IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Digital Object Identifier

Group 4 - Report

JAMES CLACKETT, MAYA CLINTON, MARC CONROY, JIN KUAN MOO, HARRY Ó CLÉIRIGH, AND CIAN REILLY
Group 4

ABSTRACT Manhattan was often the settling ground for the many immigrants that came through Ellis Island during the 19th to 20th century. It became a rich melting pot in which significant cultural, historical, and religious events, from all around the world, are celebrated right up until the present day. Events like the St. Patricks Day Parade or the NYC Pride Parade have a significant impact on the urban dynamics of Manhattan. Using historical taxi, subway and permitted events data, provided by the City of New York and the Metropolitan Transport Authority, we trained an XGBoost and Random Forest model to predict both the baseline busyness of Manhattan, and the busyness of Manhattan when large scale events are unfolding. In doing so, we were able to prove that Manhattan's urban flow and busyness changes significantly throughout the island during these events. We built a web application surrounding this discovery, named *Afterparty*, that gauges and compares the everyday busyness with days in which such significant public events occur. With a frontend of React, and a robust backend consisting of Node and Flask, we were able to develop a powerful and insightful application. Afterparty has many valuable use-cases: enterprises can predict business in relation to their location; residents can determine locations in which to engage or avoid; city planners and other governmental organisations can see the predicted flow of people, the event, and traffic, contributing the planning of all manner of things such as law enforcement presence, road closures, sanitation and so on.
URL to application: https://afterparty.solutions/

I. INTRODUCTION

Afterparty is an event impact analysis tool, with the primary goal of providing insights into how the busyness of urban areas are impacted by large-scale events. Throughout the development of Afterparty we sought to harness the power of data analysis to unravel patterns, trends, and insights that underlie the ebb and flow of human activity as a result of large events. Indeed, we wanted to build an application that would allow our users to visualise these trends and provide them with meaningful insights into how the city's "urban flow" is fundamentally altered. This term, urban flow, is a concept we used throughout the project's life cycle to refer to the interconnected relationship between human activity and movement in an urban environment. From the outset, our contention was that large scale events - particularly large-scale concerts, festivals, and parades - fundamentally altered the city's default urban flow, and we looked at leveraging data analytics and machine learning to demonstrate this impact.

During the data exploration phase of this project, we alighted upon an extensive dataset that listed all publicly registered and permitted events occurring in New York City (Manhattan inclusive), over the duration of 15 years. And while trying to cross-reference some of the larger events listed on this public register with academic reports on the impact large scale events has on urban areas, we discovered a project undertaken at CARTO, a location intelligence platform, titled "A Map of Where People Went After the NYC Pride Parade" [1]. This report examined the post-event movement of people after Pride NYC in 2016, and demonstrated that through the use of spatial clustering, it is possible to discover human movement patterns that align with public testimony from parade-goers as to where they were before and after the event. This conclusion piqued our interest, and we questioned whether the analysis carried out by CARTO in 2016 on the post-event movement of people after Pride NYC could be extended to other large scale events occurring in Manhattan?

Research exists to support the claims that large-scale events do indeed leave a signature on the busyness of an otherwise ordinary day. While many solutions can predict the recurring traffic situations [2], we address the challenge in predicting and analysing the non-recurrent congestion caused by large events. Event impact research has been at the core of event studies since its beginnings [3]. One reason is that an event has an almost ideal methodological set-up for research: during a few hours, the individuals, the community and the

region are subject to the impact of a well-defined event [4]. This means for us, that both the cause and effect are comparatively easy to define, limit and measure in terms of its spatial and temporal progression.

Armed with the insight that human movement patterns are impacted by large scale events, we set out to build Afterparty, and create a tool that could assist city planners, event organisers, and residents alike in anticipating the changes in urban flow on the day of large events. We planned to leverage machine learning to accurately predict the impact an event has on the busyness of Manhattan, and visualise this impact for our users through the medium of an interactive map. Our objective was not merely to provide our users with a means of visually interpreting the output from our machine learning models, we also wanted to provide our users with rich insights into the underlying trends in the data.

II. LITERATURE REVIEW

Predicting taxi demand has been studied in the literature. Traditional approaches include regression and time series analysis. One study [5] analysed New York City taxi data, along with additional demographic, socioeconomic, and employment data to try to determine the key factors that affect taxi demand. Utilising multiple linear regression, they found the six most influential factors were population, education, age, income, TAT (transit access time), and total jobs in modelling demand in New York. Other approaches included use of neural networks. One example [6] used recurrent neural networks in an attempt to model real-time taxi demand. They proposed a LSTM-MDN (Long Term Short Memory-Mixture Density networks) model which could get approximately 83% accuracy across the entire city. They also found extending the model to a conditional model that also takes current demands in order areas improved the model's performance and that this outperformed two other models; one based on fully connected feed-forward neural networks and the other on a naïve statistic average. Another investigated a deep-learning approach, where unstructured text data was combined with time series analysis to determine if there was a hidden performance increase that could be wrought out through data engineering [7]. They made use automated web scraping to collect data, and word embeddings and convolutional layers in order to capture patterns from the text data, and found that fusing traditional tine series analysis with deep learning can significantly reduce forecasting errors.

Understanding collective human movement dynamics during large-scale [8] by Junchuan Fan and Kathleen Stewart provide a similar methodology regarding data analytics in relation to a significant event. The authors used geo-referenced tweets to predict movement human movement during the 2017 Great American Eclipse. The event was broken down into the only possible locations across the United States in which it could be viewed, known as the Path of Totality (POT). To validate their analysis, they juxtaposed the geo-tweets with Wyoming and Idaho state traffic data to determine users moving to and from these states. They took particular care to account for the bias created in high population areas in these states, as the relatively small population sizes of the two states has a population skew towards larger urban areas. While this paper examined a country-wide event, the researchers used techniques that could be replicated in our own project, such as breaking the geo-data into more manageable and compartmentalised zones, and data engineering on the primary datasets. This paper provides honest and critical commentary of their own work in relation to data analytics, its gathering and processing, and this methodology was something we looked to reproduce from the outset.

Comprehensive examination of the relevant academic literature demonstrates the feasibility of modelling taxi demand, particularly as a measure of activity in densely populated cities like New York. Moreover, empirical findings illustrating the influence of events on movement patterns have validated our belief in our core hypothesis that events impact urban busyness. Our application fuses these two findings and explores their overlap, providing an exciting new area for research.

III. METHODOLOGY

While developing Afterparty, we adopted the agile development methodology to ensure that our cross-functional team could collaborate efficiently and build our product within the eleven week development period. An outline for an agile approach was provided by the course coordinators, who had divided the eleven weeks into a series of five two-week sprints, with an added week at the beginning of the term dedicated to data exploration and concept ideation. These pre-defined sprints articulated the learning goals and outputs required to maintain a structured progression throughout the term. Using this schema, we were able to maintain a focused development cycle, ensuring that each sprint delivered a tangible increment towards the final build of Afterparty. This approach also fostered open communication, and promoted adaptability and continuous improvement throughout the development cycle as changes and fixes were swiftly identified, discussed, and implemented.

When evaluating the various frontend architecture paradigms we could adopt for this project, both the single page application with client side rendering (SPA) and server side rendering with page pre-rendering (SSR) paradigms were considered. However, for this project we opted to follow the SPA paradigm over SSR for the following reasons:

Dynamic Page Loading: While SPAs are slower to return page content when the page initially loads, when compared to SSRs, any subsequent query to the backend does not trigger page reloads [9]. This would allow for a more fluid user experience with fewer reloads, and enable us to develop a more dynamic and reactive application.

Simple Development Structure: SPA architecture is simpler in comparison to SSRs, as the primary focus is given to client-side logic and calling restful APIs to access data provided by the backend [10].

Scalability: SPAs confer an advantage over SSRs in that they are easier to scale, given that the client handles most of the processing, and the server is primarily responsible for serving APIs and the corresponding data. The decoupling of our client-server logic enforced by SPAs architecture would allow us to scale our backend independent of frontend, allowing for a more modular system structure.

With a SPA architecture decided on, the next decision was to select an appropriate frontend framework to facilitate development on Afterparty. The decision to build our frontend using a framework over static Javascript was ultimately influenced by the modular component design and state management tools offered by modern front-end frameworks. We felt that these features would allow for a more organised and maintainable codebase and would assist in our development of Afterparty, despite being new technology to the majority of the team.

For this project, three frameworks/libraries were considered: React; Vue; Angular. In reviewing all three frameworks in light of our project goals, React became the leading candidate. This was primarily for two reasons: from a performance perspective we wanted to ensure that our application would be efficient in rendering updates to the DOM, and we felt that React's virtual DOM manipulation would assist here [11]; with regards to prior experience and familiarity with React, our backend lead had experience in using React and provided reassurances that the learning curve was relatively gentle.

The decision was taken to handle mobile device responsivity by dynamically adjusting the UI and serving our users a build tailored specifically to their device type. This was achieved by running a check on the user's browser to see if it supported landscape orientation. If the user passed this check, then they were served the mobile build, otherwise they were served the desktop build. While this entailed maintaining separate mobile and desktop builds, it also ensured that our users were served the most optimal version of our UI given their device format (see Figure 1 and Figure 2 for comparative UI rendered for user on desktop and mobile).

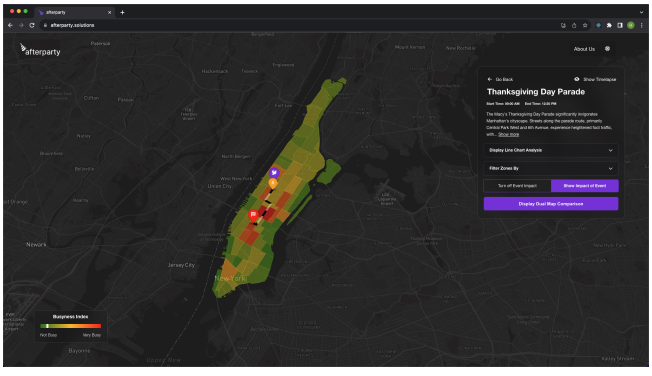

FIGURE 1: UI for desktop build of Afterparty

Regarding the back-end methodology, a two-sided approach to designing our project was proposed by our back-end lead. Rather than trying to accomplish our goals within one web application, he suggested that for separate goals, we should build separate applications. This would make our work more maintainable and scalable, and ensure its re usability in the future. The goals for our project were firstly to generate busyness scores for zones around New York City, and secondly to visualize that busyness with an interactive UI. It was thus decided that our back-end would comprise of both a Prediction API and a web application for serving our front-end.

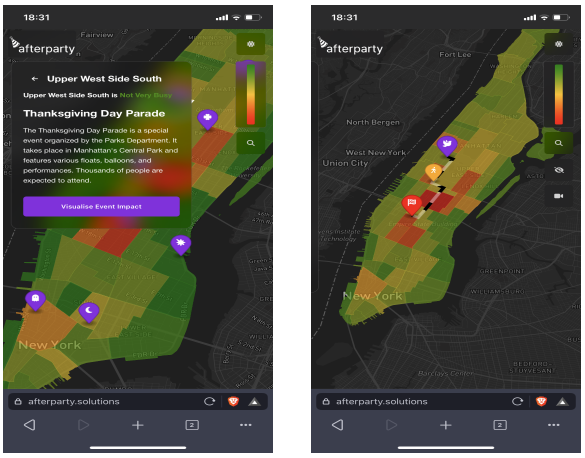

FIGURE 2: UI for mobile build of Afterparty

A number of back-end frameworks were being considered for our purposes; ExpressJS and Flask in particular. Two articles were found comparing the performance of both. The first [12] compared the performance of ExpressJS and Flask, while the other [13] also compared ASP.NET. While we were not considering ASP.NET, this articles nevertheless assisted our decision making process.

Flask and Express were being considered primarily on the assumption that they were intuitive, popular, and reasonably performant. It was also our understanding that Express was superior to Flask, but had a steeper learning curve. Challapalli et al. carried out performance tests on the two frameworks, and confirmed that Express did in fact significantly out-perform its competitor. It could handle almost 250 times as many requests as Flask [12]. Qvarnström and Jonsson also confirmed that Flask had a slower response time but a better throughput potential [13].

At the early investigative stage of our project, these articles helped us to understand the performance difference between our two candidate frameworks. With this information, it was decided that serving the front-end would require more computational power and therefore should be left to Express. Flask would be better equipped to cope with the less demanding tasks of a prediction API. In essence, the architecture of our app would be a Flask Prediction API for generating busyness scores, and an ExpressJS web server that would act as a middleman, serving the front-end, and communicating with our API (see Figure 3 below for an illustration).

There were of course other factors in our decision. For

example the benefit of Flask is that it is lightweight, which was an important consideration for us due to server constraints. It also has the benefit of using the same programming language our data team would be using. Going forward, their Python machine learning work would have a high degree of compatibility with the back-end. Similarly, as a JavaScript framework ExpressJS seemed the logical choice for an application serving a ReactJS front-end. It is also modular, performant, and an extremely popular choice among back-end engineers. As our project was ultimately an educational pursuit, gaining experience in such a popular framework was an unmissable opportunity that greatly influenced our choice.

To ensure the effectiveness of our back-end later in the project, we aimed for security, performance, and reliability. Reliability was achieved using the built-in error handling classes native to JavaScript and Python. Security was taken care of in two key ways. First, an API authentication system was implemented using JSON Web Tokens (JWT) and Bcrypt. Secondly, IP rate limiting was implemented using Express rate limit and Flask limiter. In terms of performance, ExpressJS was sufficiently powerful for our needs. Flask however was not so effective, especially when handling concurrent users. Performance was therefore enhanced with Flask Cache. It was also improved to some degree by the introduction of an SQLite Database and the SQLAlchemy O.R.M in our fourth Sprint.

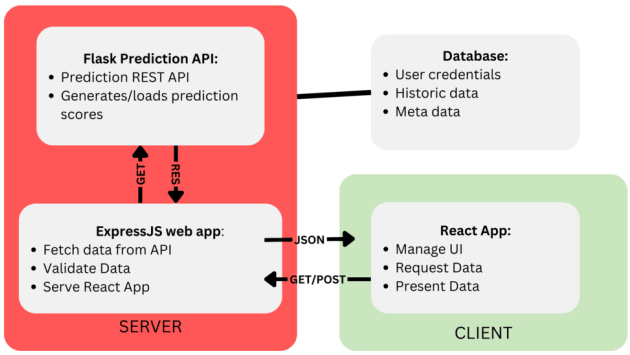

FIGURE 3: Back-end architecture of *Afterparty*

The application was mounted on an Ubuntu server provided by our academic institution. This created its own set of challenges.

The server is running a Common KVM (Kernel-based Virtual Machine) Processor, at a respectable 3.00Ghz. Unfortunately, it is single cored. This caused performance dips within our machine learning models as both XGBoost and Random Forest are naturally inclined for parallelisation with their decision tree like structure. The CPU also lacked AVX (Advanced Vector Extensions), likely as it was virtual, which further hindered our ability to process large amounts of data including the option to even install MongoDB and other databases that required this AVX feature.

Another restriction was the limited amount of RAM, which was only 2GB. This limitation caused significant problems in executing larger files. For example, our Random Forest model, which was just over 1GB in size, occupied over 70 percent of the available memory when in use. This caused not only the server itself to lag but also led to delays in the web application returning data. If this model was used concurrently, it would completely crash the python environment in which it was running, and as a result we decided against using this model.

As we did not have physical access to the server we had to thread carefully on implementing features that may sever our connection via SSH. For example, we were unable to install a firewall (UFW) for this very possibility. For security reasons, we decided to limit as many open ports as possible. To further boost security, we included an Apache Reverse Proxy Web Server configured with Certbot and Let's Encrypt (HTTPS) to act as a secure gateway to our Node backend. The reverse proxy also handled the encryption and decryption of in-going and outgoing requests. As our Node was directly communicating with the Apache Reverse Proxy we also configured it to run on HTTPS, giving it the same access to the private key and certification provided by Certbot and Let's Encrypt, the communication between these two were secured even further. The server did provide great performance when the application was molded around it which makes us believe that when upgraded hardware wise, the performance can be improved rather easily.

IV. DATA ANALYTICS AND VISUALISATION

With accurate insights being the core of our application, the choice and quality of datasets used was paramount. A comprehensive search was conducted across various websites such as NYC Open Data, data.gov, the Metropolitan Transportation Authority (MTA), Taxi and Limousine Commission's (TLC) website, and Kaggle. The following datasets were chosen and examined:

- **2022 TLC Taxi Trip Records:** This dataset details historical trips and includes fields for pickup and drop-off time and location. It also contains information on trip distances, fare amounts and tip amounts.
- **2022 MTA Subway Hourly Ridership:** This dataset gives hourly ridership for each subway stations across New York.
- **New York Historically Permitted Events:** This provides the date, time, and location of all registered events that have taken place.
- **2022 Historical Weather Data:** A supplementary dataset that was merged with the taxi and subway data, in order to determine if weather impacted ridership.

Among all, the taxi and subway data were chosen to be our main indicator of busyness. The granularity provided for by both datasets down to the individual hour was esssential in capturing the dynamically changing busyness levels across the duration of an event taking place. The event dataset is used primarily for event selection, while weather data served as an potential enhancement to our busyness prediction. It is worth mentioning that data spanning the period from 2019

to 2021 has been excluded from our data analysis, as the Covid-19 pandemic had a significant impact on transportation trends, resulting in substantial deviations from expected outcomes.

Realising the need to stress test our main hypothesis, we examined the impact of two major events, Macy's Thanksgiving Parade and St. Patrick's day parade, on Manhattan's baseline busyness. Using 2014 taxi data, partly due to its inclusion of latitude and longitude coordinates, we created a timelapse that plotted the coordinates throughout the duration of the events. As seen in Figure 4, our hypothesis became vividly apparent, effectively reaffirming a pronounced correlation between events and Manhattan's urban behaviour. The timelapse highlighted the dynamic trends, such as the parade's progression and pickup and drop-off hotspots. This stress testing boosted our confidence in the feasibility of our idea.

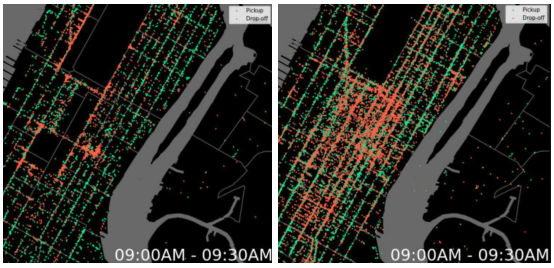

FIGURE 4: Macy's Thanksgiving Day Parade; impact (left) vs baseline (right) using taxi pick-up and drop-off locations.

Thorough data exploration and cleaning were conducted prior to analysis. For the taxi data, the most obvious data quality issues included missing, negative, and outlier values. A number of logical integrity tests were performed, including checking for dates outside of 2022, passenger counts above the legal limit, and implausible trip distances to name a few. The affected rows were dropped, as they comprised only a small proportion of the overall data.

The subway data, on the other hand, required more extensive cleaning due to a format difference between January and rest of the months. For example, January data counts a running tally of entries and exits per individual turnstile every four hours, while the data for rest of the months counts the total entries and exits per station per hour. To solve this, two external CSV files were used to link turnstiles to their corresponding stations. Entries were averaged across the 4-hour window, in order to approximate hourly entries as seen in the new format. This cleaning process facilitated a seamless merger with the newer format.

Lastly, weather data was also investigated. Rainfall appeared to have little to no impact on ridership, while heavy snowfall did. However, due to a lack of days with snowfall, the data has less significance on reflecting the actual impact it brings to the busyness, and as a result, the data was not adopted for modelling.

Extensive data analysis was performed, with the goal of identifying useful features for predicting busyness. For taxi data, drop-offs per hour were our target feature. For subway data, it was hourly ridership. Similar trends were seen across both datasets, with date-time features being the most promising for determining busyness. We can see in Figure 5 below, the hour of the day has a significant impact on ridership. Late night is by far the quietest, with an uptick in the morning, before peaking in the evening and tapering back down.

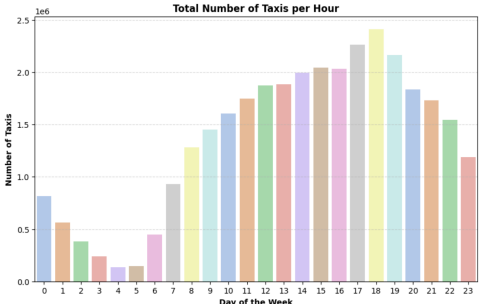

FIGURE 5: Number of taxi trips per hour of the day

Similarly, a significant amount of variation can be seen across days of the week, days of the month, and months themselves. Other trends such as holiday trends were observed. For example, Christmas week was significantly quieter comparatively than the weeks preceding it. By aggregating all pickups and drop-offs per zone for the entire year, we also observed a distinct variation across all zones (Figure 6).

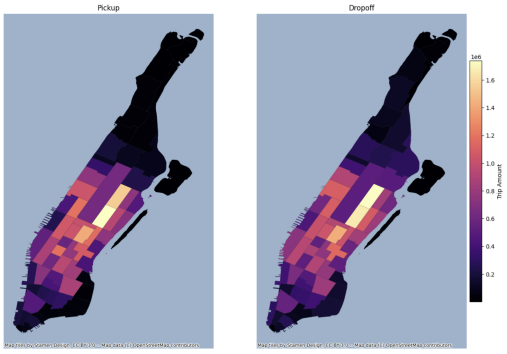

FIGURE 6: Heatmap of total pickups/dropoffs per zone

Continuous features such as trip distance and fare amount were investigated, however they showed little insight for our particular application. As such, zone Location ID and date-time information were selected as our primary features for our predictive modelling.

Integration of Subway data into the taxi dataset was essential. This was achieved by mapping stations to their respective taxi zones based on geolocation. We then aggregated the hourly taxi and subway ridership into a single busyness metric. This was then normalised to a value between 0-1. The normalisation served two purposes. It made integration with the front end easier, and relative busyness made more sense in the space of an event impact analysis tool.

For the train-test split, we separated the last 5 days of each month as test data, keeping it completely unseen during training. We employed random stratified sampling, taking 20,000 rows per month and applying an 80/20 train-test split. 5-fold cross-validation ensured even more rigorous testing. An integral component of model refinement was feature engineering. Based on the the patterns previously identified, new columns to distinguish between weekdays and weekends, as well as split hours into time blocks representing morning, afternoon, evening and night were added.

During the stage of selecting the most suitable machine learning algorithm, we explored a diverse range of models. After initial testing, Random Forest and XGBoost were chosen due to their ability to handle categorical features well. Support Vector Machines, regular gradient boost, and catboost were tested. They underperformed comparatively, so we shifted our focus to fine-tuning the former models. To strike the right balance between computational speed and precision, we further employed Randomised Search cross-validation, fine-tuning the parameters for enhanced performance. Upon assessing both models, the performance of both are visible in Table 1.

TABLE 1: Performance evaluation metrics for general busyness prediction model

	Random Forest	XGBoost
MAE	0.0091	0.01103
MSE	0.00045	0.00044
MAPE	12.52%	17.33%
R^2	0.948	0.9493

Given the context of predicting a normalised value between 0-1, these results are quite accurate. In summary, both models perform quite similarly and have very strong predictive capabilities. Note that MAPE (Mean Absolute Percentage Error) was calculated on the top 3 busiest zones, in order to avoid zero division. Initially, we opted for the Random Forest model due to its superior performance. However, the model is over 1GB which led to substantial memory overload issues on the server, ultimately causing the environment to crash. In response, we have chosen XGBoost as our final general busyness model. While XGBoost exhibits slightly lower accuracy, its computational efficiency makes it the preferred choice in ensuring smoother operation on the server.

Having the general busyness readily set up, our focus shifted to events and their associated busyness. We aimed to concentrate on events that are well-known and considered to have major impact on the busyness of the city. However, our exploration of the New York historically permitted events dataset revealed a notable challenge. Over 95% of the recorded entries consisted of minor events with negligible influence on the busyness of the city. This disparity was due to the dataset's comprehensive nature, capturing all registered events irrespective of their scale and size. Additionally,the dataset did not provide relevant insights into event sizes, introducing additional complexity to the data cleaning process. In essence, the challenge revolved around the lack of relevant information to evaluate the size of an event and its corresponding influence on the busyness of the city. The absence of this crucial information impeded our capacity to make well-informed decisions regarding whether to include or exclude specific events in our analysis.

At the end, a decision was made to manually select the events to be included by undertaking web scraping. Relevant information such as event size and popularity were gathered from official websites or online journals [14]–[26], and the frequency of occurrence of an event is also used as an indicator to evaluate the significance of an event. Under such criteria, a curated list of eight events were selected, including Thanksgiving Parade, Halloween Parade, Saint Patrick's Parade, Pride Parade, Times Square Ball, Lunar New Year Parade, Three Kings Day Parade, and Macy's Fourth of July Fireworks. By extracting the corresponding event information in year 2018 from the event data, we were able to derive busyness of each event from the previous taxi and subway data.

To that end, a similar but separate event busyness model is built on top of the general busyness model to accurately reflect the impact. The event model takes parameters including the events in the form of event ID, zones, hour,time of day (morning, afternoon, evening, night), and the output from general busyness model of same situation. Essentially, the event model functions by applying a multiplier to the general busyness prediction, accounting for the additional busyness attributed to the occurrence of an event. A diverse range of machine learning models were tested and as expected, Random Forest and XGBoost significantly outperformed the rest. The performance metrics of the following two models is shown in Table 2 . In this case, XGBoost is chosen as our event busyness prediction model.

It is noteworthy that Random Forest outperforms XGBoost in the general busyness model, while XGBoost outperforms Random Forest in the event busyness model. The main reason suspected is due to the nature that XGBoost performs better than Random Forest when data imbalance exists. Upon closer investigation of the training data, a trend emerged where certain zones consistently exhibited high levels of busyness. For example, zones in mid-Manhattan consistently displayed extreme busyness levels. Consequently, this resulted in significant data imbalance, with a few zones consistently at very high (0.9-1.0) busyness levels, while many others maintained lower (0.1-0.2) levels. This data imbalance is exacerbated during events, as these tend to attract more crowds to the zones where they are hosted. In the case of highly imbalance data, XGBoost has been favourable over Random Forest which is also observed in other researches. [27]–[29]

V. EVALUATION AND RESULTS

To test our application's performance on the client-side, we regularly ran audits on our performance using Lighthouse, an auditing tool integrated directly into Chrome DevTools. With every new stable build from the MVP onwards, a

TABLE 2: Performance evaluation metrics for event busyness prediction model

	Random Forest	XGBoost
MAE	0.025	0.024
MSE	0.0019	0.0015
RMSE	0.044	0.039
MAPE	0.165	0.136
R^2	0.932	0.944

performance audit was undertaken to understand how well the application was performing. In later weeks, the insights gleaned from routinely running these audits were instrumental in ensuring that our application became as performant as possible. The changes made to our application as a result of our performance auditing included: image compression; code minimisation; lazy loading; code splitting; library partitioning. Implementing these changes had a transformative effect on our application, resulting in, on average, a largest contentful paint (LCP) of 1.1 seconds and a relatively stable speed index of 1.4 seconds (see Figure 7). This was a remarkable increase from the average LCP speed of 2.4 seconds that we had been hitting previously.

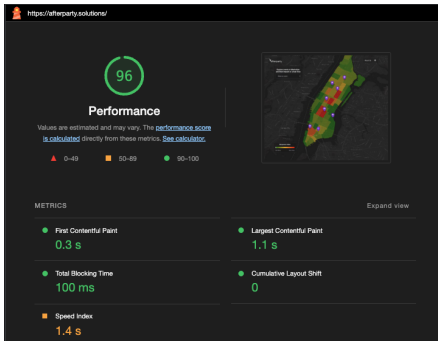

FIGURE 7: Lighthouse audit on deployed application

Evaluation on the front-end UI was also conducted. Initial user testing was carried out utilising the MVP build of our application. This assessment involved participants' engagement in a series of five tasks. For example, participants were asked to envision being interested in the activity level within the Bloomingdale neighbourhood at the start of the Thanksgiving Day parade. They were then requested to demonstrate the steps that would be taken to determine whether Bloomingdale was experiencing increased or decreased activity due to the parade. Subsequent to the tasks, the participants' performance was evaluated and categorised into several groups, ranging from "Task Successfully Completed," to "Task Abandoned." Additionally, an after-scenario questionnaire (ASQ) was administered to gain insight into the participants' perspectives on the assigned tasks.

As a result of our UX testing, improvements were made to our UI that aligned with the feedback received. Reusable accordion components were implemented to separate core features and minimise visual clutter. Similarly, a unified button design was implemented to ensure consistency throughout the interface and to simplify user-flows through the application. Users were encouraged to interact directly with map zones, resulting in dynamic changes within the line chart feature, thus eliminating the need for a drop-down menu offering 66 options (one for every zone). Further refinements encompassed the augmentation of the line chart description with event start and end times, thereby enhancing users' comprehension of the event timeline. These alterations yielded a notable increase in the rate of successful task completion by users, rising from 51% in the MVP build to 91% in the final production build of Afterparty.

To test our back-end, a combination of Apache Bench and Artillery tests were carried out. We carried out automated testing server-side with Apache Bench and ascertained that the Reverse Proxy was able to handle a significant amount of concurrent requests relative to what we as a team expected given the hardware provided. The tests also showed that the rate limiters within Express and Flask were working as expected; results over the allowed number were being filtered out or ignored.

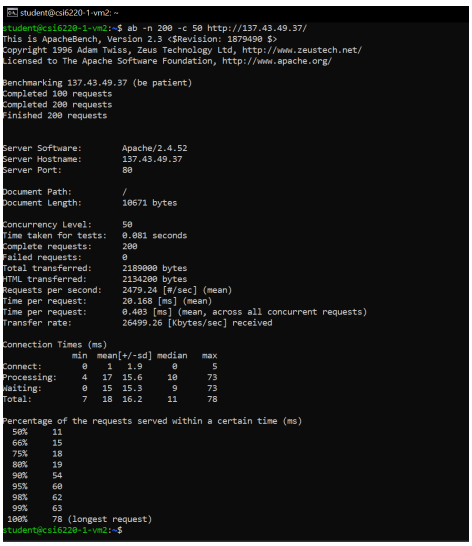

FIGURE 8: Apachebench test results with rate limit working

Local performance testing of Express and Flask were carried out using Artillery. Load, stress, spike and soak tests were run on both. The aim of this local testing was not to ascertain the real performance of our app, but rather to flag any potential issues. For the most part, our back-end serves static data, and here it performed very well, with high concurrency and low response times. However tests highlighted that there was a performance issue with our predictive functions. Concurrency was almost impossible when using our Random Forest model given its large file size, and only somewhat better when using our XGBoost model. We realised that for every concurrent request, Flask created a new instance of the model resource, rather than forcing clients to share it. This led to a major memory bottleneck. Due to the late stage of our project, only rudimentary fixes like caching could be

implemented, however, this was enough to solve our memory bottleneck for most use-cases.

VI. CONCLUSION AND FUTURE WORK

In summary, we first identified a clear positive correlation between the urban flow of the city and large-scale events through our research and data exploration. This insight, in turn, allowed us to identify an opportunity to develop a data analytics visualisation tool capable of articulating an event's impact on the busyness of Manhattan through the power of machine learning, which can help event organisers, city planners, and residents alike in anticipating the changes in urban flow on the day. The development process has been a collaborative endeavour, involving the efforts of the front end, back end, and data teams. Together, we worked to determine the most fitting technology for each aspect of the tool, followed by subsequent optimisation efforts. Afterparty is fully capable of fulfilling its designated role, though there are still room for further improvements.

Currently, the web application is experiencing performance issues when handling concurrent prediction requests. Future work could be done on optimising the use of memory by designing a system that loads models into a queue or buffer system. In addition to this, a distributed system could be built which routes requests to a second prediction API once the system load surpasses a predefined threshold. The incorporation of a load balancer within the backend infrastructure would notably enhance the scalability of the tool, subsequently bolstering its overall performance. On the other hand, leveraging a server with higher memory specifications enables the deployment of the previously developed Random Forest model, thereby improving the overall prediction accuracy.

Furthermore, significant opportunities for improvement lie in expanding the scope of events covered by the application. In the future, efforts could be directed towards including a wider variety of event types and categories, extending beyond parades to encompass sports events, music festivals, and more. This expansion could involve integrating additional datasets to enhance the precision of busyness predictions, enabling the incorporation of multi-day events as well. Given the solid foundation and established infrastructure, there is potential to expand Afterparty coverage to encompass locations beyond its current boundaries. This adaptability would enable its utilization on a global scale, making it suitable for a wide range of locations worldwide.

REFERENCES

[1] CARTO. A map of where people went after the nyc pride parade, 2016. Carto Blog, Accessed on 2023-08-08.

[2] Ruwangi Fernando. The impact of planned special events (pses) on urban traffic congestion. Technical report, Melbourne, Australia, 2019.

[3] J.R. Brent Ritchie. Assessing the impact of hallmark events: Conceptual and research issues. Journal of Travel Research, 23(1):2–11, 1984.

[4] John Armbrecht and Tommy D. Andersson. Subjects and objects of event impact analysis. Scandinavian Journal of Hospitality and Tourism, 16(2):111–114, 2016.

[5] Ci Yang and Eric J. Gonzales. Modeling Taxi Trip Demand by Time of Day in New York City. Transportation Research Record: Journal of the Transportation Research Board, 2429(1):110–120, January 2014.

[6] Jun Xu, Rouhollah Rahmatizadeh, Ladislau Bölöni, and Damla Turgut. Real-Time Prediction of Taxi Demand Using Recurrent Neural Networks. IEEE Transactions on Intelligent Transportation Systems, 19(8):2572–2581, August 2018.

[7] Filipe Rodrigues, Ioulia Markou, and Francisco C. Pereira. Combining time-series and textual data for taxi demand prediction in event areas: A deep learning approach. Information Fusion, 49:120–129, September 2019.

[8] Junchuan Fan and Kathleen Stewart. Understanding collective human movement dynamics during large-scale events using big geosocial data analytics. Computers, Environment and Urban Systems, 87:101605, 2021.

[9] Dotsquares. Pros and cons of single page applications, 2023.

[10] Patrick Roos. Which web frontend architecture fits best?, Apr 2023.

[11] GeeksforGeeks. Reactjs virtual dom, 2023.

[12] Sai Sri Nandan Challapalli, Prakarsh Kaushik, Shashikant Suman, Basu Dev Shivahare, Vimal Bibhu, and Amar Deep Gupta. Web development and performance comparison of web development technologies in node.js and python. In 2021 International Conference on Technological Advancements and Innovations (ICTAI), pages 303–307, 2021.

[13] Eric Qvarnström and Max Jonsson. A performance comparison on rest-apis in express.js, flask and asp.net core, Jun 2022.

[14] New York City Tourism + Conventions. Annual events in nyc | your guide to major events. [online]. https://www.nyctourism.com/annual-events, Accessed on: Jul. 1, 2023.

[15] Alyssa Ochs. Top 10 annual events in new york city. [online]. https://www.tripstodiscover.com/top-annual-events-new-york-city/, Accessed on: Jul. 1, 2023.

[16] Dan O'Malley. 5 biggest annual events in new york city. [online]. https://nomadworks.com/blog/5-biggest-annual-events-in-nyc/, Accessed on: Jul. 1, 2023.

[17] Eleanor Tallon. Top 15 annual events in new york. [online]. https://www.clickandgo.com/blog/2020/02/13/top-15-annual-events-in-new-york/, Accessed on: Jul. 1, 2023.

[18] Tripadvisor. Events in new york city. [online]. https://www.tripadvisor.co.uk/Attractions-g60763-Activities-c62 New_York_City_New_York.html, Accessed on: Jul. 1, 2023.

[19] Macy's. Macy's thanksgiving day parade. [online]. https://www.macys.com/social/parade/where-to-watch/, Accessed on: Jul. 1, 2023.

[20] New York's Village Halloween Parade. Village halloween parade. [online]. https://halloween-nyc.com/participate/, Accessed on: Jul. 1, 2023.

[21] TimeOut. Village halloween parade in nyc 2022 guide. [online]. https://www.timeout.com/newyork/events-festivals/village-halloween-parade-nyc-guide, Accessed on: Jul. 1, 2023.

[22] The St Patrick's Day Parade Inc. Parade information. [online]. https://www.nycstpatricksparade.org/, Accessed on: Jul. 1, 2023.

[23] Gothamist. Chinatown's lunar new year parade is on sunday. here's what new yorkers need to know. [online]. https://gothamist.com/news/chinatowns-lunar-new-year-parade-is-on-sunday-heres-what-new-yorkers-need-to-know, Accessed on: Jul. 1, 2023.

[24] El Museo Del Barrio. 46th annual three kings day parade. [online]. https://www.elmuseo.org/event/3k2023/, Accessed on: Jul. 1, 2023.

[25] New York City Tourism + Conventions. Three kings day parade. [online]. https://www.nyctourism.com/events/three-kings-day-parade, Accessed on: Jul. 1, 2023.

[26] Macy's. Macy's fourth of july fireworks. [online]. https://www.macys.com/s/fireworks/, Accessed on: Jul. 1, 2023.

[27] Petr Hajek, Mohammad Z. Abedin, and Uthayasankar Sivarajah. Fraud detection in mobile payment systems using an xgboost-based framework. Information systems frontiers, pages 1–19, 2022.

[28] Yeongah Choi, Jiho An, Seiyoung Ryu, and Jaekyeong Kim. Development and evaluation of machine learning-based high-cost prediction model using health check-up data by the national health insurance service of korea. International journal of environmental research and public health, 19(20):13672, 2022.

[29] Kommana Swathi and Subrahmanyam Kodukula. Xgboost classifier with hyperband optimization for cancer prediction based on geneselection by using machine learning techniques. Revue d'Intelligence Artificielle, 36(5):665, 2022.