Group Report

C. EGAN, M. GIRT, N. HEGARTY, C. LARKIN, Z. LI, AND Y. WEI.
Group 5

ABSTRACT This project addresses the challenge of generating busyness predictions for locations in Manhattan and translating these predictions into actionable insights for end users. Specifically, it explores the decision-making process for prospective cafe owners when selecting an optimal cafe location in Manhattan. The project's objectives involve creating a responsive and intuitive data-driven web application which predicts busyness through the use of freely available taxi, Citi bike, and subway data. The project employs React with Mapbox and Chart.js for the front-end, while the back-end relies on Django, Redis Cache, and an SQL database to manage busyness predictions and cafe-related data. The XGBoost model emerges as the most effective machine learning algorithm for generating busyness predictions. The use of caching is found to significantly improve the website performance, however the performance is hampered by Mapbox's resource-intensive demands which affects its Lighthouse score.

Project url: https://csi6220-2-vm4.ucd.ie/

I. INTRODUCTION

For this project, our objective was to create a web-based data analytics application that predicts the level of busyness of various locations within Manhattan. These predictions were to be in part based upon New York City Taxi and Limousine Commission (TLC) Data. Building upon this groundwork, we were tasked with addressing a problem of our choosing. The problem we attempted to address was faced by aspiring business owners, where they should locate their business in Manhattan. Our application was tailored to cater to potential café owners, aiming to offer them a thorough and user-friendly tool for making well-informed decisions about where to establish their business. We were aware of the fact that different users prioritise different factors when choosing a business location, so we ensured that the application was able to accommodate these differing priorities.

With all this in mind, the key objectives of the project can be summarised as follows:

- Generate predictions about busyness levels, drawing from NYC taxi data and supplemented by relevant openly accessible data.
- Construct an intuitive application that empowers prospective business owners to assess different areas within Manhattan, guided by personalised criteria.
- Ensure the website's resilience and responsiveness, while adhering to the project timeline for successful delivery.

II. LITERATURE REVIEW

Our literature review is divided into three distinct sections. The first section explores literature relating to prediction generation and machine learning (ML). The second section reviews literature on web application frameworks. The third and final section examines the relationship between location selection and business success.

A. PREDICTION GENERATION AND MACHINE LEARNING

In order to guide our approach for generating busyness predictions, we reviewed recent literature focusing on journey time and demand prediction for various shared modes of transportation, including Citi bikes, buses, the subway, taxis and transportation network companies (e.g., Uber, Lyft). We analysed their approaches, algorithms, and features for prediction generation. We will discuss each of these studies in turn, highlighting their key findings and how they can be applied to our specific use case.

Work from Dunne and McArdle (2021) regarding bus travel time estimation makes an important distinction between long-term travel time prediction, which accounts for things such as cyclical traffic trends and general weather patterns, and short-term travel time prediction which instead deals with unpredictable events such as traffic collisions and road maintenance [1]. The authors opt to focus on long-term travel time predictions, using ML algorithms which forecast the duration of an entire bus trip, combining this approach with a dynamic post-prediction segmentation technique to offer predictions for specific segments of the journey. This is found to significantly improve prediction accuracy for partial bus journey times.

Although the focus of our work is not to generate predictions regarding the duration of journeys, we can employ this distinction between long-term and short-term predictions when generating our models. As we will be seeking to provide busyness predictions to prospective cafe owners, we will provide long-term predictions, as the goal of these prospective cafe owners is to stay in business for the foreseeable future. With this in mind, the features we shall be incorporating in our model will be those relating to spatial and temporal trends, rather than more unpredictable short-term events. We

will exclude weather from our models as this would limit the predictions to the timeframe offered by weather forecasts. In order to account for temporal busyness trends we will utilise a comparable technique to the one employed by the authors. This involves ordinally encoding time of day, day of the week, and month of the year. Additionally, we are contemplating the inclusion of a week of the year variable in order to introduce a more detailed temporal feature.

In order to help improve qualitative taxi demand prediction in New York City, Terroso-Saenz et al (2019) have proposed the use of online social networks (OSNs) (e.g. Foursquare, Flickr) [2]. This approach predicts taxi demand in terms of categorical labels ("high", "medium", "low") rather than generating quantitative predictions. Although the results offered by Terroro-Saenz et al appear promising, we wish to generate quantitative predictions and seek to provide predictions for a longer time period. Nevertheless the possibility of using these OSNs could certainly be incorporated into future models, especially if we wished to have separate models for short term forecasting. In line with this short term approach, Poongodi et al (2021) have demonstrated the efficacy of using XGBoost to predict taxi trip duration in New York City [3]. We shall build upon these findings, examining whether XGBoost demonstrates a similar level of performance for long term busyness predictions.

Gerte et al (2019) have analysed the impact of TNC (Transportation Network Companies) demand on taxi, Subway, and Citi bike usage [4]. Through the use of a dynamic linear model (DLM) they found that TNCs had a substitutive effect on taxi usage and a complementary effect on subway usage. Additionally they identified three types of exogenous variables, temporal, environmental, and land use, which impacted TNC, taxi, subway, and Citi bike usage. Broadly in line with these findings, Toman et al (2020) have demonstrated the importance of incorporating both temporal, spatial, and socio-demographic factors when predicting ridership patterns for TNCs [5]. This paper investigates patterns at the TLC zone level and identifies distinct spatial usage patterns between taxis and TNCs. Notably, these differences are primarily observed outside of Manhattan. Since our research exclusively focuses on Manhattan, these variations are of lesser relevance to our project. Nevertheless, both of these papers can help inform our approach to generating busyness predictions. Given that the usage patterns between taxis and TNCs has been found to be substitutionary and thus broadly similar, we will focus on examining the busyness patterns of subway and Citi bike usage, as it appears that they will likely differ from that of taxi usage.

Finally, it is worth highlighting the study by Dey et al. (2021), where they focused on modelling TNC demand and zone preference. In their Multiple Discrete-Continuous Model System, they included the number of restaurants and sidewalk cafés in a given TLC zone as a variable [6]. Their findings indicated that this variable had a positive and statistically significant impact on zone preference. This finding appears to raise the possibility of a virtuous cycle. Our underlying assumption is that the level of activity, as measured by taxi, Citi Bike, and subway usage, can contribute to the success of a café. What this particular discovery seems to suggest is that the presence of cafés, among other activity opportunities, can play a role in positively influencing zone preference. In other words, the existence of a café might enhance the attractiveness of a zone, thus leading to an increase in demand.

B. WEB APPLICATION FRAMEWORKS

Turning now towards literature pertaining to web application frameworks, David Johansson, a researcher at Linkoping University, posits that the crucial elements of making a web application include reusing functionality, adding new features, and fixing errors which React helps achieve [7]. However, to do this with apps of greater complexity it is necessary to use an architectural pattern. While his research is focused on finding the most optimal architectural pattern for building React applications, he says for simple web applications using Reacts prop drilling to pass data downwards through the component tree is more than enough. For a smaller application such as Café Compass, the component tree is not as deep and the data flow is less. If necessary, React's built in context API can avoid prop drilling and allow data to be globally accessible through the application. This proved ideal for the scope of our application, however, if we were to expand the complexity of the application Johansson urges developers to use react-redux or the Apollo Client which he says is the most maintainable based on a variety of metrics including lines of code.

Julia Plekhanova from Temple University stresses the importance of back-end frameworks stating they automate some of the backend, add structure to the code and increase readability and reusable components [8]. After evaluating a variety of frameworks, Plekhanova found that Django's use of Templates which separate application and logic from design make it easier for designers to work with developers. Based on her evaluation criteria Django came out as the best framework with especially high scores in user interface development, maintainability, and data management. Due to these reasons we went with Django over other frameworks.

C. BUSINESS LOCATION AND SUCCESS

In an ever-evolving business landscape, the symbiotic relationship between a company's geographical positioning and its success cannot be underestimated. To further explore this relationship we examined literature related to the link between location and business success.

What ultimately influenced our decision to create a web application to help prospective cafe owners find the ideal location to open a cafe was the literature demonstrating that the location ultimately correlates to the success of a business. Researchers have used a Gravity model to identify ideal locations for restaurants. Positive inputs in the model included traffic volume while detractors are the average level of unemployment and menu price. They found that more traffic

around a location provides more exposure to a restaurant thus benefiting sales [9]. Due to this analysis it was logical to use a variety of transit links in Manhattan as a proxy for traffic. Proximity to Bus stops, subway stops, and Citi bikes would be beneficial as more people would see the businesses near these locations. In addition to this, we included the average property price in each zone to allow users to see the general price point for that location. While unemployment records on a granular level were not available to us for Manhattan, we went with showing crime density as an undesirable trait for a location.

III. METHODOLOGY

In this section we will discuss the various technologies and frameworks used in our web application, detailing the rationale behind our choices, as well as the ways in which this architecture interacts. We will begin by discussing the use of the agile methodology in the project, turning then to front-end and UI decisions, after that we will move on to discuss the back-end and the use of caching, before finishing up with a discussion of how the application was deployed.

A. AGILE METHODOLOGY

Our overarching development strategy followed the principles of the agile methodology. We operated in five two week sprints, holding standup meetings three times a week, with sprint planning and sprint retrospective meetings, at the start and end of each sprint. In addition to our standup meetings, we also had meetings with our project manager and academic mentors, which provided us with valuable insights and guidance. To assist with our adherence to the agile methodology, we opted to use Atlassian's project management tools (Jira and Confluence), for their intuitiveness and extensive market popularity [10].

B. FRONT-END

The front-end of our application utilised React, a JavaScript framework used for building single page web applications. The reason we went with React is that it offered a variety of features that vanilla JavaScript was lacking. React's component based design made it easy to collaborate as the simultaneous creation of features did not impact other features as the code is encapsulated in each component. Additionally, React's hooks, a collection of functional components, allowed us to write cleaner code as it allows for more reusability in our logic. React also allows us the option of porting our web application to mobile devices using React Native.

One benefit of using React over vanilla JavaScript is its increased performance via the Virtual Document Object Model (DOM). JavaScript uses the DOM and whenever an element on the page updates JavaScript updates the entire DOM tree. Since React uses a copy of the DOM when an element is updated only the relevant node on the DOM tree is updated reducing the performance impact that would exist if the entire tree had to be updated [11].

When deciding how to display busyness data in Manhattan we wanted to use a map, thus a key decision was whether to use Google Maps or Mapbox. We had no plans on showing routes on our map which would have made Google Maps the clear choice, so instead we opted to use MapBox which gave us greater control in displaying data and customising the look of our map. Mapbox has a variety of built in functions that allows us to add GeoJSON data to our map as layers which we used to display markers on the map and create a custom heatmap.

The front-end of our application was broken down into different components. The focal point of our application is the heat-map. We used the existing TLC zones, filtered out all zones where the borough did not equal Manhattan, and added them as a layer to our Mapbox map. Mapbox supports GeoJSON data and we added the TLC zones as a GeoJSON object to our map. Each zone represented a polygon and to have the zones displayed on the map we added a fill colour to the polygons, a border line, and added hover effects. The mouseover hover effect also allowed us to gain information from each polygon which we would later use to display information about each zone.

To create the heatmap we used the existing layer with the taxi zones and changed the colour based on each zone's busyness prediction. Predictions were retrieved using Axios, a promise-based HTTP client for Javascript. We chose to use Axios over other HTTP request clients because of its automatic transformation of data into a JSON format (Fetch does not have that feature). We passed the data down our component tree using the useContext react hook. As long as all our components were wrapped in the context provider the data would be accessible through the site. This was pivotal because if the data changed that change would be reflected across all components. Additionally, since we needed access to our cafe and prediction data which were being fetched from the backend in multiple components, it made the code cleaner as the alternative would have been passing props through each component. For instance, our busyness data needed to be added to the map TLC layer in order to assign colours depending on those predictions, however, we also needed access to the busyness data in our Chart.js graph which was in its own component.

Once we had access to the busyness predictions we altered the TLC GeoJSON file that was being used to add layers to the map. For every zone we assigned a ranking based on its relative busyness prediction. The zone with the highest busyness prediction was ranked 1. Then, a Hue Saturation Light (HSL) colour was assigned based on a ranking where the HSL would get increasingly lighter as the ranking increased. Using the addLayer function with Mapbox allowed us to change the colour of each GeoJSON polygon based on the HSL value we added to each TLC zone in the GeoJSON. The rankings and values would be updated whenever a different busyness timeframe was selected.

In order to give the user more choice we designed the busyness slider so the user could see how busyness would

change over time in each zone. The slider would update the heat map depending on its position. We allowed the user to switch between the current day's busyness, the current week's busyness, and the average monthly busyness for the current year.

In order to further customise the heatmap we followed the same principles outlined above for our other metrics. Each TLC zone was ranked for their cafe density, crime density, property price and transit links. The purpose of this was to display heatmaps for each of these metrics. In order to allow the user to select multiple metrics, whenever another metric was selected, a combined ranking was created. For example, if busyness and cafe density was created, then each zone would combine the rankings of the selected metrics and a suitability ranking would be assigned to each zone after they were filtered by their combined ranking. Whichever had the lowest combined ranking would have the 1st ranking for suitability score. This was updated automatically in our GeoJSON file whenever a new metric was selected. This was possible due to React's useEffect hook which would run our heatmap creation code whenever a new ranking was selected. Selecting or removing a metric would update a state variable. Whenever the state variable was updated the heat-map would be recreated to reflect the new selected metrics.

We used the heatmap customization logic to also provide our users with top suggested zones. Based on what metrics a user valued when opening a cafe we could provide them the best zones based on our suitability ranking. This information would be displayed in our drawer which had further detailed information when clicking on a zone or cafe. The drawers had to be conditionally rendered depending on what our user interacted with. If a zone was clicked we had to show the zone info drawer. If the top zone button was clicked we showed the top zone drawer, and if a cafe was clicked we showed the cafe drawer. This allowed us to keep the user from being overwhelmed with information as only the relevant information was shown when appropriate.

A tutorial was included using the React-Joyride library. Joyride allowed us to highlight HTML elements on the page and add accompanying text. This removed any ambiguity from our site and allowed our users to immediately become familiar with all the tools to find their ideal location. We also chose to use the React JS library to display our busyness data in a line chart because of its seamless interaction with React, responsiveness, and customisability.

One benefit of using Mapbox was we could easily integrate their Isochrone API to adapt it into our takeaway radius. The Isochrone API lets a user select a mode of transportation and a time to see how far they can travel in any given direction. This is pivotal to our website as it allows our users to see potential customer bases.

C. UI DECISIONS

Our UI decisions followed a user-centric approach, based on our earlier surveys investigating and identifying our target users' needs, priorities and values. Figures 1 to 6 exhibit the iterative process this design underwent over the course of this project, with substantial differences between early and late-stage designs. Our customer lead completed an introductory UI course, its core principles of visibility, action feedback, grouping and constraint becoming the foundations of our choices [12].

The map is the primary feature of the design, with the remaining UI designed to complement it, without occluding it. For this reason we pushed most elements towards page borders to place emphasis on the map through central placement, isolation and contrast [13] with its bright-green colours that aren't present anywhere else on the page. We also picked green due to humans associating the colour with positivity, with greener map zones indicating more positive characteristics [14].

Our site's elements are highly visible, with each item accessible in 2 clicks or less. We balanced visibility with cluttering of the screen by confining advanced information to an info drawer, that can be tucked away when not in use, and refocus the user on the map. The content of these drawers was balanced based on the rule of thirds at higher desktop widths, asymmetrical balancing at medium widths, before resolving to scrolling at narrow widths [13]. Responsiveness was achieved down to 200px widths through extensive use of media queries, but we neglected to follow the mobile-first approach, a lesson-learned that will improve efficiency in future projects[15].

We also used symmetrical balancing for the map controls either side of the screen, who's unified style and proximity are an example of our use of grouping. Other examples of proximity-based grouping include the busyness slider adjacent to the timeframe selector, which have complimentary functionalities [12][13]. Our use of a slider is also an example of positive user constraint, preventing errors caused by open-ended inputs. Unified styling can be seen for all of the page's main buttons, which extends to action feedback to promote consistency for the user. The buttons are primarily icons instead of text, to simplify and abstract our app, also seen in the use of ratings in the top-zone function instead of number values. For confused users, the tutorial explains what each icon represents, and the currently selected heatmap is labelled in the map legend also.

D. BACK-END

Django was the selected backend framework for our project. In addition to our team's familiarity with Python and the reasons outlined in our literature review, there were a number of other reasons why we opted for Django. Chief among these were Django's views function (which is used to process web requests and return a web response), streamlined URL dispatcher, capacity for serialisation, and strong support for object-relational mapping, all of which made Django an ideal fit for our application. We will discuss each of these aspects in turn.

Django's URL dispatcher integrated with our urls.py file where we defined our API endpoints. These API endpoints

were mapped to relevant Python functions, serving up the appropriate data (e.g. prediction data, Yelp cafe data) as required. We had three API endpoints for prediction data (daily, weekly, and monthly), one for the cafe locations, and one for cafe reviews. These were each mapped to functions in the "views.py" file. Whenever one of these endpoints is called, the corresponding function in the "views.py" is executed.

Django's Object-Relational Mapper (ORM) facilitated painless database management. The use of the ORM eliminated the need for SQL code as we could create, update, migrate and delete our tables with Python code alone. We took advantage of this capability in both our models.py and views.py file. In models.py we defined and created the tables in our database through the use of classes. Inside each class we further defined their attributes and data types and mapped them to database columns. In our database, we had four classes defined with their respective attributes. The use of the Django database migrations enabled us to propagate these changes to our database. Similarly, in views.py, we made use of ORM by retrieving all the café items from the database to inform us on the count of these items. This was done to ensure we retrieved all the cafes from the Yelp API, as there was a limit of 50 cafés per API call.

When sending responses via the views function, we employed the use of serialisation when reading data from our database, so that these responses could be formatted as JSON responses. This was in keeping with the requirements stipulated by our front-end team.

E. PREDICTION GENERATION AND CACHING

As mentioned in the previous subsection, we employed the use of Django's view function in order to provide prediction and Yelp café data to the front-end. A year's worth of busyness predictions were generated and stored in our AWS PostgreSQL database, eliminating the need to unpickle the models to serve predictions.

To improve the performance of our backend, we implemented caching through Redis, chosen for its support for persistence (unlike Memcache) [16]. Both the prediction data and the cafe data were cached with suitable expiries to prevent the cache from filling up and ensuring we complied with Yelp terms and conditions (data is not allowed to be cached for longer than 24 hours according to Yelp terms and conditions [17]). The use of caching not only helped to improve the performance of our website, it also reduced the number of API calls we made, ensuring that we did not hit the limits placed on the Yelp API.

We handled the possibility of cache inaccessibility by implementing error handling to retrieve data from alternative sources like our database or via a Yelp API call. This can be seen in Figure 7. As there were expiries placed on the cached data and the data required for the 24 hour predictions would change each day, we also had to account for the fact that the data may not always be cached. To avoid this issue, we implemented automated caching through the use of Celery and Celery Beat. We had a number of Celery tasks which were automatically run each day at 2am (when website traffic was assumed to be low), ensuring that the cache was constantly populated with the required data. The addition of automatic caching through the use of Celery and Celery Beat ensures that data should always be served from our cache so long as our cache is accessible. Our overall backend architecture can be seen in Figure 8.

F. DEPLOYMENT OF THE WEB APPLICATION

In order to place our project online we had to deploy it. The tools we used were Docker and Apache HTTP Server. Docker is a containerisation tool that packages code and its dependencies (such as libraries, frameworks, and modules) together into a container. In this way, the project in the container is able to run in different environments, independent of the operating system or external environment, providing good portability to the software. The use of Docker easily enabled us to run our project on the server side. In addition, the Docker hub provided by Docker served as a good solution to the problem of transferring containers between the server and our local machines. After uploading the local packaged project image to the Docker hub, we could download and run it directly on the server.

We used an Apache HTTP server to handle incoming HTTP requests from management servers. In order to make our project accessible on the server we had to use a reverse proxy through Apache to account for any possible restrictions placed on public network HTTP connections. In reverse proxy, Apache will proxy the server to accept the HTTP request, forward it to the back-end server, and then return the response to the client. In this way, the request from the public network will not be transmitted directly to the server, thus bypassing the server's restrictions. In addition, this proxy is also a more secure way of dealing with requests. By acting as an intermediate layer between the server and the external network we can filter malicious network requests and hide the IP address of the back-end server. In our project, we also configured SSL/TSL certificates in the Apache HTTP Server, which allows the website to be requested over HTTPS, further improving the website's security.

IV. DATA ANALYTICS AND VISUALISATION

As stipulated in the project requirements, the foundational dataset used to generate the busyness predictions must be a Yellow Taxi dataset from the TLC. In keeping with this requirement, our data team agreed that we should use the most recent dataset available (in order to best capture the latest busyness trends) whilst also ensuring that there was at least a year's worth of data to account for temporal variations in busyness levels. In order to meet these requirements, we opted to use 15 taxi datasets in total, one for each month in 2022, as well as the first three months in 2023 (this was the most recent data available) [18]. We joined these datasets together resulting in 49,040,585 rows (with one row representing one trip) and 19 columns.

The TLC divides up New York City into 263 different taxi zones, 69 of which are in Manhattan. 3 of these 69 Manhattan TLC zones are islands (Governor's Island, Ellis Island, and Liberty Island) and are thus dropped from our analysis, resulting in 66 different TLC zones in Manhattan. Please see Figure 9. In accordance with this zonal division, the taxi datasets indicate which pickup and dropoff zones each trip occurred in. However, as we are interested in busyness, which we take to be represented by the number of transport users in a given TLC zone in a given hour, distinguishing between pickup and dropoff locations is not relevant. Instead, we wish to see the number of taxi passengers in a given zone, in a given hour. In order to extract this information for both pickup and dropoff location, we generated two dataframes from our merged taxi dataset, one for taxi pickups and one for taxi dropoffs (ensuring that only one record was kept for rows which had the same pickup and dropoff zone). The zones of each row in both of these datasets were checked and those which were outside of Manhattan were dropped. These datasets were then rejoined resulting in 85,042,668 rows. Duplicate rows, irrelevant columns, and corrupted data (e.g. negative passenger count) were dropped. There was a relatively small percentage of missing passenger numbers (3.14%) and these were imputed using the mean.

As we wished to examine whether the busyness trends in taxi data were observed in other modes of transport, we performed a similar set of steps on Citi bike and Subway data. These modes of transportation were selected both for their availability and, as mentioned in our literature review, our belief that the busyness trends in these modes of transport would differ from that of taxi data, as suggested by Gerte et al (2019) [14].

Data relating to Citi bike, which is New York's official bike sharing scheme [19], was taken from Citi bike's website. As with the taxi data, there were fifteen datasets in total, and there was a distinction made between the start and end location for each trip. In order to enable a comparison between the bike and taxi trips, the latitude and longitude for each station was mapped to the corresponding Manhattan TLC zone, and only rows pertaining to Manhattan were kept. Duplicate rows and corrupted data were again dropped. After this cleaning and preparation there were 50,182,690 rows.

Subway data was taken from the Metropolitan Transportation Authority (MTA) website [20] and provided hourly ridership figures for each station from February 2022. As there was no January data available for subway ridership, January data for bike and taxi usage will be excluded from the comparison. This meant that our data went from the 1st of February 2022 to the 1st of April 2023. As with the Citi bike data, it was necessary to map subway stations to corresponding TLC zones. After cleaning was completed, there were 5,015,198 rows. The considerable difference in the number of rows compared to taxi and bike data is explained by the fact that these rows represent hourly ridership in each TLC zone and there was no distinction (and thus doubling) of start and end locations.

In order to compare bike and taxi data with that of subway data, it was necessary to round timestamps to the nearest hour and group by TLC zone. This meant that both taxi and bike data now had the total number of users grouped by both hour and TLC zone, enabling comparison with subway data. All three modes of transport were consolidated into one dataset which was formatted as seen in Figure 10. The total transport users in a given TLC zone in a given hour was also calculated.

After data cleaning and consolidation, we compared spatial and temporal busyness patterns across all three modes of transport. Figure 11 highlights the considerable difference in the number of users of the subway network when compared to the other modes of transport. In order to more easily discern trends, when plotting all three modes of transport on the one graph, we employ a secondary axis for subway data. Graph "smoothness" can be explained by the fact that data has been grouped by hour.

Examining the hourly busyness trends in Figure 12 we can see that subway and bike usage display similar trends, with clear peaks in the morning and evening, possibly explained by commuters going to work. By contrast, taxi usage continuously rises throughout the day, with a peak in the evening in a similar manner to that of bike and subway. Daily trends (Figure 13) show midweek peaks for all modes, dipping on weekends with varying extents. In Figure 14, which examines monthly usage, we can see that taxi and subway usage broadly follows a similar pattern, while bike usage peaks during the summer months. Intuitively this makes sense as bike usage is likely more greatly influenced by weather conditions than that of taxi or subway usage. In summary, all three graphs have demonstrated differing temporal usage trends between the three modes of transport, bolstering the case for using all three when generating busyness predictions.

Turning now towards the spatial patterns, Figures 15, 16, 17, show hourly usage patterns for each TLC zone, for taxi, subway, and bike. The legend on each graph shows the top 5 busiest zones for each mode of transportation. As can be seen, the top 5 zones differ substantially for all three modes of transportation. The TLC zones themselves also demonstrate different busyness patterns. Similar patterns were observed for both daily and monthly usage also. As suspected, this suggests that there is spatial variation in terms of busyness in addition to the temporal variation. Once again, this supports the case for using all three modes of transportation in order to generate busyness predictions. Owing to the idiosyncratic nature of these busyness patterns we will opt to generate separate models for each TLC zone, rather than simply opting for one unified model.

Finally, it is worth noting that the data used for both crime rates and property value were extracted from NYC open data [21][22]. The relevant rankings were calculated in the manner discussed in the methodology section.

A. MODEL GENERATION

With clear spatial-temporal differences in peak zones and hours for all three transportation modes established, we now

focus on busyness prediction. Busyness is defined as users within a TLC zone per hour. Predictions are based on subway, Citi bike, and taxi data from February 2022 to March 2023. In line with the approach adopted by Dunne and McArdle (2021), we ordinally encoded time of day, day of the week, month of the year, and also week of the year (to add additional temporal granularity) [1]. As stated in the literature review, we will be generating "long term" predictions and thus have opted to avoid including features such as weather which would drastically limit how far into the future we can offer busyness predictions. Having performed these steps, our data appears as seen in Figure 18, with transport total acting as our target feature.

We will explore six different models initially, Random Forests, Linear Regression, Decision Trees, Gradient Boosting, K-Nearest Neighbours, and XGBoost. Model evaluation will encompass Mean Absolute Percentage Error (MAPE), Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), K-Fold Mean Absolute Error and R Squared. These metrics align with established literature conventions [1]. K-Fold MAE refers to a process in which we divide the dataset into k subsets (in our case 5) and train and evaluate the model k times, using a different subset for validation each time. K-fold MAE was included as it provides a more robust and unbiased estimate of a model's performance by averaging the Mean Absolute Errors calculated during each validation iteration of k-fold cross-validation. Based on the literature reviewed [1]3], we expect that either XGBoost or Random Forests will be our best performing model. The results we obtained are discussed in the next section.

V. EVALUATION AND RESULTS

TABLE 1. Performance Comparison of Machine Learning Models

Model	RF	LR	DT	GB	KNN	XGB
MAPE	12.23	247.75	9.12	37.84	22.56	15.81
MAE	78.88	616.332	63.35	161.44	136.86	78.67
RMSE	166.34	836.60	179.81	256.21	229.25	144.13
K-Fold MAE	16.96	80.04	19.92	18.89	25.92	15.94
R^2	0.96	0.27	0.95	0.90	0.92	0.96

* RF: Random Forest, LR: Linear Regression, DT: Decision Trees, GB: Gradient Boosting, KNN: K-Nearest Neighbours, XGB: XGBoost

TABLE 2. Performance Comparison of Machine Learning Models (With comparable maximum depth and number of estimators)

	Maching Learning Models		
	RF-MD:20,E:10	**XGB-MD:6,E:100**	**RF-MD:6,E:100**
MAPE	17.83	15.81	27.00
MAE	82.15	78.67	170.27
RMSE	173.60	144.13	301.47
K-Fold MAE	16.98	15.94	27.99
R^2	0.96	0.96	0.87

* RF: Random Forest, XGB: XGBoost, MD: Maximum Depth, E: Number of Estimators

As discussed in the previous section, we initially used six different ML algorithms, to generate hourly predictions, with a separate model for each TLC zone. We opted to use a 70/30 test train split each time. The initial results can be seen in Table 1. These results represent the average metrics for 66 different models using each ML algorithm. As can be seen from these results, Random Forests, Decision Trees, and XGBoost were found to be the best performing models. These models had the lowest error rates, as measured by MAPE, MAE, K-Fold MAE, and RMSE and also displayed the highest accuracy, as measured by R Squared.

These top three models were subjected to additional testing, introducing a US public holiday dummy variable and a weekend dummy variable separately, but these additions didn't noticeably influence model performance, so we opted to exclude these variables. The two most performant models, Random Forests and XGBoost were examined for signs of overfitting by adjusting their depth and estimator counts. Although Random Forests and XGBoost differ in their approaches (Random Forests builds multiple decision trees and averages their predictions, while XGBoost sequentially adds trees having corrected errors made by previous ones [23]), we nevertheless attempted to perform a fair comparison and found that XGBoost significantly outperforms Random Forests while guarding against overfitting, as can be seen in Table 2. Employing the same max depth and estimator counts, XGBoost consistently outperformed Random Forests across all metrics, exhibiting lower errors and a higher R Squared value. This trend persisted even when juxtaposed with the Random Forests model that achieved a balance between accuracy and generalisation (attained through evaluating various Random Forests models with varying max depths and estimator counts). As a result of these findings, we opted to adopt XGBoost as our ML model. We generated 66 different models, one for each TLC zone, and pickled these models. Additionally, we generated a year's worth of prediction data based on these models (as previously mentioned).

A. FRONT-END TESTING

We chose to conduct Lighthouse testing due to its popularity in a number of research articles, and to highlight overlooked issues. Our final testing returned consistent 100/100 in the fields of accessibility, SEO and best practices, with an average of 80/100 for performance. The app also meets Lighthouse's criteria as a progressive web application, following the integration of a service worker and manifest file to our React architecture [24][25][26].

B. BACKEND TESTING

We used Selenium and JMeter as our functional and load testing tools. Selenium's test scripts were used to simulate user actions (such as clicks), allowing us to check page functionality and interaction integrity. JMeter carried out load stress tests, enabling us to evaluate the servers responses and to gauge performance under varying stress levels. As can be seen in Figures 19, 20, 21, the server responds well at low load stresses, but as the load stresses increase, the server is unable to handle a large number of requests, most of the requests go unanswered, and the latency increases

significantly. Based on these performance results, our web page is suitable for a small number of concurrent users. Since our project is mainly for enterprise, the current load capacity is acceptable, if we need to expand the user group or develop more features, we need to add extra servers to increase the load capacity.

Finally, it is worth mentioning that we implemented unit tests for all our backend functionality. Unit testing for all Django-related functionality was done through the use of the built-in unit test module, while testing for Celery was performed via Pytest. Django's unit test module uses a dedicated test database to ensure that there was no disruption to our production database. This testing helped detect minor glitches and reinforced the resilience of our backend.

VI. CONCLUSION AND FUTURE WORK

Overall we were pleased with the final outcome of our project, feeling that we met, if not exceeded the project requirements. We will now discuss our key findings and discuss possible improvements.

With regards to ML and busyness predictions, there were two main takeaways:

1) As we expected, there were different spatial-temporal busyness trends across the three modes of transportation we examined (taxi, Citi bike, and subway)
2) XGBoost was found to be our best performing model, demonstrating the highest accuracy and lowest error rates, while remaining robust to overfitting.

Regarding our UI design decisions, we conducted an After Scenario Questionnaire (ASQ) and Computer Systems Usability Questionnaire (CSUQ) using Google Forms. We found that users were satisfied with the performance of the application and were able to explain and perform their assigned user tasks. However, we also found some minor bugs, such as an issue with resetting the tutorial. Through this process we also identified features that users would like to see added, such as additional map markers (e.g. parking, universities, and green spaces), additional heatmap information (e.g. population demographics), as well as expansion into other businesses such as bars and restaurants.

If we were to redo the project, we would certainly bear these findings in mind, adjusting our application to reflect the feedback we received. Finally, we would opt to use test driven development to improve the maintainability and quality of our code.

REFERENCES

[1] L.Dunne and McArdle.G, "A novel post prediction segmentation technique for urban bus travel time estimation."Proceedings Tenth International Workshop on Urban Computing. 2021

[2] F. Terroso-Saenz, A. Muñoz, and J. M. Cecilia, "QUADRIVEN: A framework for qualitative taxi demand prediction based on Time-Variant Online Social Network data analysis," Sensors, vol. 19, no. 22, p. 4882, Nov. 2019, doi: 10.3390/s19224882.

[3] M. Poongodi et al., "New York City taxi trip duration prediction using MLP and XGBoost," International Journal of Systems Assurance Engineering and Management, vol. 13, no. S1, pp. 16–27, Jul. 2021, doi: 10.1007/s13198-021-01130-x.

[4] R. Gerte, K. C. Konduri, N. Ravishanker, A. Mondal, and N. Eluru, "Understanding the Relationships between Demand for Shared Ride Modes: Case Study using Open Data from New York City," Transportation Research Record, vol. 2673, no. 12, pp. 30–39, Jun. 2019, doi: 10.1177/0361198119849584.

[5] P. Toman, J. Zhang, N. Ravishanker, and K. C. Konduri, "Spatiotemporal Analysis of Ridesourcing and Taxi Demand by Taxi zones in New York City," arXiv (Cornell University), Aug. 2020, doi: 10.48550/arxiv.2008.00568.

[6] B. K. Dey, S. D. Tirtha, and N. Eluru, "Transport networking Companies demand and flow estimation in New York City," Transportation Research Record, vol. 2675, no. 9, pp. 139–153, Mar. 2021, doi: 10.1177/03611981211000752.

[7] D. Johansson, "Building maintainable web applications using React," MA Thesis, Linköping University.

[8] J. Plekhanova, "Evaluating web development frameworks: Django, Ruby on Rails and CakePHP," Temple University, Sep. 2009. Accessed: Aug. 16, 2023. [Online]. Available: https://ibit.temple.edu/wp-content/uploads/2011/03/IBITWebframeworks.pdf

[9] J. P. Dock, W. Song, and J. Lu, "Evaluation of dine-in restaurant location and competitiveness: Applications of gravity modeling in Jefferson County, Kentucky," Applied Geography, vol. 60, pp. 204–209, Jun. 2015, doi: 10.1016/j.apgeog.2014.11.008.

[10]] "Atlassian Customers," Atlassian. Available: https://www.atlassian.com/customers

[11] Educative, "What is the difference between virtual and real DOM (React)?," Educative: Interactive Courses for Software Developers. https://www.educative.io/answers/what-is-the-difference-between-virtual-and-real-dom-react

[12] "Introduction to ui design," Coursera, https://www.coursera.org/learn/ui-design (accessed June. 24, 2023).

[13] J. George and A. Walker, The Principles of Beautiful Web Design. Fitzroy, VIC, Australia: SitePoint Pty. Ltd., 2020.

[14] S. Bouhassoun, M. Naveau, N. Delcroix, and N. Poirel, "Approach in green, avoid in red? examining interindividual variabilities and personal color preferences through continuous measures of specific meaning associations," Psychological Research, vol. 87, no. 4, pp. 1232–1242, 2022. doi:10.1007/s00426-022-01732-5

[15] S. Gupta, "UX Design: What is mobile-first design and why should you ... - springboard," Springboard.com, https://www.springboard.com/blog/design/what-is-mobile-first-design/ (accessed Aug. 17, 2023).

[16] C. Vicente, "Redis vs Memcached: which one to pick?," Blog | Imaginary Cloud, Mar. 2023, [Online]. Available: https://www.imaginarycloud.com/blog/redis-vs-memcached/

[17] "Frequently asked questions," Yelp Developers. https://docs.developer.yelp.com/docs/fusion-faq: :text=You

[18] "TLC Trip Record Data - TLC." https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page

[19] "Citi Bike System Data | Citi Bike NYC," Citi Bike NYC. https://citibikenyc.com/system-data

[20] "MTA Subway hourly ridership: Beginning February 2022 | State of New York," Aug. 2023. https://data.ny.gov/Transportation/MTA-Subway-Hourly-Ridership-Beginning-February-202/wujg-7c2s

[21] City of New York, "Crime Map," NYC Open Data, https://data.cityofnewyork.us/Public-Safety/Crime-Map-/5jvd-shfj. [Accessed July 14, 2023]

[22] City of New York, "Revised Notice of Property Value (RNOPV)," NYC Open Data. [Online]. Available: https://data.cityofnewyork.us/City-Government/Revised-Notice-of-Property -Value-RNOPV-/8vgb-zm6e. [Accessed October 26, 2022]

[23] "XGBoost versus Random Forest | Qwak's Blog." https://www.qwak.com/post/xgboost-versus-random-forest: :text=RF

[24] M. Siahaan and V. O. Vianto, "Comparative Analysis Study of Front-End JavaScript Frameworks Performance Using Lighthouse Tool," Manajemen, Teknologi Informatika dan Komunikasi (Mantik), vol. 6, no. 3, Nov. 2022. doi:https://doi.org/10.35335/mantik.v6i3.3131

[25] A. Agarwal, S. Satish Rao, and B. M. Mahendra3, "Gamification Based Learning," International Journal of Research in Engineering, Science and Management , vol. 3, no. 5, May 2020.

[26] W. K. Dharma and D. Anggraini, "Development of Modern Web Application Frontend Structures Using Micro Frontends," International Research Journal of Advanced Engineering and Science, vol. 7, no. 1, pp. 149–155, 2022.

VII. APPENDIX

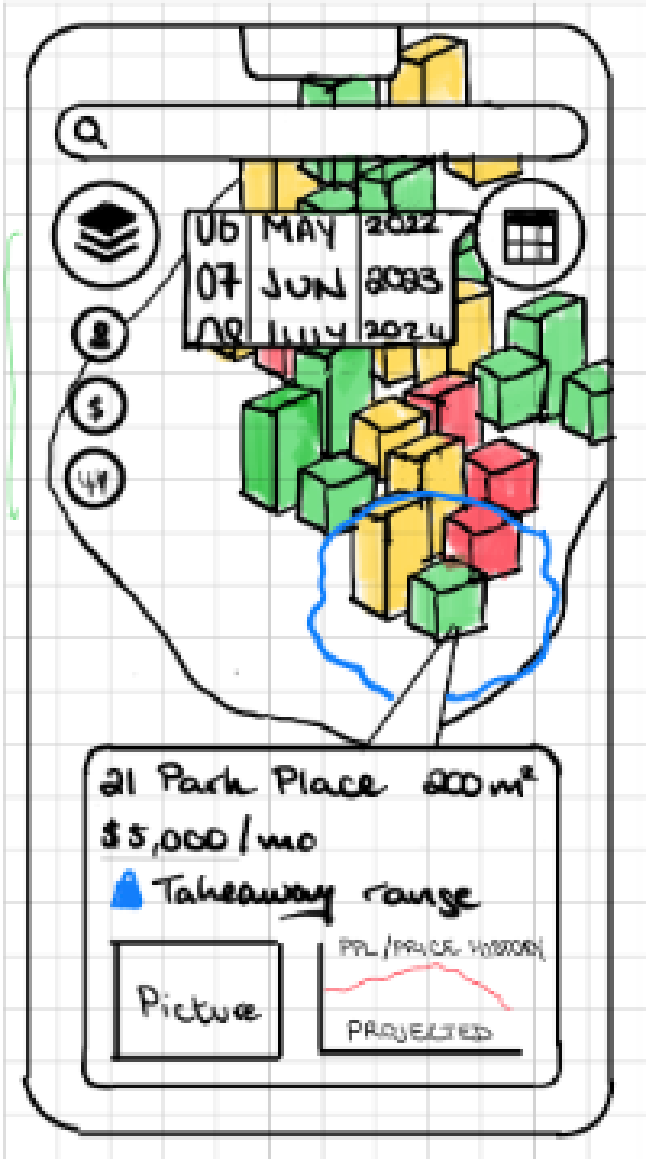

FIGURE 1. Initial Concept Sketch

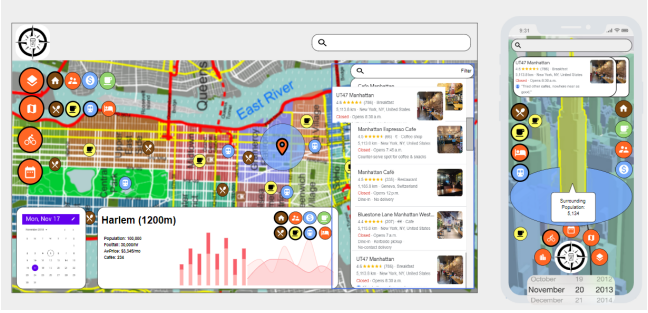

FIGURE 2. Initial Concept Mock-up

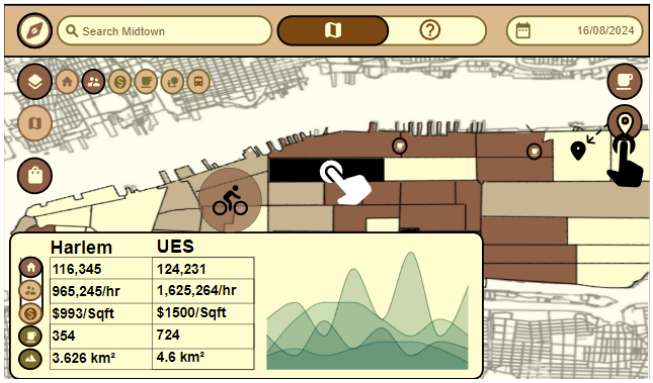

FIGURE 3. Mock-up following feature selection decisions

FIGURE 4. Non-React Prototype

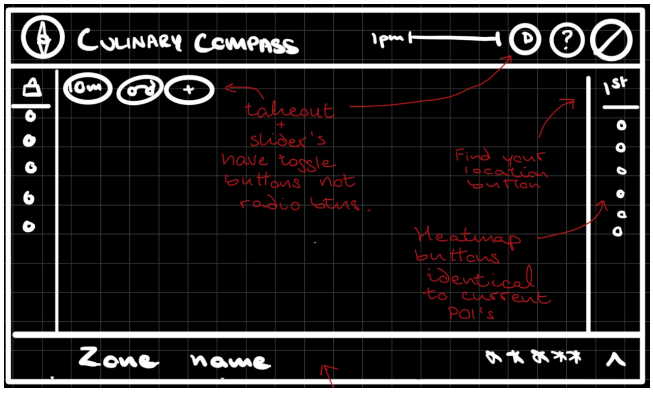

FIGURE 5. Sketch for re-worked positioning

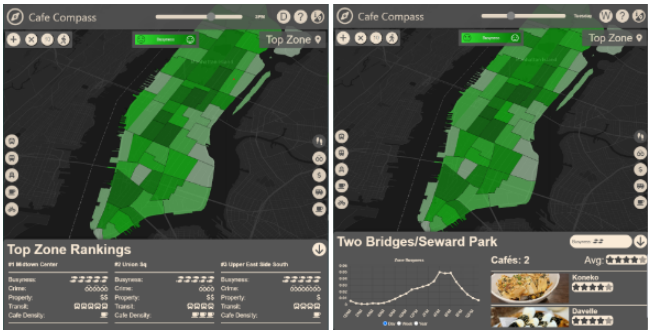

FIGURE 6. Final Design

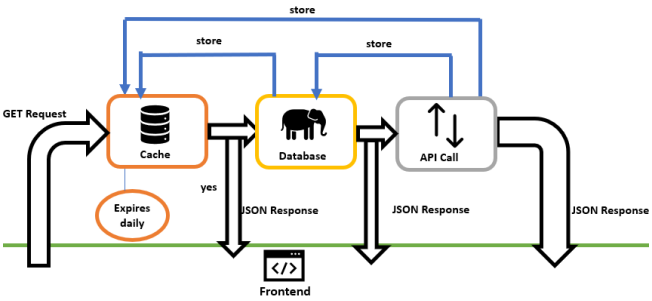

FIGURE 7. Caching and alternative data sources

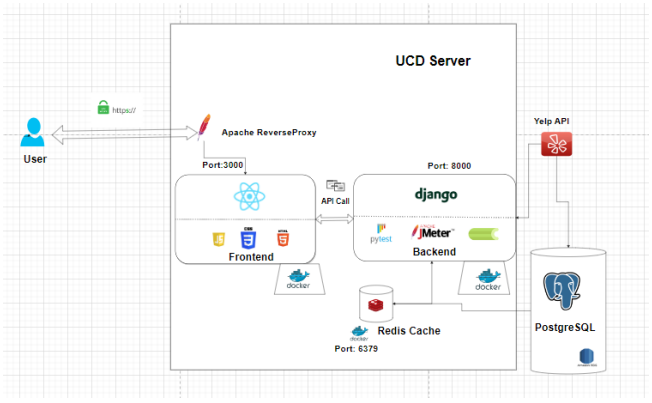

FIGURE 8. Overall project architecture

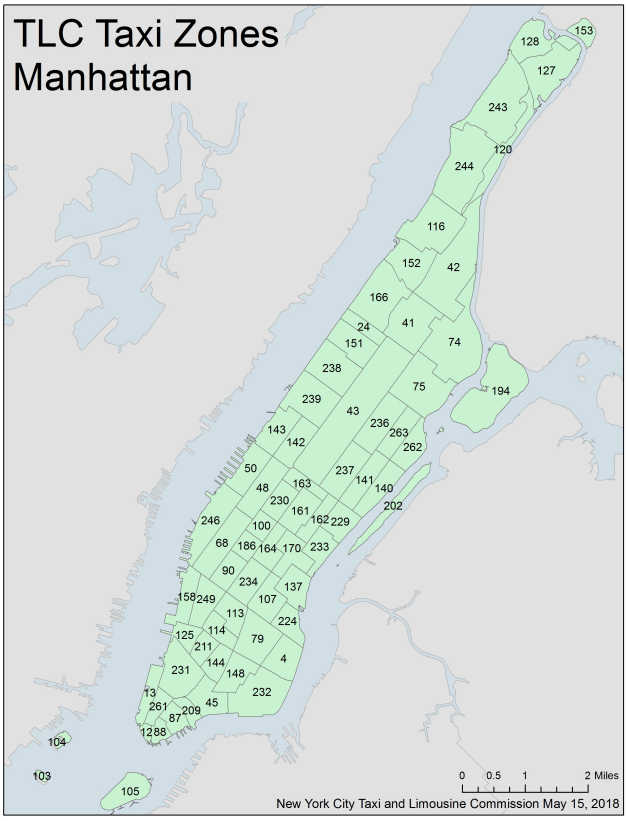

FIGURE 9. Manhattan TLC Taxi Zones

index	LocationID	datetime	bike_count	passenger_count	ridership	transport_total
381195	161	2023-02-14T1...	249	726	27660	28635
378982	161	2022-11-09T17...	298	664	26944	27906
379603	161	2022-12-06T1...	63	469	27351	27883
381036	161	2023-02-07T1...	222	737	26881	27840
380719	161	2023-01-24T1...	190	725	26843	27758
381673	161	2023-03-07T1...	238	847	26287	27372
380403	161	2023-01-10T17...	223	624	26394	27241

FIGURE 10. Consolidated transport data

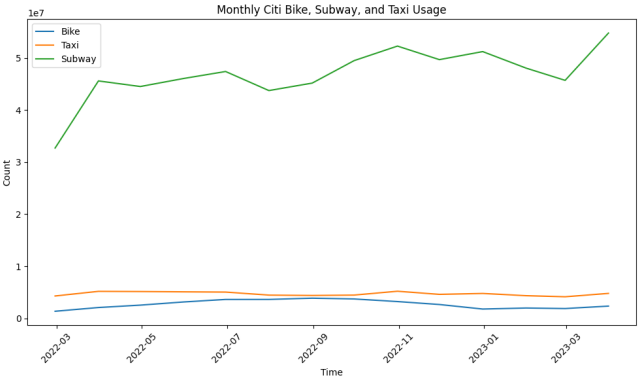

FIGURE 11. Monthly usage for all three modes of transportation with the same scale

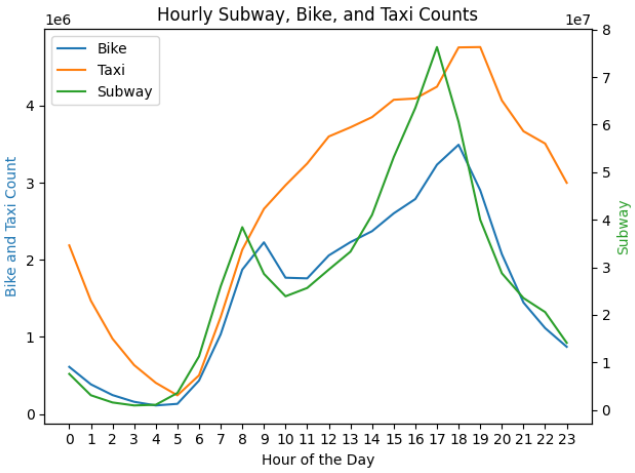

FIGURE 12. Hourly subway, bike, and taxi usage

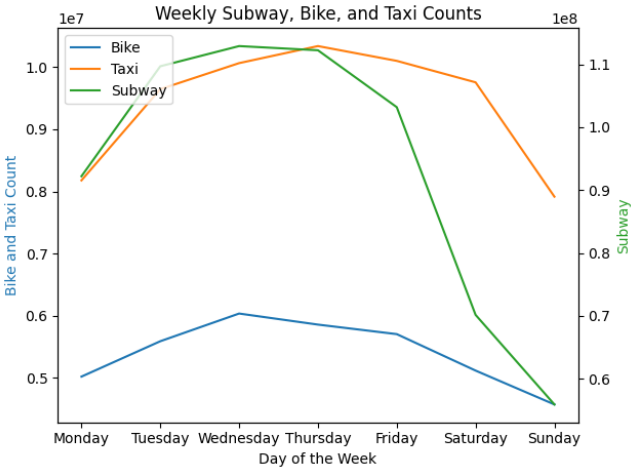

FIGURE 13. Daily subway, bike, and taxi usage

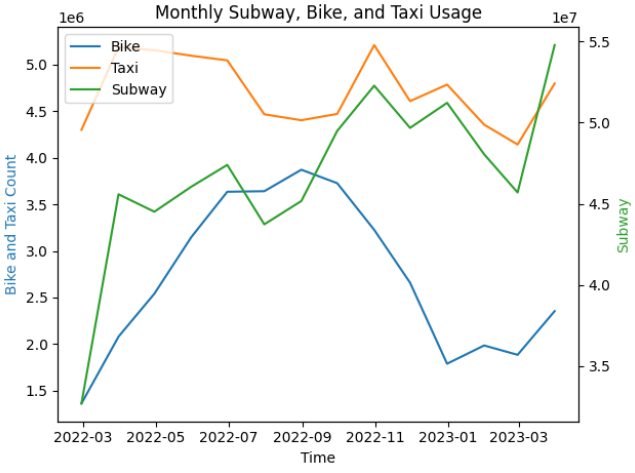

FIGURE 14. Monthly subway, bike, and taxi usage

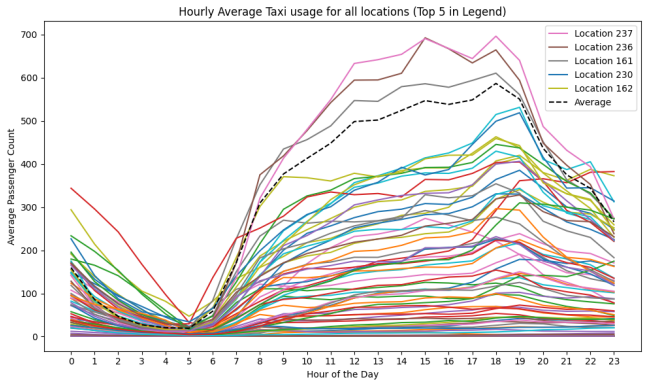

FIGURE 15. Hourly taxi usage

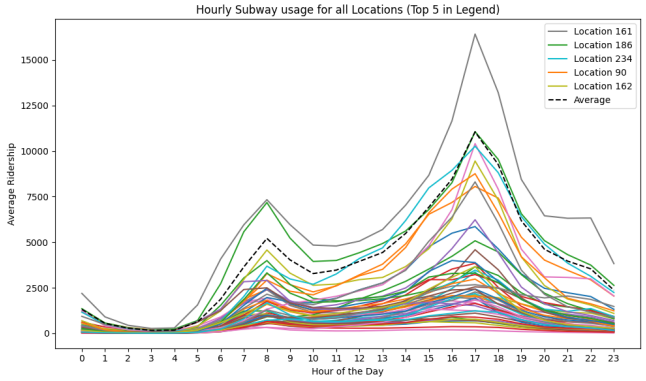

FIGURE 16. Hourly subway usage

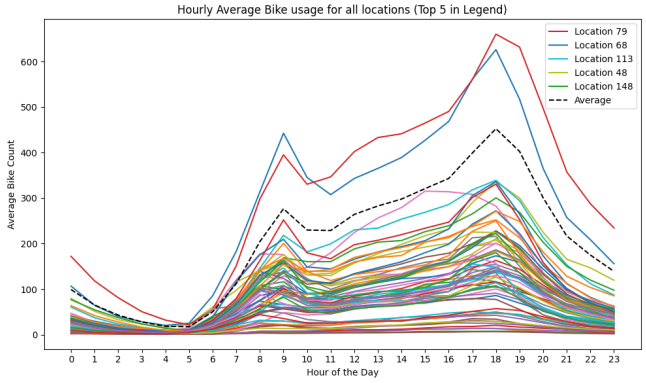

FIGURE 17. Hourly bike usage

	index	LocationID	transport_total	hour	day	month	week_of_year
0	725	4	31	0	1	2	5
1	726	4	13	1	1	2	5
2	727	4	9	2	1	2	5
3	728	4	3	3	1	2	5
4	729	4	6	4	1	2	5
5	730	4	1	5	1	2	5
6	731	4	8	6	1	2	5

FIGURE 18. Data inputted to models

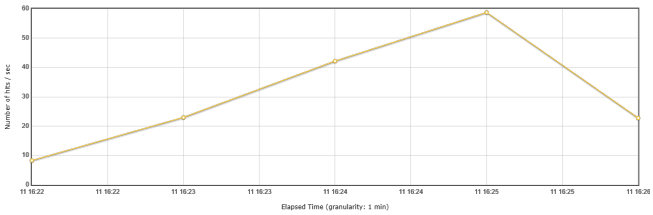

FIGURE 19. Hits per second

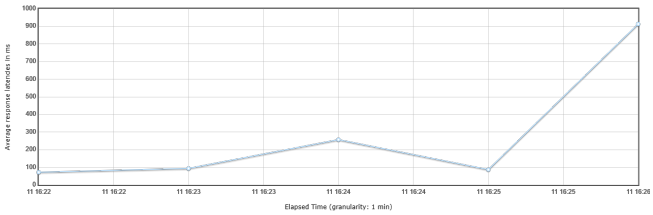

FIGURE 20. Latency over time

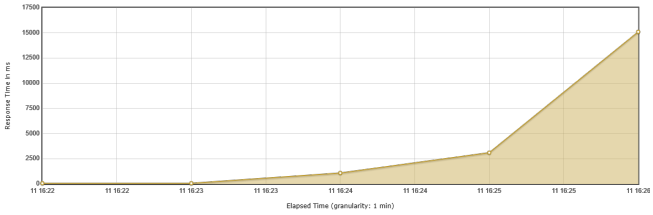

FIGURE 21. Response time over time