Alain Tamazian, Alexander Chavez

DSCI551: Project Final Report

Project Title

Park It: Safe and Simple

Topic

As the name suggests, our project topic is safety-focused parking assistant; drivers can use the web-app to find the best (and safest) parking option for them. They can sort their preferences based on proximity, duration, and a custom safety metric. There are also various filtering options regarding availability, parking type, valet vs not, whether a parking area is restricted to only guests/customers, as well as a safety threshold. The safety metric will be calculated with considerations such as security, whether it is on a main/active road, the parking type, history of crime near it, and current potentially dangerous incidents in its immediate vicinity. The parking occupancy data will be based on user input, where people will be able to indicate whether there was or wasn't space available at a particular parking location – like how the navigation app Waze updates its traffic conditions. Information about when the latest availability update was made will also be reported to the user so that they can make the most informed decision.

Motivation

Our motivation for this project was two-fold. The first was simple utility, to make the parking process simpler for drivers. Parking can often be one of the more troublesome aspects for drivers – such as myself. This includes finding an available and appropriate spot (near your destination) – as well as understanding if the various policies and restrictions apply to you at that specific date and time. More significantly, there is also the issue of safety. The FBI has found that approximately one third of assaults and murders (in recent years) occurred in parking facilities. According to the FBI's Crime Data Explorer (database), in the last 5 years there were 157,092 violent crimes in parking lots/garages (FBI). Unfortunately, these statistics don't even account for the risks of walking to and from street parking – which is another common location for crimes of opportunity. From research, as well as personal experiences, I know that this is a significant consideration when parking – especially for women and after sundown. Therefore, our hope is that our "Park It: Safe and Simple" app will help drivers better find an available parking spot, which is convenient and most importantly safe.

Architecture and Components of the App

Web-Scraping, Preprocessing, and Database Creation

A significant portion of our project involved a lot of background work that isn't directly part of the backend or frontend code – such as the collection, preprocessing, and structuring of the data for our Realtime Firebase database. Due to the time constraint, we decided to limit the scope of our parking app to two zip codes – 90028 and 90038 (Appendix A). Unfortunately, there was no existing dataset to provide all of the parking information we needed. Therefore, we identified all the potential parking locations as well as collected their standard details – like price, time, type, and address – by web-scraping Parkopedia. We chose this website since we found it to be the most comprehensive source for such data. Since "parkopedia.com" is dynamic (e.g., using JavaScript for its rendering), we had to use a combination of the "Selenium" and "BeautifulSoup" Python libraries to collect and parse the required information. After a lengthy process of web-scraping, we created a dictionary with the basic information for each of the 335 locations.

Finally, we could start calculating our safety scores, using a simple subtractive formula that is initialized with a score of 100. Although parking facilities may (often) be safer for cars than parking on the street, that is surprisingly not the case for the risk to the actual drivers (FBI). Based on our research (and available crime data), the order of safety for each parking location – while all else equal – is street parking, parking lot, and garage. Garages are often considered least safe due to their design, which creates an enclosed, isolated place with easy access, poor lighting, and security. Therefore, we subtract 5 and 10 from our safety score if their parking types are "Lot" or "Garage", respectively. However, we also considered the security in those facilities. Specifically, if the garage or lot includes human employees (such as attendants, valets, and security), we add 5 to the score; that acts as a deterrent for crime. Then, if the parking location is not on an active/main street, another 10 is subtracted. It is generally known that crimes of opportunity are much less likely when there is a lot of foot and car traffic – such as on a popular street like Vine, rather than a smaller, remote street like Gower. This is especially true for street parking.

Most significant to our custom score is the calculation of the crime index for each parking location. Appendix B shows a screenshot of an LAPD hosted dataset, cataloguing all crime that occurred in LA from the beginning of 2020 to the present day – with 458,547 instances (LAPD OpenData). After deletion of all instances that had null values for relevant attributes or that did not occur in 90028, 90038, or any of their neighboring zip codes, the dataset became 32,465 rows. Using the Google Maps Geocoding API (through the "googlemaps" library"), we utilized forward geocoding to find the latitude and longitude of every one of our parking locations; we tested various other geocoding API such as "OpenCage" and "positionstack" as well but found "Google Maps" be most accurate for this. Since the crime dataset included coordinate attributes, we were able to use Python's GeoPy library – specifically "geopy.distance.geodesic" – to measure the pairwise distance between each of the 335 parking spots and 32,465 crime locations. This was a very lengthy and computationally expensive process.

For each parking location in our complete dataset, using the acquired distances, we counted the number of crimes that occurred within a quarter-mile radius (which is approximately a 5-minute walk). When calculating the crime index for the parking locations, we used the "Crm Cd Desc", "Premis Desc", and "Weapon Desc" attributes from the LAPD dataset to add weights to each incident – based on its relevance to parking safety (Appendix B). The three attributes give information for crime description, premises description, and weapon description. To start, all crimes in the radius are each worth 1 point that will be added together to make up the crime index. However, if the crime was related to cars or occurred in a parking location, we multiplied the point by 2; such as, if the "Crm Cd Desc" is "VEHICLE STOLEN" or the "Premis Desc" is "GARAGE" (Appendix B). Similarly, using "Crm Cd Desc" and "Weapon Desc", we identified violent crimes (like assault, rape, and murder) and applied a 3x multiplier to the corresponding point (Appendix B). So, with our specific calculations, our crime index weighs serious crimes that would affect our app users' parking safety much higher than relatively minor incidents like vandalism and petty theft. Most of the resulting indices are in the thousands; so, we mathematically mapped all of them to their corresponding number in a range of 0 and 40. We finalize the base safety score by subtracting the mapped crime index from it. The final component of our safety calculations is not included as part of our database safety score and will be discussed later as part of the backend implementation. Although the specific numbers in our scoring formula are a little arbitrary and could benefit from the insights of a math and domain expert, the reasoning behind it is rather intuitive and well-researched.

Once the base safety score was finalized, creating the Firebase database was very simple. After computing the parking geocoding data and safety, our finalized dictionary includes information about address, coordinates, parking type, number of spots, valet service, parking restrictions, flat fees, rates, maximum price, maximum stay allowed, and the safety score for all 335 locations. We also have an "availability" item, which simply has a default value of "True" for all. Using "json.dumps", the dictionary was converted to a JSON string. The "requests" library's "put" method took the address for our (empty) Realtime Firebase along with the JSON string as arguments and created our completed cloud database. You can see several screenshots of our Firebase Database and its structure in Appendix C. If you want to further inspect our Realtime Database, you can visit "https://dsci551-project-parking-app-default-rtdb.firebaseio.com/".

Backend Functions (and Dataflow)

Our app has three main functionalities that needed to be developed on the backend (the code for which can be accessed through the Google Drive link in Appendix D). The first functionality addresses the standard situation of when a driver is looking for the most suitable parking spot for them (Appendix E Image 2). More specifically, this is accomplished in the backend with the "parking_list" function, which is found in the "functionality3.py" script (Appendix D). This function takes the user's (app) inputs for street number, street name, zip code, city, and state as required parameters. The function also offers a series of optional inputs for the various filtering and sorting conditions that they can set during their parking location search. The first one is the parking type, which can be set to one of four strings: "Lot", "Garage", "Street parking", and "Not street parking". Is the user want to restrict the outputs to only places with confirmed availability, they can set "availability_cond" to "True". Similarly, they can filter out any locations that have parking restrictions by setting the "no_restrictions_cond" input to "True". Whether a valet service is offered or not is another possible criterion. The user may also choose a threshold for the safety score (between 0 and 100), so that locations with a score below their input are not displayed. The parameter value for the sorting condition can be "distance", "duration", and "safety"; if nothing is chosen, the sorting will be done based on the safety score. Also, there is an optional parameter to determine the max number of matching locations to display; its default is 5.

After these arguments are passed into "parking_list", our Firebase database is queried with a get request (Appendix D). However, one of the limitations of Firebase is that it does not take multiple "orderBy" conditions when querying. So, all but one of the filtering conditions need to be implemented locally. To maximize computational efficiency, I identified and ranked the filtering conditions based on how many parking location items each returns. For example, if the user includes two filters (e.g. valet is offered and there are no restrictions) it would be more advantageous to query our cloud database with the valet filter; valet_cond == True returns only 24 items while no_restrictions_cond == True returns 311. With that logic (and aforementioned ranking), I wrote an if/elif conditions block for executing the Firebase "get" request, which minimizes local computation for the filtering. If there are no filtering conditions, then the entire database is returned. After JSON string is acquired and converted to a dictionary (with the "json" library"), we loop through each element and delete it, if it doesn't meet any of the filtering conditions. Also, the parking location being currently open is a crucial implicit filter condition, which must also be considered. The current time is acquired in the backend using the "datetime" library's "datetime.now()". After the appropriate deletions are concluded, sorting is conducted. By default, all of the remaining parking locations are sorted in descending order of their safety score and then sliced based on the max output limit (e.g. 5). Then, using the Google Maps Distance Matrix API, the distance and duration between the user address and each of the (5) parking locations is calculated. If the sorting condition is “duration” or “distance” instead, these pair-wise Google Maps queries would need to be conducted for all post-filter locations. This is a much larger number of API requests, which slows the app’s loading time a little. At the end, Google Maps Geocoding API is used to get the coordinates of the user, which will be used for the front-end UI map – explained later. Finally, the “parking_list” function returns these coordinates and a filtered/sorted “pandas” DataFrame with each matching locations’ “ID", "Address", "Parking Type", "Distance", "Duration", "Safety Score", "Lat", "Long".

If the user wants more specifics for a particular location, they can search for it (in a space we provide below the DataFrame) by inputting its “ID”, which is column one of the DataFrame (Appendix E Image 7). This action prompts the backend execution of the “location_description” function. This is our app’s second functionality which can be executed by either inputting a location’s specific ID or entire address (Appendix D). Generally, this functionality is for situations such as when a driver has already parked or is simply looking for general information about a specific parking location. The user would then input their address in the search fields and the detailed information for the corresponding parking location is outputted (Appendix E Image 2 and 3). Other than the address’s street number, street name, zip code, and city, the user can also specify the parking type. Including the parking type with the address is optional. However, it can help ensure that an approximate input address still yields the results for the user’s desired parking location. Like before, we first need to connect to our Firebase to get the location data and find which one corresponds with the user’s search criteria. If the parameter for parking type is left blank, we cut the number of queried items from Firebase by (approximately) one half – using zip code as the “orderBy” condition. However, if the user specifies that the parking type is a “Lot” or “Garage”, the function uses the “type” key as the “orderBy” condition since it only returns 37 and 40 locations, respectively. After the “get” request is sent and the resulting JSON converted to a dictionary, certain geo-computations are required to find the parking with the smallest distance from the user’s input. We first conduct geocoding to get the latitude and longitude of the user’s address. Then we find its distance from each of the locations in our dictionary using “geopy.distance.geodesic”. The item with the smallest distance is our target parking location. Since these computations take time, efficiently querying the minimum number of items from Firebase is important, since geocoding 37 locations instead of 335 makes a big difference. For the scenario (mentioned at the beginning of this functionality) where a specific ID is inputted, “location_description” directly queries the database for that single location – since ID is it’s the Firebase parent key. This bypasses all of the aforementioned looping and geo-computations. Once the specific location dictionary is identified, the function parses through the available data to then output it in a clear and intuitive format.

However, the final component of the safety score needs to be calculated before it is reported. We calculate the current risk at the specific parking location based on live, crime updates from Citizen. Real-time scraping of “citizen.com” is not only very challenging due to its dynamic front-end design, but it would also take too long and be error prone. So, we created a Gmail account that receives alerts for all LA incidents. Whenever a user query is made to see the details of a specific location, the function uses the “imaplib” and “email” libraries to parse all Citizen emails (less than a day old) to see if any of them recently occurred in that area. Incidents that occurred more than 12 hours ago is ignored. Any other email that matches one of our predefined keywords is selected. The keywords are: "shot", "slash", "armed", "fire", "gun", "stab", "knife", "blaze", "homicide", "rob", "carjack", "police", "vehicle". These keywords are needed in order to prevent alerts like "Have You Seen Kevin?" – which have nothing to do with risk – from inaccurately decreasing the safety score. Then, like we did for the crime dataset, we find the pairwise distances using the GeoPy library, and only select crimes that are within half-mile from the chosen parking. Since currently occurring crimes are dynamic, I used a larger radius of consideration than for the LAPD dataset. For every new crime that occurred within 1, 4, and 12 hours, we subtract 25, 10, and 5 from the safety score (respectively). Finally, all of the parking location details are returned as three strings variables and are used as inputs for the front-end code (Appendix D). The first string has the address, location type, number of spots, availability, last updated (time), and safety score information. The second variable has the details on the relevant Citizen crimes that affect the safety score; it can be empty. The third string states whether the location is open and provides the pricing data.

In the app, once the user searches for the parking address in this "Specific Parking Information" page and receives the results, they can update the availability information (Appendix E Image 3). This Waze-like functionality is the last, main component of our app. The backend function that implements this is called "update_availability" and is found in the "functionality1.py" script (Appendix D). This function takes the parking location "ID" and whether the availability is being updated to True/False as the arguments. Once the user executes the update in the app, the values for these arguments are passed to this backend function as inputs. Using the "patch" method from the "requests" library, a patch query is sent to the database. The value for the "availability" key is thus updated and a new key:value item is created in Firebase showing the exact update time. The time upon execution (for all the functions) is acquired using "datetime.now()". The next time the "location_description" function, this updated information will be reflected in the displayed results. An extra function in this same Python file is called "daily_reset". This function takes no inputs and requires no user interaction. As the name suggest, it is executed daily (at midnight) and by using the "patch" and "delete" methods resets the availability information for all parking locations to the default of an unconfirmed True. On a Mac, this can be done by scheduling the execution task using CronJobs. Since most parking facilities and street parking do not allow overnight parking, this daily reset keeps the availability information more objectively accurate for the new day.

Frontend Design and User-Interface

In implementing the front-end portion of our application, we decided that each of our three pages, 'Mission Statement', 'Specific Parking & Availability Updates', and 'Locate Nearby Parking' would each have their own function that when called, would render the page and its features. Furthermore, for specific functionality related to the backend, such as updating availability or querying for locations based on index, functions representing them in the backend are imported over and nested within each page's function as needed. Users are first presented the Mission Statement page by default, and on the left, there is a radio widget which allows users to navigate to other pages of the app by clicking on the icon associated with it. On the homepage, there is a section dedicated to explaining our motivation for building Park It, along with a background on the data used and a dedicated section explaining the 3 core functionalities we provide (Appendix E Image 1).

The remaining two pages the user can navigate to contain the core functionality of our application. On the Specific Parking and Availability Updates page, users are first presented with a form and prompted to enter a parking spot's specific address. In the scenario that a user already knows the address of a parking location and its type (Street Parking, Garage, Lot, Not Street Parking), they are able to fill out the form with these details and click the search button (Appendix E Image 2). These user inputs are then passed as parameters to a backend function, "location_description", which queries our database on those values to return various strings corresponding to different types of information on that location. The user is then notified that their query is loading while they wait by using a "st.spinner" function that disappears once a function is finished running.

Once the query is complete, its contents are printed out to the application containing the address, type of parking, valet status, safety score, availability, and price (Appendix Image 3). Below this, two options for the user are available once they finish looking at the specific parking information: a checkbox (i.e. "st.checkbox") to clear the query results, and another checkbox to proceed towards the availability update function of our application. When the latter is clicked on by the user, they are then prompted to enter the availability for that specific location they looked up. The assumption is that a user may have looked at the information for a parking spot marked as available, but when they drive there to look for parking, all spots are occupied. Therefore, a user can notify other drivers that there is currently no space available in real-time by selecting occupied from the drop-down menu. Once selected, their response is sent to the backend to update the availability value for that location in our Firebase database to match what they input. They are then notified of this change if successful, and they are prompted to then click on a checkbox to clear the query result. If a user re-submits the query the changes made to the location's availability status are made (Appendix E Image 4). Whether a user decides to update availability or not, they are prompted in both scenarios to clear the query result as not only is it more appealing visually afterwards, but it is also important to certain functions of the Streamlit library that are later discussed in our section of the report explaining the challenges associated with using Streamlit.

The final page of our application is the Locate Nearby Parking page, which allows a user to input their location and desired filters to find parking locations that fit their criteria (Appendix E Image 5). Like the previous page, the user is presented with an identical form to fill out an address, except instead of a location's address they input theirs. The user is then prompted below to select any filters they might have as requirements for locations that could be returned as results. These filters range from parking type, valet preference, restrictions, availability, and a desired safety score. In addition to this, users can select how to sort the locations that match their criteria. Commonly available sorting criteria, such as distance and duration, are options for the user, but keeping in mind our motivation to help driver safety, we also allow user to sort locations by its safety score. Finally, the user can select how many results they would like to view from 5-25 in increments of 5.

Once the form is filled out, the user clicks the search button and like the previous page, gets a notification that their query is loading. Their various responses are then stored in variables and passed to the appropriate backend function. In this case, the "parking_list" function is called first, which queries the database and sorts the results based on user criteria, then returns a DataFrame of the results and a dictionary containing the coordinates for the user's location. These values are then passed into the Map( ) function for our frontend which uses "pydeck", a high-scale spatial rendering library, to display a map to the user. This map is done in a scatterplot style with two layers. The first, is where the user's longitude and latitude values from the dictionary are plotted as a green point on the map. For the locations in the DataFrame, the columns containing latitude and longitude data for each instance are passed as data for the map, and they are plotted as red points on the map as the second layer. The map is interactive and allows the user to zoom in/out and drag the map around (Appendix E Image 6). Unfortunately, we were not able to include labels for points on our interactive map, as it required an html template and other configurations to implement, which would potentially increase the load times for our application.

At the same time the map appears, a table representing the DataFrame of locations is also outputted to the user (Appendix E Image 6). Each location has individual columns for address, distance from the user, the time it takes to driver there (duration), its safety score, and an ID value representing its index in our Firebase database. The latitude and longitude columns used for the map are dropped in the front-end code before displaying the DataFrame, given that they are irrelevant for the user. Like the map, this table is also interactive to users and allows them to sort rows, one attribute on a time, by clicking on column headers. Once the user has looked at the results and decided on a location they would like to park at, they can find more detailed information on its pricing and hours in the drop-down menu below the table. Here they are prompted to select one location by selecting a number that corresponds to the index provided in the table. Once an index is selected, it is passed to a different version of the "location_description" provided by the backend, which takes in an index and queries Firebase for the information related to the corresponding location. This returns the same values as displayed in the Specific Parking Information and Availability page (Appendix E Image 7). If a user decides to instead view another location, they can click on the x next to the index in the drop-down menu to clear their search to select a new index. Once a user views one location's specific information, a checkbox appears below the information to clear the entire query result once they are finished looking at all locations, they are interested in.

Challenges Faced

Web Scraping, Data Preprocessing, Database Querying, and Backend Implementation

One of the challenges we faced early on was that, despite its appearance, the Parkopedia website is dynamically structured, which set severe limitations on crawling and scraping using the standard "requests" and "BeautifulSoup" libraries. I had to use "Selenium" to interact with the JavaScript rendered pages and reach the desired parking data. Unfortunately, I didn't have experience with "Selenium", so I had to first familiarize myself with this library. Another problem was the website's small query limit, which their robots.txt didn't provide any information about. Even when spacing out the queries one minute apart, Parkopedia would block me after approximately 16 requests. The websites dynamic nature contributed to this problem, since getting to our desired data page was only possible after two "clicks". This made web scraping the data for several hundred parking locations a long process, which we recently completely. Overall, we had underestimated the difficulty of this task and the time it would take. Unfortunately, the format of the retrieved data also wasn't very organized (i.e. tabular), which additionally required a lot of preprocessing to format all the information into our desired parking attributes. Similarly, the program for preprocessing the LAPD crime dataset (including the various geocoding tasks) took several days to finish running.

We also had to deal with the limitations of the Realtime Firebase database. We chose Firebase because of its real-time applications, and it was the cloud database we were most familiar with at the time (through class). The initial simplicity for setting Firebase up through a single JSON string's "put" request was also a benefit we considered. Unfortunately, the "get" requests for information retrieval are rather inefficient and don't support using complex querying conditions. Due to an absence of an operator like "not" or "in", sometimes two separate requests were necessary in the backend code. For example, when we have to only select locations that are "Not street parking", we need the items that match with "Garage" and "Lot". With many databases, an "or", "not", or "in" operator could be used to retrieve the desired data with a single query; unfortunately, this did not seem to be the case for Realtime Firebase. Also, as mentioned before, the Firebase query allows selecting only one “orderBy” key, which was not ideal for our project since our various functionalities often had multiple filtering criteria. We had to adapt and use if/elif blocks to maximize the benefit of the single condition-key query – by minimizing the number of returned location items. Still, this Firebase characteristic was problematic and required more local filtering and sorting computations, thus increasing the app’s loading time for the results.

Our third member (Karthik Sivanadiyan) abruptly leaving the team before the Midterm Report was the core obstacle that we encountered; this forced us to not only work harder to complete the app in time, but also to reevaluate the inclusion of various intended functionalities and reduce their complexity. Some, more time-consuming details that weren’t crucial, such as related to formatting, had to be skipped or simplified. For example, we originally considered including even more variables in our safety score function – such as for the presence of security cameras. This information proved difficult to find (much less automate) and manually visiting these locations is unrealistic. So, we couldn’t include this. Also, in addition to the Parkopedia data, we had found two relevant parking datasets, which we had hoped to use for crosschecking and perhaps integrate with the Firebase database (Appendix F and G). The “LADOT Metered Parking” dataset also had a related API called “LADOT Parking Meter Occupancy”, which updates occupancy information for those spaces in real time (Appendix F). Implementing this would have increased our app’s utility. Unfortunately, due to how time-consuming preprocessing data is and our lack of time and manpower, these additional app components had to be abandoned (for now).

Frontend

As stated, this project was designed for a three- person team, taking into consideration all of each other’s’ specialties and shortcomings. Since Sivanadiyan had various front-end responsibilities, his sudden departure forced Alain Tamazian to work on the web-scraping, preprocessing, and backend code alone, while team member Alexander Chavez handled the entire frontend development portion of our application. Given Chavez’s inexperience with web development, it took longer than expected to learn and create the user interface with the “Streamlit” library – as it was originally intended to be developed by our previous group member who had substantial front-end experience. Fortunately, since our team size became two, our UI did not need to be accessible from the internet and is instead hosted on our local machine – which made the application deployment step straight forward.

In developing our application, we faced challenges with the Streamlit library that made it more difficult to develop our UI in the specific way we desired. One of the first challenges involved printing strings and other text to the application. Streamlit does not render formatted Python strings correctly, as newline characters (“\n”) in a string are required to be a 2-space distance away from the actual content that you wish to put on a new line to work properly. Tabbing/indenting lines with the corresponding python character (“\t”) are not rendered at all, and after extensive searching for potential solutions, only the solution for newline characters could be found. Unfortunately, this bug and its work-around are not mentioned within Streamlit’s documentation for the “st.markdown” and “st.write” functions, both of which print out text and data to the user. As a result, we could not format all the information outputted to the user in as formal of a design as we wanted.

Another issue we encountered was with Streamlit’s use of the “pydeck” library for more interactive maps instead of the default “st.map” function. While “pydeck” allowed us to center the map by default to the user’s location and control other aspects such as icon size and zoom, there were constraints when implementing multiple layers. To display text or other values when a user hovers over a point on the map, this required a specific HTML file to be referenced as the tooltip= value for the map. However, since one layer was for location data and the other layer was for user data, multiple HTML files would have to be specified as values, which “pydeck” did not allow. Given time-constraints for the app’s loading speed, and the complexity of integrating multiple HTML files with both the back/front end, this feature unfortunately could not be implemented.

The biggest challenge faced when using the Streamlit library was being able to maintain session state when working with button widgets to prevent pages from reloading and erasing user inputted data. As acknowledged by Streamlit itself, there are issues with its “st.button” function for button widgets. Prior to the button being clicked on by the user, all functionality and interactions made are saved onto the page. This is still saved even after the button is clicked; however, the caveat is that all functionality after a button press can only be non-interactive to preserve the page’s session with user data. For example, in the page for Specific Parking Information and Availability Updates, I originally had a user fill out a form, click on a search button, and then display the information for the queried location below. However, once we decided to implement an availability update feature, trying to add it as a checkbox below the displayed information proved difficult to use at first. This was because every time the user clicked on the checkbox to procced to the availability update function, it would reload the page and reset the query results that had been displayed up to that point. After extensive searches on Streamlit forums to try and find the solution, it turned out that in one of their forum posts, they acknowledge that the button widget does not work as a developer using Streamlit would intend for it to. This is because the button is tied to a dictionary hidden from the user, which has a Boolean value corresponding to the status of the button. This is Streamlit’s way of managing session state, and by default, Streamlit always re-runs the entire code for a page once an input is received. In the case of the button widget, re-running the page sets the status of the button in session state to be reset to False, as if the user had not pressed it, even though they did previously. Because writing is the only Streamlit functionality that does not require re-running the page, initially only print statements could be performed after a button press. Therefore, to maintain the session on the page, the dictionary value for “button” had to be set to True permanently once a user clicks on it, so that even when a page is re-run, the value in session state remains True. While this allows for unlimited user interactions and functions to run after a button is pressed, the value for the button must also be set to False once the session is complete. Otherwise, the saved session will automatically run every time the user refreshes the page, or when another user interacts with the address and parking type inputs before clicking search. This means that results will appear even if the user does not click the search button or has finished entering inputs. Therefore, to avoid a poor UI design for both pages with user interaction, users are prompted to clear the query result. If they click on the checkbox, then the value of the button’s state in the session state of Streamlit is set to False, and then any other interaction with the page will clear the session, allowing the user and others to properly interact with the page. In the case of our Locate Nearby Parking page, this had to be considered when deciding how to allow a viewer to select a specific parking spot from the many results their search returned (Appendix E Image 6 and 7). Due to the complexity of maintaining session state and its sensitivity to user interaction, it was decided that it would be better to have parking information for a user-selected location to display on the same page rather than redirecting to the Specific Parking Information page.

Other minor problems experienced with Streamlit include limited interaction in displaying DataFrames, as the interactive tables cannot be sorted on multiple column headers, and some columns refuse to sort properly. However, outside of the previously mentioned complications, Streamlit provided a streamlined way for us to develop the frontend of our application and integrate with the backend functions by just using the Python language. While the library has several quirks and features that have not been developed as much as we would have liked during the creation of our application's frontend, it allowed us to overcome the challenge of losing our previous teammate with frontend experience. Overall, despite the various challenges we faced throughout this project (and some compromises that had to be made), we successfully completed our "Park It: Safe and Simple" web-app along with all of the core functionalities (for safety and utility) that we hoped to include.

Team Members and Responsibilities

Alain Tamazian:

- Dataset collection, integration, and preprocessing – including web-scraping and geocoding API queries for Realtime Firebase database creation
- Backend Python programming for the functionalities described in the above section; receiving inputs (from users), processing data, outputting results
- Lead on the reports

Alexander Chavez:

- Frontend implementation of the web-app (using Streamlit)
- Integration between the backend and frontend code
- Lead on the demo video/presentation

References

LADOT. "Ladot Metered Parking Inventory & Policies." *Catalog.data.gov*, Data.lacity.org, 29 Nov. 2021, https://catalog.data.gov/dataset/ladot-metered-parking-inventory-policies.

LA GeoHub. "City Owned Parking Lots." *Https://Geohub.lacity.org/*, City of Los Angeles Hub, 12 Aug. 2016, https://geohub.lacity.org/datasets/lahub::city-owned-parking-lots/explore.

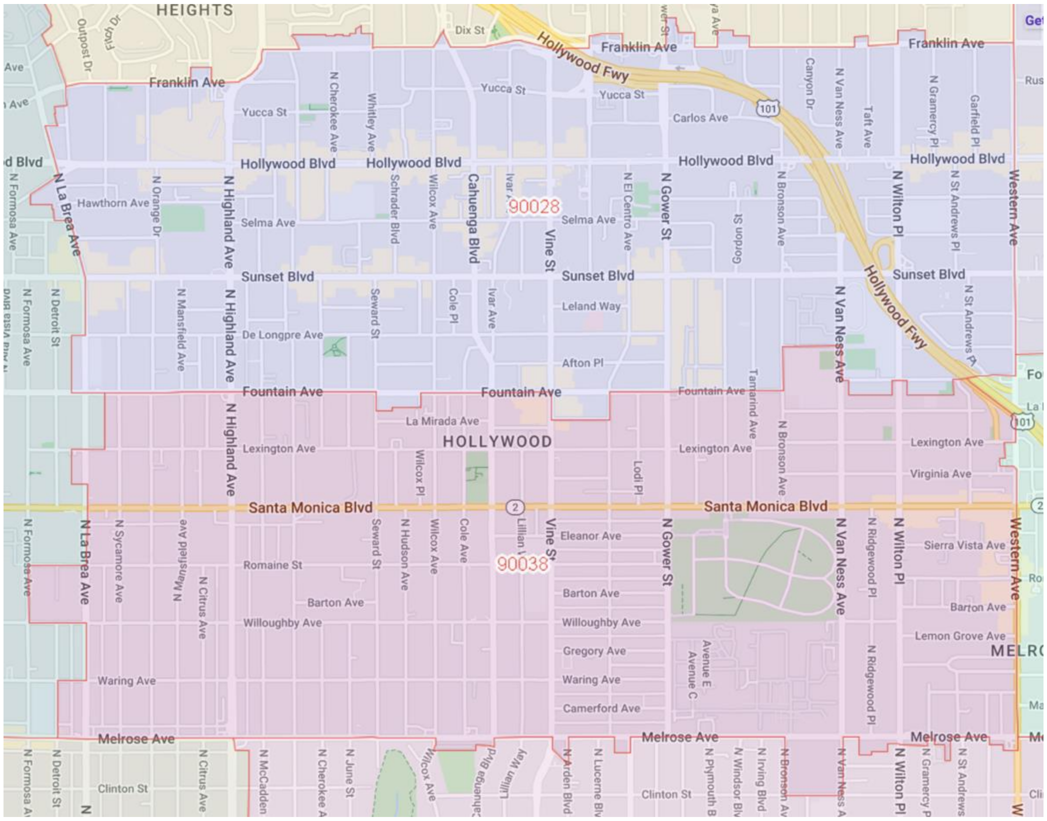
LAPD OpenData. "Crime Data from 2020 to Present." *catalog.data.gov*, data.lacity.org, 7 Feb. 2022, https://catalog.data.gov/dataset/crime-data-from-2020-to-present.

"FBI Data Confirms Risk in Hospital Parking Garage after Attack - Security." *Www.healthcarefacilitiestoday.com*, Healthcare Facilities Today, 11 Feb. 2019, https://www.healthcarefacilitiestoday.com/posts/FBI-data-confirms-risk-in-hospital-parking-garage-after-attack--20637.

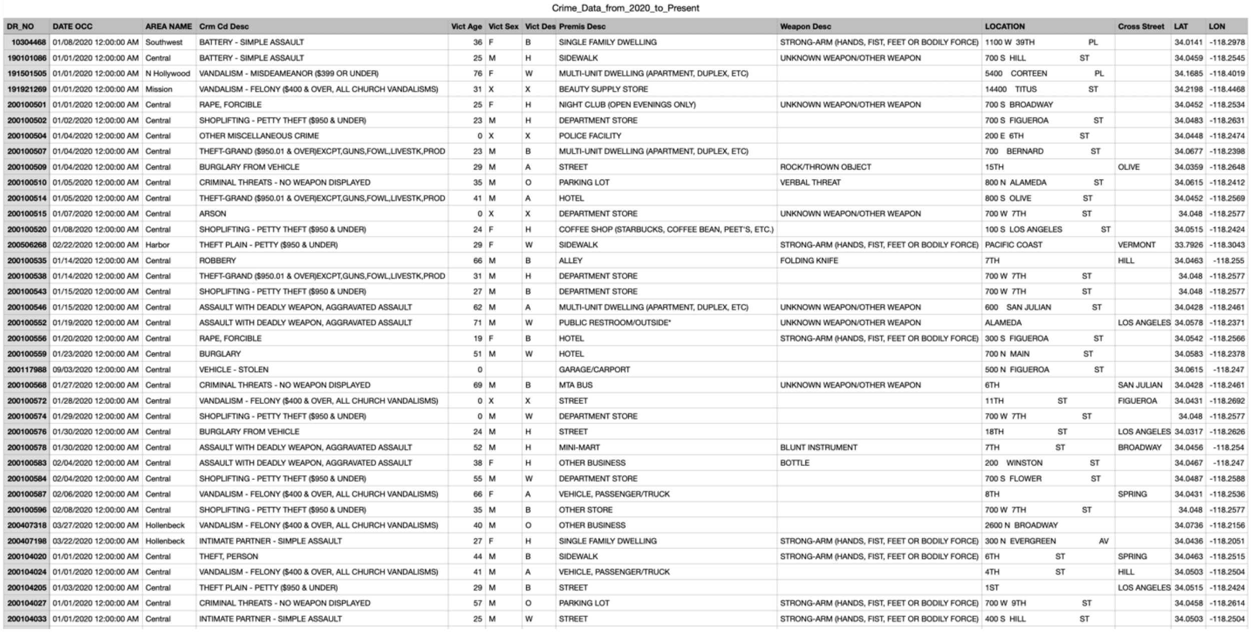
FBI. "Trend of Violent Crime from 2010 to 2020." *Crime-Data-Explorer.fr*, Federal Bureau of Investigation Crime Data Explorer, 31 Dec. 2020, https://crime-data-explorer.fr.cloud.gov/pages/explorer/crime/crime-trend.

Appendices

Appendix A



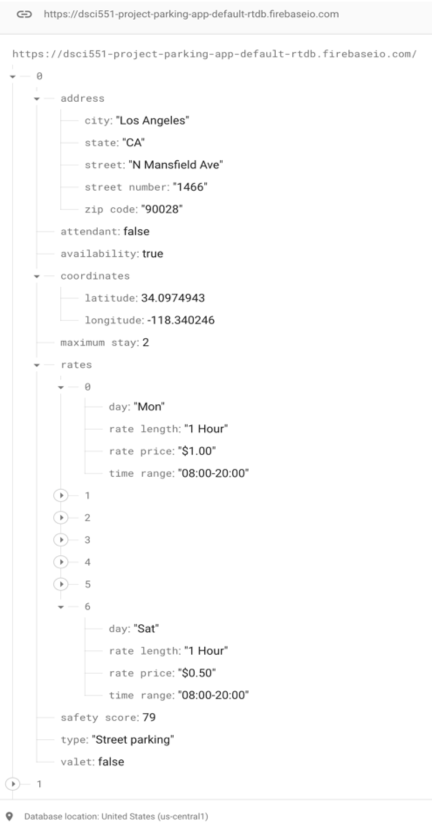
Appendix B



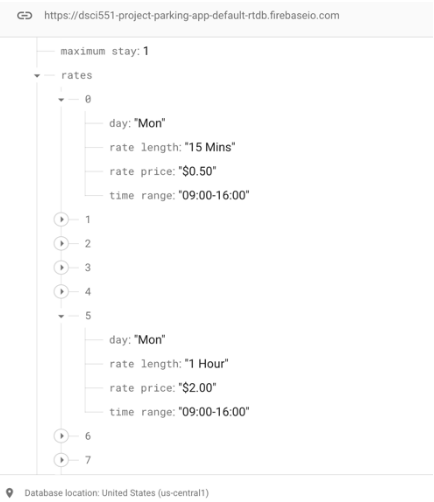
Crime_Data_from_2020_to_Present

DR_NO	DATE OCC	AREA NAME	Crm Cd Desc	Vict Age	Vict Sex	Vict Des	Premis Desc	Weapon Desc	LOCATION	Cross Street	LAT	LON
10304468	01/08/2020 12:00:00 AM	Southwest	BATTERY - SIMPLE ASSAULT	36	F	B	SINGLE FAMILY DWELLING	STRONG-ARM (HANDS, FIST, FEET OR BODILY FORCE)	1100 W 39TH PL		34.0141	-118.2978
190101086	01/01/2020 12:00:00 AM	Central	BATTERY - SIMPLE ASSAULT	25	M	H	SIDEWALK	UNKNOWN WEAPON/OTHER WEAPON	700 S HILL ST		34.0459	-118.2545
191501505	01/01/2020 12:00:00 AM	N Hollywood	VANDALISM - MISDEAMEANOR ($399 OR UNDER)	76	F	W	MULTI-UNIT DWELLING (APARTMENT, DUPLEX, ETC)		5400 CORTEEN PL		34.1685	-118.4019
191921269	01/01/2020 12:00:00 AM	Mission	VANDALISM - FELONY ($400 & OVER, ALL CHURCH VANDALISMS)	31	X	X	BEAUTY SUPPLY STORE		14400 TITUS ST		34.2198	-118.4468
200100501	01/01/2020 12:00:00 AM	Central	RAPE, FORCIBLE	25	F	H	NIGHT CLUB (OPEN EVENINGS ONLY)	UNKNOWN WEAPON/OTHER WEAPON	700 S BROADWAY		34.0452	-118.2534
200100502	01/02/2020 12:00:00 AM	Central	SHOPLIFTING - PETTY THEFT ($950 & UNDER)	23	M	H	DEPARTMENT STORE		700 S FIGUEROA ST		34.0483	-118.2631
200100504	01/04/2020 12:00:00 AM	Central	OTHER MISCELLANEOUS CRIME	0	X	X	POLICE FACILITY		200 E 6TH ST		34.0448	-118.2474
200100507	01/04/2020 12:00:00 AM	Central	THEFT-GRAND ($950.01 & OVER)EXCPT,GUNS,FOWL,LIVESTK,PROD	23	M	B	MULTI-UNIT DWELLING (APARTMENT, DUPLEX, ETC)		700 BERNARD ST		34.0677	-118.2398
200100509	01/04/2020 12:00:00 AM	Central	BURGLARY FROM VEHICLE	29	M	A	STREET	ROCK/THROWN OBJECT	15TH	OLIVE	34.0359	-118.2648
200100510	01/05/2020 12:00:00 AM	Central	CRIMINAL THREATS - NO WEAPON DISPLAYED	35	M	O	PARKING LOT	VERBAL THREAT	800 N ALAMEDA ST		34.0615	-118.2412
200100514	01/05/2020 12:00:00 AM	Central	THEFT-GRAND ($950.01 & OVER)EXCPT,GUNS,FOWL,LIVESTK,PROD	41	M	A	HOTEL		800 S OLIVE ST		34.0452	-118.2569
200100515	01/07/2020 12:00:00 AM	Central	ARSON	0	X	X	DEPARTMENT STORE	UNKNOWN WEAPON/OTHER WEAPON	700 W 7TH ST		34.048	-118.2577
200100520	01/08/2020 12:00:00 AM	Central	SHOPLIFTING - PETTY THEFT ($950 & UNDER)	24	F	H	COFFEE SHOP (STARBUCKS, COFFEE BEAN, PEET'S, ETC.)		100 S LOS ANGELES ST		34.0515	-118.2424
200506268	02/22/2020 12:00:00 AM	Harbor	THEFT PLAIN - PETTY ($950 & UNDER)	29	F	W	SIDEWALK	STRONG-ARM (HANDS, FIST, FEET OR BODILY FORCE)	PACIFIC COAST	VERMONT	33.7926	-118.3043
200100535	01/14/2020 12:00:00 AM	Central	ROBBERY	66	M	B	ALLEY	FOLDING KNIFE	7TH	HILL	34.0463	-118.255
200100538	01/14/2020 12:00:00 AM	Central	THEFT-GRAND ($950.01 & OVER)EXCPT,GUNS,FOWL,LIVESTK,PROD	31	M	H	DEPARTMENT STORE		700 W 7TH ST		34.048	-118.2577
200100543	01/15/2020 12:00:00 AM	Central	SHOPLIFTING - PETTY THEFT ($950 & UNDER)	27	M	B	DEPARTMENT STORE		700 W 7TH ST		34.048	-118.2577
200100546	01/15/2020 12:00:00 AM	Central	ASSAULT WITH DEADLY WEAPON, AGGRAVATED ASSAULT	62	M	A	MULTI-UNIT DWELLING (APARTMENT, DUPLEX, ETC)	UNKNOWN WEAPON/OTHER WEAPON	600 SAN JULIAN ST		34.0428	-118.2461
200100552	01/19/2020 12:00:00 AM	Central	ASSAULT WITH DEADLY WEAPON, AGGRAVATED ASSAULT	71	M	W	PUBLIC RESTROOM/OUTSIDE*	UNKNOWN WEAPON/OTHER WEAPON	ALAMEDA	LOS ANGELES	34.0578	-118.2371
200100556	01/20/2020 12:00:00 AM	Central	RAPE, FORCIBLE	19	F	B	HOTEL	STRONG-ARM (HANDS, FIST, FEET OR BODILY FORCE)	300 S FIGUEROA ST		34.0542	-118.2566
200100559	01/23/2020 12:00:00 AM	Central	BURGLARY	51	M	W	HOTEL		700 N MAIN ST		34.0583	-118.2378
200117988	09/03/2020 12:00:00 AM	Central	VEHICLE - STOLEN	0			GARAGE/CARPORT		500 N FIGUEROA ST		34.0615	-118.247
200100568	01/27/2020 12:00:00 AM	Central	CRIMINAL THREATS - NO WEAPON DISPLAYED	69	M	B	MTA BUS	UNKNOWN WEAPON/OTHER WEAPON	6TH	SAN JULIAN	34.0428	-118.2461
200100572	01/28/2020 12:00:00 AM	Central	VANDALISM - FELONY ($400 & OVER, ALL CHURCH VANDALISMS)	0	X	X	STREET		11TH ST	FIGUEROA	34.0431	-118.2692
200100574	01/29/2020 12:00:00 AM	Central	SHOPLIFTING - PETTY THEFT ($950 & UNDER)	0	M	W	DEPARTMENT STORE		700 W 7TH ST		34.048	-118.2577
200100576	01/30/2020 12:00:00 AM	Central	BURGLARY FROM VEHICLE	24	M	H	STREET		18TH ST	LOS ANGELES	34.0317	-118.2626
200100578	01/30/2020 12:00:00 AM	Central	ASSAULT WITH DEADLY WEAPON, AGGRAVATED ASSAULT	52	M	H	MINI-MART	BLUNT INSTRUMENT	7TH ST	BROADWAY	34.0456	-118.254
200100583	02/04/2020 12:00:00 AM	Central	ASSAULT WITH DEADLY WEAPON, AGGRAVATED ASSAULT	38	F	H	OTHER BUSINESS	BOTTLE	200 WINSTON ST		34.0467	-118.247
200100584	02/04/2020 12:00:00 AM	Central	SHOPLIFTING - PETTY THEFT ($950 & UNDER)	55	M	W	DEPARTMENT STORE		700 S FLOWER ST		34.0487	-118.2588
200100587	02/06/2020 12:00:00 AM	Central	VANDALISM - FELONY ($400 & OVER, ALL CHURCH VANDALISMS)	66	F	A	VEHICLE, PASSENGER/TRUCK		8TH	SPRING	34.0431	-118.2536
200100596	02/08/2020 12:00:00 AM	Central	SHOPLIFTING - PETTY THEFT ($950 & UNDER)	35	M	B	OTHER STORE		700 W 7TH ST		34.048	-118.2577
200407318	03/27/2020 12:00:00 AM	Hollenbeck	VANDALISM - FELONY ($400 & OVER, ALL CHURCH VANDALISMS)	40	M	O	OTHER BUSINESS		2600 N BROADWAY		34.0736	-118.2156
200407198	03/22/2020 12:00:00 AM	Hollenbeck	INTIMATE PARTNER - SIMPLE ASSAULT	27	F	H	SINGLE FAMILY DWELLING	STRONG-ARM (HANDS, FIST, FEET OR BODILY FORCE)	300 N EVERGREEN AV		34.0436	-118.2051
200104020	01/01/2020 12:00:00 AM	Central	THEFT, PERSON	44	M	B	SIDEWALK	STRONG-ARM (HANDS, FIST, FEET OR BODILY FORCE)	6TH ST	SPRING	34.0463	-118.2515
200104024	01/01/2020 12:00:00 AM	Central	VANDALISM - FELONY ($400 & OVER, ALL CHURCH VANDALISMS)	41	M	A	VEHICLE, PASSENGER/TRUCK		4TH ST	HILL	34.0503	-118.2504
200104205	01/03/2020 12:00:00 AM	Central	THEFT PLAIN - PETTY ($950 & UNDER)	29	M	B	STREET		1ST	LOS ANGELES	34.0515	-118.2424
200104027	01/01/2020 12:00:00 AM	Central	CRIMINAL THREATS - NO WEAPON DISPLAYED	57	M	O	PARKING LOT	STRONG-ARM (HANDS, FIST, FEET OR BODILY FORCE)	700 W 9TH ST		34.0458	-118.2614
200104033	01/01/2020 12:00:00 AM	Central	INTIMATE PARTNER - SIMPLE ASSAULT	25	M	W	STREET	STRONG-ARM (HANDS, FIST, FEET OR BODILY FORCE)	400 S HILL ST		34.0503	-118.2504

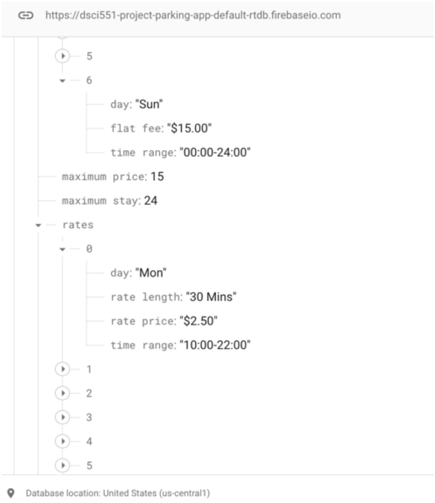
Appendix C



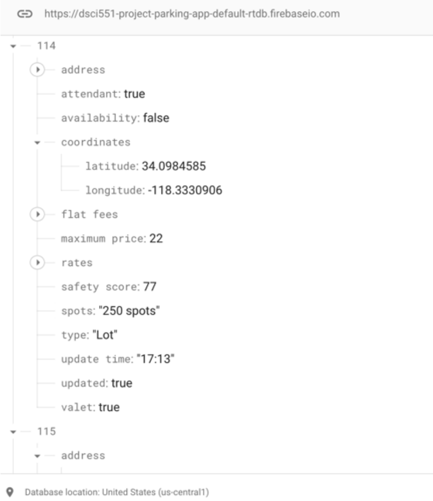
```
https://dsci551-project-parking-app-default-rtdb.firebaseio.com

https://dsci551-project-parking-app-default-rtdb.firebaseio.com/
- 0
    - address
        city: "Los Angeles"
        state: "CA"
        street: "N Mansfield Ave"
        street number: "1466"
        zip code: "90028"
    attendant: false
    availability: true
    - coordinates
        latitude: 34.0974943
        longitude: -118.340246
    maximum stay: 2
    - rates
        - 0
            day: "Mon"
            rate length: "1 Hour"
            rate price: "$1.00"
            time range: "08:00-20:00"
        1
        2
        3
        4
        5
        - 6
            day: "Sat"
            rate length: "1 Hour"
            rate price: "$0.50"
            time range: "08:00-20:00"
    safety score: 79
    type: "Street parking"
    valet: false
1

Database location: United States (us-central1)
```



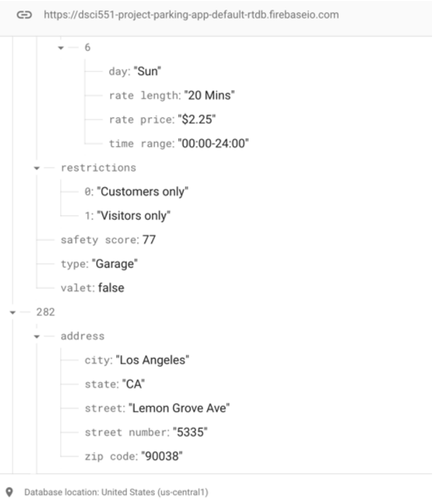
```
https://dsci551-project-parking-app-default-rtdb.firebaseio.com

    maximum stay: 1
    - rates
        - 0
            day: "Mon"
            rate length: "15 Mins"
            rate price: "$0.50"
            time range: "09:00-16:00"
        1
        2
        3
        4
        - 5
            day: "Mon"
            rate length: "1 Hour"
            rate price: "$2.00"
            time range: "09:00-16:00"
        6
        7

Database location: United States (us-central1)
```



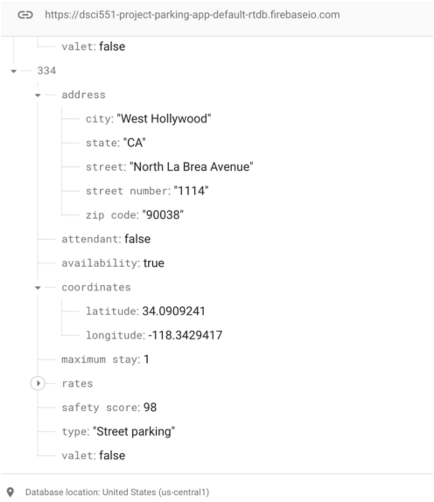
```
https://dsci551-project-parking-app-default-rtdb.firebaseio.com

        5
        - 6
            day: "Sun"
            flat fee: "$15.00"
            time range: "00:00-24:00"
    maximum price: 15
    maximum stay: 24
    - rates
        - 0
            day: "Mon"
            rate length: "30 Mins"
            rate price: "$2.50"
            time range: "10:00-22:00"
        1
        2
        3
        4
        5

Database location: United States (us-central1)
```



```
https://dsci551-project-parking-app-default-rtdb.firebaseio.com

- 114
    address
    attendant: true
    availability: false
    - coordinates
        latitude: 34.0984585
        longitude: -118.3330906
    flat fees
    maximum price: 22
    rates
    safety score: 77
    spots: "250 spots"
    type: "Lot"
    update time: "17:13"
    updated: true
    valet: true
- 115
    - address

Database location: United States (us-central1)
```



```
https://dsci551-project-parking-app-default-rtdb.firebaseio.com

        - 6
            day: "Sun"
            rate length: "20 Mins"
            rate price: "$2.25"
            time range: "00:00-24:00"
    - restrictions
        0: "Customers only"
        1: "Visitors only"
    safety score: 77
    type: "Garage"
    valet: false
- 282
    - address
        city: "Los Angeles"
        state: "CA"
        street: "Lemon Grove Ave"
        street number: "5335"
        zip code: "90038"

Database location: United States (us-central1)
```



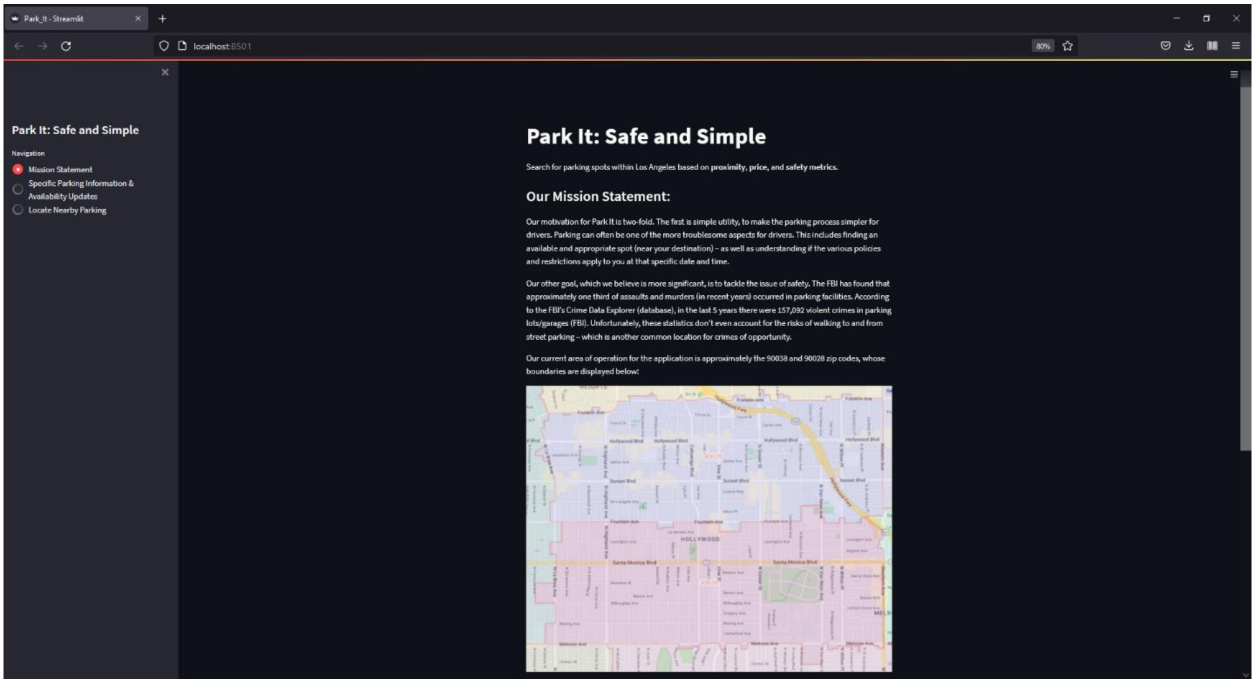
```
https://dsci551-project-parking-app-default-rtdb.firebaseio.com

    valet: false
- 334
    - address
        city: "West Hollywood"
        state: "CA"
        street: "North La Brea Avenue"
        street number: "1114"
        zip code: "90038"
    attendant: false
    availability: true
    - coordinates
        latitude: 34.0909241
        longitude: -118.3429417
    maximum stay: 1
    rates
    safety score: 98
    type: "Street parking"
    valet: false

Database location: United States (us-central1)
```

Appendix D

https://drive.google.com/drive/folders/1ryS-eDtU4dAx3Ueks3QD_wyoz1dM6ljd
(Google Drive link for project code submission)

Appendix E
Images of Park It UI

Image 1: Homepage with mission statement, boundaries of Park It's operation, and description of core functionalities



Park It: Safe and Simple

Search for parking spots within Los Angeles based on **proximity, price,** and **safety metrics.**

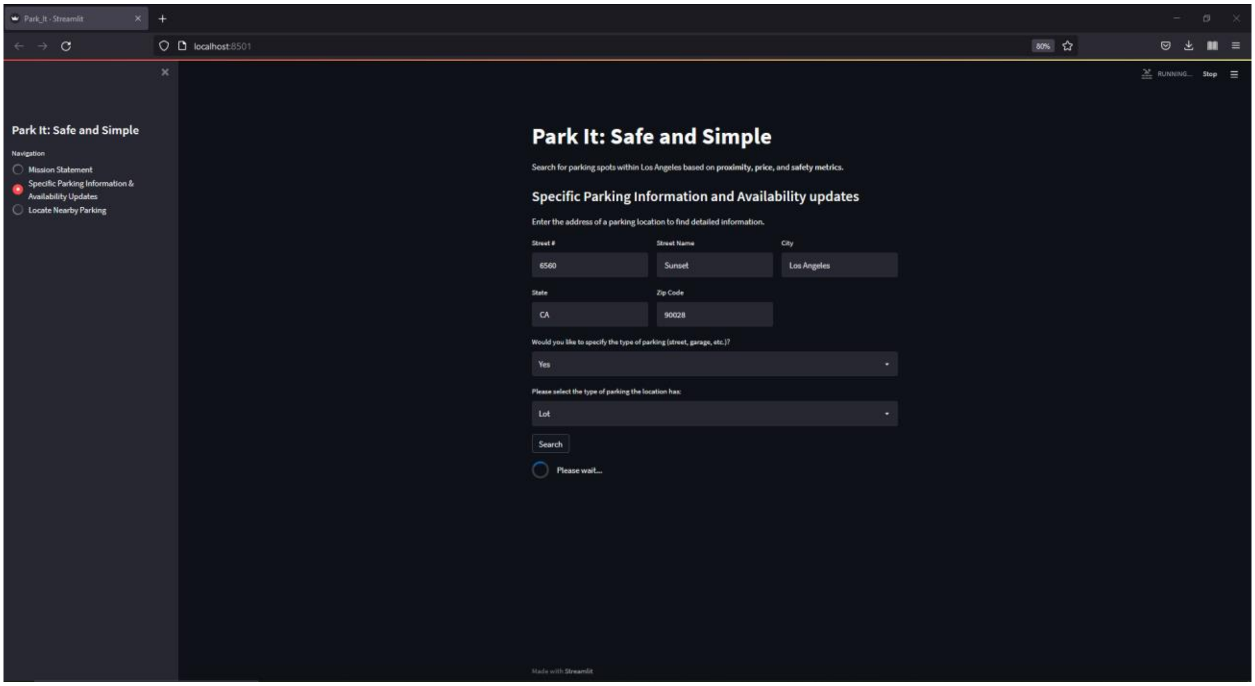
Our Mission Statement:

Our motivation for Park It is two-fold. The first is simple utility, to make the parking process simpler for drivers. Parking can often be one of the more troublesome aspects for drivers. This includes finding an available and appropriate spot (near your destination) – as well as understanding if the various policies and restrictions apply to you at that specific date and time.

Our other goal, which we believe is more significant, is to tackle the issue of safety. The FBI has found that approximately one third of assaults and murders (in recent years) occurred in parking facilities. According to the FBI's Crime Data Explorer (database), in the last 5 years there were 157,092 violent crimes in parking lots/garages (FBI). Unfortunately, these statistics don't even account for the risks of walking to and from street parking – which is another common location for crimes of opportunity.

Our current area of operation for the application is approximately the 90038 and 90028 zip codes, whose boundaries are displayed below:

Image 2: Specific Parking Information page structure, and sample user query



Park It: Safe and Simple

Search for parking spots within Los Angeles based on **proximity, price,** and **safety metrics.**

Specific Parking Information and Availability updates

Enter the address of a parking location to find detailed information.

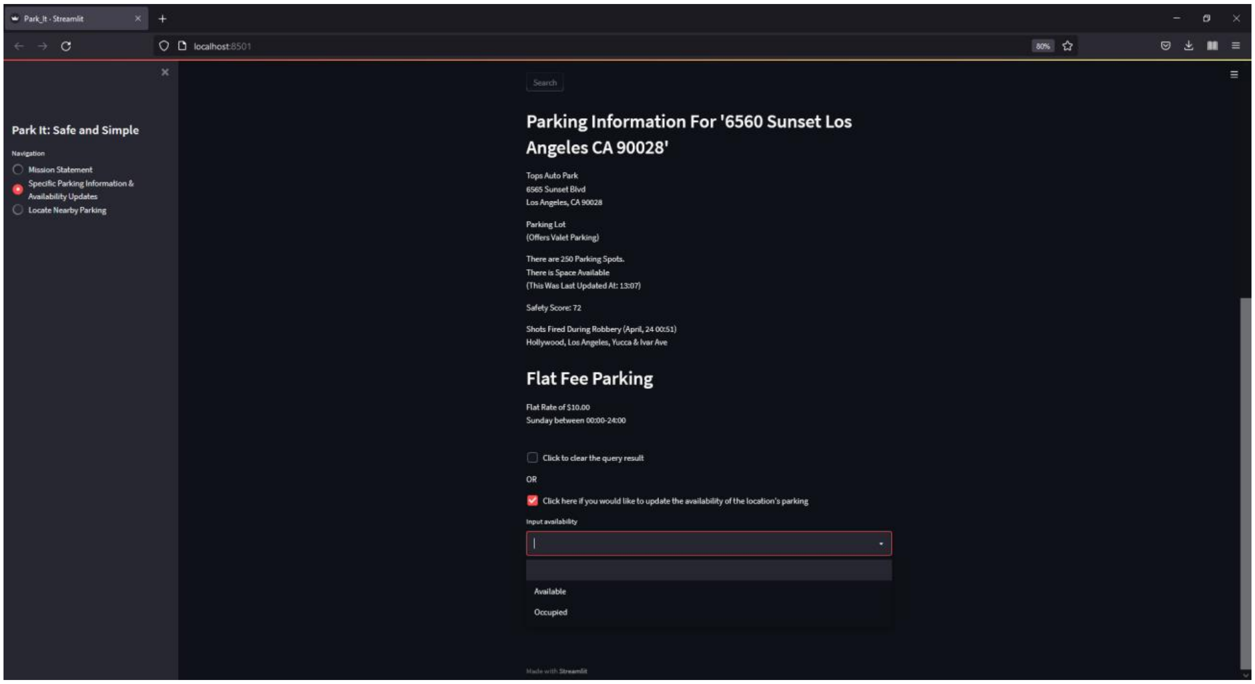
Street #: 6560
Street Name: Sunset
City: Los Angeles
State: CA
Zip Code: 90028

Would you like to specify the type of parking (street, garage, etc.)?: Yes

Please select the type of parking the location has: Lot

Please wait...

Image 3: Information retrieved from sample user query and subsequent availability update feature



Park It: Safe and Simple

Navigation

○ Mission Statement
● Specific Parking Information & Availability Updates
○ Locate Nearby Parking

Parking Information For '6560 Sunset Los Angeles CA 90028'

Tops Auto Park
6565 Sunset Blvd
Los Angeles, CA 90028

Parking Lot
(Offers Valet Parking)

There are 250 Parking Spots.
There is Space Available
(This Was Last Updated At: 13:07)

Safety Score: 72

Shots Fired During Robbery (April, 24 00:51)
Hollywood, Los Angeles, Yucca & Ivar Ave

Flat Fee Parking

Flat Rate of $10.00
Sunday between 00:00-24:00

□ Click to clear the query result

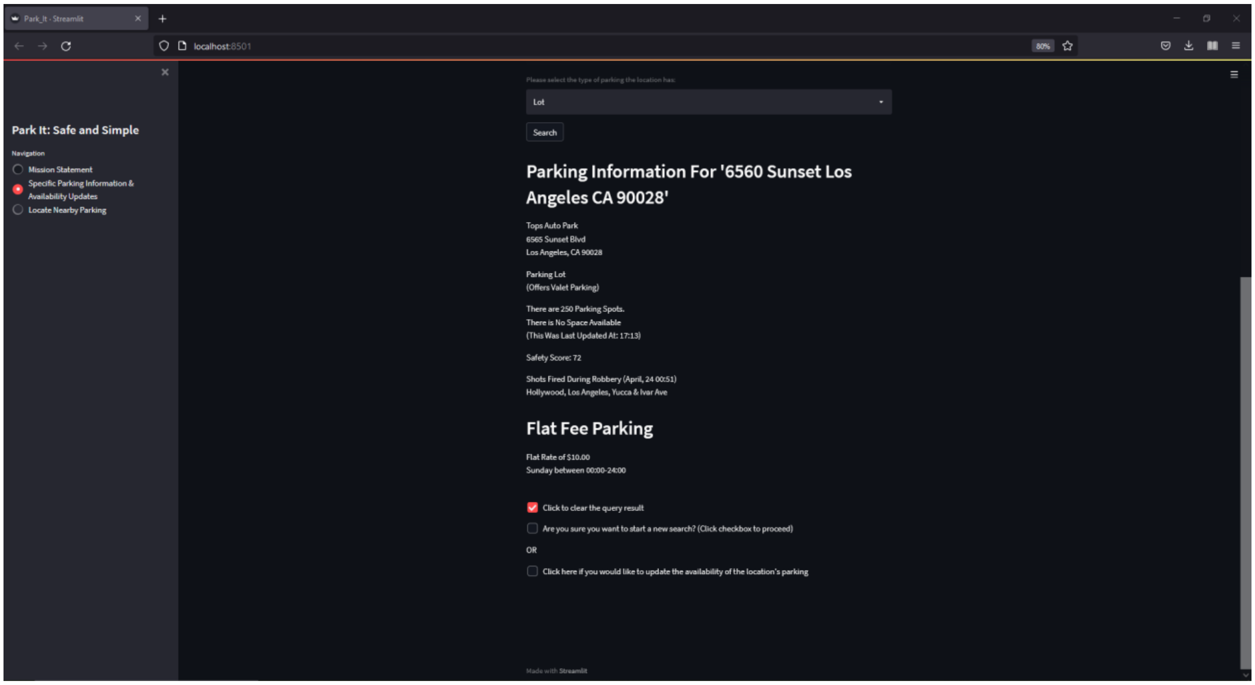
OR

☑ Click here if you would like to update the availability of the location's parking

Input availability

Available
Occupied

Image 4: An updated availability status for the sample user location, and prompt for clearing user query results.



Park It: Safe and Simple

Navigation

○ Mission Statement
● Specific Parking Information & Availability Updates
○ Locate Nearby Parking

Please select the type of parking the location has:

Lot

Parking Information For '6560 Sunset Los Angeles CA 90028'

Tops Auto Park
6565 Sunset Blvd
Los Angeles, CA 90028

Parking Lot
(Offers Valet Parking)

There are 250 Parking Spots.
There is No Space Available
(This Was Last Updated At: 17:13)

Safety Score: 72

Shots Fired During Robbery (April, 24 00:51)
Hollywood, Los Angeles, Yucca & Ivar Ave

Flat Fee Parking

Flat Rate of $10.00
Sunday between 00:00-24:00

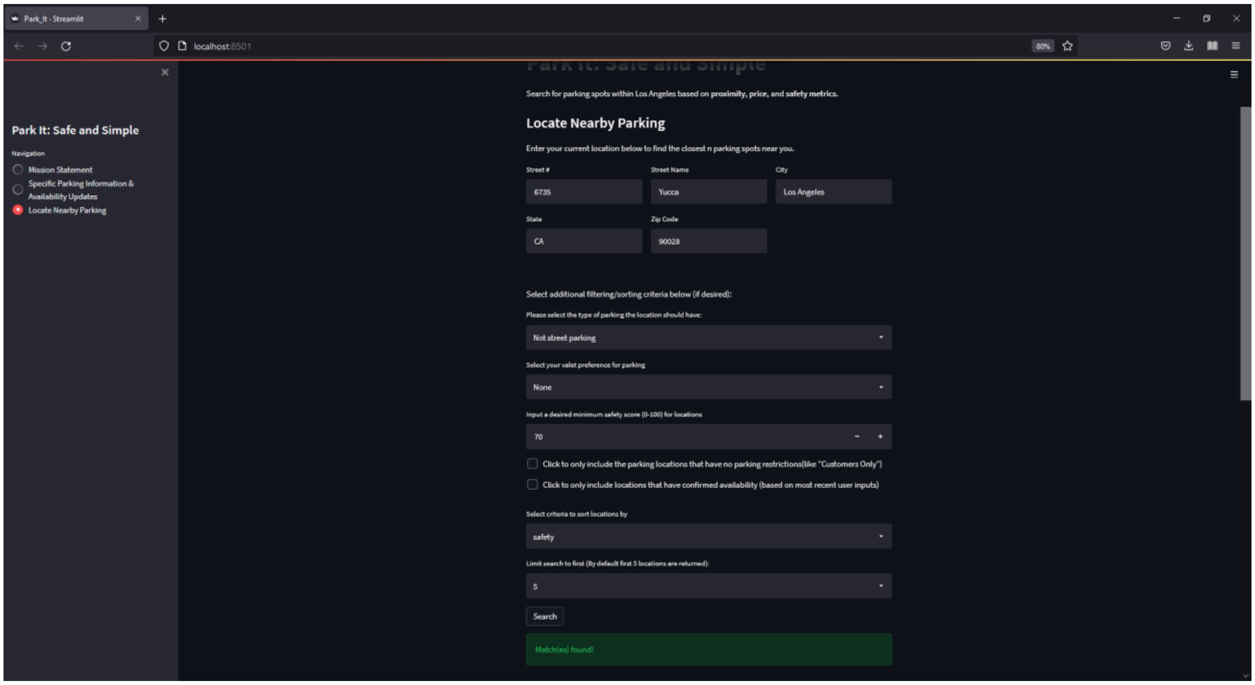
☑ Click to clear the query result

□ Are you sure you want to start a new search? (Click checkbox to proceed)

OR

□ Click here if you would like to update the availability of the location's parking

Image 5: Locate Nearby Parking page structure, and sample user query



Park It: Safe and Simple

Navigation

○ Mission Statement

○ Specific Parking Information & Availability Updates

● Locate Nearby Parking

Park It: Safe and Simple

Search for parking spots within Los Angeles based on **proximity, price,** and **safety metrics.**

Locate Nearby Parking

Enter your current location below to find the closest n parking spots near you.

Street #	Street Name	City
6735	Yucca	Los Angeles

State	Zip Code
CA	90028

Select additional filtering/sorting criteria below (if desired):

Please select the type of parking the location should have: Not street parking

Select your valet preference for parking: None

Input a desired minimum safety score (0-100) for locations: 70

□ Click to only include the parking locations that have no parking restrictions(like "Customers Only")

□ Click to only include locations that have confirmed availability (based on most recent user inputs)

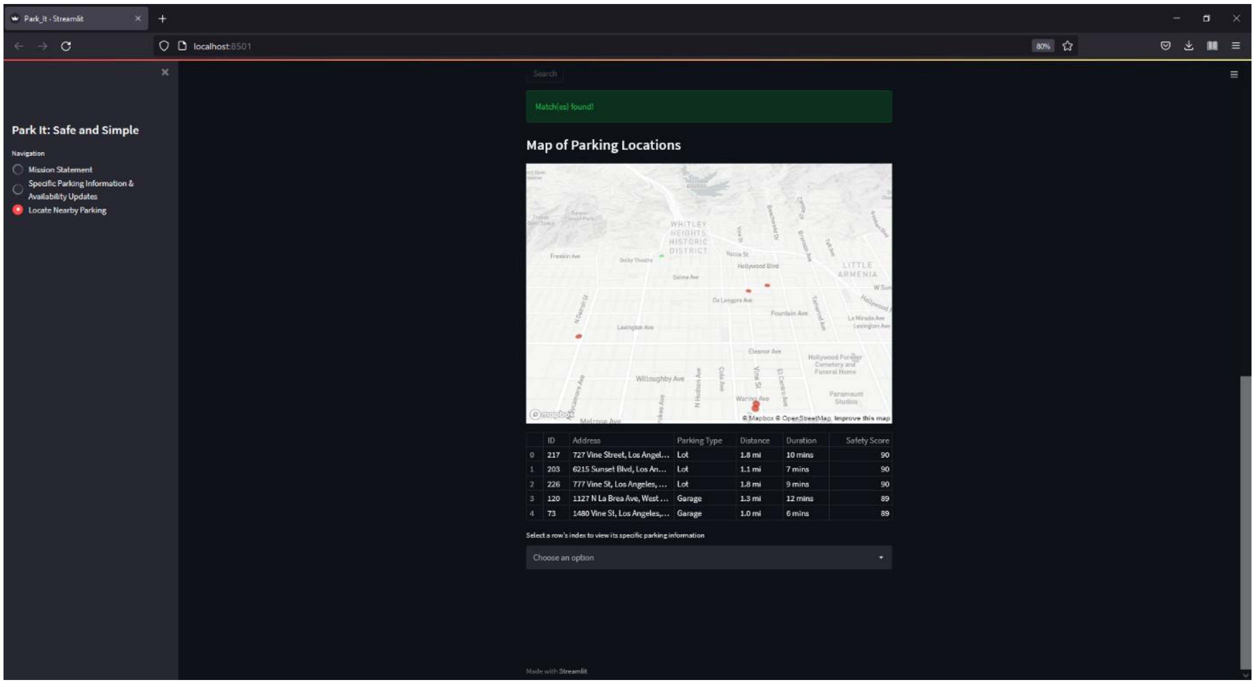
Select criteria to sort locations by: safety

Limit search to first (By default first 5 locations are returned): 5

Search

Match(es) found!

Image 6: Interactive map and table displaying the results of the sample user query



Park It: Safe and Simple

Navigation

○ Mission Statement

○ Specific Parking Information & Availability Updates

● Locate Nearby Parking

Match(es) found!

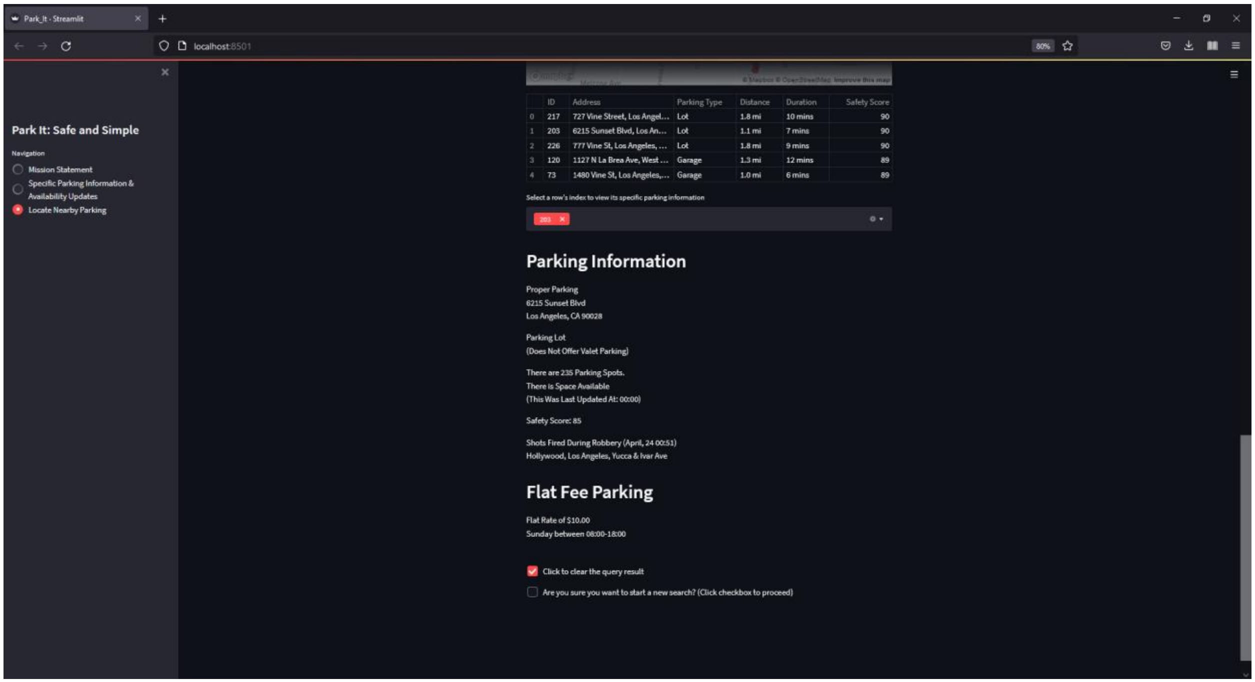
Map of Parking Locations

	ID	Address	Parking Type	Distance	Duration	Safety Score
0	217	727 Vine Street, Los Angel…	Lot	1.8 mi	10 mins	90
1	203	6215 Sunset Blvd, Los An…	Lot	1.1 mi	7 mins	90
2	226	777 Vine St, Los Angeles, …	Lot	1.8 mi	9 mins	90
3	120	1127 N La Brea Ave, West …	Garage	1.3 mi	12 mins	89
4	73	1480 Vine St, Los Angeles,…	Garage	1.0 mi	6 mins	89

Select a row's index to view its specific parking information: Choose an option

Made with Streamlit

Image 7: User search for a specific location's information using its index value (ID column) from table + Option to clear query



Park It: Safe and Simple

Navigation

○ Mission Statement
○ Specific Parking Information & Availability Updates
● Locate Nearby Parking

	ID	Address	Parking Type	Distance	Duration	Safety Score
0	217	727 Vine Street, Los Angel…	Lot	1.8 mi	10 mins	90
1	203	6215 Sunset Blvd, Los An…	Lot	1.1 mi	7 mins	90
2	226	777 Vine St, Los Angeles, …	Lot	1.8 mi	9 mins	90
3	120	1127 N La Brea Ave, West …	Garage	1.3 mi	12 mins	89
4	73	1480 Vine St, Los Angeles,…	Garage	1.0 mi	6 mins	89

Select a row's index to view its specific parking information

203 ×

Parking Information

Proper Parking
6215 Sunset Blvd
Los Angeles, CA 90028

Parking Lot
(Does Not Offer Valet Parking)

There are 235 Parking Spots.
There is Space Available
(This Was Last Updated At: 00:00)

Safety Score: 85

Shots Fired During Robbery (April, 24 00:51)
Hollywood, Los Angeles, Yucca & Ivar Ave

Flat Fee Parking

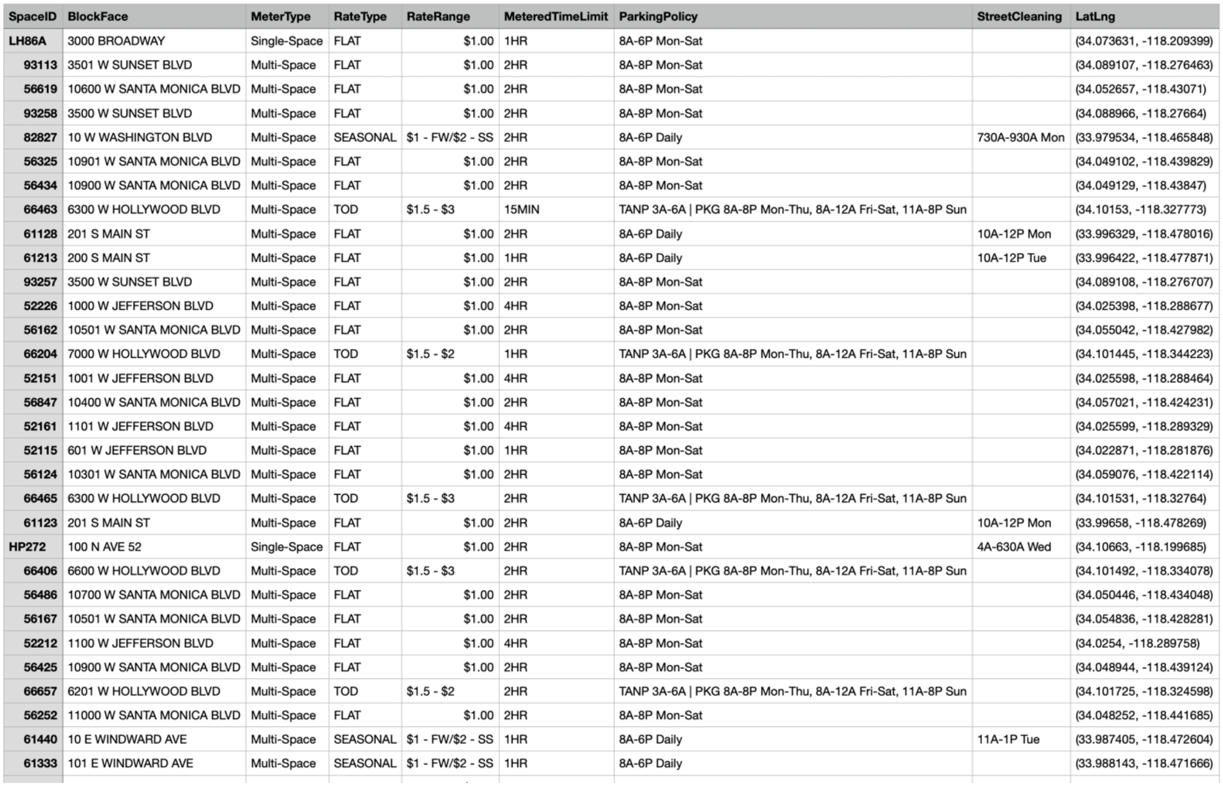
Flat Rate of $10.00
Sunday between 08:00-18:00

☑ Click to clear the query result

☐ Are you sure you want to start a new search? (Click checkbox to proceed)

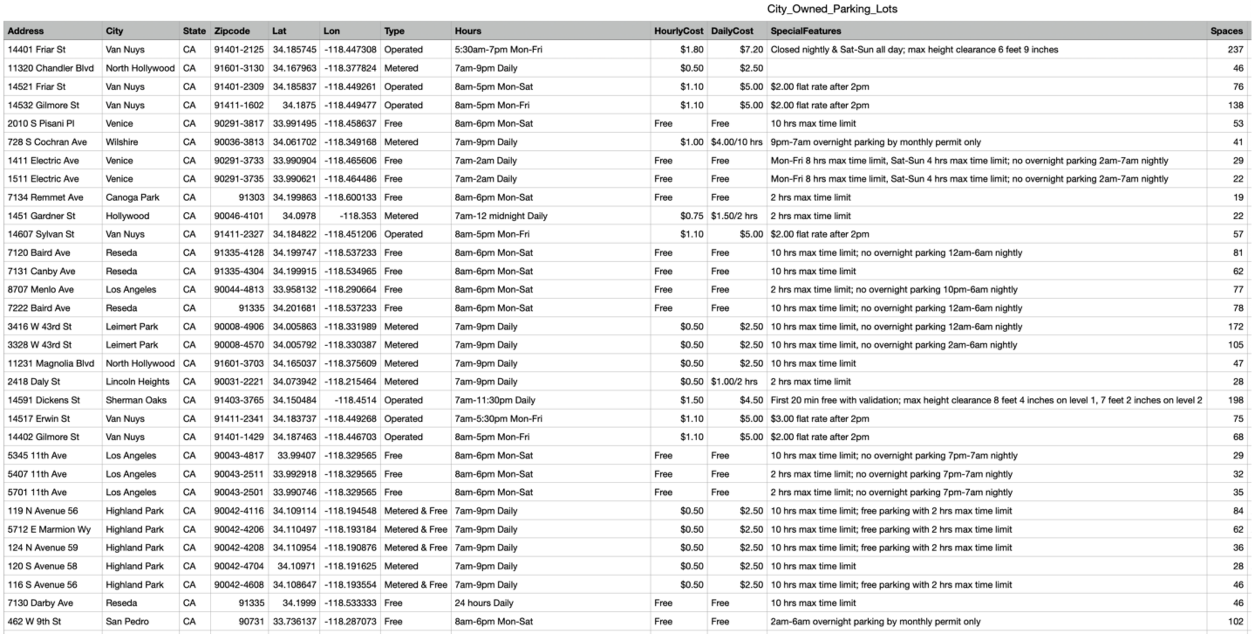
Appendix F

LADOT_Metered_Parking_Inventory___Policies



SpaceID	BlockFace	MeterType	RateType	RateRange	MeteredTimeLimit	ParkingPolicy	StreetCleaning	LatLng
LH86A	3000 BROADWAY	Single-Space	FLAT	$1.00	1HR	8A-6P Mon-Sat		(34.073631, -118.209399)
93113	3501 W SUNSET BLVD	Multi-Space	FLAT	$1.00	2HR	8A-8P Mon-Sat		(34.089107, -118.276463)
56619	10600 W SANTA MONICA BLVD	Multi-Space	FLAT	$1.00	2HR	8A-8P Mon-Sat		(34.052657, -118.43071)
93258	3500 W SUNSET BLVD	Multi-Space	FLAT	$1.00	2HR	8A-8P Mon-Sat		(34.088966, -118.27664)
82827	10 W WASHINGTON BLVD	Multi-Space	SEASONAL	$1 - FW/$2 - SS	2HR	8A-6P Daily	730A-930A Mon	(33.979534, -118.465848)
56325	10901 W SANTA MONICA BLVD	Multi-Space	FLAT	$1.00	2HR	8A-8P Mon-Sat		(34.049102, -118.439829)
56434	10900 W SANTA MONICA BLVD	Multi-Space	FLAT	$1.00	2HR	8A-8P Mon-Sat		(34.049129, -118.43847)
66463	6300 W HOLLYWOOD BLVD	Multi-Space	TOD	$1.5 - $3	15MIN	TANP 3A-6A | PKG 8A-8P Mon-Thu, 8A-12A Fri-Sat, 11A-8P Sun		(34.10153, -118.327773)
61128	201 S MAIN ST	Multi-Space	FLAT	$1.00	2HR	8A-6P Daily	10A-12P Mon	(33.996329, -118.478016)
61213	200 S MAIN ST	Multi-Space	FLAT	$1.00	1HR	8A-6P Daily	10A-12P Tue	(33.996422, -118.477871)
93257	3500 W SUNSET BLVD	Multi-Space	FLAT	$1.00	2HR	8A-8P Mon-Sat		(34.089108, -118.276707)
52226	1000 W JEFFERSON BLVD	Multi-Space	FLAT	$1.00	4HR	8A-8P Mon-Sat		(34.025398, -118.288677)
56162	10501 W SANTA MONICA BLVD	Multi-Space	FLAT	$1.00	2HR	8A-8P Mon-Sat		(34.055042, -118.427982)
66204	7000 W HOLLYWOOD BLVD	Multi-Space	TOD	$1.5 - $2	1HR	TANP 3A-6A | PKG 8A-8P Mon-Thu, 8A-12A Fri-Sat, 11A-8P Sun		(34.101445, -118.344223)
52151	1001 W JEFFERSON BLVD	Multi-Space	FLAT	$1.00	4HR	8A-8P Mon-Sat		(34.025598, -118.288464)
56847	10400 W SANTA MONICA BLVD	Multi-Space	FLAT	$1.00	2HR	8A-8P Mon-Sat		(34.057021, -118.424231)
52161	1101 W JEFFERSON BLVD	Multi-Space	FLAT	$1.00	4HR	8A-8P Mon-Sat		(34.025599, -118.289329)
52115	601 W JEFFERSON BLVD	Multi-Space	FLAT	$1.00	1HR	8A-8P Mon-Sat		(34.022871, -118.281876)
56124	10301 W SANTA MONICA BLVD	Multi-Space	FLAT	$1.00	2HR	8A-8P Mon-Sat		(34.059076, -118.422114)
66465	6300 W HOLLYWOOD BLVD	Multi-Space	TOD	$1.5 - $3	2HR	TANP 3A-6A | PKG 8A-8P Mon-Thu, 8A-12A Fri-Sat, 11A-8P Sun		(34.101531, -118.32764)
61123	201 S MAIN ST	Multi-Space	FLAT	$1.00	2HR	8A-6P Daily	10A-12P Mon	(33.99658, -118.478269)
HP272	100 N AVE 52	Single-Space	FLAT	$1.00	2HR	8A-8P Mon-Sat	4A-630A Wed	(34.10663, -118.199685)
66406	6600 W HOLLYWOOD BLVD	Multi-Space	TOD	$1.5 - $3	2HR	TANP 3A-6A | PKG 8A-8P Mon-Thu, 8A-12A Fri-Sat, 11A-8P Sun		(34.101492, -118.334078)
56486	10700 W SANTA MONICA BLVD	Multi-Space	FLAT	$1.00	2HR	8A-8P Mon-Sat		(34.050446, -118.434048)
56167	10501 W SANTA MONICA BLVD	Multi-Space	FLAT	$1.00	2HR	8A-8P Mon-Sat		(34.054836, -118.428281)
52212	1100 W JEFFERSON BLVD	Multi-Space	FLAT	$1.00	4HR	8A-8P Mon-Sat		(34.0254, -118.289758)
56425	10900 W SANTA MONICA BLVD	Multi-Space	FLAT	$1.00	2HR	8A-8P Mon-Sat		(34.048944, -118.439124)
66657	6201 W HOLLYWOOD BLVD	Multi-Space	TOD	$1.5 - $2	2HR	TANP 3A-6A | PKG 8A-8P Mon-Thu, 8A-12A Fri-Sat, 11A-8P Sun		(34.101725, -118.324598)
56252	11000 W SANTA MONICA BLVD	Multi-Space	FLAT	$1.00	2HR	8A-8P Mon-Sat		(34.048252, -118.441685)
61440	10 E WINDWARD AVE	Multi-Space	SEASONAL	$1 - FW/$2 - SS	1HR	8A-6P Daily	11A-1P Tue	(33.987405, -118.472604)
61333	101 E WINDWARD AVE	Multi-Space	SEASONAL	$1 - FW/$2 - SS	1HR	8A-6P Daily		(33.988143, -118.471666)

Appendix G



City_Owned_Parking_Lots

Address	City	State	Zipcode	Lat	Lon	Type	Hours	HourlyCost	DailyCost	SpecialFeatures	Spaces
14401 Friar St	Van Nuys	CA	91401-2125	34.185745	-118.447308	Operated	5:30am-7pm Mon-Fri	$1.80	$7.20	Closed nightly & Sat-Sun all day; max height clearance 6 feet 9 inches	237
11320 Chandler Blvd	North Hollywood	CA	91601-3130	34.167963	-118.377824	Metered	7am-9pm Daily	$0.50	$2.50		46
14521 Friar St	Van Nuys	CA	91401-2309	34.185837	-118.449261	Operated	8am-5pm Mon-Sat	$1.10	$5.00	$2.00 flat rate after 2pm	76
14532 Gilmore St	Van Nuys	CA	91411-1602	34.1875	-118.449477	Operated	8am-5pm Mon-Fri	$1.10	$5.00	$2.00 flat rate after 2pm	138
2010 S Pisani Pl	Venice	CA	90291-3817	33.991495	-118.458637	Free	8am-6pm Mon-Sat	Free	Free	10 hrs max time limit	53
728 S Cochran Ave	Wilshire	CA	90036-3813	34.061702	-118.349168	Metered	7am-9pm Daily	$1.00	$4.00/10 hrs	9pm-7am overnight parking by monthly permit only	41
1411 Electric Ave	Venice	CA	90291-3733	33.990904	-118.465606	Free	7am-2am Daily	Free	Free	Mon-Fri 8 hrs max time limit, Sat-Sun 4 hrs max time limit; no overnight parking 2am-7am nightly	29
1511 Electric Ave	Venice	CA	90291-3735	33.990621	-118.464486	Free	7am-2am Daily	Free	Free	Mon-Fri 8 hrs max time limit, Sat-Sun 4 hrs max time limit; no overnight parking 2am-7am nightly	22
7134 Remmet Ave	Canoga Park	CA	91303	34.199863	-118.600133	Free	8am-6pm Mon-Sat	Free	Free	2 hrs max time limit	19
1451 Gardner St	Hollywood	CA	90046-4101	34.0978	-118.353	Metered	7am-12 midnight Daily	$0.75	$1.50/2 hrs	2 hrs max time limit	22
14607 Sylvan St	Van Nuys	CA	91411-2327	34.184822	-118.451206	Operated	8am-5pm Mon-Fri	$1.10	$5.00	$2.00 flat rate after 2pm	57
7120 Baird Ave	Reseda	CA	91335-4128	34.199747	-118.537233	Free	8am-6pm Mon-Sat	Free	Free	10 hrs max time limit; no overnight parking 12am-6am nightly	81
7131 Canby Ave	Reseda	CA	91335-4304	34.199915	-118.534965	Free	8am-6pm Mon-Sat	Free	Free	10 hrs max time limit	62
8707 Menlo Ave	Los Angeles	CA	90044-4813	33.958132	-118.290664	Free	8am-6pm Mon-Sat	Free	Free	2 hrs max time limit; no overnight parking 10pm-6am nightly	77
7222 Baird Ave	Reseda	CA	91335	34.201681	-118.537233	Free	8am-6pm Mon-Sat	Free	Free	10 hrs max time limit; no overnight parking 12am-6am nightly	78
3416 W 43rd St	Leimert Park	CA	90008-4906	34.005863	-118.331989	Metered	7am-9pm Daily	$0.50	$2.50	10 hrs max time limit, no overnight parking 12am-6am nightly	172
3328 W 43rd St	Leimert Park	CA	90008-4570	34.005792	-118.330387	Metered	7am-9pm Daily	$0.50	$2.50	10 hrs max time limit, no overnight parking 2am-6am nightly	105
11231 Magnolia Blvd	North Hollywood	CA	91601-3703	34.165037	-118.375609	Metered	7am-9pm Daily	$0.50	$2.50	10 hrs max time limit	47
2418 Daly St	Lincoln Heights	CA	90031-2221	34.073942	-118.215464	Metered	7am-9pm Daily	$0.50	$1.00/2 hrs	2 hrs max time limit	28
14591 Dickens St	Sherman Oaks	CA	91403-3765	34.150484	-118.4514	Operated	7am-11:30pm Daily	$1.50	$4.50	First 20 min free with validation; max height clearance 8 feet 4 inches on level 1, 7 feet 2 inches on level 2	198
14517 Erwin St	Van Nuys	CA	91411-2341	34.183737	-118.449268	Operated	7am-5:30pm Mon-Fri	$1.10	$5.00	$3.00 flat rate after 2pm	75
14402 Gilmore St	Van Nuys	CA	91401-1429	34.187463	-118.446703	Operated	8am-5pm Mon-Fri	$1.10	$5.00	$2.00 flat rate after 2pm	68
5345 11th Ave	Los Angeles	CA	90043-4817	33.99407	-118.329565	Free	8am-6pm Mon-Sat	Free	Free	10 hrs max time limit; no overnight parking 7pm-7am nightly	29
5407 11th Ave	Los Angeles	CA	90043-2511	33.992918	-118.329565	Free	8am-6pm Mon-Sat	Free	Free	2 hrs max time limit; no overnight parking 7pm-7am nightly	32
5701 11th Ave	Los Angeles	CA	90043-2501	33.990746	-118.329565	Free	8am-6pm Mon-Sat	Free	Free	2 hrs max time limit; no overnight parking 7pm-7am nightly	35
119 N Avenue 56	Highland Park	CA	90042-4116	34.109114	-118.194548	Metered & Free	7am-9pm Daily	$0.50	$2.50	10 hrs max time limit; free parking with 2 hrs max time limit	84
5712 E Marmion Wy	Highland Park	CA	90042-4206	34.110497	-118.193184	Metered & Free	7am-9pm Daily	$0.50	$2.50	10 hrs max time limit; free parking with 2 hrs max time limit	62
124 N Avenue 59	Highland Park	CA	90042-4208	34.110954	-118.190876	Metered & Free	7am-9pm Daily	$0.50	$2.50	10 hrs max time limit; free parking with 2 hrs max time limit	36
120 S Avenue 58	Highland Park	CA	90042-4704	34.10971	-118.191625	Metered	7am-9pm Daily	$0.50	$2.50	10 hrs max time limit	28
116 S Avenue 56	Highland Park	CA	90042-4608	34.108647	-118.193554	Metered & Free	7am-9pm Daily	$0.50	$2.50	10 hrs max time limit; free parking with 2 hrs max time limit	46
7130 Darby Ave	Reseda	CA	91335	34.1999	-118.533333	Free	24 hours Daily	Free	Free	10 hrs max time limit	46
462 W 9th St	San Pedro	CA	90731	33.736137	-118.287073	Free	8am-6pm Mon-Sat	Free	Free	2am-6am overnight parking by monthly permit only	102