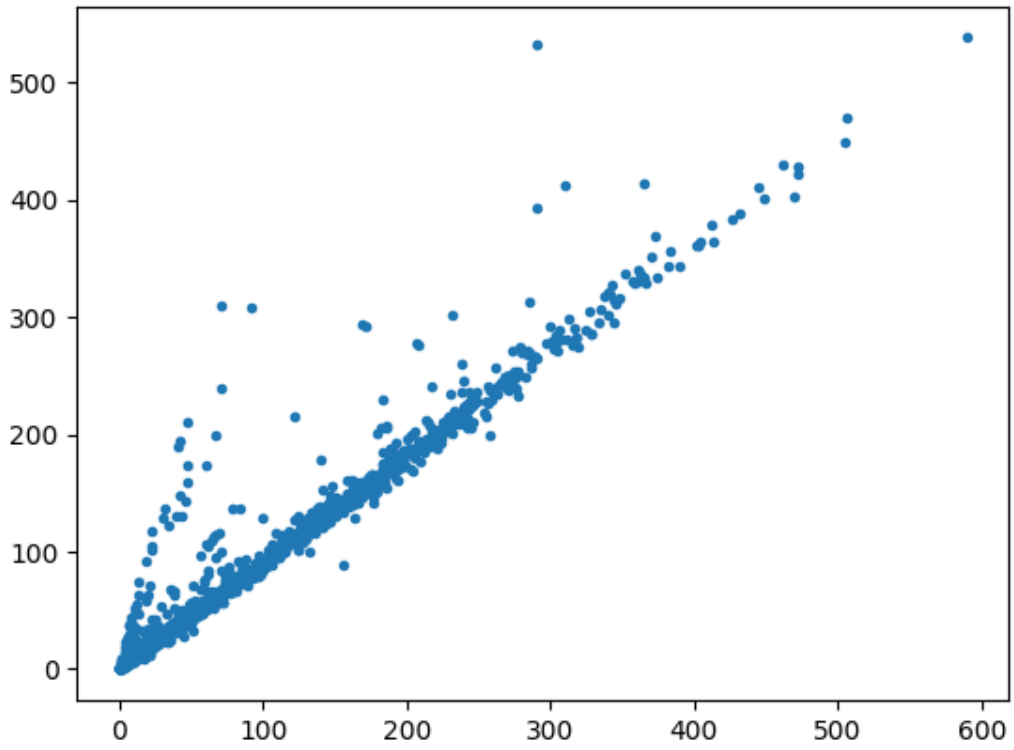
Out[1246]:

	id	index_x	hour_x	counts_x	level_0	index_y	hour_y	counts_y	index_
0	2733.03	379069.0	266.0	9.95	684570.0	684570.0	266.0	8.85	325242.0
1	2782.02	379088.0	266.0	13.65	684589.0	684589.0	266.0	13.85	325261.0
2	2821.05	379107.0	266.0	15.65	684608.0	684608.0	266.0	14.80	325280.0
3	2832.03	379126.0	266.0	15.60	684627.0	684627.0	266.0	16.00	325299.0
4	2861.02	379145.0	266.0	14.25	684646.0	684646.0	266.0	13.25	325318.0
...	...	...	...	...	...	...	...	...	...
1915	JC109	415872.0	266.0	0.05	0.0	0.0	0.0	0.00	0.0
1916	JC110	415891.0	266.0	0.10	0.0	0.0	0.0	0.00	0.0
1917	JC115	415910.0	266.0	0.30	721088.0	721088.0	266.0	0.15	361342.0
1918	JC116	415929.0	266.0	0.15	721107.0	721107.0	266.0	0.10	361361.0
1919	JC093	0.0	0.0	0.00	720974.0	720974.0	266.0	0.10	361228.0

1920 rows × 9 columns

In [124…
```
plt.plot(join['counts_x'],join['counts_y'],'.')
```

Out[1247]:
```
[<matplotlib.lines.Line2D at 0x28e74199848>]
```



In [124…
```
np.sum(join['counts_x'],axis=0)
```

Out[1248]:
```
147062.5
```

In [124…
```
np.sum(join['counts_y'],axis=0)
```

Out[1249]:
```
141053.75
```

In [125…
```
np.sum(join['counts_y'],axis=0)-np.sum(join['counts_x'],axis=0)
```

Out[1250]:
```
-6008.75
```

normalize so the net flow is the same because assuming flow from 5am to midnight is the same. Conservation of Flow must hold.

In [125…
```
# This code will normalize the columns that start with 'numb' so that their sum equals the sum of 'numbe
sum_take_aways = np.sum(mus_filtered['counts'])

# Normalize the columns
lambdas_filtered['counts_normalized'] = (lambdas_filtered['counts'] / lambdas_filtered['counts'].sum())
```

In [125…
```
mus = mus_filtered
lambdas = lambdas_filtered
lambdas['counts'] = lambdas['counts_normalized'].copy()
lambdas
```

Out[1252]:

	index	hour	id	counts	counts_normalized
241	241	5	2733.03	0.143871	0.143871
242	242	5	2782.02	0.047957	0.047957
243	243	5	2821.05	0.095914	0.095914
244	244	5	2832.03	0.047957	0.047957
245	245	5	2861.02	0.000000	0.000000
...	...	...	...	...	...
41597	41597	23	JC105	0.000000	0.000000
41598	41598	23	JC109	0.000000	0.000000
41599	41599	23	JC110	0.000000	0.000000
41600	41600	23	JC115	0.000000	0.000000
41601	41601	23	JC116	0.000000	0.000000

36461 rows × 5 columns

In [125…
```
np.sum(lambdas['counts'])
```

Out[1253]:
```
141053.75000000003
```

In [125…
```
np.sum(mus['counts'])
```

Out[1254]:
```
141053.75
```

Now, for each station in the system, compute the optimal number of bikes to place at the station by 5am to minimize outages throughout a typical weekday. Also compute the net flow of bikes over a typical weekday at all stations (this entails computing one number for each station, and is essentially how many bikes would need to be moved overnight per station per day to keep the system in balance).

To compute the optimal number of bikes at each station, we want to minimize the number of unhappy people. People are unhappy when there are no bikes at the station they want or there is nowhere to dock a bike at the station they want.

Citibike makes income from 1 time users or yearly subscription.

We want to minimize the number of angry bikers

Decisions

- rebalancing = overnight bike movement on trucks and rush periods where bikes trailors can tow more bikes
- dock allocation = where to put bikes

Model

Fluid model at each station.

For each "period" (hour) t, and each station i, let:

$\mu_i(t)$ = rate of bikes taken from station (bikes/hr)

$\lambda_i(t)$ = rate of bikes returnes (bikes/hr)

c_i = capacity of station i

$\text{angry bikers}(t) = \text{failed returns} + \text{failed pickups} = y_i(t) = \text{unhappy people at the end of period t}$

$x_i(t)$ = number of bikes at time t

So, $x_i(t+1) = x_i(t) + (\lambda_i(t) - \mu_i(t)$ (pinned to $[0, c_i]$

Therefore,
$y_i(t) = \text{failed returns} + \text{failed pickups} = d * [x_i(t) + (\lambda_i(t) - \mu_i(t)) - c_i]^+ + p * [x_i(t) + (\lambda_i(t) - \mu_i(t))]^-$

where p and d are some multipliers to determine the relative importance of minimizing failed dropoffs or failed returns. For the sake of this model we will assume $p = d = 1$, or the unhappiness from inability to drop off is equal to the unhappiness from inability to pick up

The problem to determine the optimal number of bikes at each station at 5am to minimize the number of unhappy people is therefore:

$$min_x \sum_{i \in \text{stations}} \sum_{t=5\text{am}}^{11\text{pm}} y_i(t)$$

This problem will give the bike allocation and fleet size optimally. There are no constraints needed to link stations, and we already know the departure rate and arrival rate of each bike station at east time $\lambda_i(t) \text{and} \mu_i(t)$.

It is therefore easy to calculate the objective at each station by breaking down the calculation by station and enumerating.

In [125…

```python
#GPT
def calculate_unhappiness(initial_bikes, mus_i, lambda_i, capacity_i, start_hour=14):
    """
    Calculate the cumulative unhappiness for the rest of the day from start_hour until the end of the da

    Parameters:
    initial_bikes: int - the number of bikes at the station at start_hour
    mus_i: DataFrame - the number of bikes taken each hour for station i
    lambda_i: DataFrame - the number of bikes returned each hour for station i
    capacity_i: int - the capacity of station i
    start_hour: int - the hour to start calculations (default is 14 for 2 pm)

    Returns:
    int - the cumulative unhappiness for the rest of the day
    """

    # Initialize variables
    current_bikes = initial_bikes
    cumulative_unhappiness = 0

    # Loop through each hour from start_hour to 23
    for t in range(start_hour, 24):
        lam = float(lambda_i.loc[t,'counts'])
        m = float(mus_i.loc[t,'counts'])
        x_t = current_bikes + lam - m
        x_t_clamped = max(min(x_t, capacity_i), 0)
```

```
        xs.append(x_t_clamped)

        failed_returns = max(x_t - capacity_i, 0)
        failed_pickups = max(-x_t, 0)
        cumulative_unhappiness += failed_returns + failed_pickups

        # Update the current number of bikes for the next hour, clamping between 0 and capacity
        current_bikes = x_t_clamped

    return cumulative_unhappiness
```

In [125…

```
# original logic written by me, modified by Chat GPT
bike_impact = []
x_is=[]
for i in station_list['id']:
    capacity_i = int(capacity[capacity['end_station_id'] == str(i)]['capacity'].iloc[0])
    mus_i = mus[mus['id'] == i][['hour', 'counts']].copy()
    all_hours = pd.DataFrame({'hour': range(5, 24)})  # Adjusted to include 5 am
    mus_i = pd.merge(all_hours, mus_i, on='hour', how='left').fillna(0)
    mus_i.set_index('hour', inplace=True)

    lambda_i = lambdas[lambdas['id'] == i][['hour', 'counts']].copy()
    all_hours = pd.DataFrame({'hour': range(5, 24)})  # Adjusted to include 5 am
    lambda_i = pd.merge(all_hours, lambda_i, on='hour', how='left').fillna(0)
    lambda_i.set_index('hour', inplace=True)
    unhappiness = []
    for x in range(0,capacity_i+1):
        unhappiness_with_current = calculate_unhappiness(x, mus_i, lambda_i, capacity_i, start_hour=5)
        unhappiness.append(unhappiness_with_current)

    if np.sum(unhappiness)==0:
        x_is.append(0)
    else:
        min_value = np.min(unhappiness)
        min_indices = np.where(unhappiness == min_value)[0]
        x_min = (min_indices[0] + min_indices[-1])/2
        x_is.append(round(x_min))

print(np.sum(x_is))
```

```
30720
```

In [125…

```
station_list['number_bikes_5am'] = x_is
```

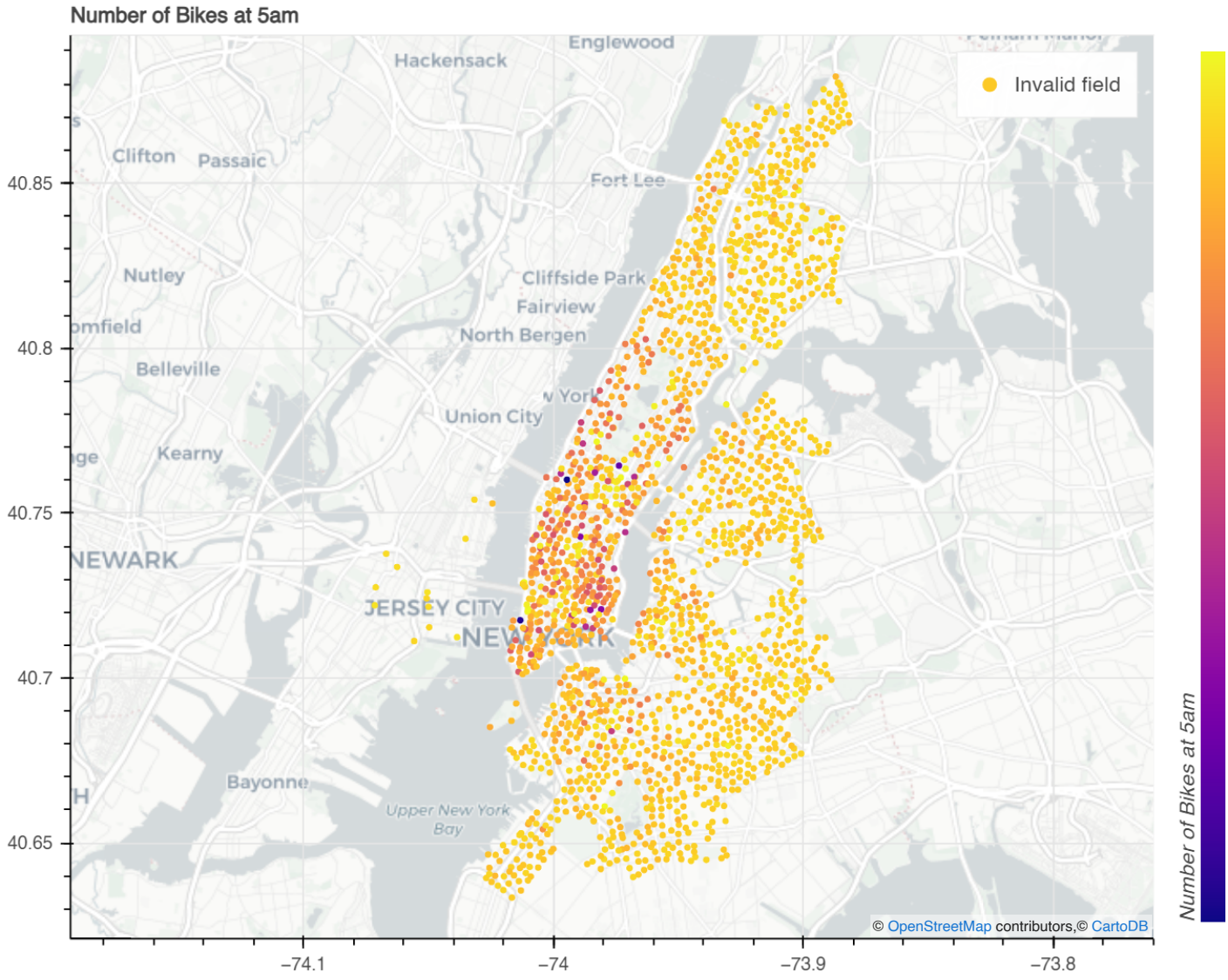
In [128…

```
def plotNetwork(nodes, title='Plot of Graph', on_map=True):
    output_notebook()

    # Prepare the data for plotting
    nodes_source = ColumnDataSource(nodes)

    # Create a color mapper for 'counts' column with a high and low bound
    color_mapper = LinearColorMapper(palette='Plasma256', low=max(nodes['number_bikes_5am']), high=min(r

    # Define the plot
    plot = figure(title=title,
                  x_axis_type="mercator", y_axis_type="mercator",
                  width=800, height=600,
                  toolbar_location=None, tools="pan,wheel_zoom,reset")

    # Add map tile if requested
    if on_map:
        plot.add_tile(get_provider(Vendors.CARTODBPOSITRON))

    # Add the nodes to the plot with color mapped by 'counts'
    plot.circle(x='x', y='y', size=3, color=transform('number_bikes_5am', color_mapper),
                legend_field='criteria', source=nodes_source)

    # Add a color bar to the side of the plot to show the color mapping
```

```
    color_bar = ColorBar(color_mapper=color_mapper, label_standoff=12, location=(0,0), title='Number of
    plot.add_layout(color_bar, 'right')

    # Show the plot
    show(layout(plot))

# Call the function with the prepared data
plotNetwork(nodes=station_list,  title='Number of Bikes at 5am')
```

Loading BokehJS ...

ERROR:bokeh.core.validation.check:E-1001 (BAD_COLUMN_NAME): Glyph refers to nonexistent column name. Th
is could either be due to a misspelling or typo, or due to an expected column being missing. : LegendIt
em(id='1570216', ...)



In [125…
```
print('Fleet Size', np.sum(station_list['number_bikes_5am']))
```

```
Fleet Size 30720
```

In [126…
```
# original logic written by me, bug was fixed by Chat GPT.
hours = range(6, 24)
column_names = [f'number_bikes_{hour}am' if hour <= 12 else f'number_bikes_{hour-12}pm' for hour in hour
df = pd.DataFrame(0, index=station_list['id'], columns=column_names)

unhappiness = []
for i in station_list['id']:
    capacity_i = int(capacity[capacity['end_station_id'] == str(i)]['capacity'].iloc[0])
```

```
    # Retrieve the counts of bikes taken and returned for each hour
    mus_i = mus[mus['id'] == i][['hour', 'counts']].copy()
    lambda_i = lambdas[lambdas['id'] == i][['hour', 'counts']].copy()

    # Ensure the DataFrame covers all hours from 5 am to 11 pm
    all_hours = pd.DataFrame({'hour': range(5, 24)})
    mus_i = pd.merge(all_hours, mus_i, on='hour', how='left').fillna(0).set_index('hour')
    lambda_i = pd.merge(all_hours, lambda_i, on='hour', how='left').fillna(0).set_index('hour')

    # Initial number of bikes at 5 am
    x_0 = float(station_list[station_list['id']==i]['number_bikes_5am'])

    xs = [x_0]
    y = 0
    for t in range(6, 24):  # Loop over each hour from 6 am to 11 pm
        lam = lambda_i.loc[t, 'counts']
        mu = mus_i.loc[t, 'counts']
        x_t = xs[-1] + lam - mu  # Calculate the new number of bikes

        # Clamp x_t to be within the station capacity
        x_t_clamped = max(min(x_t, capacity_i), 0)
        xs.append(x_t_clamped)

        # Calculate failed returns and pickups
        failed_returns = max(x_t - capacity_i, 0)
        failed_pickups = max(-x_t, 0)

        y += failed_returns + failed_pickups  # Update unhappiness score

    df.loc[i] = xs[1:]  # Store the number of bikes for each hour, excluding the initial value
    unhappiness.append(y)  # Store the total unhappiness score for the station
```

In [126…

```
df_a = df.reset_index()
station_list = station_list.reset_index()
station_list = pd.concat([station_list,df_a],axis=1,join='inner')
station_list
```

Out[1261]:

	index	lat	lng	id	x	y	number_bikes_5am	id	number_bikes_6a
0	0	40.811432	-73.951878	7756.10	-8.232285e+06	4.984568e+06	13	7756.10	13.9712
1	1	40.804372	-73.951475	7670.09	-8.232241e+06	4.983529e+06	12	7670.09	10.3387
2	6	40.827075	-73.945909	8033.09	-8.231621e+06	4.986869e+06	14	8033.09	13.0959
3	7	40.824814	-73.951868	7981.16	-8.232284e+06	4.986536e+06	14	7981.16	12.6836
4	8	40.781257	-73.949838	7286.01	-8.232058e+06	4.980130e+06	42	7286.01	37.4437
…	…	…	…	…	…	…	…	…	
1916	2029927	40.726012	-74.050389	JC081	-8.243252e+06	4.972012e+06	6	JC081	6.0000
1917	3038654	40.721630	-74.049968	JC076	-8.243205e+06	4.971368e+06	8	JC076	8.0000
1918	3073462	40.752961	-74.024353	HB202	-8.240353e+06	4.975971e+06	16	HB202	16.0000
1919	3169641	40.722104	-74.071455	JC095	-8.245597e+06	4.971438e+06	6	JC095	6.0000
1920	3178380	40.711242	-74.055701	JC052	-8.243843e+06	4.969843e+06	10	JC052	10.0000

1921 rows × 26 columns

In [126…

```
np.sum(station_list,axis=0)
```

Out[1262]:

```
index                                                130281684
lat                                                  78266.618544
lng                                                -142063.053667
id                  7756.107670.098033.097981.167286.015669.106303...
x                                                -15814386794.771019
y                                                  9555936766.050743
number_bikes_5am                                             30720
id                  7756.107670.098033.097981.167286.015669.106303...
number_bikes_6am                                     30332.702003
number_bikes_7am                                     29788.085261
number_bikes_8am                                     29163.304085
number_bikes_9am                                     29450.181354
number_bikes_10am                                    29429.779867
number_bikes_11am                                    29264.025569
number_bikes_12am                                    29186.642413
number_bikes_1pm                                     29127.899149
number_bikes_2pm                                     29124.816867
number_bikes_3pm                                     29181.015462
number_bikes_4pm                                      29007.47784
number_bikes_5pm                                     28982.545844
number_bikes_6pm                                     29715.196875
number_bikes_7pm                                     30600.942741
number_bikes_8pm                                     31451.947586
number_bikes_9pm                                     32084.621006
number_bikes_10pm                                    32581.407754
number_bikes_11pm                                    33072.105293
dtype: object
```

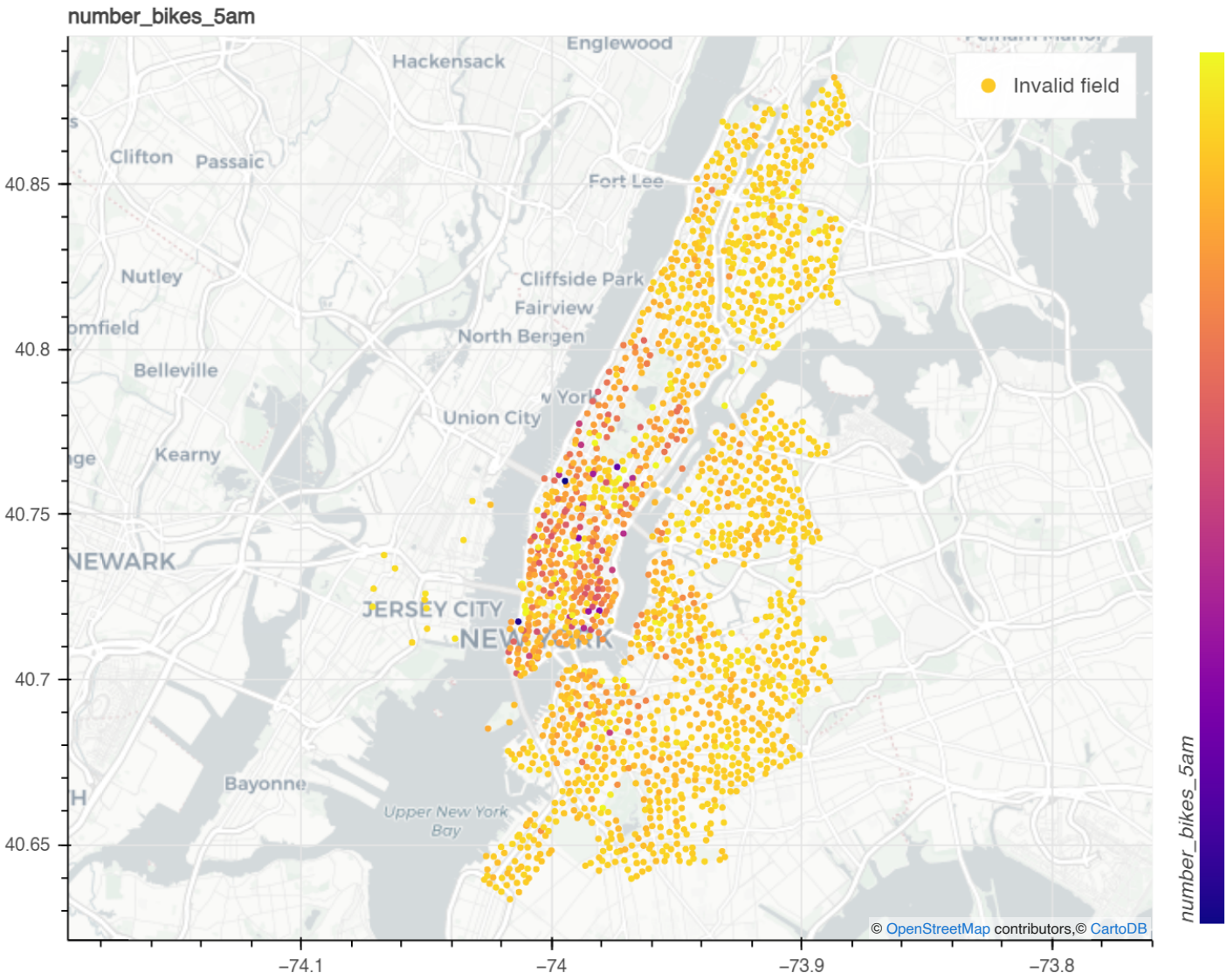
In [128…

```python
def plotNetwork(nodes, time, on_map=True):
    output_notebook()

    # Prepare the data for plotting
    nodes_source = ColumnDataSource(nodes)
    if time <= 12:
        time = f'number_bikes_{time}am'
    else:
        time =  f'number_bikes_{time-12}pm'
    print(time)
    # Create a color mapper for 'counts' column with a high and low bound
    color_mapper = LinearColorMapper(palette='Plasma256', low=max(nodes[time]), high=min(nodes[time]))

    # Define the plot
    plot = figure(title=time,
                  x_axis_type="mercator", y_axis_type="mercator",
                  width=800, height=600,
                  toolbar_location=None, tools="pan,wheel_zoom,reset")

    # Add map tile if requested
    if on_map:
        plot.add_tile(get_provider(Vendors.CARTODBPOSITRON))

    # Add the nodes to the plot with color mapped by 'counts'
    plot.circle(x='x', y='y', size=3, color=transform(time, color_mapper),
                legend_field='criteria', source=nodes_source)

    # Add a color bar to the side of the plot to show the color mapping
    color_bar = ColorBar(color_mapper=color_mapper, label_standoff=12, location=(0,0), title=time)
    plot.add_layout(color_bar, 'right')

    # Show the plot
    show(layout(plot))

for time in range(5,24):
# Call the function with the prepared data
    plotNetwork(nodes=station_list, time=time)
```

Loading BokehJS ...

number_bikes_5am

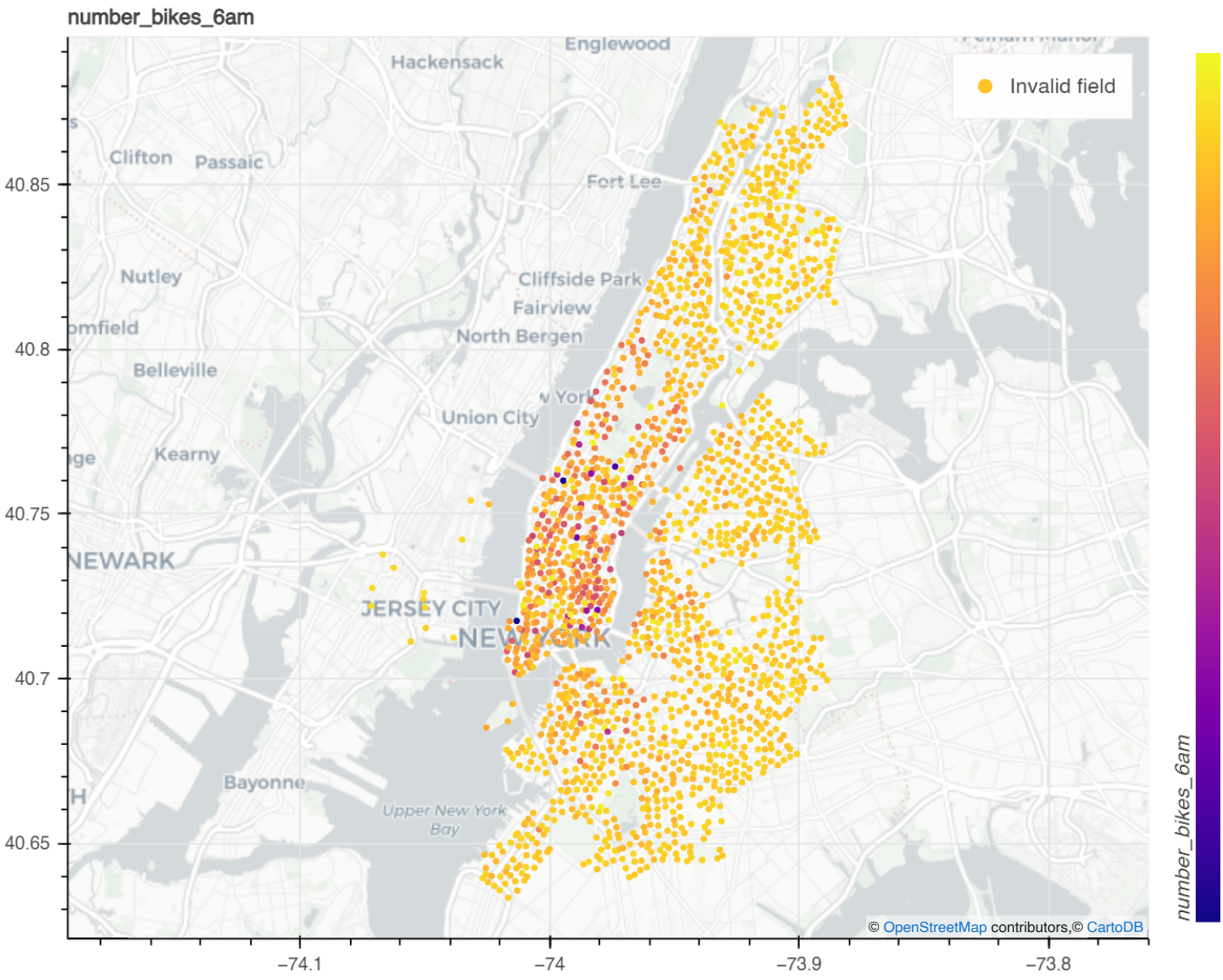
```
ERROR:bokeh.core.validation.check:E-1001 (BAD_COLUMN_NAME): Glyph refers to nonexistent column name. Th
is could either be due to a misspelling or typo, or due to an expected column being missing. : LegendIt
em(id='1465241', ...)
```



Loading BokehJS ...

number_bikes_6am

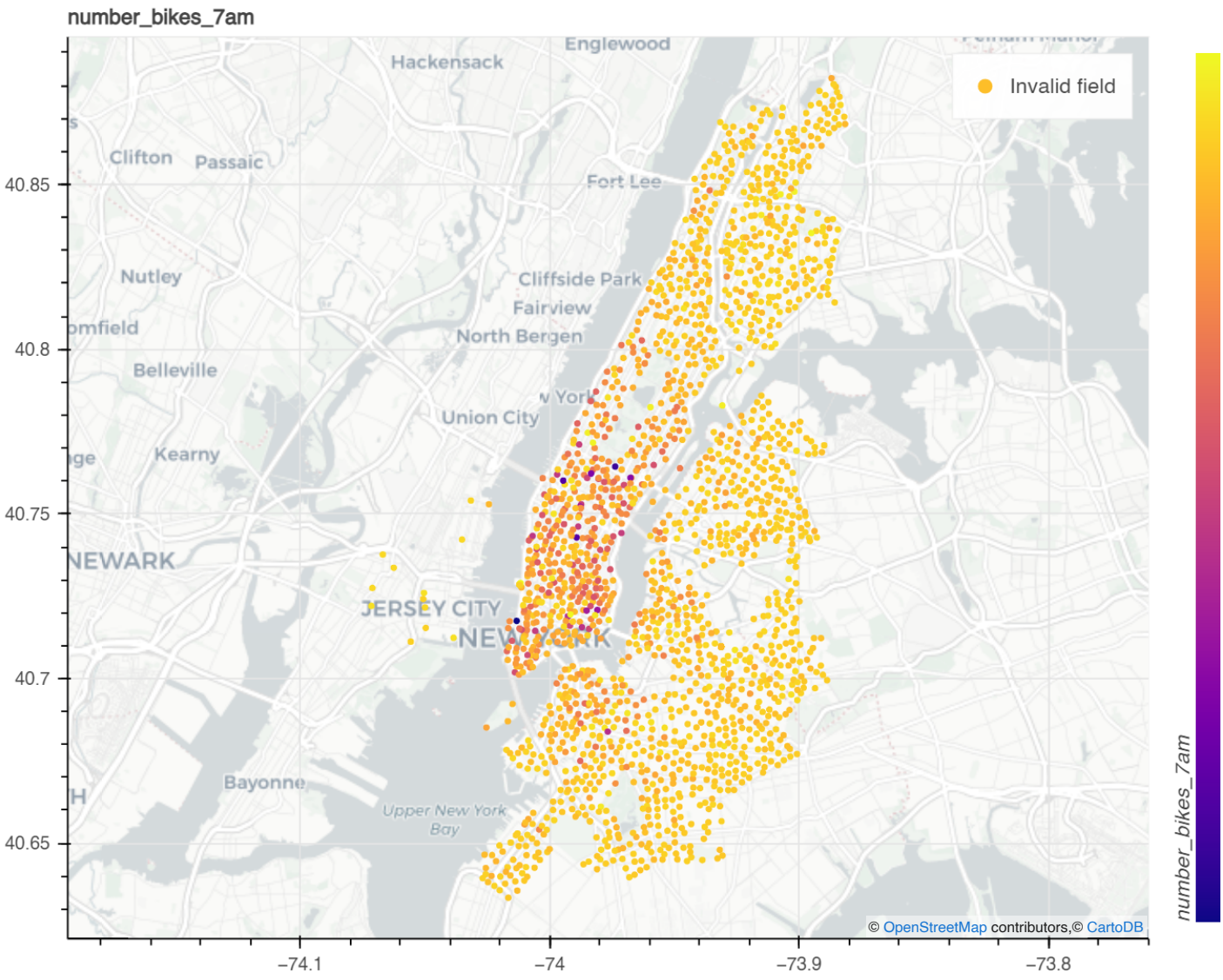
```
ERROR:bokeh.core.validation.check:E-1001 (BAD_COLUMN_NAME): Glyph refers to nonexistent column name. Th
is could either be due to a misspelling or typo, or due to an expected column being missing. : LegendIt
em(id='1470676', ...)
```



BokehJS 2.4.3 successfully loaded.

```
number_bikes_7am
```

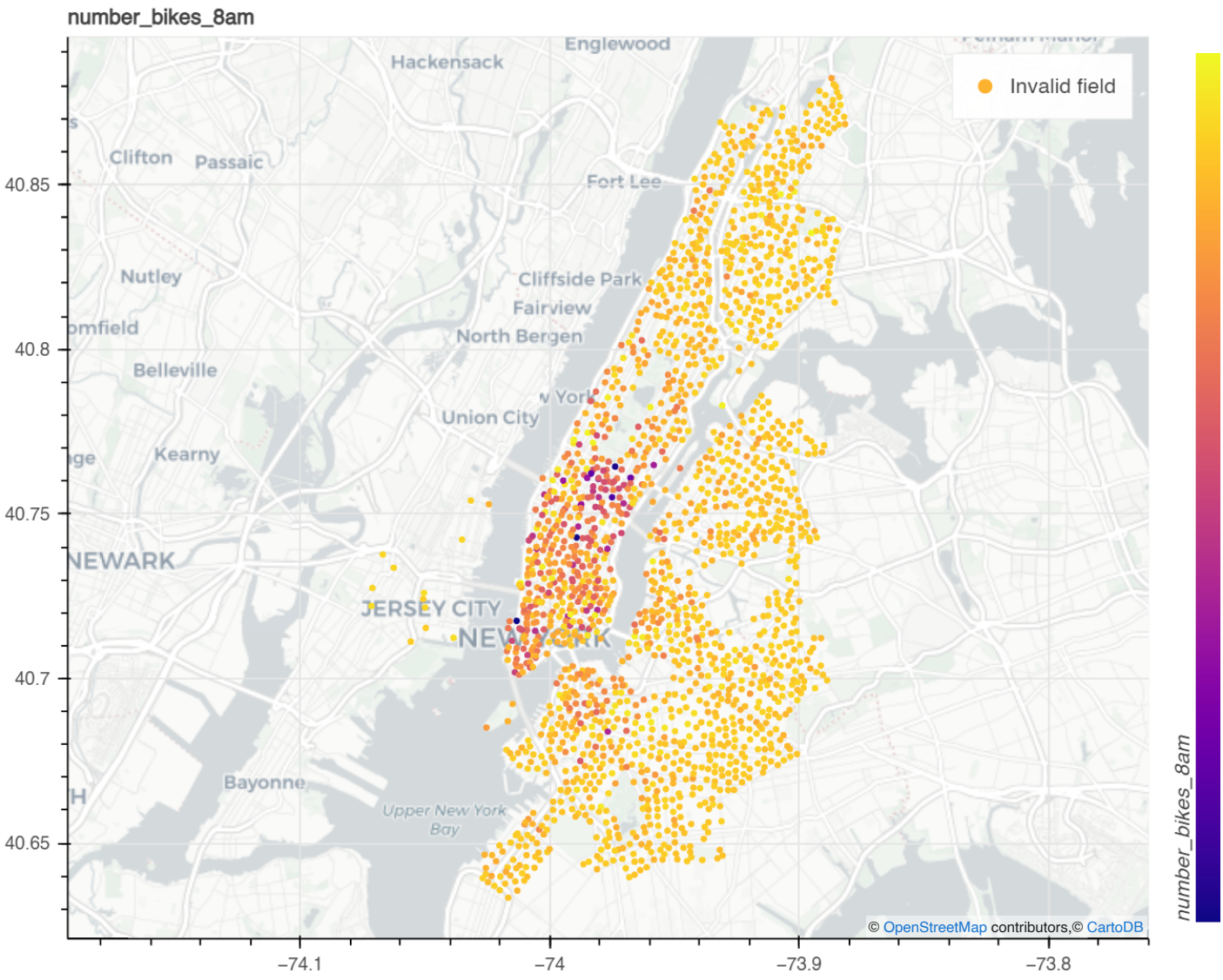
```
ERROR:bokeh.core.validation.check:E-1001 (BAD_COLUMN_NAME): Glyph refers to nonexistent column name. Th
is could either be due to a misspelling or typo, or due to an expected column being missing. : LegendIt
em(id='1476121', ...)
```



BokehJS 2.4.3 successfully loaded.

```
number_bikes_8am
```

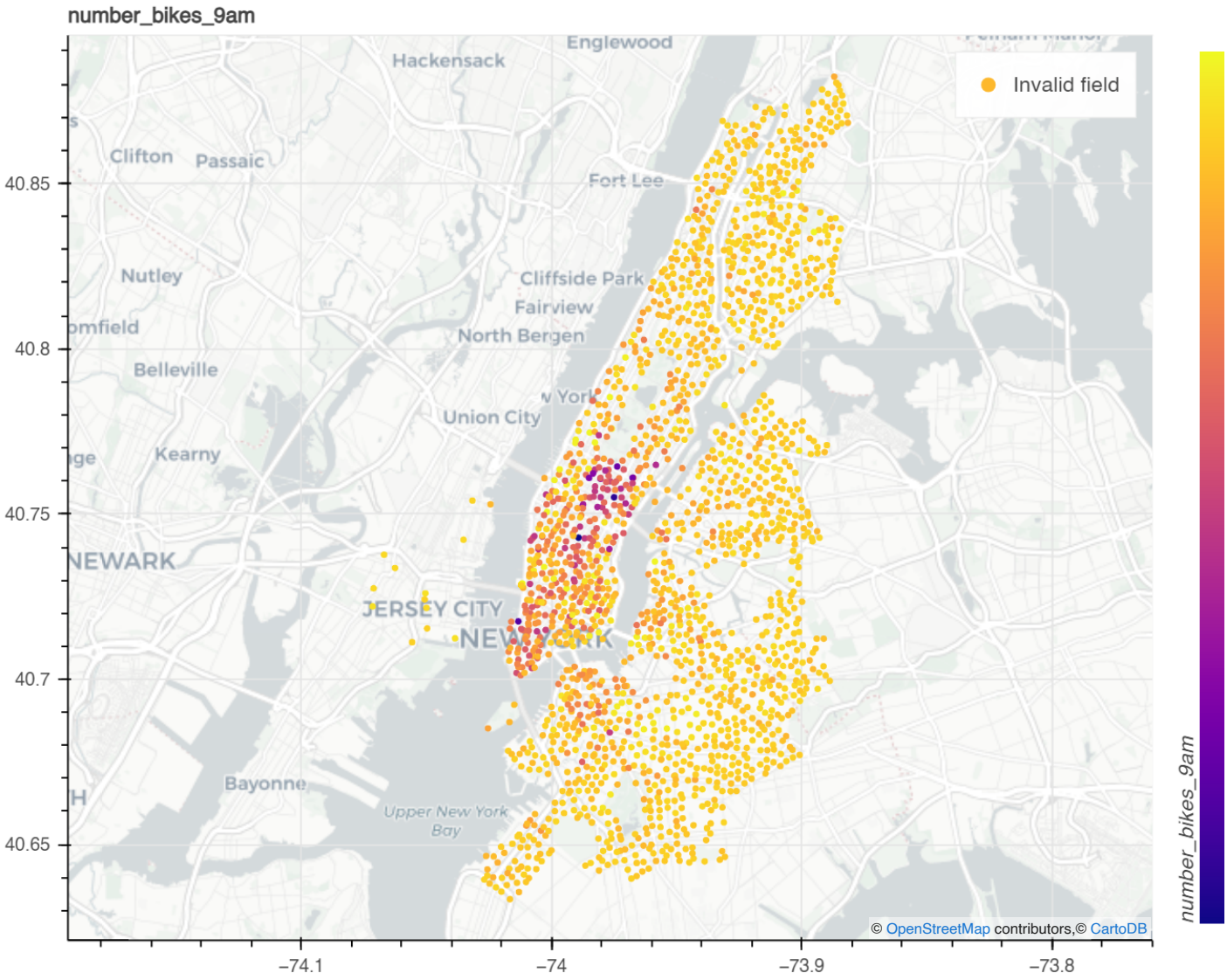
```
ERROR:bokeh.core.validation.check:E-1001 (BAD_COLUMN_NAME): Glyph refers to nonexistent column name. Th
is could either be due to a misspelling or typo, or due to an expected column being missing. : LegendIt
em(id='1481576', ...)
```



BokehJS 2.4.3 successfully loaded.

```
number_bikes_9am
```

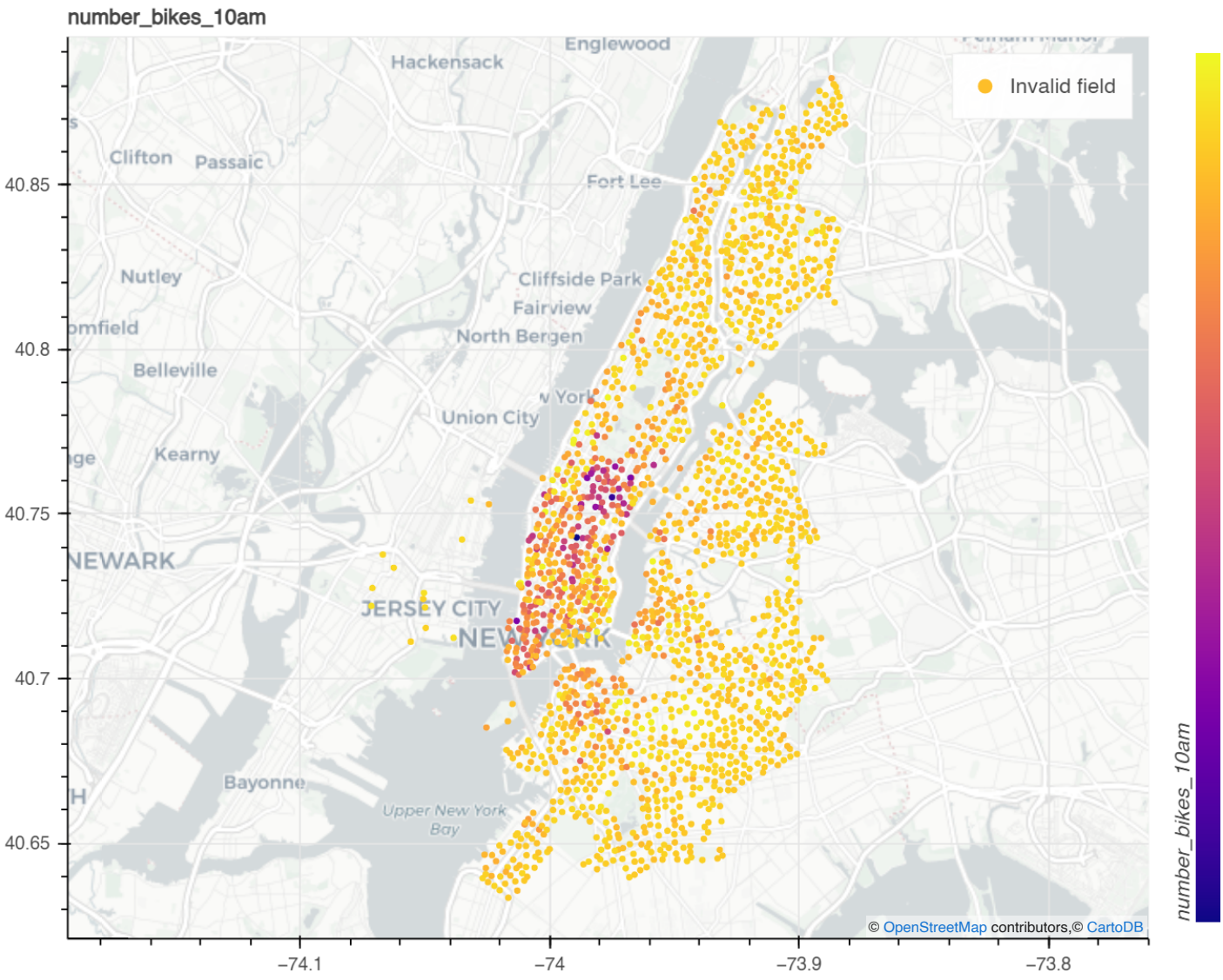
```
ERROR:bokeh.core.validation.check:E-1001 (BAD_COLUMN_NAME): Glyph refers to nonexistent column name. Th
is could either be due to a misspelling or typo, or due to an expected column being missing. : LegendIt
em(id='1487041', ...)
```



BokehJS 2.4.3 successfully loaded.

```
number_bikes_10am
```

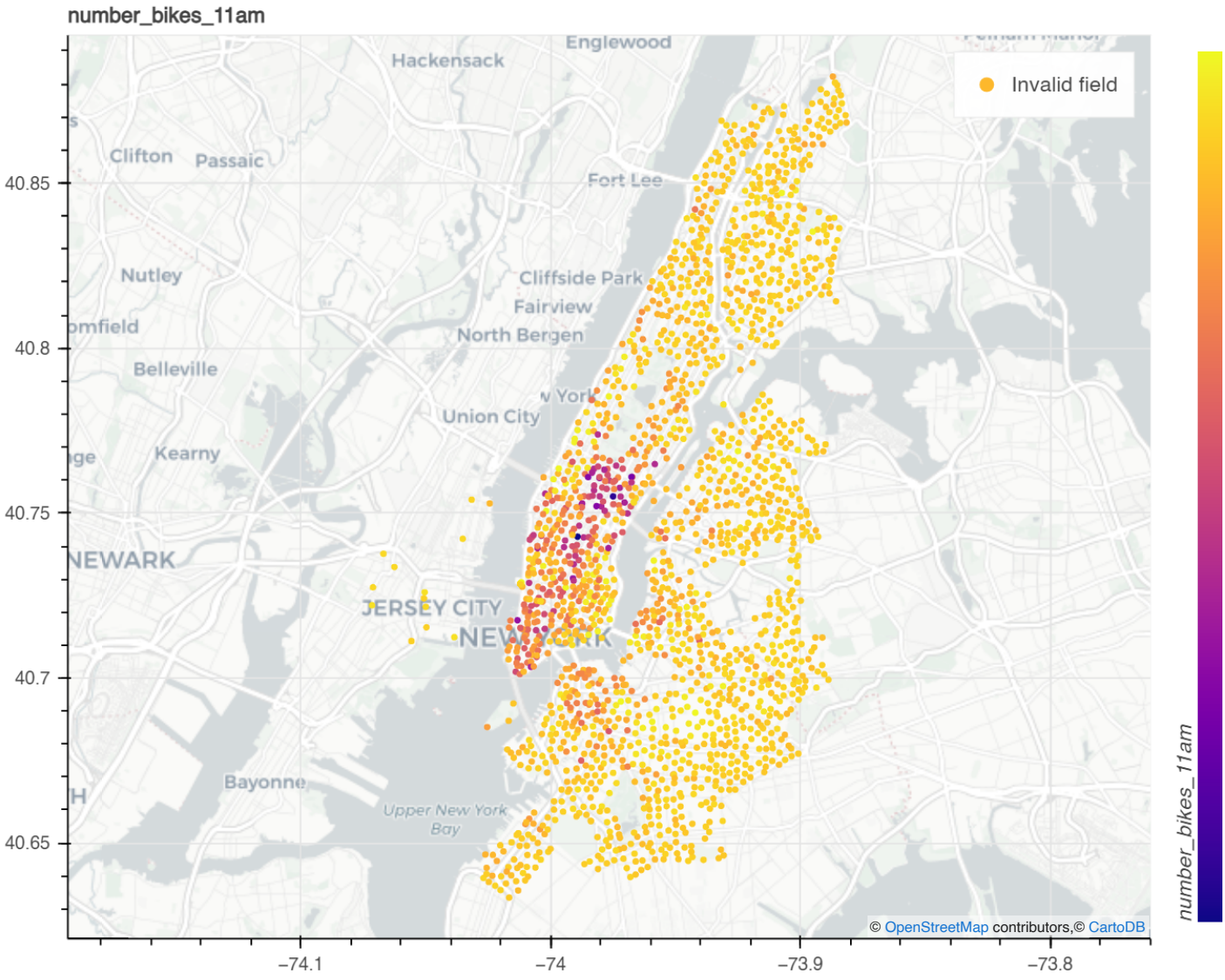
```
ERROR:bokeh.core.validation.check:E-1001 (BAD_COLUMN_NAME): Glyph refers to nonexistent column name. Th
is could either be due to a misspelling or typo, or due to an expected column being missing. : LegendIt
em(id='1492516', ...)
```



BokehJS 2.4.3 successfully loaded.

```
number_bikes_11am
```

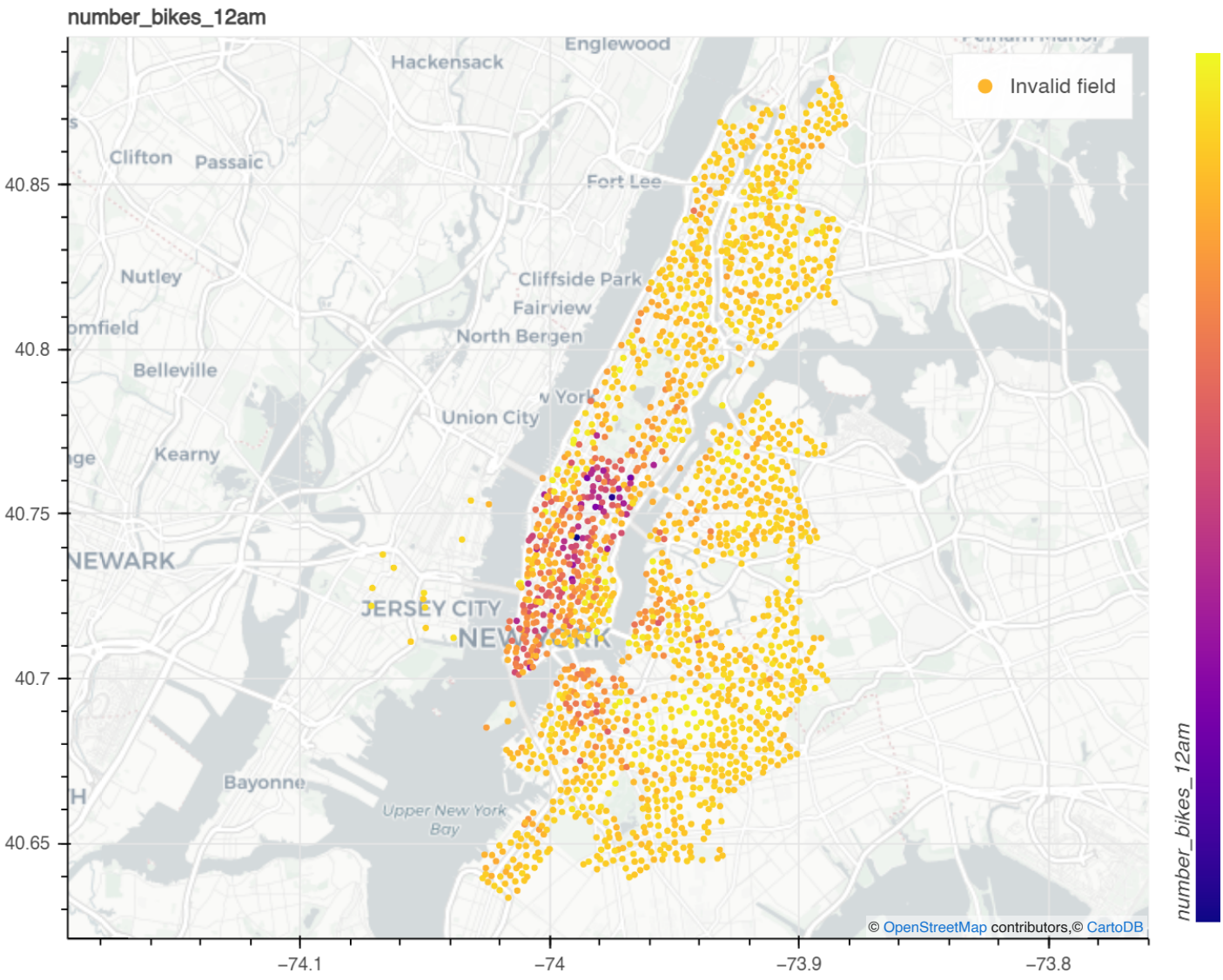
```
ERROR:bokeh.core.validation.check:E-1001 (BAD_COLUMN_NAME): Glyph refers to nonexistent column name. Th
is could either be due to a misspelling or typo, or due to an expected column being missing. : LegendIt
em(id='1498001', ...)
```



BokehJS 2.4.3 successfully loaded.

```
number_bikes_12am
```

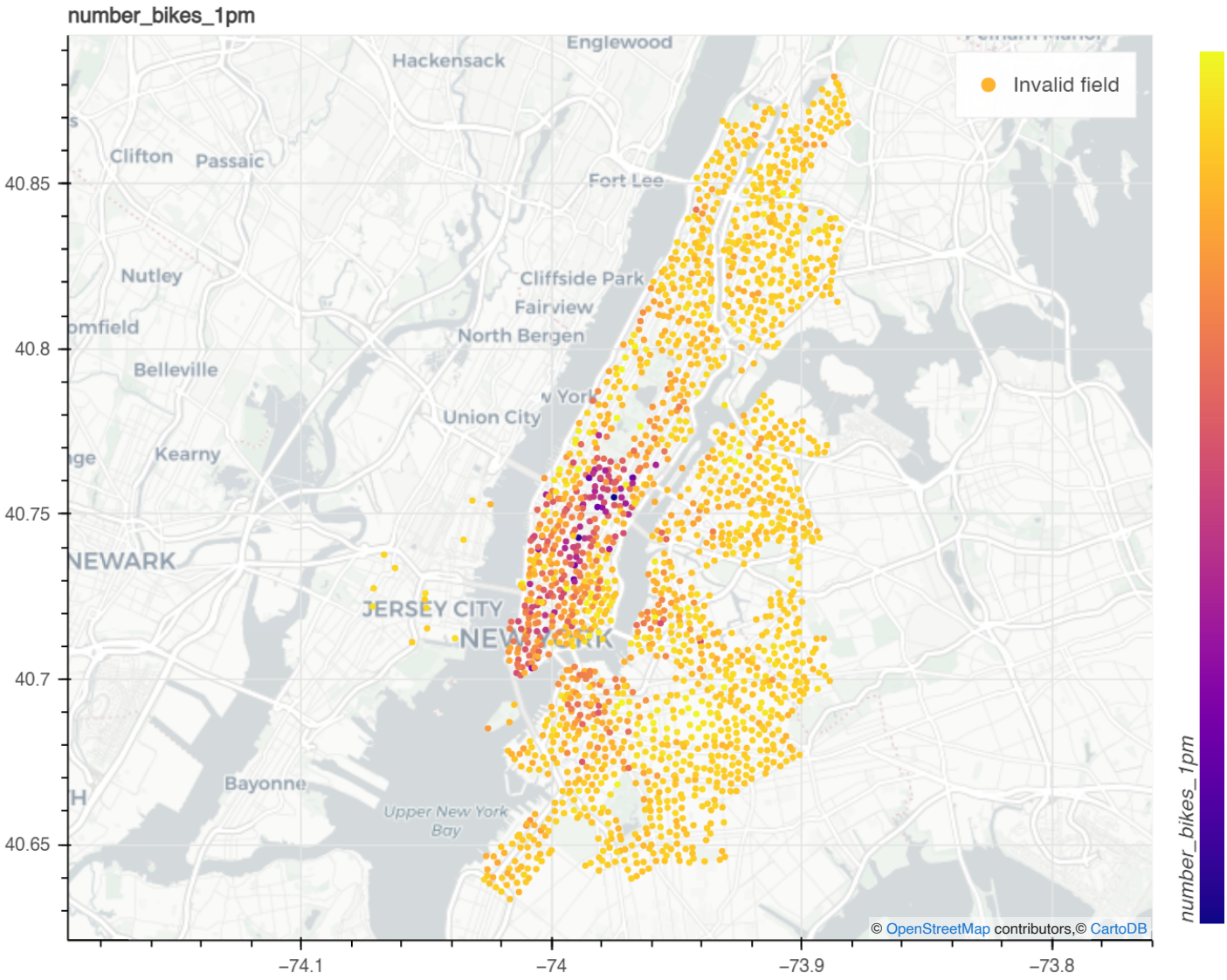
```
ERROR:bokeh.core.validation.check:E-1001 (BAD_COLUMN_NAME): Glyph refers to nonexistent column name. Th
is could either be due to a misspelling or typo, or due to an expected column being missing. : LegendIt
em(id='1503496', ...)
```



BokehJS 2.4.3 successfully loaded.

```
number_bikes_1pm
```

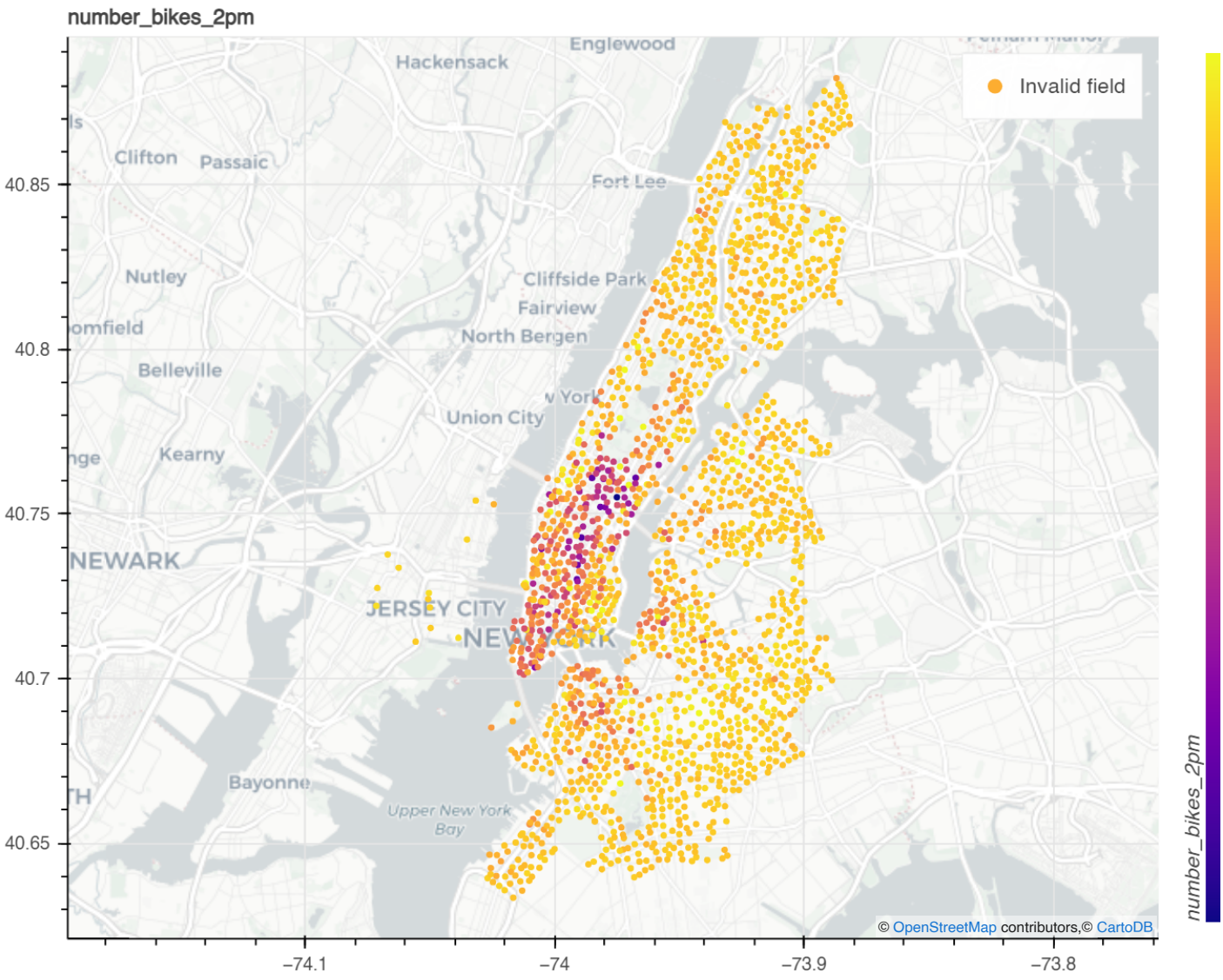
```
ERROR:bokeh.core.validation.check:E-1001 (BAD_COLUMN_NAME): Glyph refers to nonexistent column name. Th
is could either be due to a misspelling or typo, or due to an expected column being missing. : LegendIt
em(id='1509001', ...)
```



BokehJS 2.4.3 successfully loaded.

```
number_bikes_2pm
```

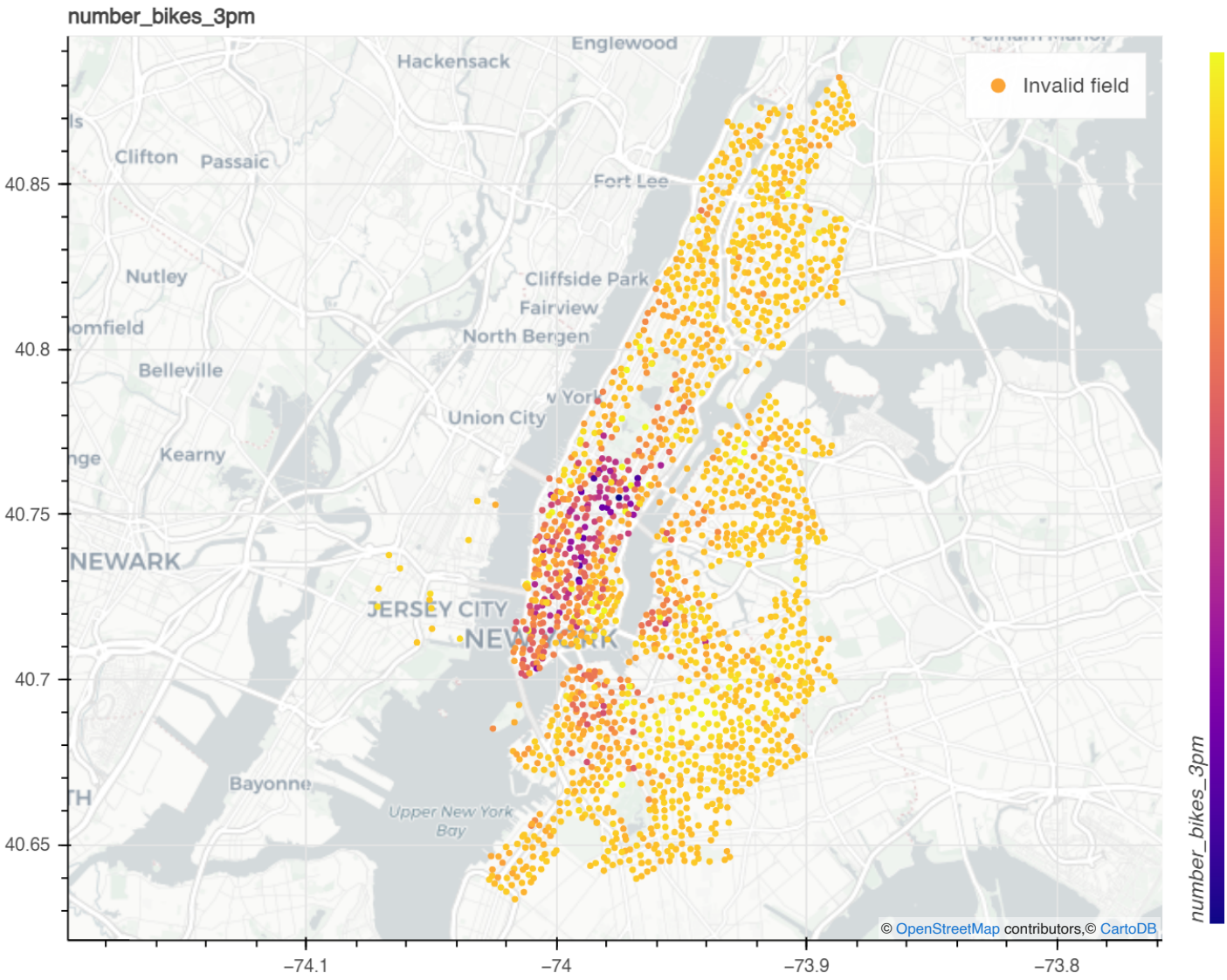
```
ERROR:bokeh.core.validation.check:E-1001 (BAD_COLUMN_NAME): Glyph refers to nonexistent column name. Th
is could either be due to a misspelling or typo, or due to an expected column being missing. : LegendIt
em(id='1514516', ...)
```



BokehJS 2.4.3 successfully loaded.

```
number_bikes_3pm
```

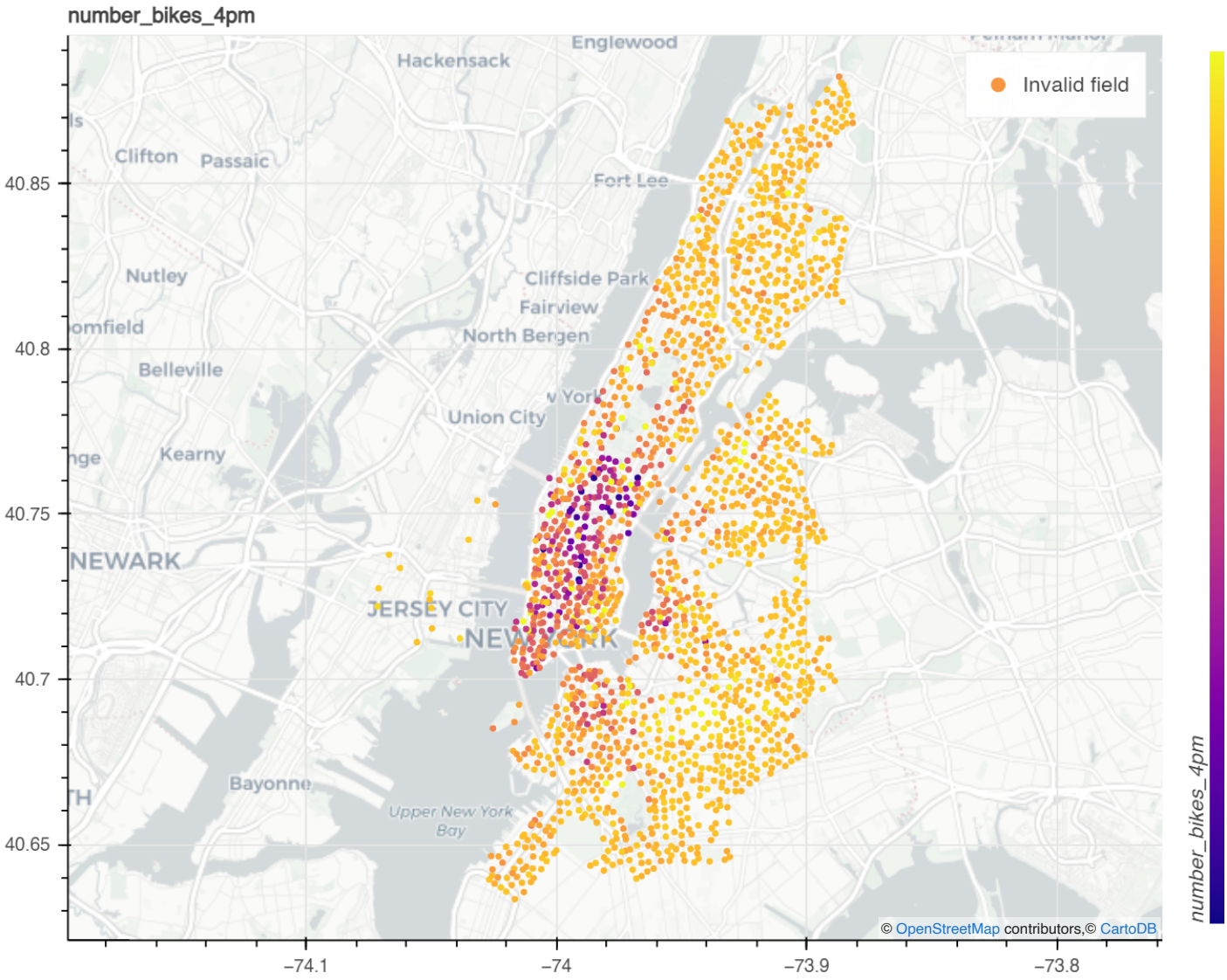
```
ERROR:bokeh.core.validation.check:E-1001 (BAD_COLUMN_NAME): Glyph refers to nonexistent column name. Th
is could either be due to a misspelling or typo, or due to an expected column being missing. : LegendIt
em(id='1520041', ...)
```



BokehJS 2.4.3 successfully loaded.

```
number_bikes_4pm
```

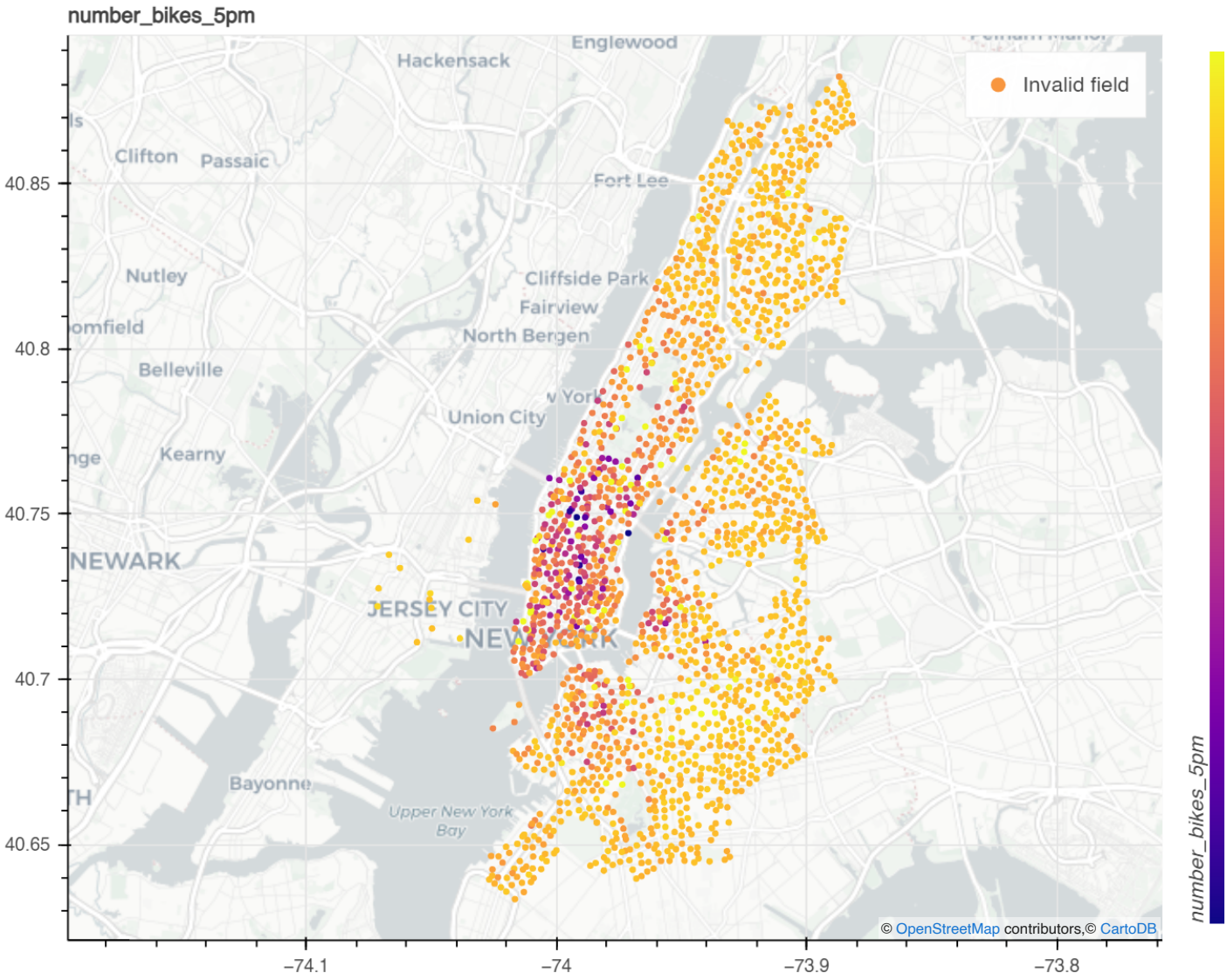
```
ERROR:bokeh.core.validation.check:E-1001 (BAD_COLUMN_NAME): Glyph refers to nonexistent column name. Th
is could either be due to a misspelling or typo, or due to an expected column being missing. : LegendIt
em(id='1525576', ...)
```



BokehJS 2.4.3 successfully loaded.

```
number_bikes_5pm
```

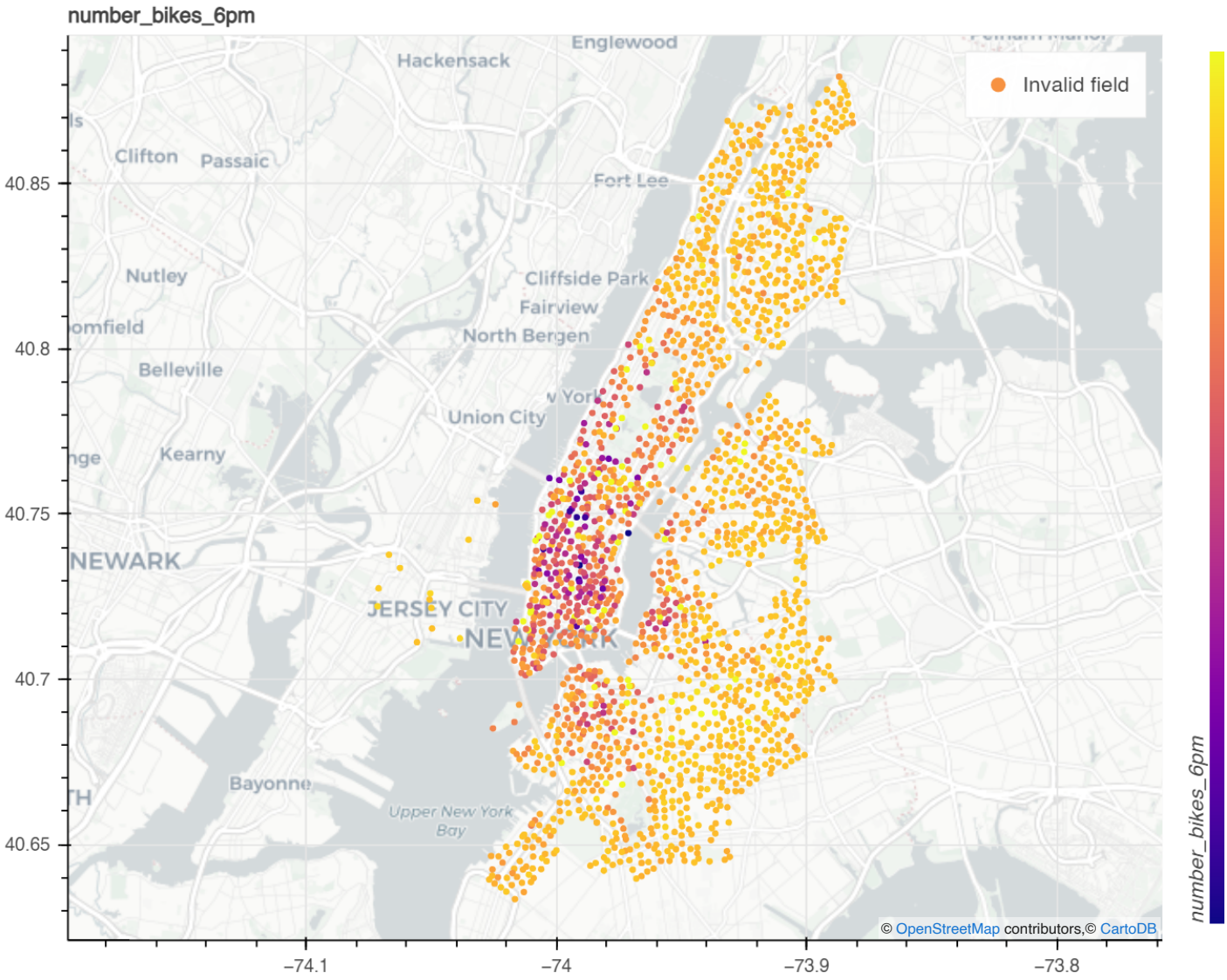
```
ERROR:bokeh.core.validation.check:E-1001 (BAD_COLUMN_NAME): Glyph refers to nonexistent column name. Th
is could either be due to a misspelling or typo, or due to an expected column being missing. : LegendIt
em(id='1531121', ...)
```



BokehJS 2.4.3 successfully loaded.

```
number_bikes_6pm
```

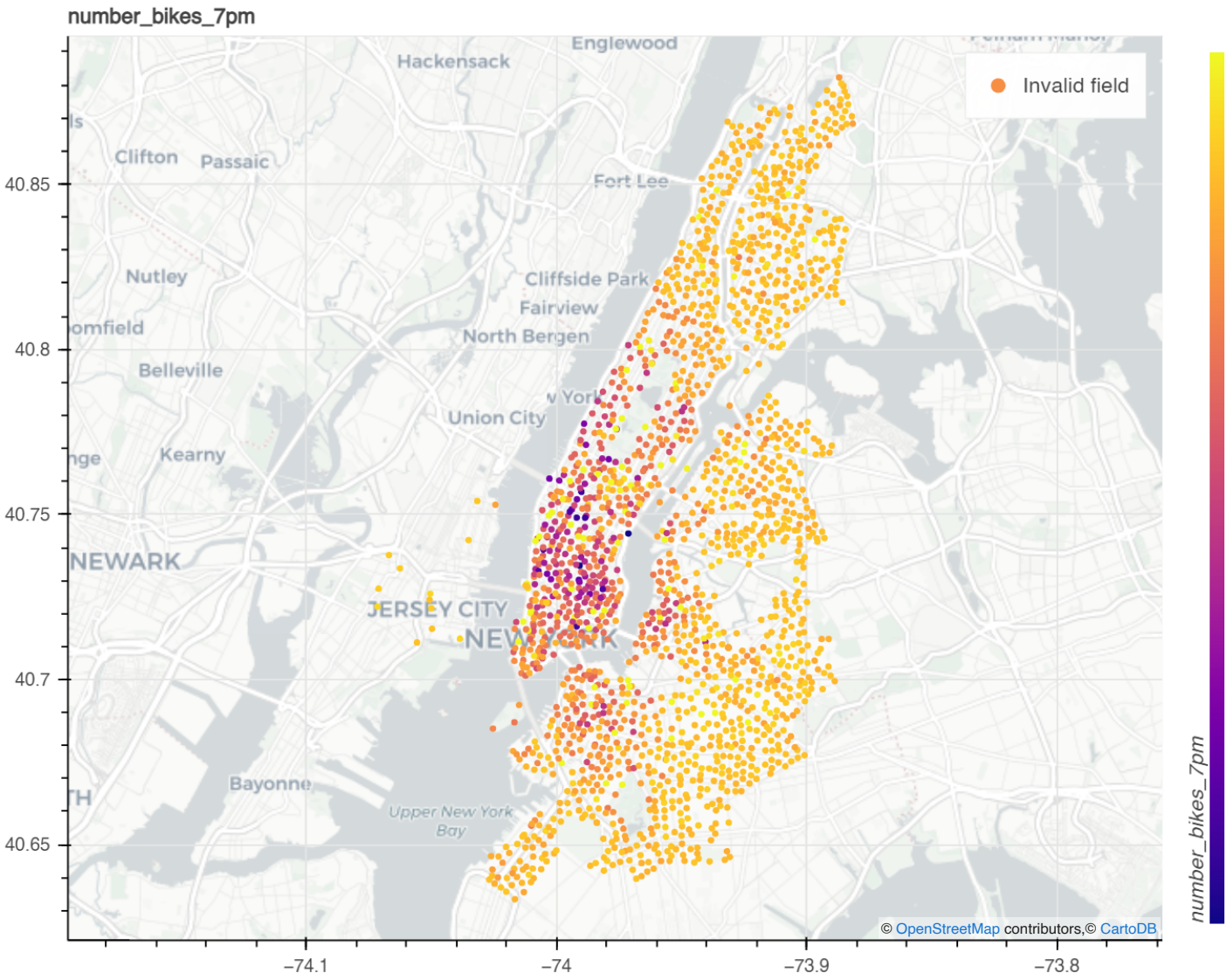
```
ERROR:bokeh.core.validation.check:E-1001 (BAD_COLUMN_NAME): Glyph refers to nonexistent column name. Th
is could either be due to a misspelling or typo, or due to an expected column being missing. : LegendIt
em(id='1536676', ...)
```



BokehJS 2.4.3 successfully loaded.

```
number_bikes_7pm
```

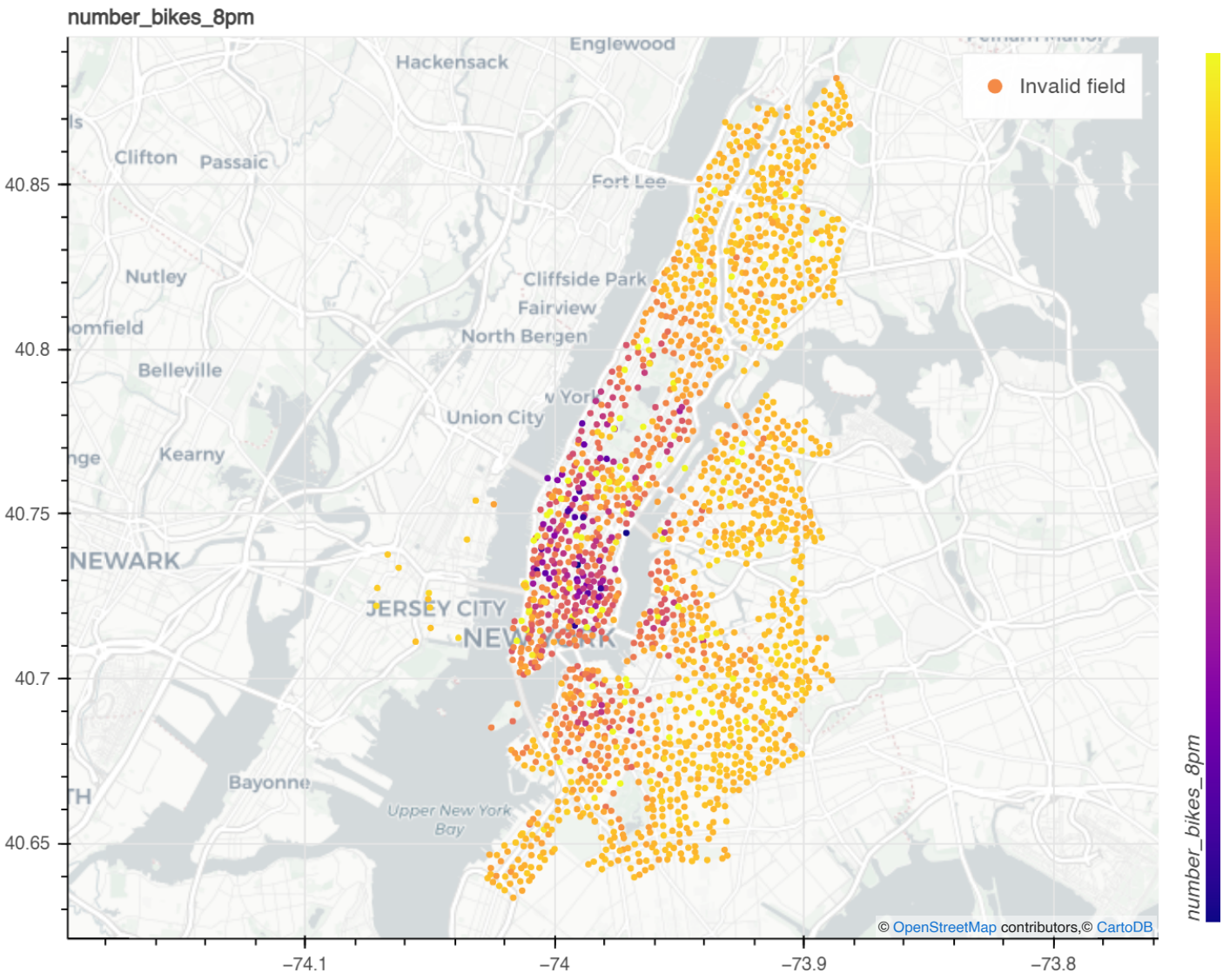
```
ERROR:bokeh.core.validation.check:E-1001 (BAD_COLUMN_NAME): Glyph refers to nonexistent column name. Th
is could either be due to a misspelling or typo, or due to an expected column being missing. : LegendIt
em(id='1542241', ...)
```



BokehJS 2.4.3 successfully loaded.

```
number_bikes_8pm
```

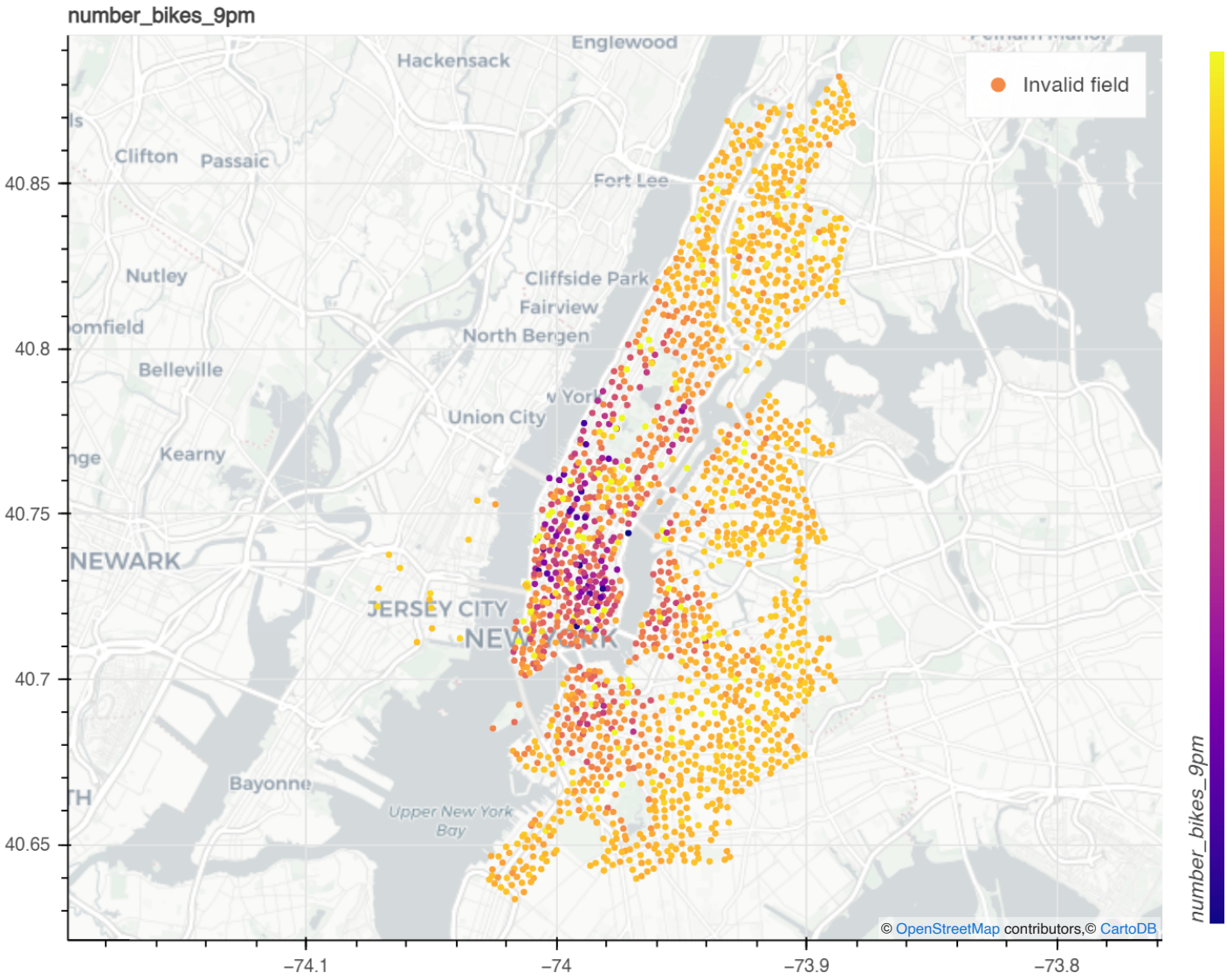
```
ERROR:bokeh.core.validation.check:E-1001 (BAD_COLUMN_NAME): Glyph refers to nonexistent column name. Th
is could either be due to a misspelling or typo, or due to an expected column being missing. : LegendIt
em(id='1547816', ...)
```



BokehJS 2.4.3 successfully loaded.

```
number_bikes_9pm
```

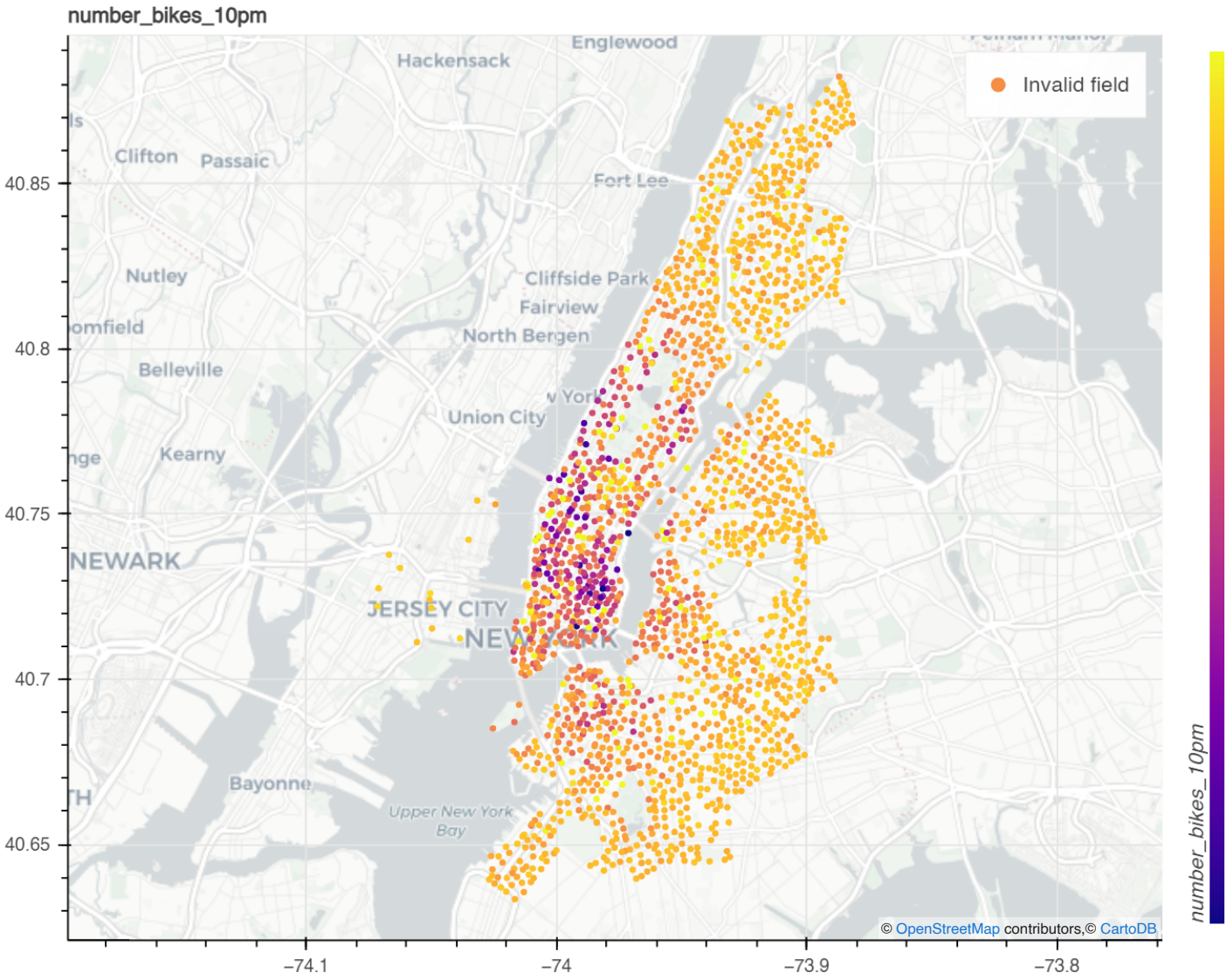
```
ERROR:bokeh.core.validation.check:E-1001 (BAD_COLUMN_NAME): Glyph refers to nonexistent column name. Th
is could either be due to a misspelling or typo, or due to an expected column being missing. : LegendIt
em(id='1553401', ...)
```



BokehJS 2.4.3 successfully loaded.

```
number_bikes_10pm
```

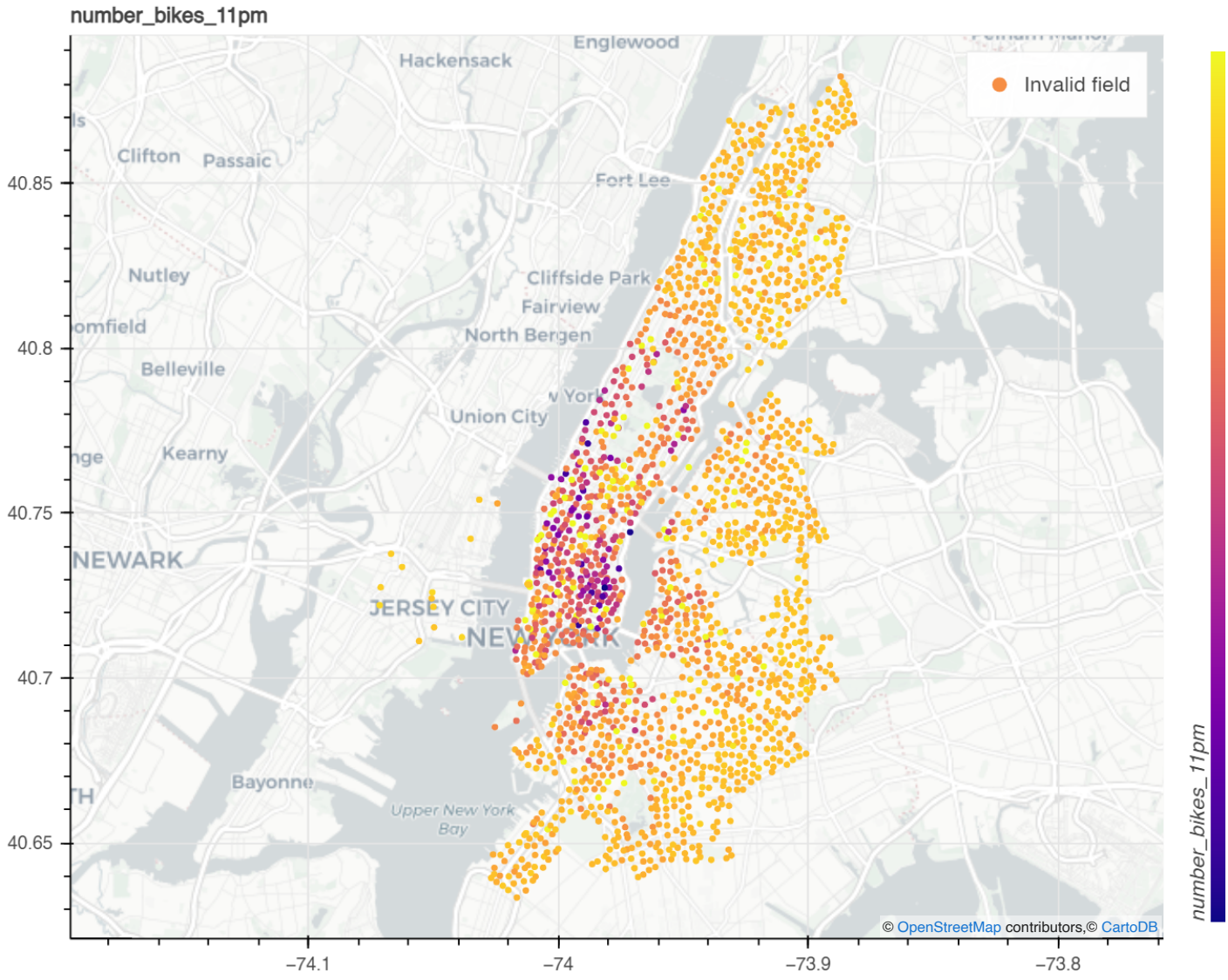
```
ERROR:bokeh.core.validation.check:E-1001 (BAD_COLUMN_NAME): Glyph refers to nonexistent column name. Th
is could either be due to a misspelling or typo, or due to an expected column being missing. : LegendIt
em(id='1558996', ...)
```



BokehJS 2.4.3 successfully loaded.

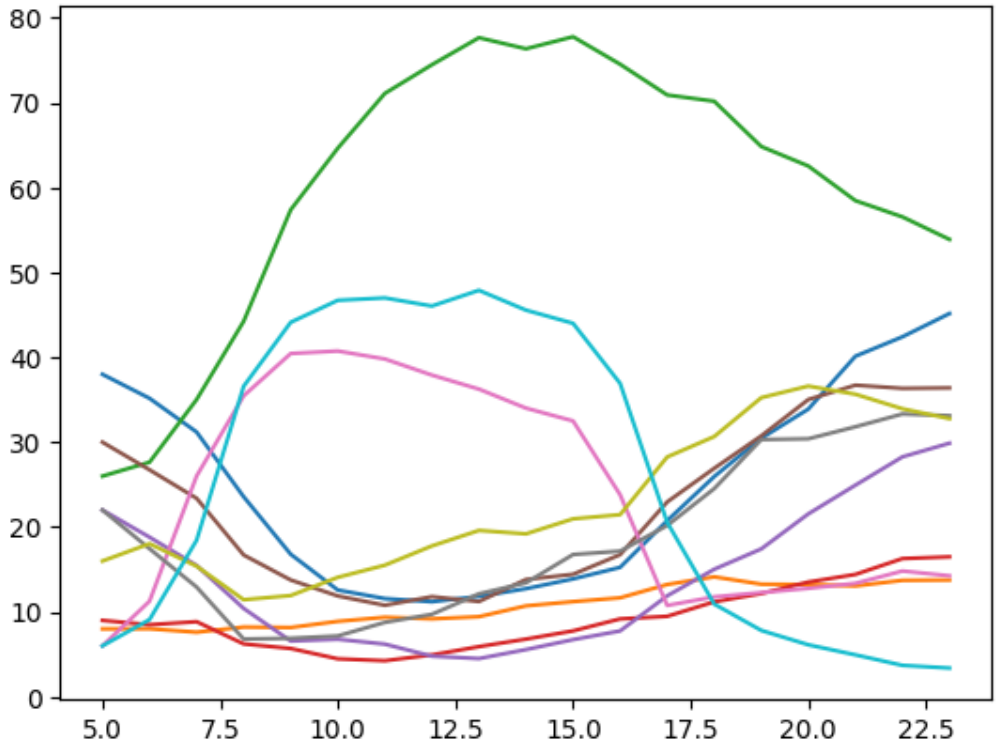
```
number_bikes_11pm
```

```
ERROR:bokeh.core.validation.check:E-1001 (BAD_COLUMN_NAME): Glyph refers to nonexistent column name. Th
is could either be due to a misspelling or typo, or due to an expected column being missing. : LegendIt
em(id='1564601', ...)
```



In [126…

```
plot_stations = station_list[['id','number_bikes_5am',
       'number_bikes_6am', 'number_bikes_7am', 'number_bikes_8am',
       'number_bikes_9am', 'number_bikes_10am', 'number_bikes_11am',
       'number_bikes_12am', 'number_bikes_1pm', 'number_bikes_2pm',
       'number_bikes_3pm', 'number_bikes_4pm', 'number_bikes_5pm',
       'number_bikes_6pm', 'number_bikes_7pm', 'number_bikes_8pm',
       'number_bikes_9pm', 'number_bikes_10pm', 'number_bikes_11pm']].iloc[40:50,:]

for i in range(len(plot_stations)):
    plt.plot(range(5,24),plot_stations.iloc[i,2:])
```



error because of model assumptions that bikes dissapear and appear (no rerouting to other stations when failed returns or failed pickups) that we end the day with more bikes than we started with. will assume that the end of the day bikes are the same as the start of day bikes.

In [126…

```
# This code will normalize the columns that start with 'numb' so that their sum equals the sum of 'numbe
sum_number_bikes_5am = station_list['number_bikes_5am'].sum()

# Normalize the columns
for i in station_list.columns:
    if i.startswith('numb'):
        station_list[i] = (station_list[i] / station_list[i].sum()) * sum_number_bikes_5am

np.sum(station_list,axis=0)
```

Out[1265]:

```
index                                                130281684
lat                                                  78266.618544
lng                                                 -142063.053667
id                  7756.107670.098033.097981.167286.015669.106303...
x                                                -15814386794.771019
y                                                  9555936766.050743
number_bikes_5am                                             30720.0
id                  7756.107670.098033.097981.167286.015669.106303...
number_bikes_6am                                             30720.0
number_bikes_7am                                             30720.0
number_bikes_8am                                             30720.0
number_bikes_9am                                             30720.0
number_bikes_10am                                            30720.0
number_bikes_11am                                            30720.0
number_bikes_12am                                            30720.0
number_bikes_1pm                                             30720.0
number_bikes_2pm                                             30720.0
number_bikes_3pm                                             30720.0
number_bikes_4pm                                             30720.0
number_bikes_5pm                                             30720.0
number_bikes_6pm                                             30720.0
number_bikes_7pm                                             30720.0
number_bikes_8pm                                             30720.0
number_bikes_9pm                                             30720.0
number_bikes_10pm                                            30720.0
number_bikes_11pm                                            30720.0
dtype: object
```

In [128…

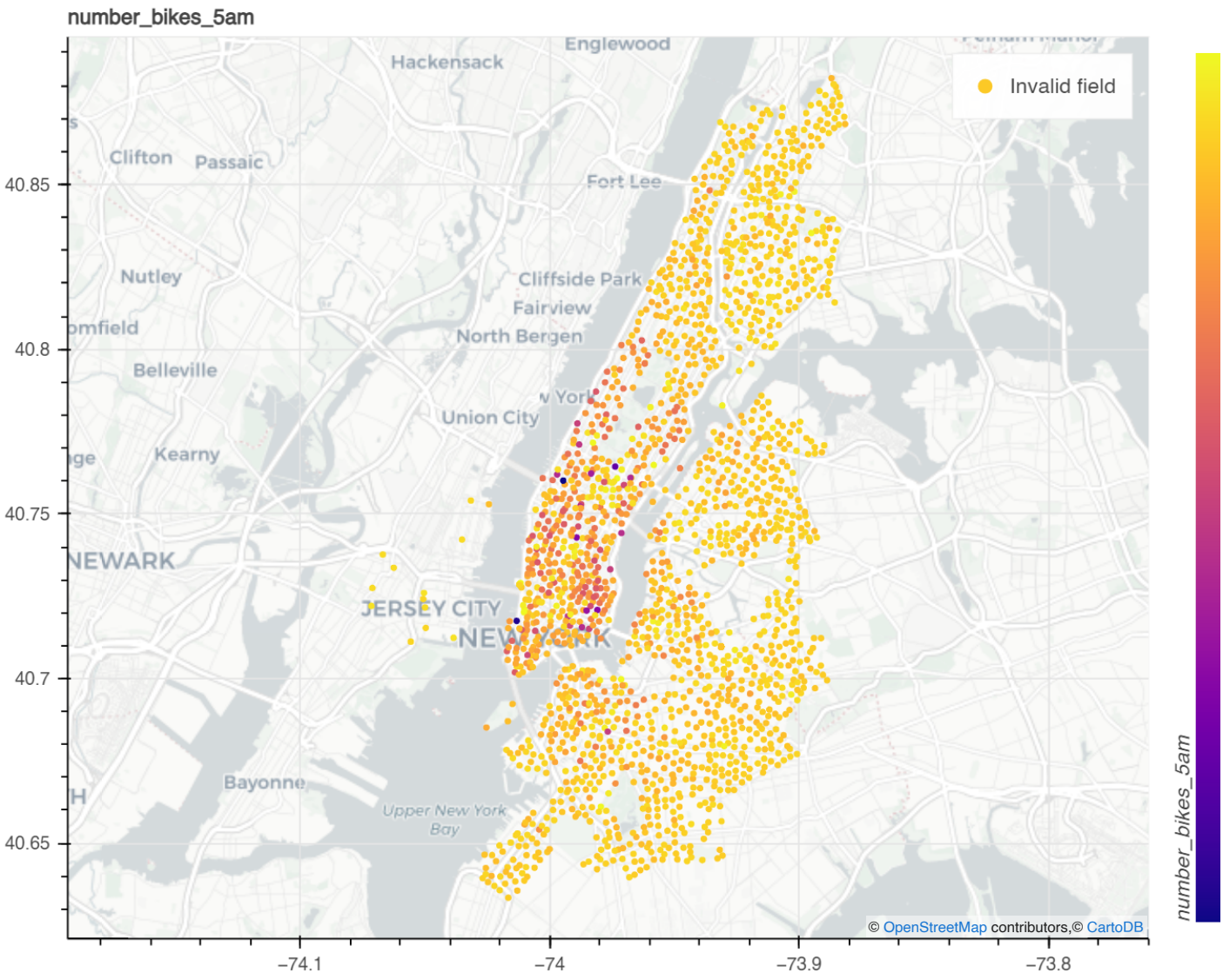
```
!pip install selenium
from bokeh.io import export_png
def plotNetwork(nodes, time, on_map=True):
    output_notebook()

    # Prepare the data for plotting
    nodes_source = ColumnDataSource(nodes)
    if time <= 12:
        time = f'number_bikes_{time}am'
    else:
        time =  f'number_bikes_{time-12}pm'
    print(time)
    # Create a color mapper for 'counts' column with a high and low bound
    color_mapper = LinearColorMapper(palette='Plasma256', low=max(nodes[time]), high=min(nodes[time]))

    # Define the plot
    plot = figure(title=time,
                  x_axis_type="mercator", y_axis_type="mercator",
                  width=800, height=600,
                  toolbar_location=None, tools="pan,wheel_zoom,reset")

    # Add map tile if requested
    if on_map:
        plot.add_tile(get_provider(Vendors.CARTODBPOSITRON))

    # Add the nodes to the plot with color mapped by 'counts'
    plot.circle(x='x', y='y', size=3, color=transform(time, color_mapper),
                legend_field='criteria', source=nodes_source)

    # Add a color bar to the side of the plot to show the color mapping
    color_bar = ColorBar(color_mapper=color_mapper, label_standoff=12, location=(0,0), title=time)
    plot.add_layout(color_bar, 'right')

    # Show the plot
    show(layout(plot))
    #export_png(layout(plot), filename=f'{time}.png')


for time in range(5,24):
# Call the function with the prepared data
    plotNetwork(nodes=station_list, time=time)
```

```
Requirement already satisfied: selenium in c:\users\julia\anaconda3\envs\engri_1101\lib\site-packages
(4.11.2)
Requirement already satisfied: trio~=0.17 in c:\users\julia\anaconda3\envs\engri_1101\lib\site-packages
(from selenium) (0.22.2)
Requirement already satisfied: trio-websocket~=0.9 in c:\users\julia\anaconda3\envs\engri_1101\lib\site
-packages (from selenium) (0.11.1)
Requirement already satisfied: certifi>=2021.10.8 in c:\users\julia\anaconda3\envs\engri_1101\lib\site-
packages (from selenium) (2022.12.7)
Requirement already satisfied: urllib3[socks]<3,>=1.26 in c:\users\julia\anaconda3\envs\engri_1101\lib
\site-packages (from selenium) (1.26.14)
Requirement already satisfied: cffi>=1.14 in c:\users\julia\anaconda3\envs\engri_1101\lib\site-packages
(from trio~=0.17->selenium) (1.15.1)
Requirement already satisfied: attrs>=20.1.0 in c:\users\julia\anaconda3\envs\engri_1101\lib\site-packa
ges (from trio~=0.17->selenium) (22.1.0)
Requirement already satisfied: outcome in c:\users\julia\anaconda3\envs\engri_1101\lib\site-packages (f
rom trio~=0.17->selenium) (1.3.0.post0)
Requirement already satisfied: exceptiongroup>=1.0.0rc9 in c:\users\julia\anaconda3\envs\engri_1101\lib
\site-packages (from trio~=0.17->selenium) (1.1.3)
Requirement already satisfied: idna in c:\users\julia\anaconda3\envs\engri_1101\lib\site-packages (from
trio~=0.17->selenium) (3.4)
Requirement already satisfied: sniffio in c:\users\julia\anaconda3\envs\engri_1101\lib\site-packages (f
rom trio~=0.17->selenium) (1.2.0)
Requirement already satisfied: sortedcontainers in c:\users\julia\anaconda3\envs\engri_1101\lib\site-pa
ckages (from trio~=0.17->selenium) (2.4.0)
Requirement already satisfied: wsproto>=0.14 in c:\users\julia\anaconda3\envs\engri_1101\lib\site-packa
ges (from trio-websocket~=0.9->selenium) (1.2.0)
Requirement already satisfied: PySocks!=1.5.7,<2.0,>=1.5.6 in c:\users\julia\anaconda3\envs\engri_1101
\lib\site-packages (from urllib3[socks]<3,>=1.26->selenium) (1.7.1)
Requirement already satisfied: pycparser in c:\users\julia\anaconda3\envs\engri_1101\lib\site-packages
(from cffi>=1.14->trio~=0.17->selenium) (2.21)
Requirement already satisfied: h11<1,>=0.9.0 in c:\users\julia\anaconda3\envs\engri_1101\lib\site-packa
ges (from wsproto>=0.14->trio-websocket~=0.9->selenium) (0.14.0)
Requirement already satisfied: typing-extensions in c:\users\julia\anaconda3\envs\engri_1101\lib\site-p
ackages (from h11<1,>=0.9.0->wsproto>=0.14->trio-websocket~=0.9->selenium) (4.7.1)
```

BokehJS 2.4.3 successfully loaded.

number_bikes_5am

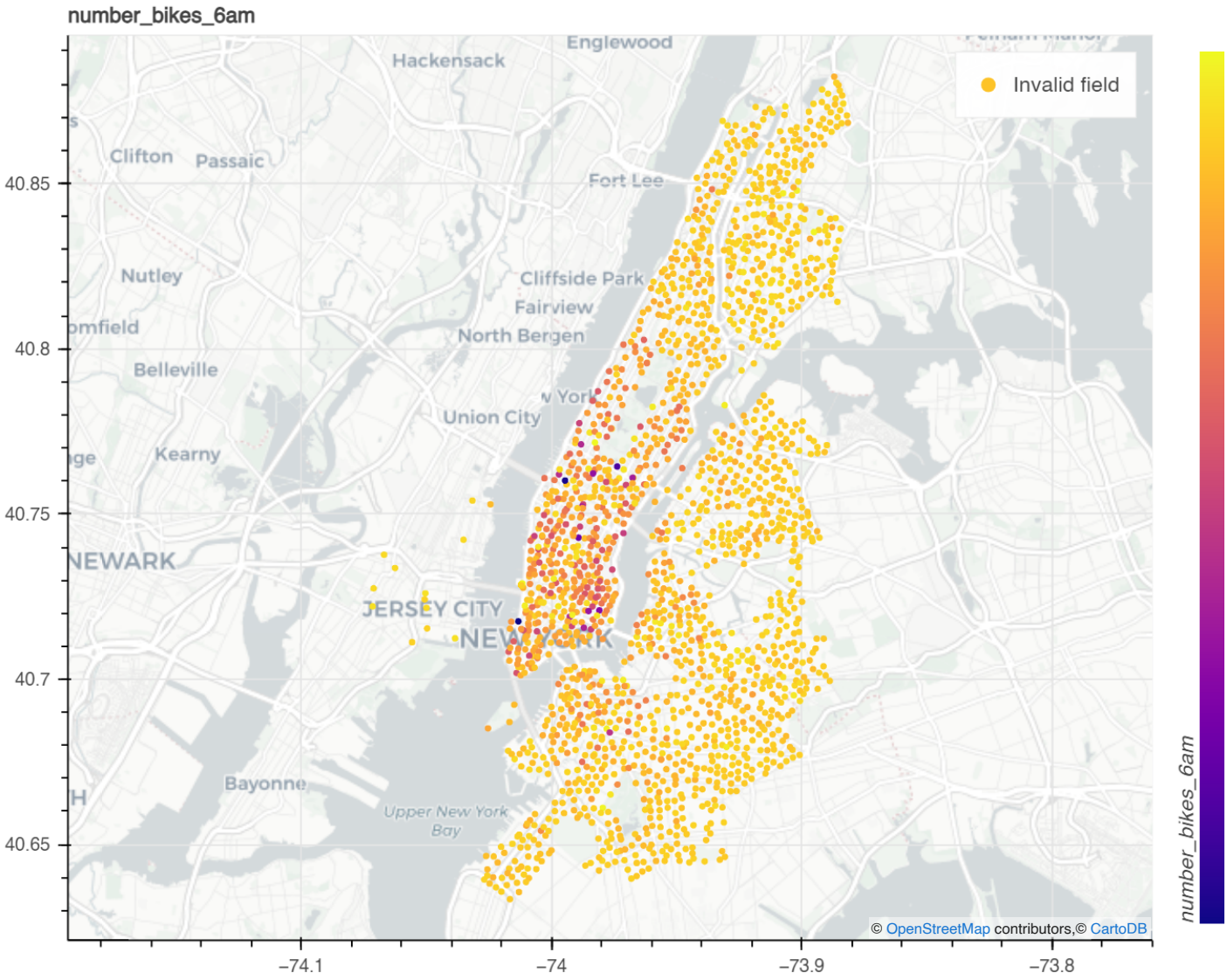
```
ERROR:bokeh.core.validation.check:E-1001 (BAD_COLUMN_NAME): Glyph refers to nonexistent column name. Th
is could either be due to a misspelling or typo, or due to an expected column being missing. : LegendIt
em(id='1363876', ...)
```



BokehJS 2.4.3 successfully loaded.

```
number_bikes_6am
```

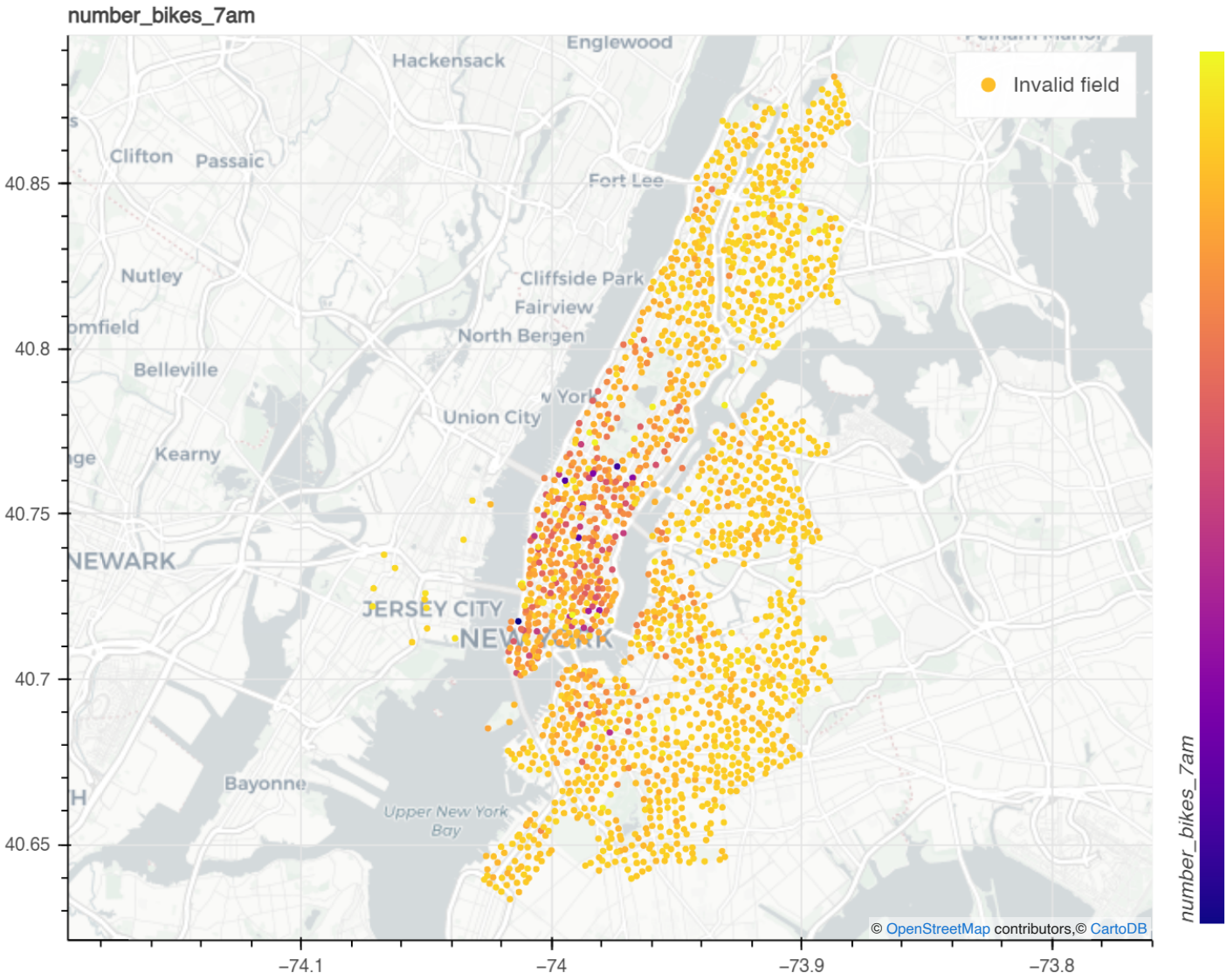
```
ERROR:bokeh.core.validation.check:E-1001 (BAD_COLUMN_NAME): Glyph refers to nonexistent column name. Th
is could either be due to a misspelling or typo, or due to an expected column being missing. : LegendIt
em(id='1369121', ...)
```



BokehJS 2.4.3 successfully loaded.

```
number_bikes_7am
```

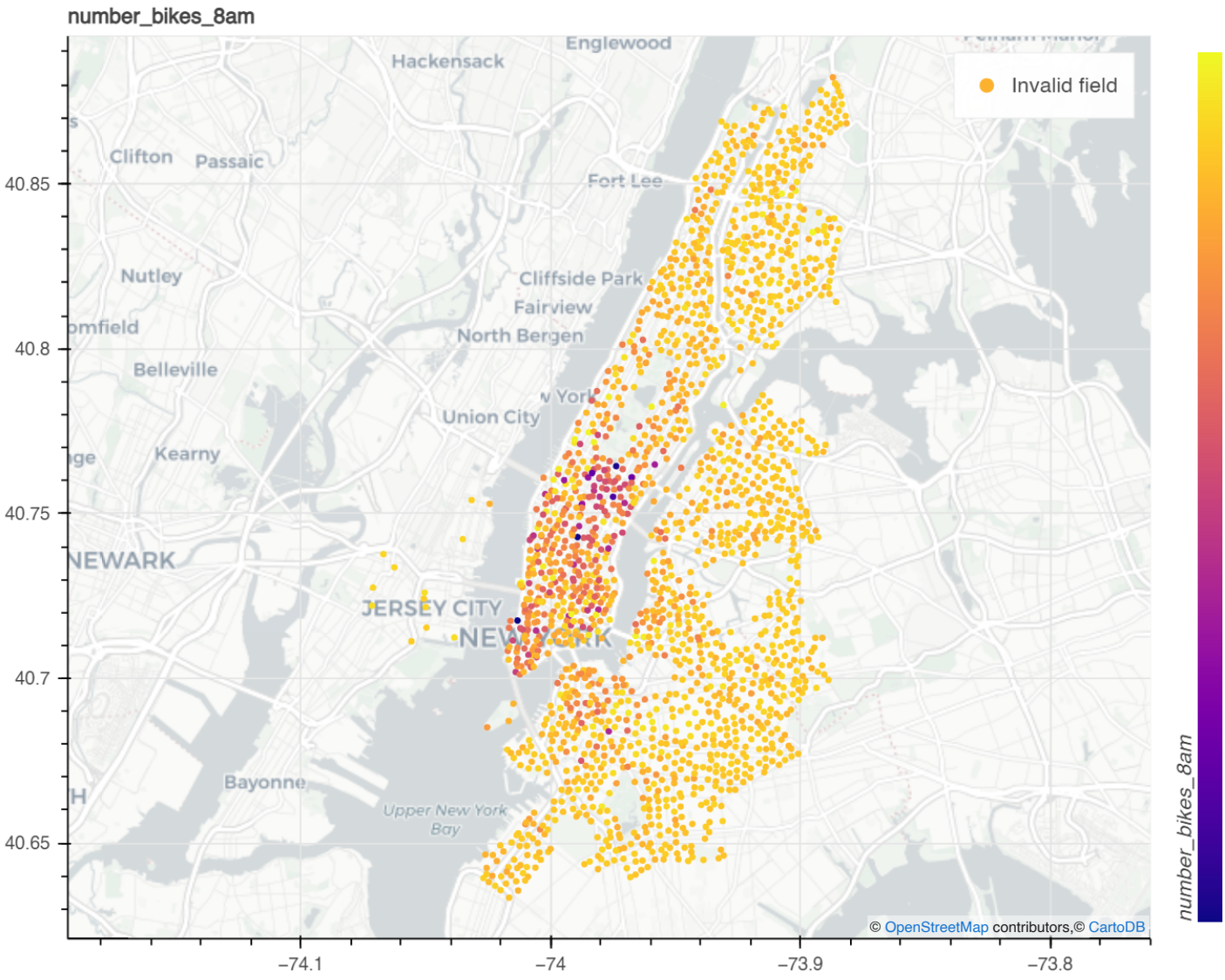
```
ERROR:bokeh.core.validation.check:E-1001 (BAD_COLUMN_NAME): Glyph refers to nonexistent column name. Th
is could either be due to a misspelling or typo, or due to an expected column being missing. : LegendIt
em(id='1374376', ...)
```



BokehJS 2.4.3 successfully loaded.

```
number_bikes_8am
```

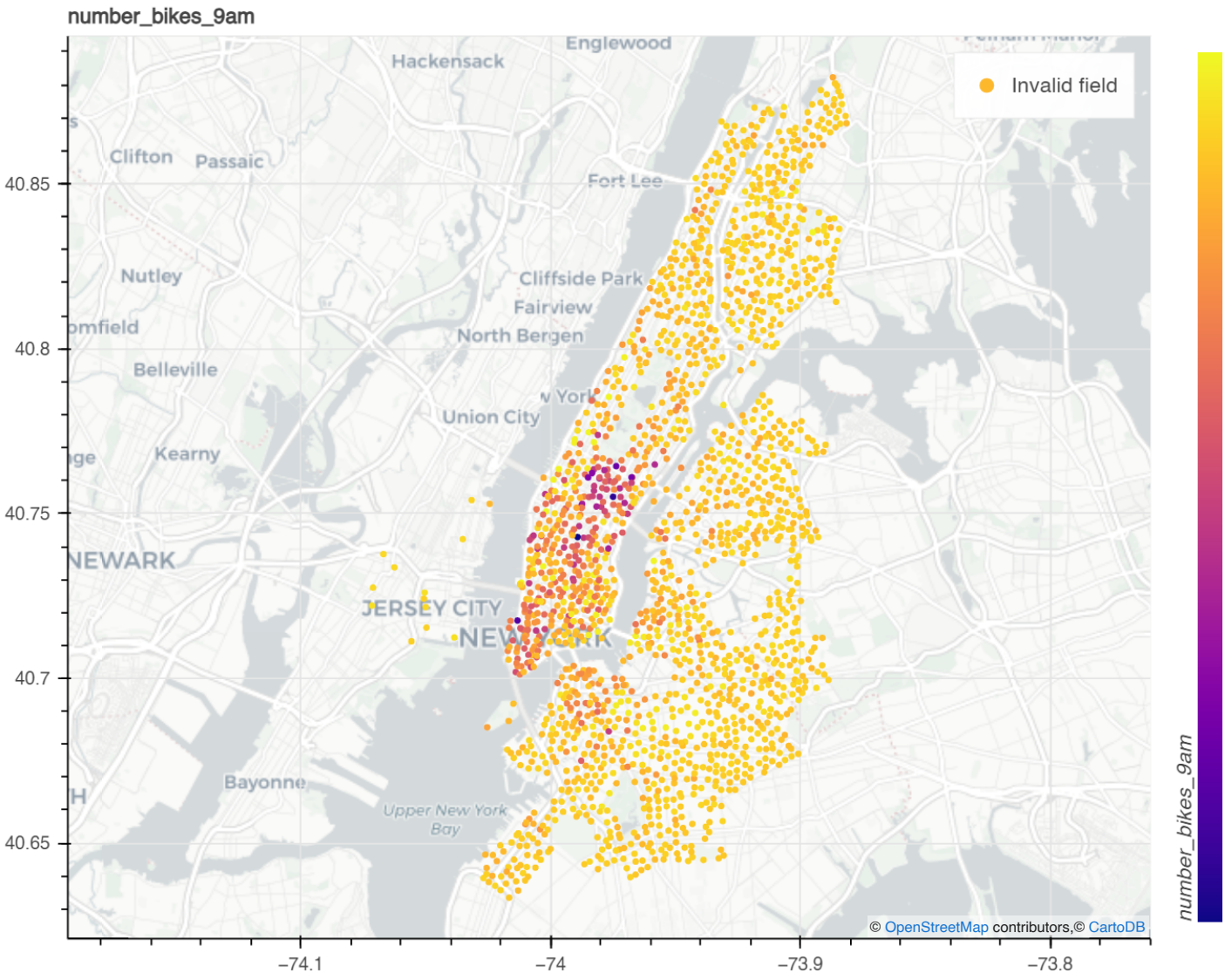
```
ERROR:bokeh.core.validation.check:E-1001 (BAD_COLUMN_NAME): Glyph refers to nonexistent column name. Th
is could either be due to a misspelling or typo, or due to an expected column being missing. : LegendIt
em(id='1379641', ...)
```



BokehJS 2.4.3 successfully loaded.

```
number_bikes_9am
```

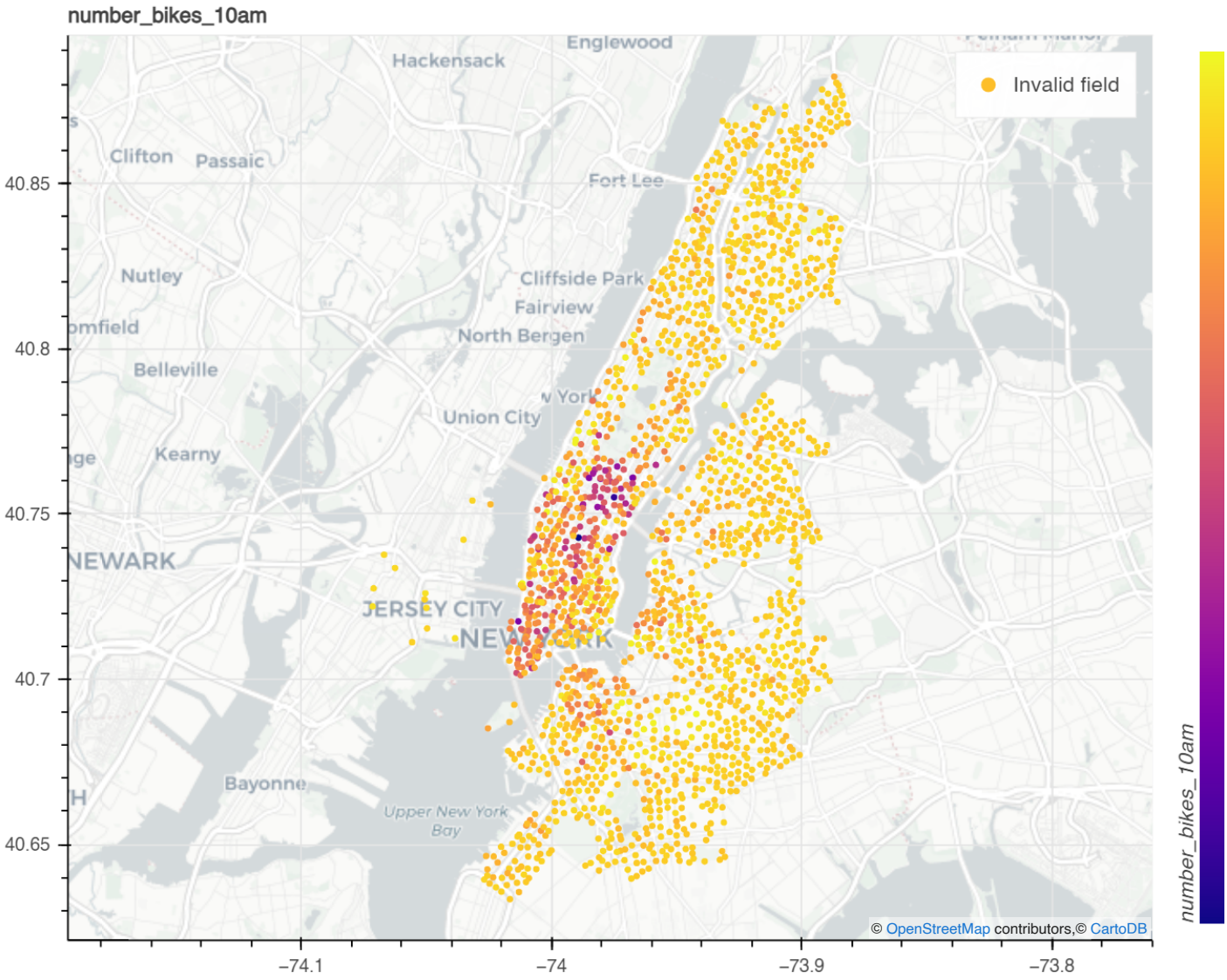
```
ERROR:bokeh.core.validation.check:E-1001 (BAD_COLUMN_NAME): Glyph refers to nonexistent column name. Th
is could either be due to a misspelling or typo, or due to an expected column being missing. : LegendIt
em(id='1384916', ...)
```



BokehJS 2.4.3 successfully loaded.

```
number_bikes_10am
```

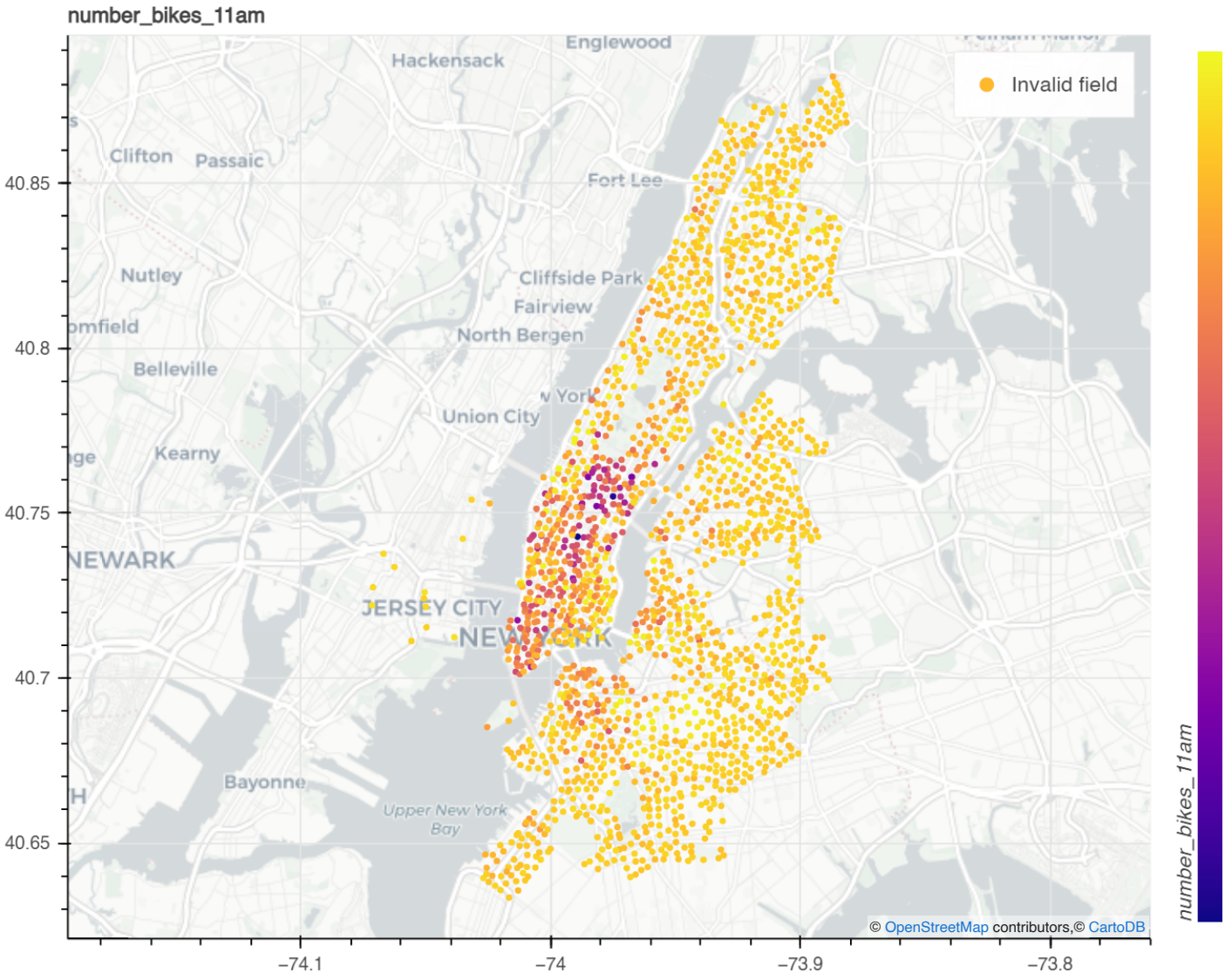
```
ERROR:bokeh.core.validation.check:E-1001 (BAD_COLUMN_NAME): Glyph refers to nonexistent column name. Th
is could either be due to a misspelling or typo, or due to an expected column being missing. : LegendIt
em(id='1390201', ...)
```



BokehJS 2.4.3 successfully loaded.

```
number_bikes_11am
```

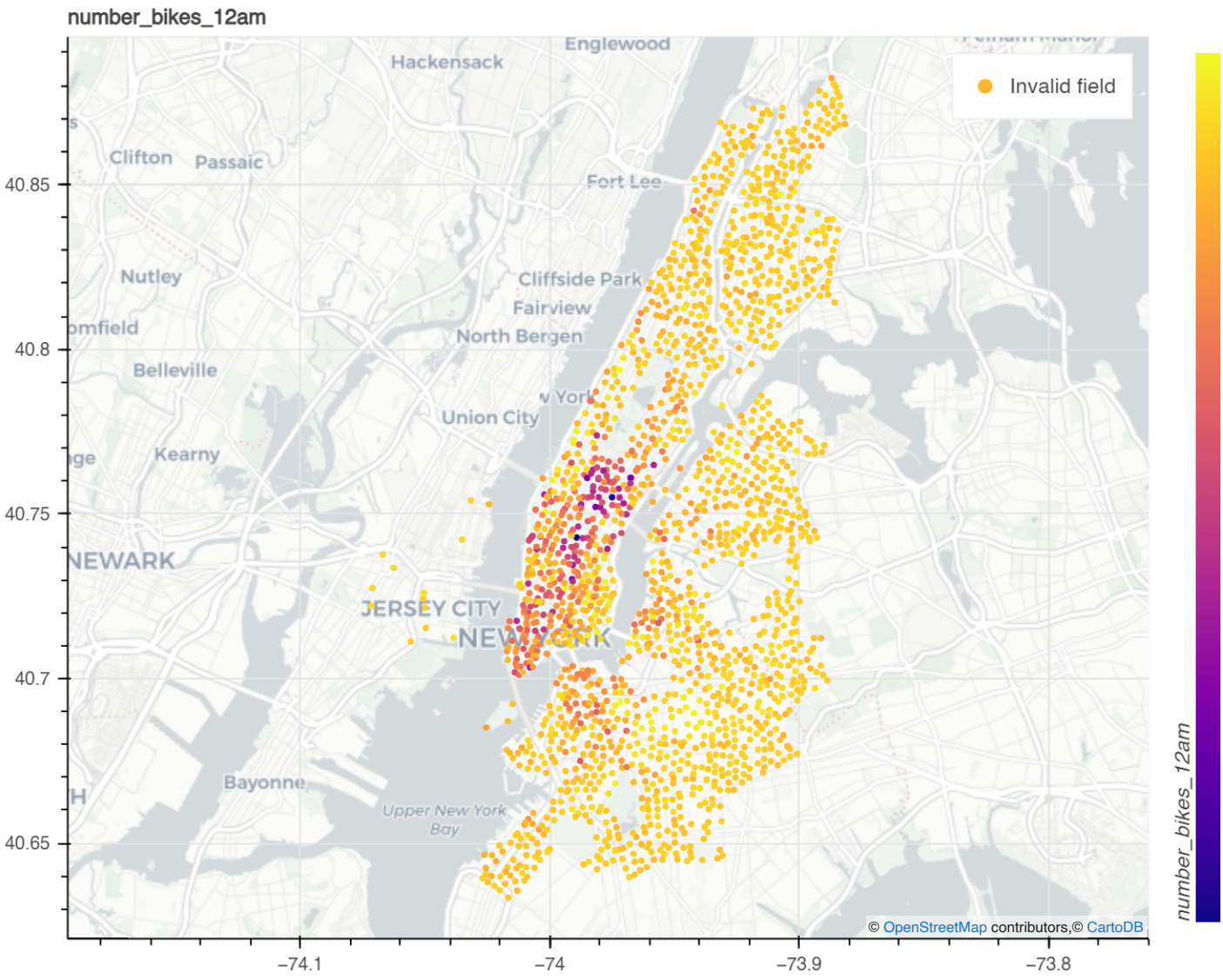
```
ERROR:bokeh.core.validation.check:E-1001 (BAD_COLUMN_NAME): Glyph refers to nonexistent column name. Th
is could either be due to a misspelling or typo, or due to an expected column being missing. : LegendIt
em(id='1395496', ...)
```



BokehJS 2.4.3 successfully loaded.

```
number_bikes_12am
```

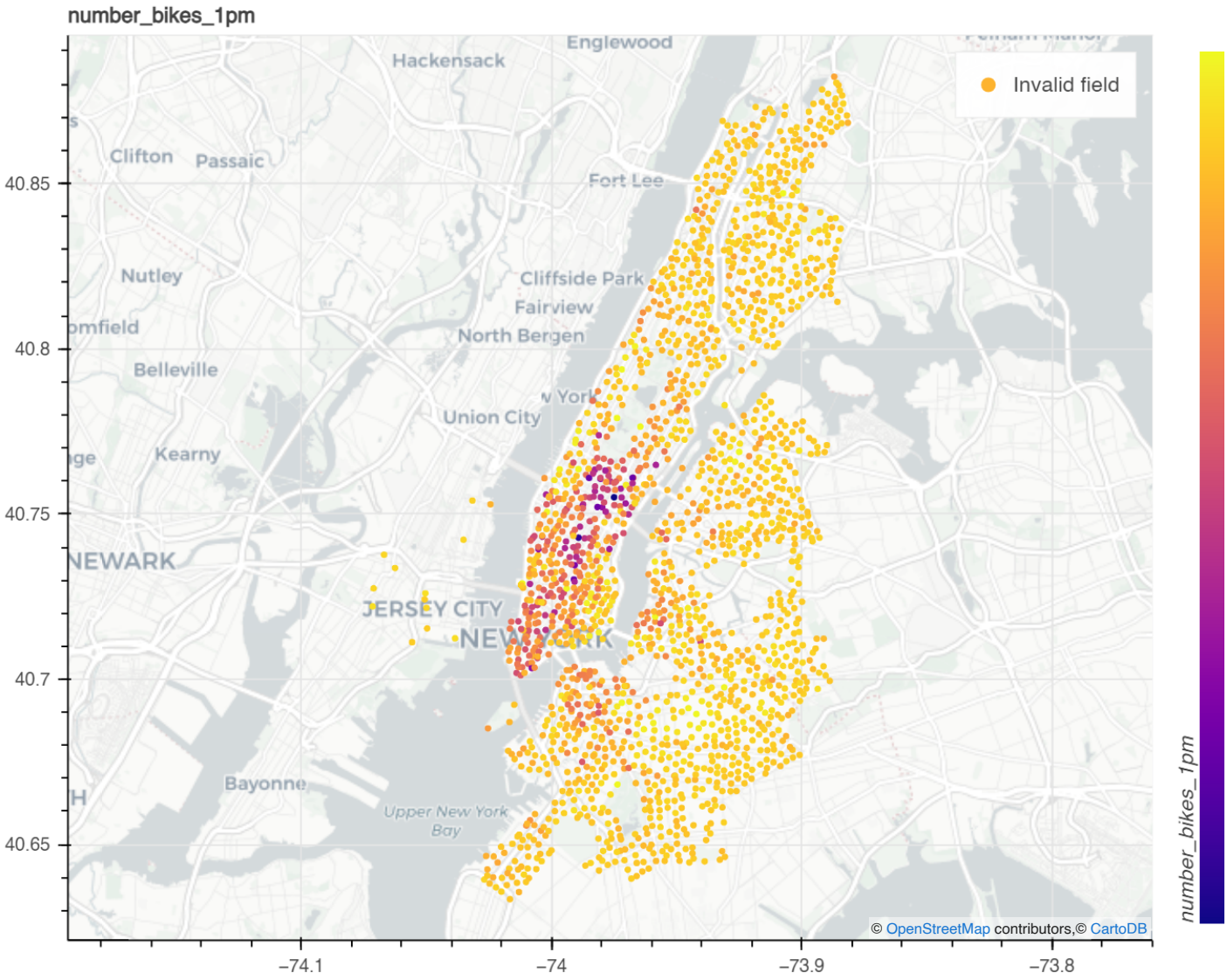
```
ERROR:bokeh.core.validation.check:E-1001 (BAD_COLUMN_NAME): Glyph refers to nonexistent column name. Th
is could either be due to a misspelling or typo, or due to an expected column being missing. : LegendIt
em(id='1400801', ...)
```



BokehJS 2.4.3 successfully loaded.

```
number_bikes_1pm
```

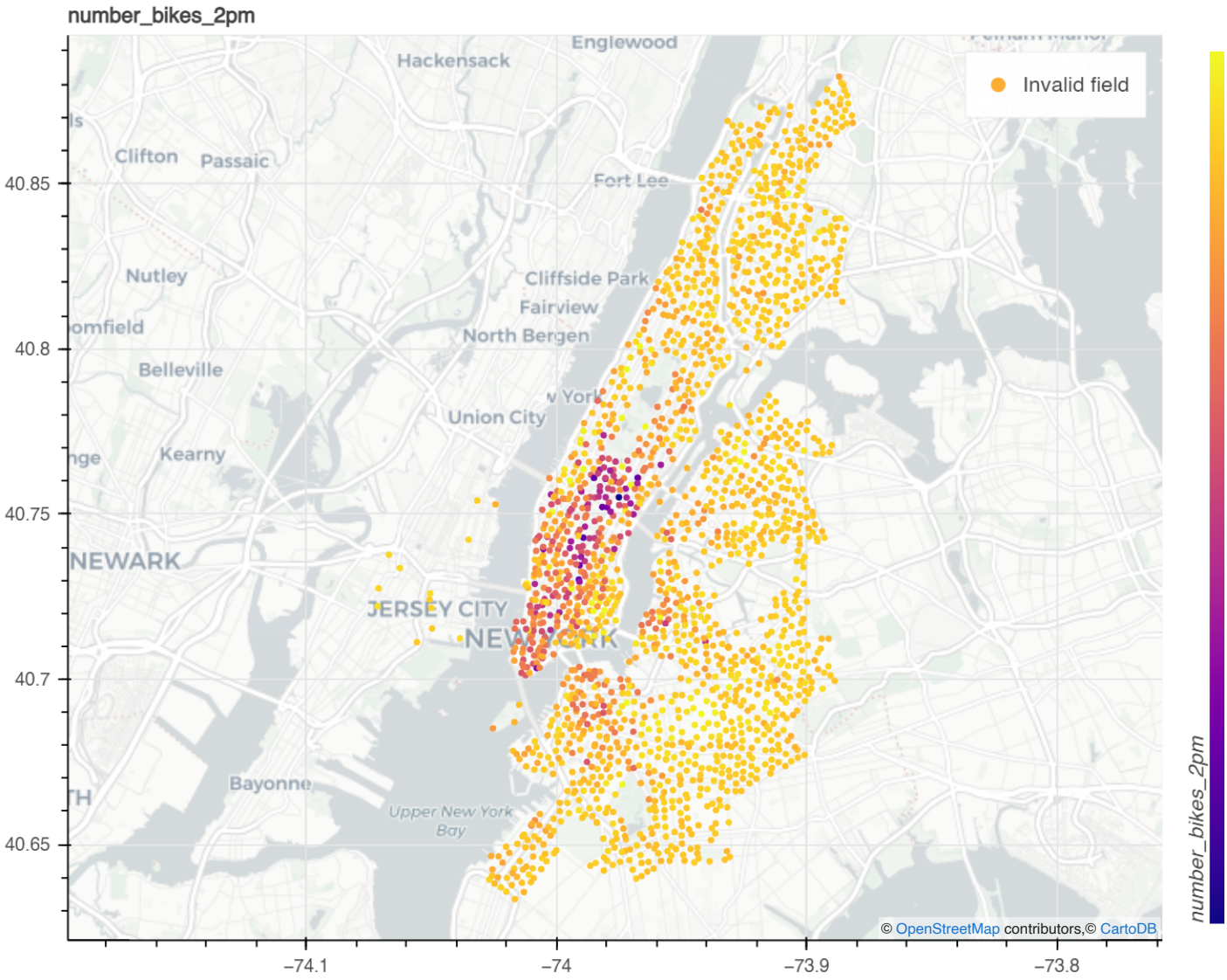
```
ERROR:bokeh.core.validation.check:E-1001 (BAD_COLUMN_NAME): Glyph refers to nonexistent column name. Th
is could either be due to a misspelling or typo, or due to an expected column being missing. : LegendIt
em(id='1406116', ...)
```



BokehJS 2.4.3 successfully loaded.

```
number_bikes_2pm
```

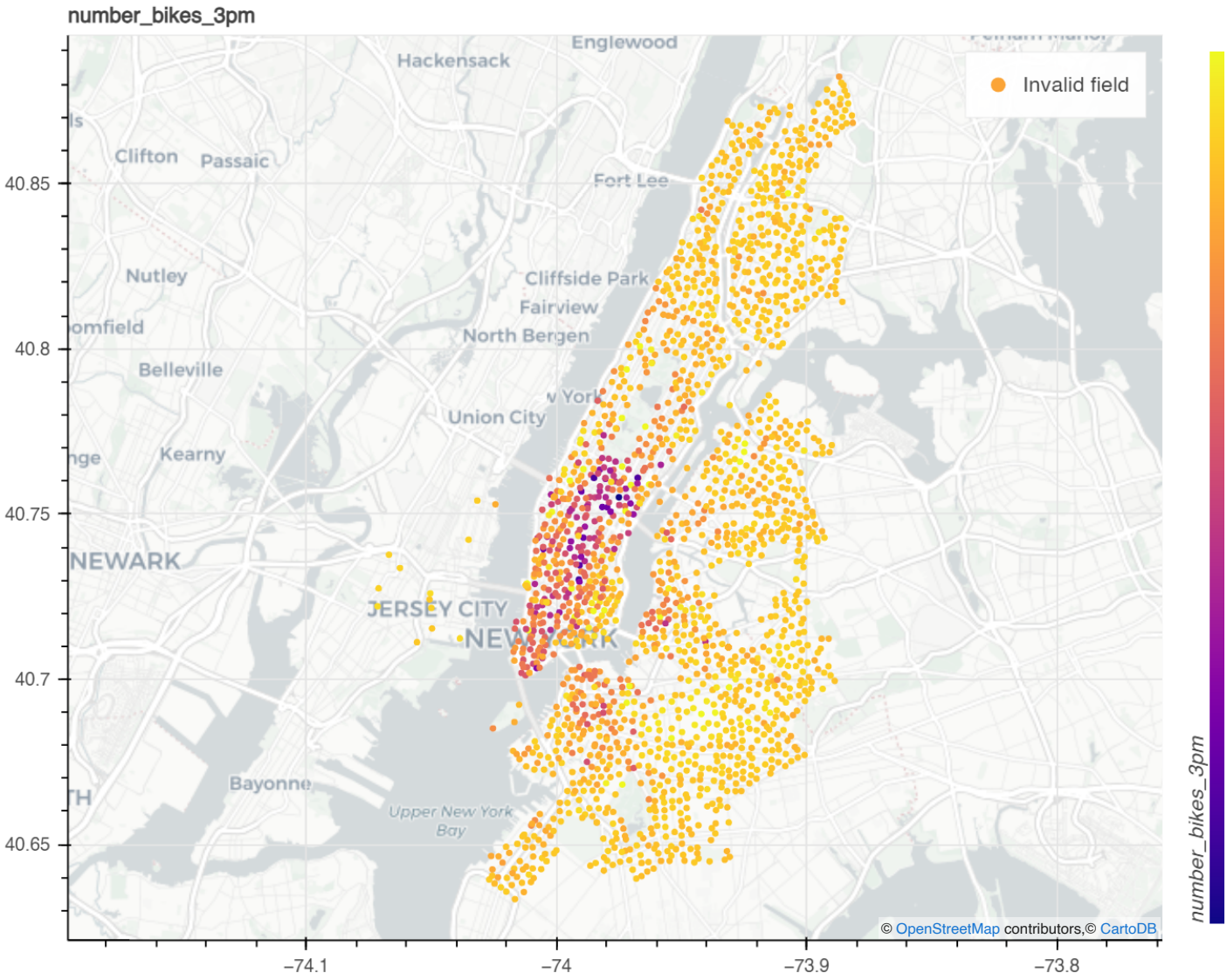
```
ERROR:bokeh.core.validation.check:E-1001 (BAD_COLUMN_NAME): Glyph refers to nonexistent column name. Th
is could either be due to a misspelling or typo, or due to an expected column being missing. : LegendIt
em(id='1411441', ...)
```



BokehJS 2.4.3 successfully loaded.

```
number_bikes_3pm
```

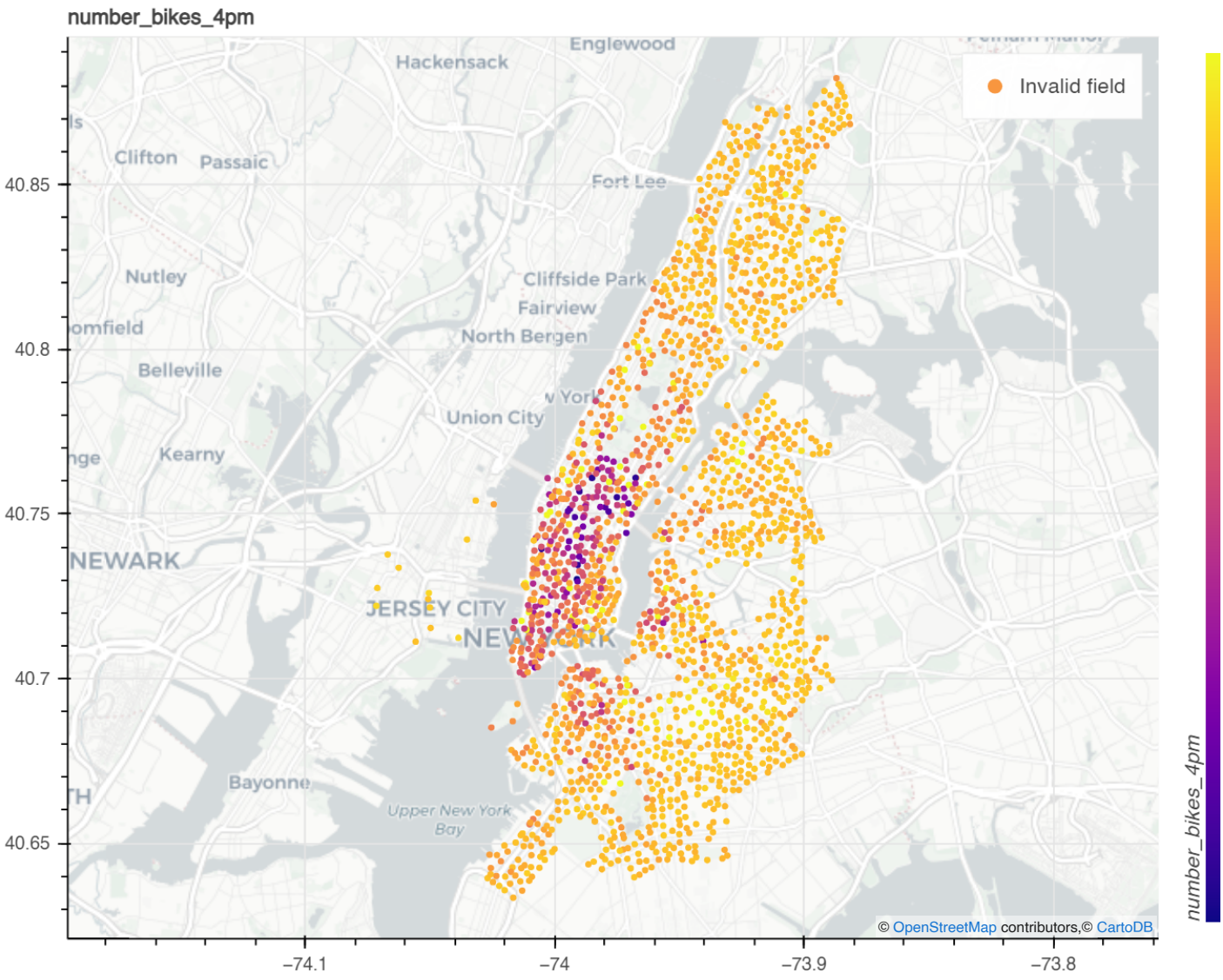
```
ERROR:bokeh.core.validation.check:E-1001 (BAD_COLUMN_NAME): Glyph refers to nonexistent column name. Th
is could either be due to a misspelling or typo, or due to an expected column being missing. : LegendIt
em(id='1416776', ...)
```



BokehJS 2.4.3 successfully loaded.

```
number_bikes_4pm
```

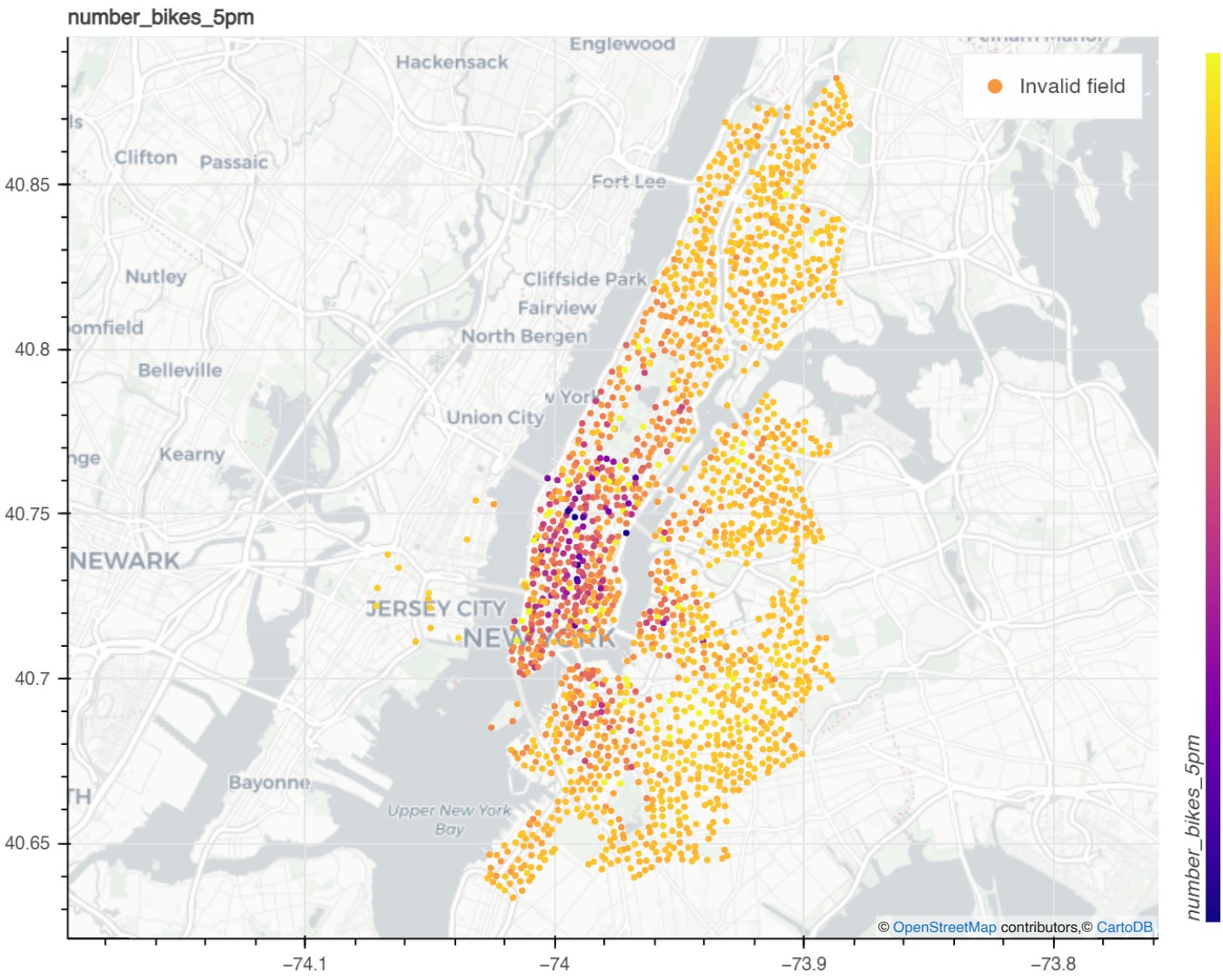
```
ERROR:bokeh.core.validation.check:E-1001 (BAD_COLUMN_NAME): Glyph refers to nonexistent column name. Th
is could either be due to a misspelling or typo, or due to an expected column being missing. : LegendIt
em(id='1422121', ...)
```



BokehJS 2.4.3 successfully loaded.

```
number_bikes_5pm
```

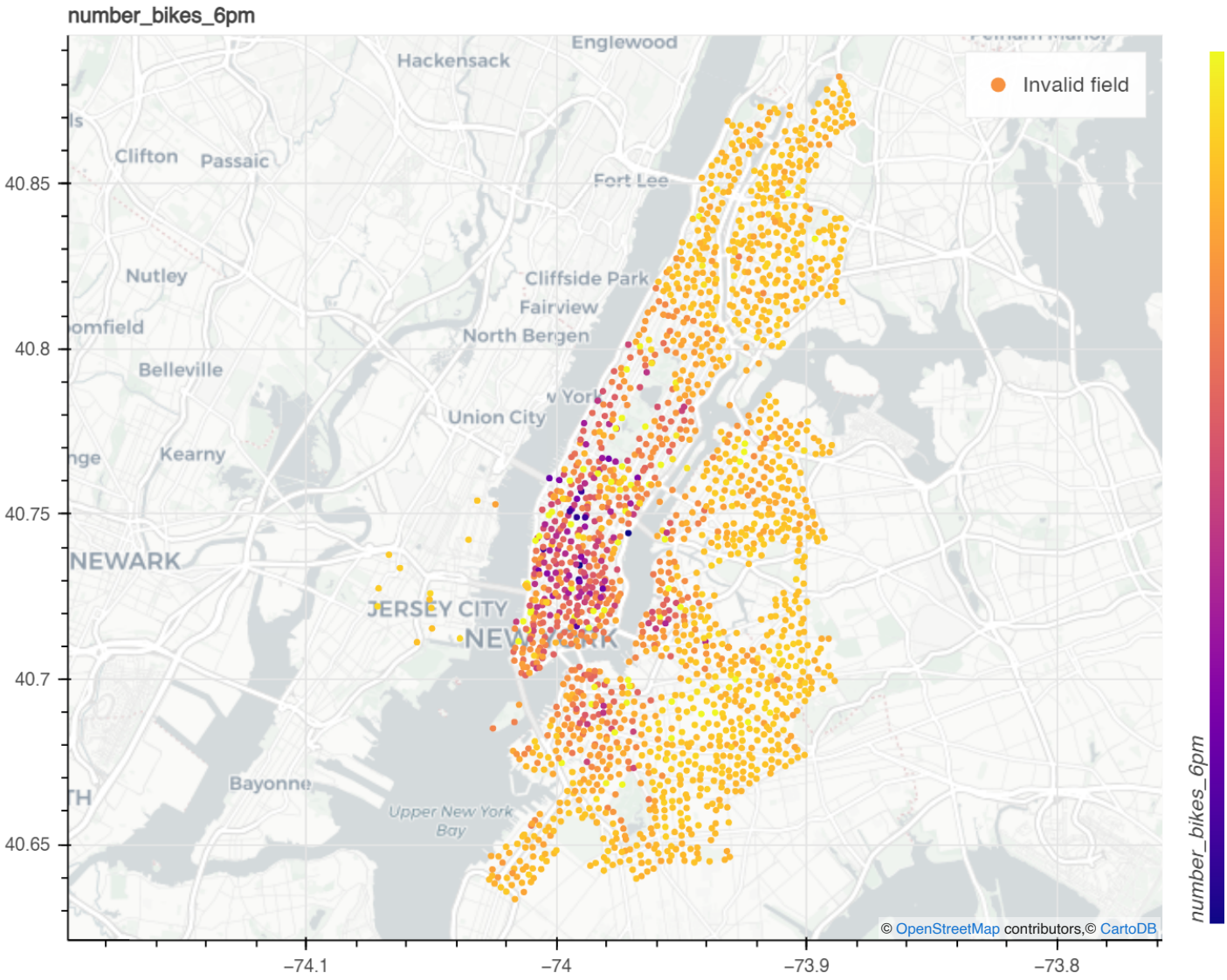
```
ERROR:bokeh.core.validation.check:E-1001 (BAD_COLUMN_NAME): Glyph refers to nonexistent column name. Th
is could either be due to a misspelling or typo, or due to an expected column being missing. : LegendIt
em(id='1427476', ...)
```



BokehJS 2.4.3 successfully loaded.

```
number_bikes_6pm
```

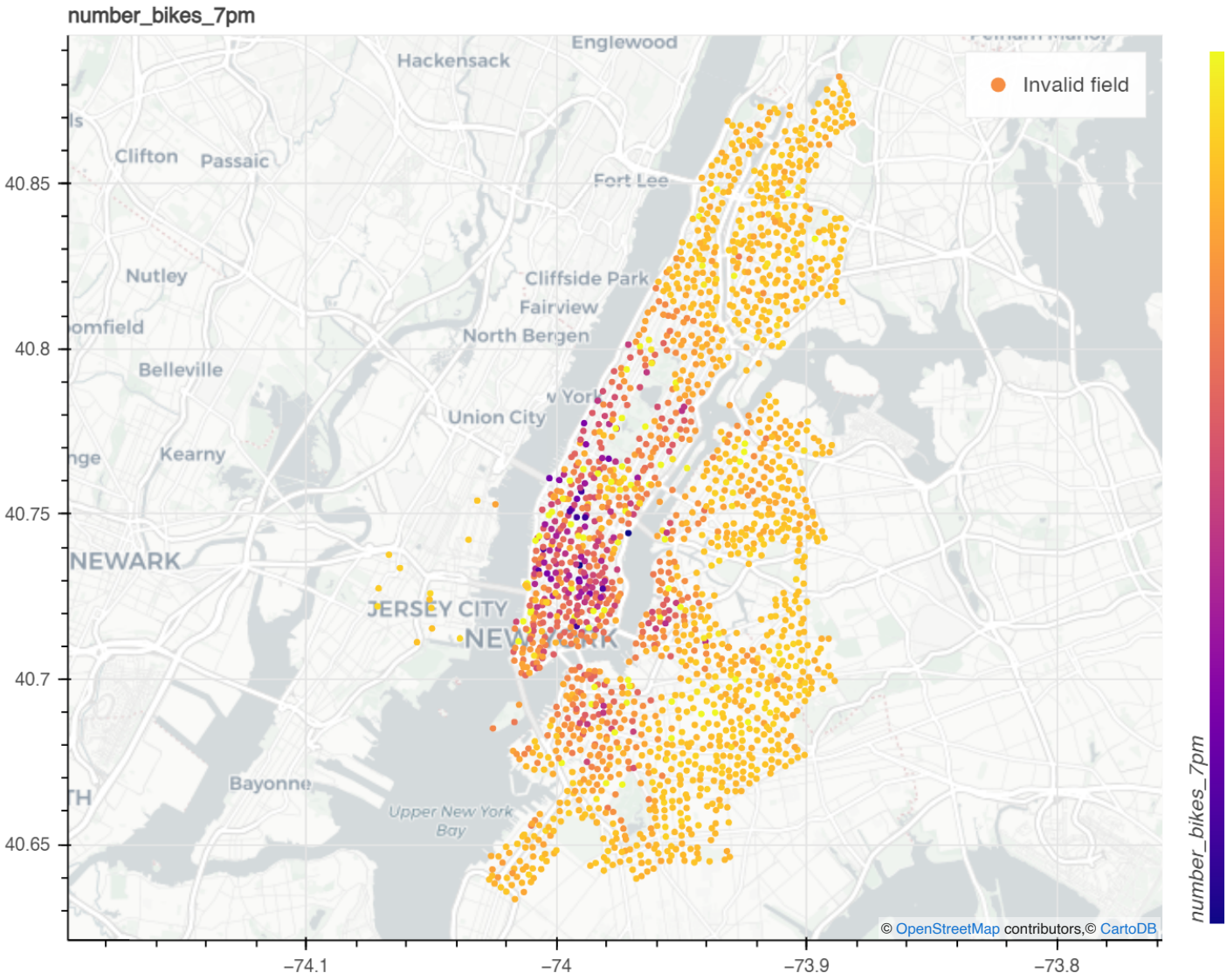
```
ERROR:bokeh.core.validation.check:E-1001 (BAD_COLUMN_NAME): Glyph refers to nonexistent column name. Th
is could either be due to a misspelling or typo, or due to an expected column being missing. : LegendIt
em(id='1432841', ...)
```



BokehJS 2.4.3 successfully loaded.

```
number_bikes_7pm
```

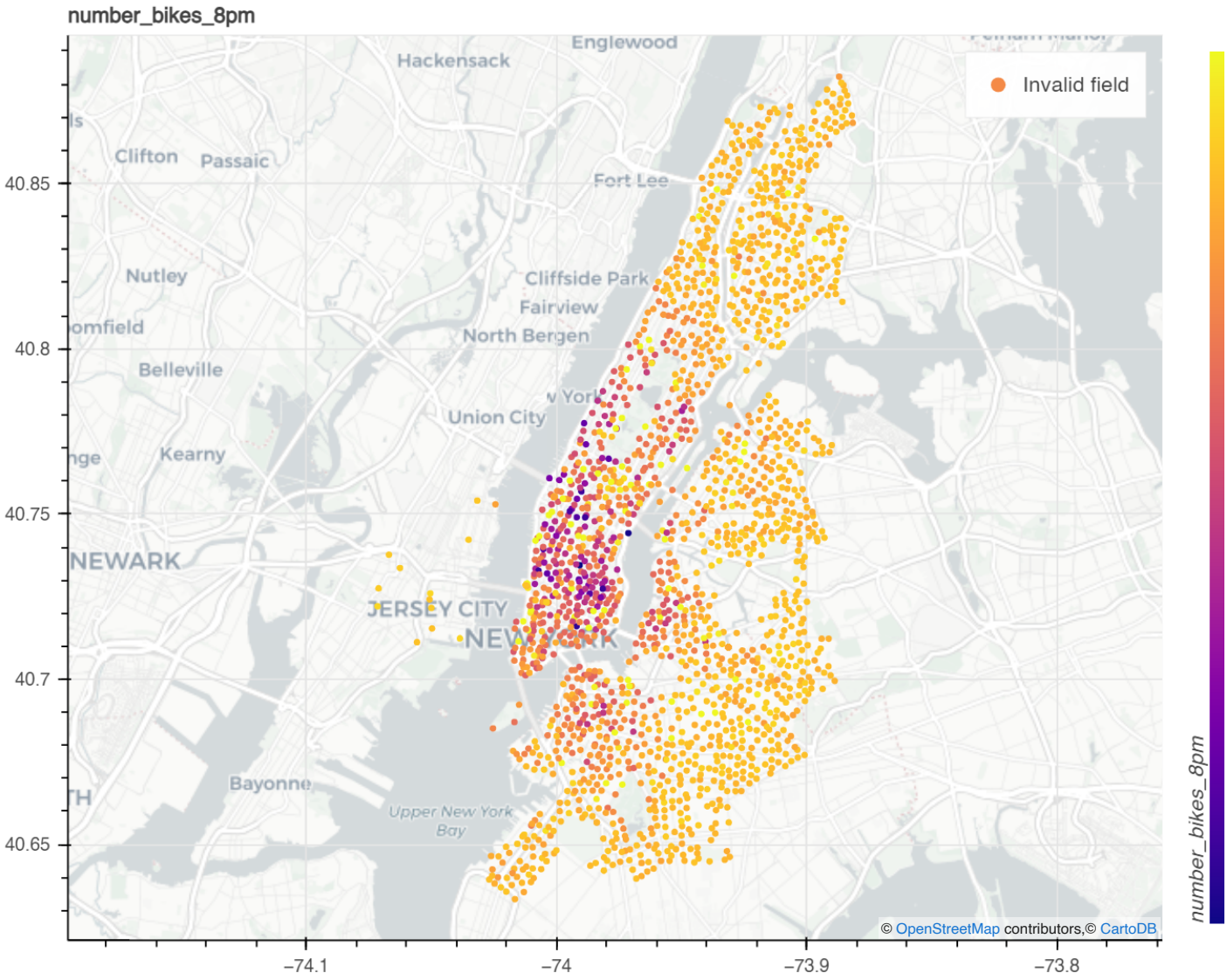
```
ERROR:bokeh.core.validation.check:E-1001 (BAD_COLUMN_NAME): Glyph refers to nonexistent column name. Th
is could either be due to a misspelling or typo, or due to an expected column being missing. : LegendIt
em(id='1438216', ...)
```



BokehJS 2.4.3 successfully loaded.

```
number_bikes_8pm
```

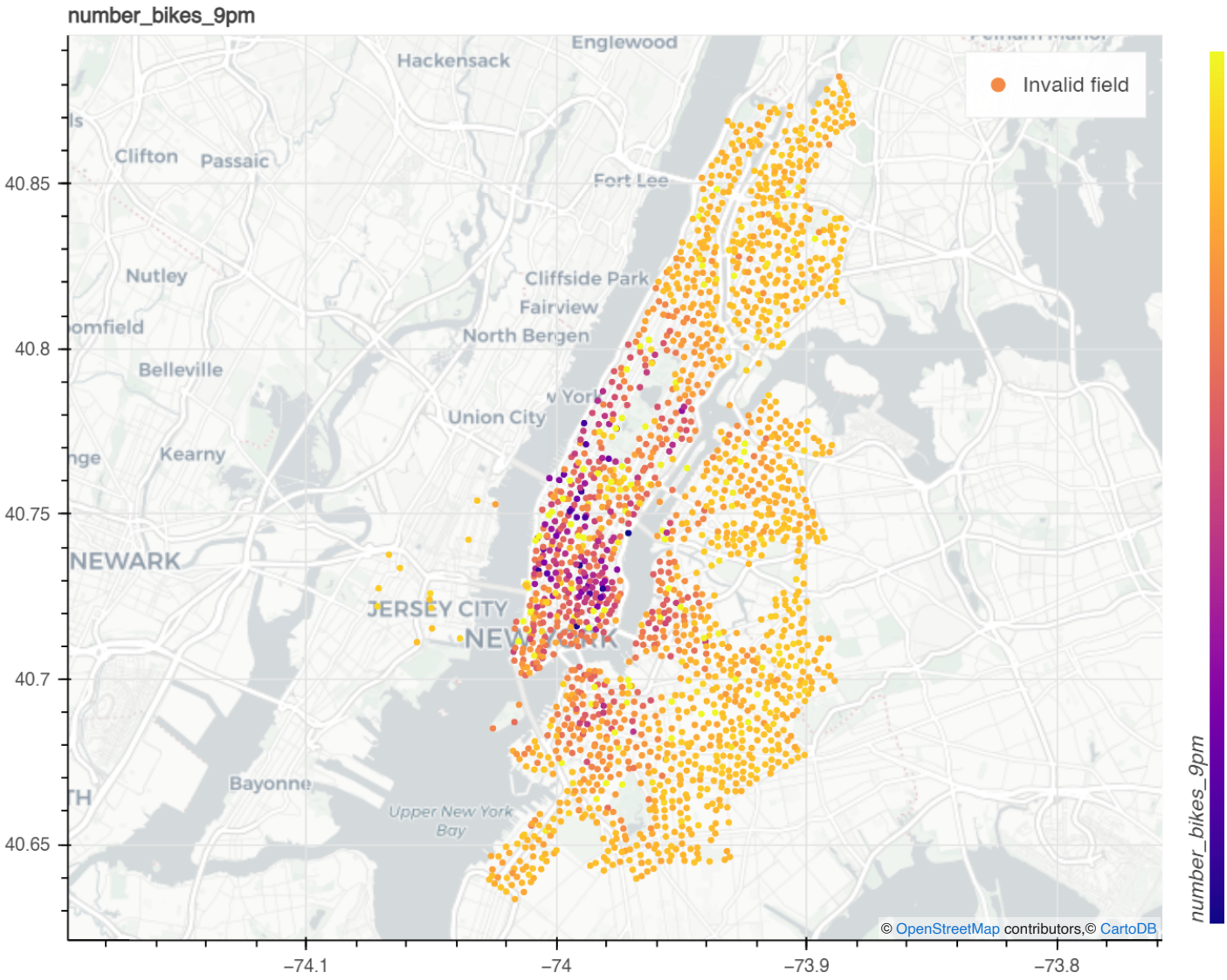
```
ERROR:bokeh.core.validation.check:E-1001 (BAD_COLUMN_NAME): Glyph refers to nonexistent column name. Th
is could either be due to a misspelling or typo, or due to an expected column being missing. : LegendIt
em(id='1443601', ...)
```



BokehJS 2.4.3 successfully loaded.

```
number_bikes_9pm
```

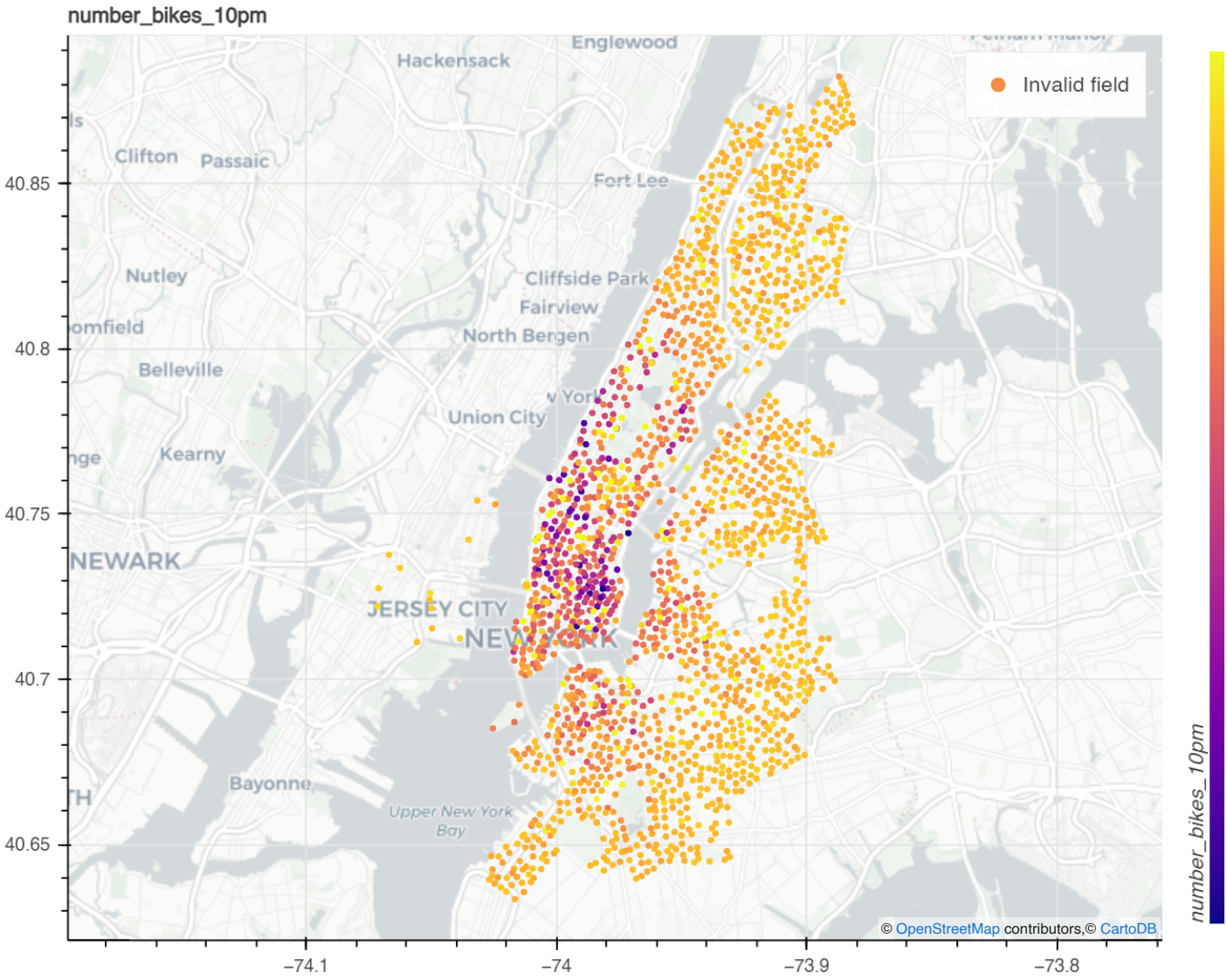
```
ERROR:bokeh.core.validation.check:E-1001 (BAD_COLUMN_NAME): Glyph refers to nonexistent column name. Th
is could either be due to a misspelling or typo, or due to an expected column being missing. : LegendIt
em(id='1448996', ...)
```



BokehJS 2.4.3 successfully loaded.

```
number_bikes_10pm
```

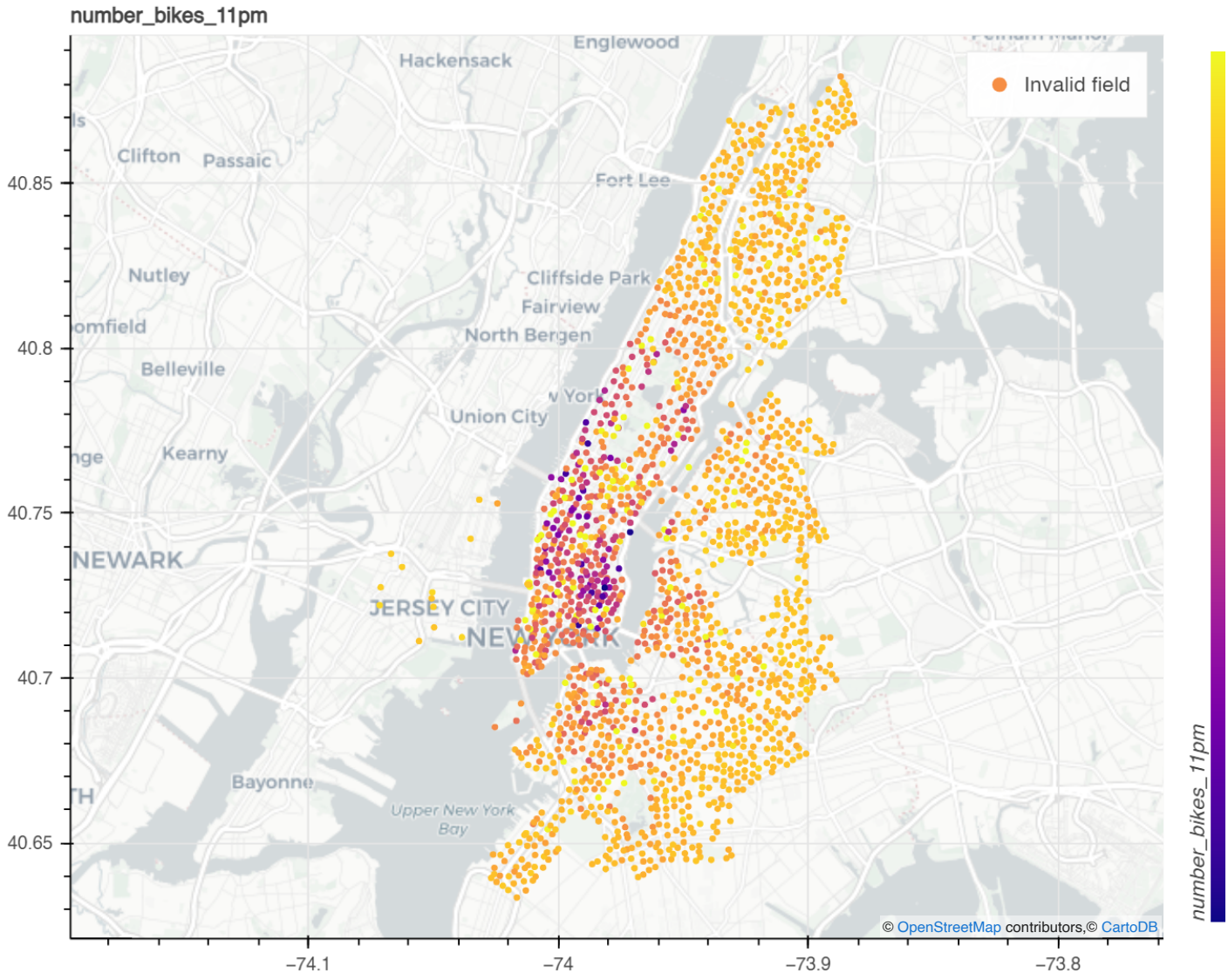
```
ERROR:bokeh.core.validation.check:E-1001 (BAD_COLUMN_NAME): Glyph refers to nonexistent column name. Th
is could either be due to a misspelling or typo, or due to an expected column being missing. : LegendIt
em(id='1454401', ...)
```



BokehJS 2.4.3 successfully loaded.

```
number_bikes_11pm
```

```
ERROR:bokeh.core.validation.check:E-1001 (BAD_COLUMN_NAME): Glyph refers to nonexistent column name. Th
is could either be due to a misspelling or typo, or due to an expected column being missing. : LegendIt
em(id='1459816', ...)
```

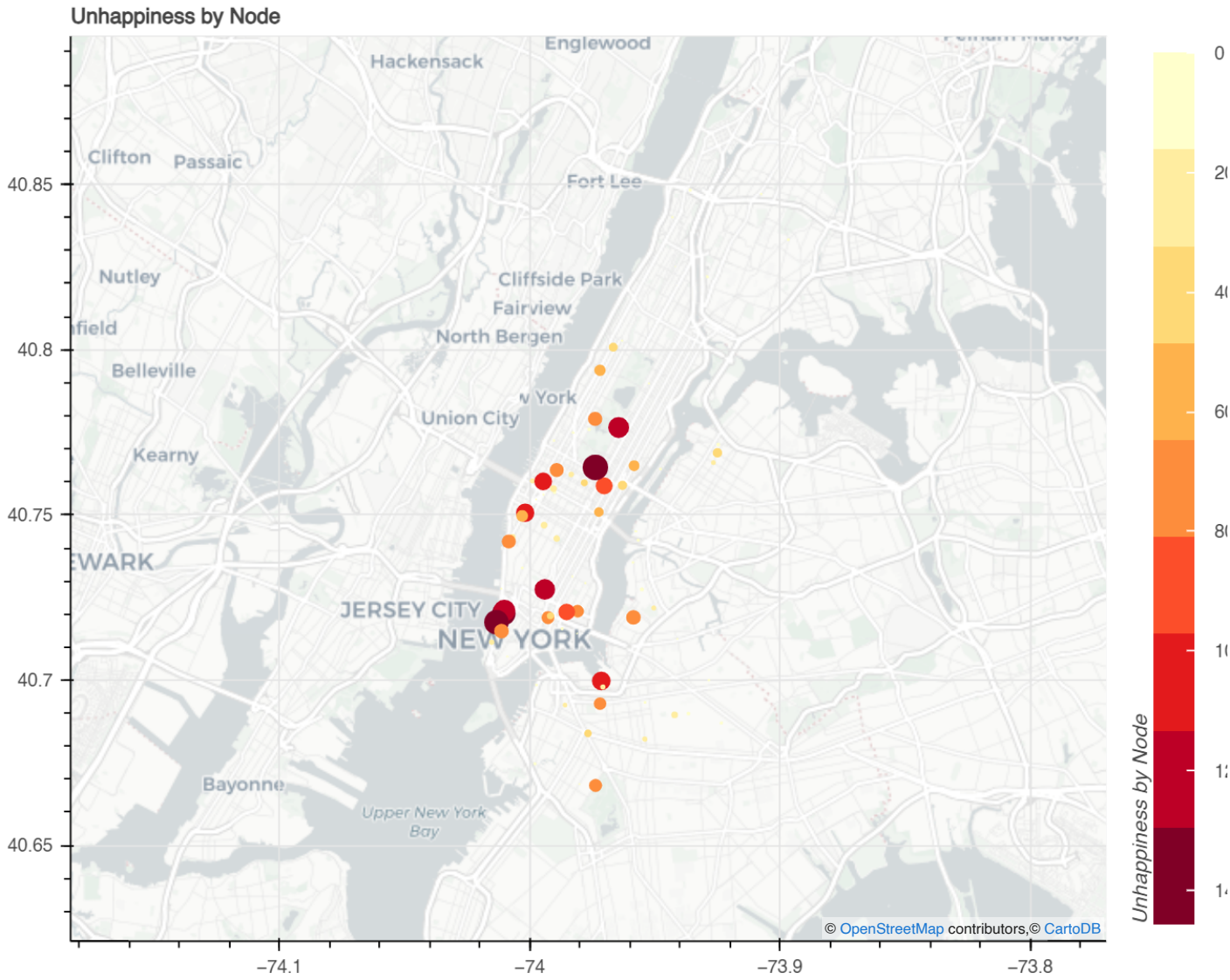


In [126…
```
station_list['unhappiness'] = unhappiness
```

In [128…
```
from bokeh.palettes import YlOrRd
def plotNetwork(nodes, on_map=True):
    output_notebook()
    nodes['size'] = nodes['unhappiness']/10
    # Prepare the data for plotting
    nodes_source = ColumnDataSource(nodes)
    # Create a color mapper for 'counts' column with a high and low bound
    color_mapper = LinearColorMapper(palette=YlOrRd[9], low=max(nodes['unhappiness']), high=min(nodes['u

    # Define the plot
    plot = figure(title='Unhappiness by Node',
                  x_axis_type="mercator", y_axis_type="mercator",
                  width=800, height=600)

    # Add map tile if requested
    if on_map:
        plot.add_tile(get_provider(Vendors.CARTODBPOSITRON))

    # Add the nodes to the plot with color mapped by 'counts'
    plot.circle(x='x', y='y', size='size', color=transform('unhappiness', color_mapper), source=nodes_sc

    # Add a color bar to the side of the plot to show the color mapping
    color_bar = ColorBar(color_mapper=color_mapper, label_standoff=12, location=(0,0), title='Unhappines
    plot.add_layout(color_bar, 'right')
```

```
    # Show the plot
    show(layout(plot))
    #export_png(layout(plot), filename='unhappiness.png')


plotNetwork(nodes=station_list)
```

BokehJS 2.4.3 successfully loaded.



In [126…
```
total_outages = np.sum(station_list['unhappiness'])
print('total outages ',total_outages)
rides_per_day = np.sum(mus['counts'])
print('rides per day', rides_per_day)
percent_outage = total_outages/rides_per_day*100
print('percent outage',percent_outage)
```

```
total outages  2927.9617458563534
rides per day 141053.75
percent outage 2.075777315992204
```

In [127…
```
movement = np.abs(station_list['number_bikes_11pm'] - station_list['number_bikes_5am'])
print(np.sum(movement))
```

```
10438.433026722694
```

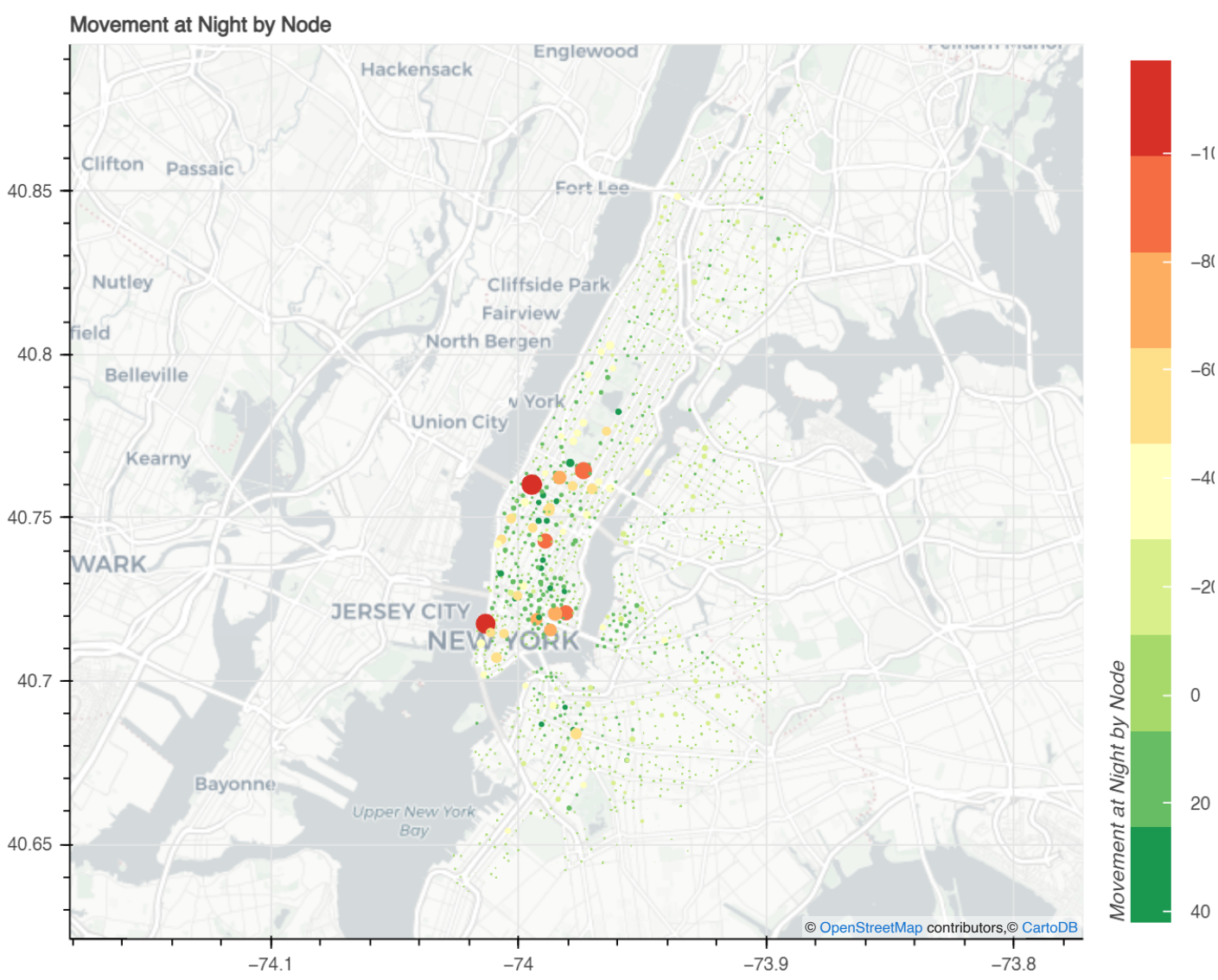
In [127…
```
too_many = np.min(station_list['number_bikes_11pm'] - station_list['number_bikes_5am'],0)
too_few = np.min(station_list['number_bikes_5am'] - station_list['number_bikes_11pm'],0)
movement = station_list['number_bikes_11pm'] - station_list['number_bikes_5am']
```

```
# stations with too many will have positive values
# stations with too few will have negative values
```

In [129…

```
from bokeh.palettes import RdYlGn
def plotNetwork(nodes, movement, on_map=True):
    output_notebook()
    nodes['movement'] = movement
    nodes['size'] = np.abs(movement/10)
    # Prepare the data for plotting
    nodes_source = ColumnDataSource(nodes)
    # Create a color mapper for 'counts' column with a high and low bound
    color_mapper = LinearColorMapper(palette=RdYlGn[9], low=max(nodes['movement']), high=min(nodes['move

    # Define the plot
    plot = figure(title='Movement at Night by Node',
                  x_axis_type="mercator", y_axis_type="mercator",
                  width=800, height=600)

    # Add map tile if requested
    if on_map:
        plot.add_tile(get_provider(Vendors.CARTODBPOSITRON))

    # Add the nodes to the plot with color mapped by 'counts'
    plot.circle(x='x', y='y', size='size', color=transform('movement', color_mapper), source=nodes_sourc

    # Add a color bar to the side of the plot to show the color mapping
    color_bar = ColorBar(color_mapper=color_mapper, label_standoff=12, location=(0,0), title='Movement a
    plot.add_layout(color_bar, 'right')

    # Show the plot
    show(layout(plot))
    #export_png(layout(plot), filename='movement.png')


plotNetwork(nodes=station_list, movement=movement)
```

BokehJS 2.4.3 successfully loaded.



In [129…

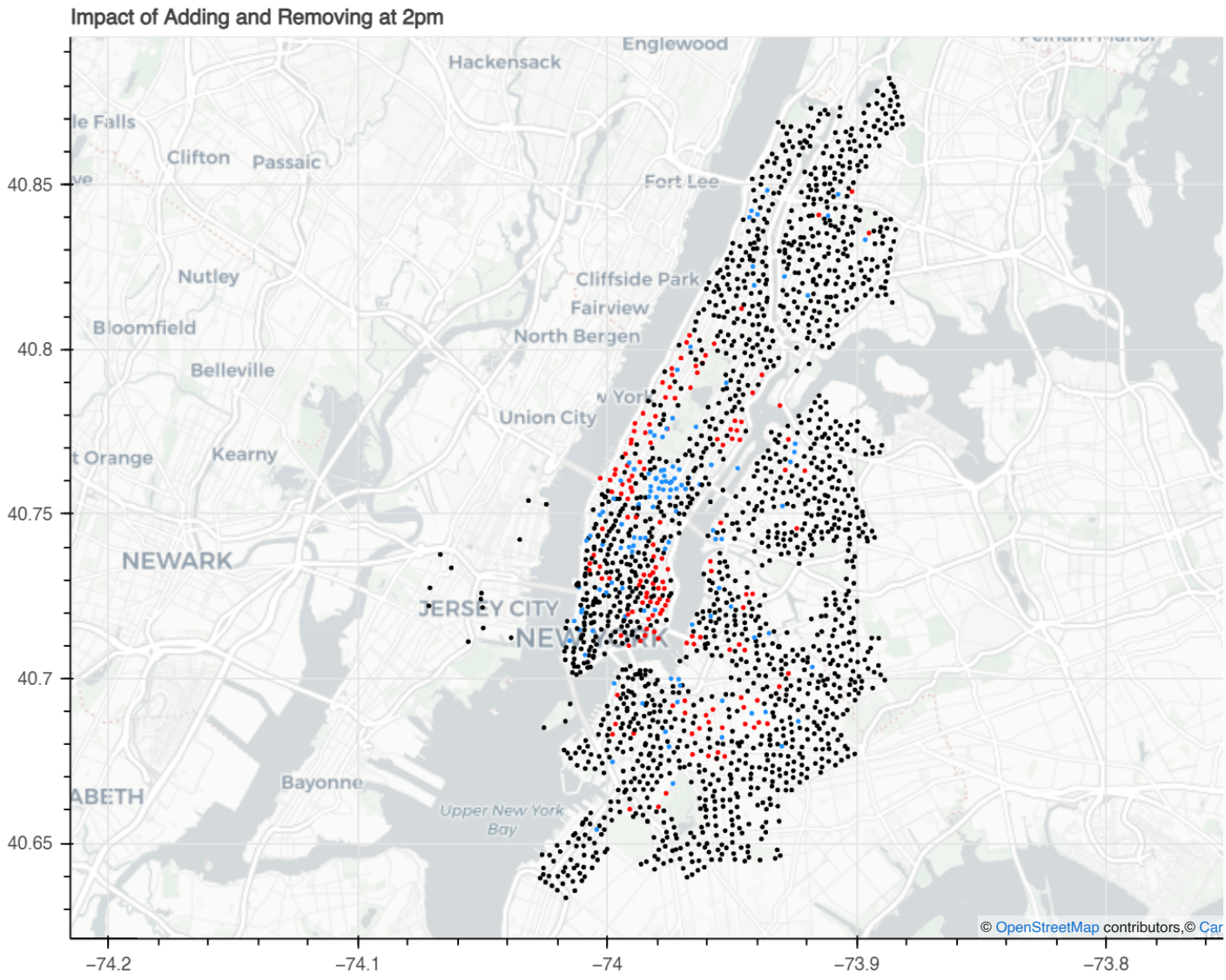
```
# original logic written by me, modified by Chat GPT to include analysis for Bike Angels.
bike_impact = []

for i in station_list['id'].iloc[:,0]:
    capacity_i = int(capacity[capacity['end_station_id'] == str(i)]['capacity'].iloc[0])
    mus_i = mus[mus['id'] == i][['hour', 'counts']].copy()
    all_hours = pd.DataFrame({'hour': range(5, 24)})  # Adjusted to include 5 am
    mus_i = pd.merge(all_hours, mus_i, on='hour', how='left').fillna(0)
    mus_i.set_index('hour', inplace=True)

    lambda_i = lambdas[lambdas['id'] == i][['hour', 'counts']].copy()
    all_hours = pd.DataFrame({'hour': range(5, 24)})  # Adjusted to include 5 am
    lambda_i = pd.merge(all_hours, lambda_i, on='hour', how='left').fillna(0)
    lambda_i.set_index('hour', inplace=True)
    half_full = capacity_i//2
    x_at_2pm = half_full

    unhappiness_with_one_fewer = calculate_unhappiness(x_at_2pm - 1, mus_i, lambda_i, capacity_i, start_
    unhappiness_with_one_more = calculate_unhappiness(x_at_2pm + 1, mus_i, lambda_i, capacity_i, start_h
    unhappiness_with_current = calculate_unhappiness(x_at_2pm, mus_i, lambda_i, capacity_i, start_hour=1

    # Determine which action (add, remove, or no change) minimizes unhappiness
    min_unhappiness = min(unhappiness_with_one_fewer, unhappiness_with_one_more, unhappiness_with_currer
    if min_unhappiness == unhappiness_with_current:
        color = 'black'  # No change needed
    elif min_unhappiness == unhappiness_with_one_more:
        color = 'dodgerblue'   # Adding a bike is better
    else:
```

```
        color = 'red'    # Removing a bike is better

    bike_impact.append((i, color))

df_bike_impact = pd.DataFrame(bike_impact, columns=['id', 'color_impact'])
```

In [129…

```
from bokeh.palettes import RdYlGn
def plotNetwork(nodes, impact, on_map=True):
    output_notebook()
    nodes['impact'] = impact
    # Prepare the data for plotting
    nodes_source = ColumnDataSource(nodes)

    # Define the plot
    plot = figure(title='Impact of Adding and Removing at 2pm',
                  x_axis_type="mercator", y_axis_type="mercator",
                  width=800, height=600)

    # Add map tile if requested
    if on_map:
        plot.add_tile(get_provider(Vendors.CARTODBPOSITRON))

    # Add the nodes to the plot with color mapped by 'counts'
    plot.circle(x='x', y='y', size=2, color='impact', source=nodes_source)

    # Show the plot
    show(layout(plot))


plotNetwork(nodes=station_list, impact=df_bike_impact['color_impact'])
```

BokehJS 2.4.3 successfully loaded.



In [129…

```python
def clamp(n, min, max):
    if n < min:
        return min
    elif n > max:
        return max
    else:
        return n
```

In [129…

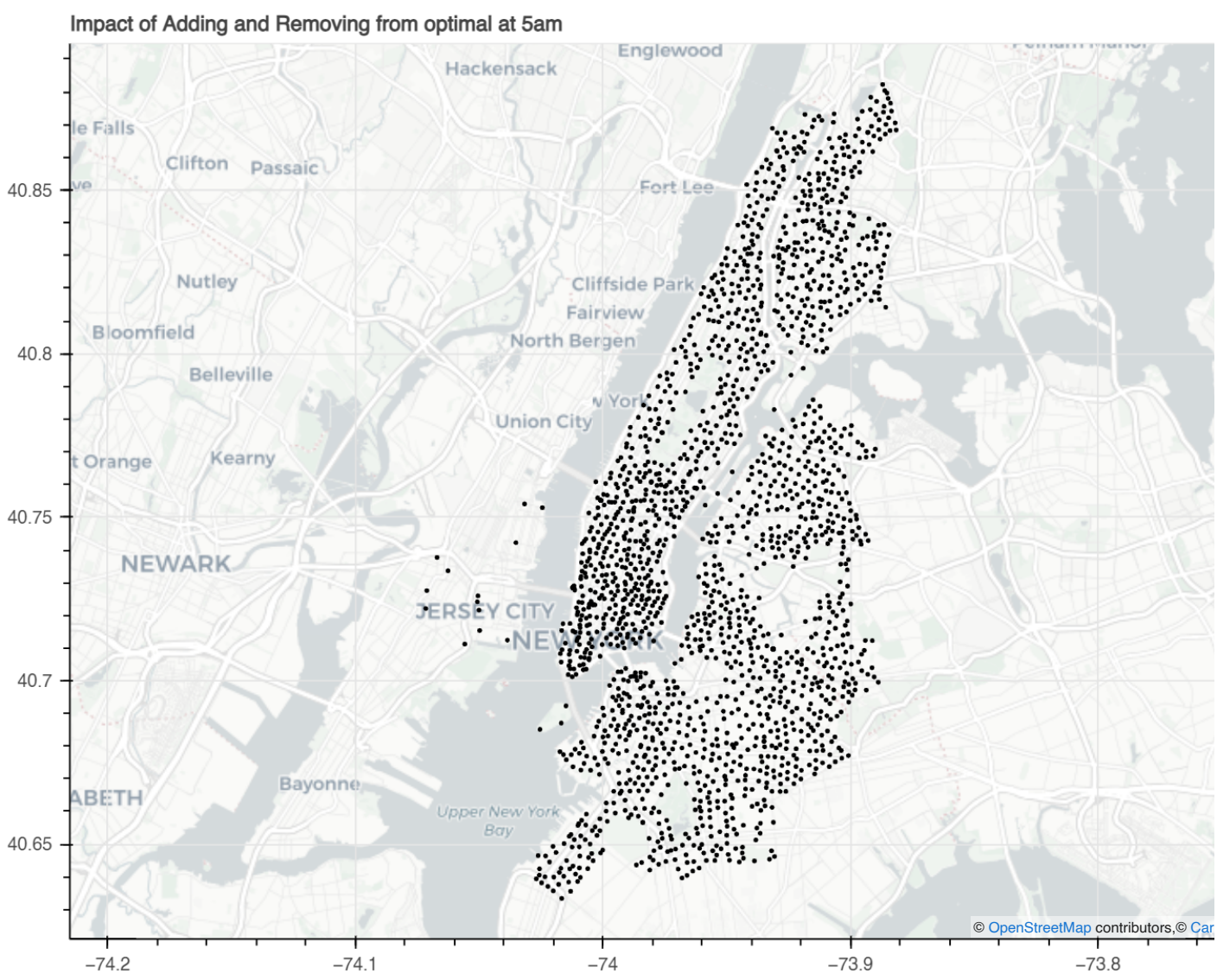
```python
# original logic written by me, modified by Chat GPT to include analysis for Bike Angels.
bike_impact = []
j = 0
for i in station_list['id'].iloc[:,0]:
    capacity_i = int(capacity[capacity['end_station_id'] == str(i)]['capacity'].iloc[0])
    mus_i = mus[mus['id'] == i][['hour', 'counts']].copy()
    all_hours = pd.DataFrame({'hour': range(5, 24)})  # Adjusted to include 5 am
    mus_i = pd.merge(all_hours, mus_i, on='hour', how='left').fillna(0)
    mus_i.set_index('hour', inplace=True)

    lambda_i = lambdas[lambdas['id'] == i][['hour', 'counts']].copy()
    all_hours = pd.DataFrame({'hour': range(5, 24)})  # Adjusted to include 5 am
    lambda_i = pd.merge(all_hours, lambda_i, on='hour', how='left').fillna(0)
    lambda_i.set_index('hour', inplace=True)
    half_full = x_is[j]
    x_at_2pm = half_full

    one_more = clamp(x_at_2pm,0,capacity_i)
    one_less = clamp(x_at_2pm,0,capacity_i)
```

```
    unhappiness_with_one_fewer = calculate_unhappiness(one_less, mus_i, lambda_i, capacity_i, start_hour
    unhappiness_with_one_more = calculate_unhappiness(one_more, mus_i, lambda_i, capacity_i, start_hour=
    unhappiness_with_current = calculate_unhappiness(x_at_2pm, mus_i, lambda_i, capacity_i, start_hour=5

    # Determine which action (add, remove, or no change) minimizes unhappiness
    min_unhappiness = min(unhappiness_with_current,unhappiness_with_one_fewer, unhappiness_with_one_more
    if min_unhappiness == unhappiness_with_current:
        color = 'black'  # No change needed
    elif min_unhappiness == unhappiness_with_one_more:
        color = 'dodgerblue'   # Adding a bike is better
        print(x_at_2pm)
        print(unhappiness_with_one_more)
        print(unhappiness_with_current)
    else:
        color = 'red'    # Removing a bike is better
        print(x_at_2pm)
        print(unhappiness_with_one_fewer)
        print(unhappiness_with_current)

    bike_impact.append((i, color))
    j += 1

df_bike_impact = pd.DataFrame(bike_impact, columns=['id', 'color_impact'])
```

In [129…

```
from bokeh.palettes import RdYlGn
def plotNetwork(nodes, impact, on_map=True):
    output_notebook()
    nodes['impact'] = impact
    # Prepare the data for plotting
    nodes_source = ColumnDataSource(nodes)

    # Define the plot
    plot = figure(title='Impact of Adding and Removing from optimal at 5am',
                  x_axis_type="mercator", y_axis_type="mercator",
                  width=800, height=600)

    # Add map tile if requested
    if on_map:
        plot.add_tile(get_provider(Vendors.CARTODBPOSITRON))

    # Add the nodes to the plot with color mapped by 'counts'
    plot.circle(x='x', y='y', size=2, color='impact', source=nodes_source)

    # Show the plot
    show(layout(plot))


plotNetwork(nodes=station_list, impact=df_bike_impact['color_impact'])
```

BokehJS 2.4.3 successfully loaded.


Impact of Adding and Removing from optimal at 5am
Englewood
Hackensack
Clifton
Passaic
Fort Lee
Nutley
Cliffside Park
Fairview
Bloomfield
North Bergen
Belleville
Union City
Kearny
NEWARK
JERSEY CITY
Bayonne
Upper New York Bay
40.85
40.8
40.75
40.7
40.65
−74.2
−74.1
−74
−73.9
−73.8
© OpenStreetMap contributors,© Car

In [ ]: