Fall 2023 CS 4624 Capstone Project:

Parking Garage Occupancy Prediction

Adi Hanumaiah, Kevin He, Eugene Kwon, Antonio Lau, Josh Marcelin

Client: Mohamed Farag

Virginia Tech

Table of Contents

Table of Contents **1**
Abstract **2**
Introduction **3**
Requirements **4**
1. Login: 4
2. Prediction Model Integration: 4
3. Interactive Map: 4
4. Notification System and Spot Availability Forecast: 4
5. History and Analytics: 5
Design **6**
Frontend 6
Middleware 7
Backend 7
Implementation **7**
Testing/Evaluation/Assessment **8**
First iteration using LGBMRegressor 8
Users' Manual **10**
Developer's Manual **10**
Database Schema 10
Running Back End Locally 10
Accessing Database 11
Data provided 11
Libraries used for machine learning model 12
Lessons Learned **13**
Acknowledgements **13**
Client 13
References **13**
Appendices **14**
Plans going forward: **14**

Abstract

Across Virginia Tech's campus, finding parking is consistently a source of frustration for students and faculty. During peak hours, locating free parking spots becomes a challenging task; leading to significant delays and increased traffic around campus. Leveraging modern data-driven technologies such as Smart City infrastructure and Intelligent Transportation, we can alleviate some of the school's congestion and enhance the parking experience for Virginia Tech residents. The proposed solution is a web app that users can integrate into their daily commute. With the help of live data, the app will give real-time parking recommendations as well various other helpful insights. It will analyze the live data at each of the garages, to predict the occupancy of the garages at a given time of arrival. Machine learning will allow us to estimate the occupancy of each of the garages a given time into the future, depending on the distance to each garage, and provide a recommendation for which garage to target. The application will also allow for more effective collection of data for parking services and could eventually take into account more factors such as schedules and live traffic.

Introduction

The “Parking Spaces Occupancy Prediction” web app helps address issues students face when looking for parking on campus. Our web app will utilize real time data collection to build a prediction model that will inform drivers which area to target for parking. This will help improve driver experience by providing insight into which parking lots or garages have open spots available.

Client & Motivation:

Under the guidance of Dr. Farag, a research associate in the Center for Sustainable Mobility, our goal is to reduce the time drivers spend searching for parking. With limited parking available and as the enrollment at Virginia Tech grows, finding parking becomes a more pervasive challenge. Utilizing the prediction made from our prediction model, drivers can search for parking more efficiently.

Problem & General Approach:

Oftentimes searching for parking can be an arduous task. Drivers can expect to spend a considerable amount of time searching for open spaces especially during peak hours. Our project consists of a web app with a backend server and machine learning model. Our frontend built using Svelte connects to the backend which is built using Node js and Flask. As the user makes inputs such as selecting an arrival time, calls are made to the backend. Flask works with the machine learning model to make predictions on the number of parking spaces available by the user's arrival time. The number of parking spots available and the probability of finding open spots is then sent back to the user. Using this information drivers can more accurately assess which areas have more parking spaces available.

Requirements

Features

1. Login:

Our first requirement is a user-friendly login system. This ensures user data privacy and personalization of data. This will help facilitate individual user preferences and history retention of parking lot visitation data for enhanced service provision.

2. Prediction Model Integration:

Spot Availability Prediction: We are implementing a sophisticated machine learning algorithm that predicts the availability of parking spots. The model considers various features from the data such as historical availability and time-based attributes like the day of the week. This helps in capturing the weekly parking pattern. This prediction is based on an aggregation of historical data that was provided form our client.

3. Interactive Map:

Users will be able to access an interactive map which shows the various parking lot options and their available spaces that we are given the data for. We will also provide information on the routing data as well as distance from the user's location to the potential parking spot. This provides users with a holistic view of all available options, allowing them to make an informed choice.

4. Notification System and Spot Availability Forecast:

As part of ensuring a seamless experience, users will be notified of the availability of parking spaces in real-time. Furthermore, they can also access forecasts on spot availability, aiding them in planning ahead. Our app will alert users to change in recommended parking spots based on availability. We will also alert users to leave on time in order to arrive on time and find parking.

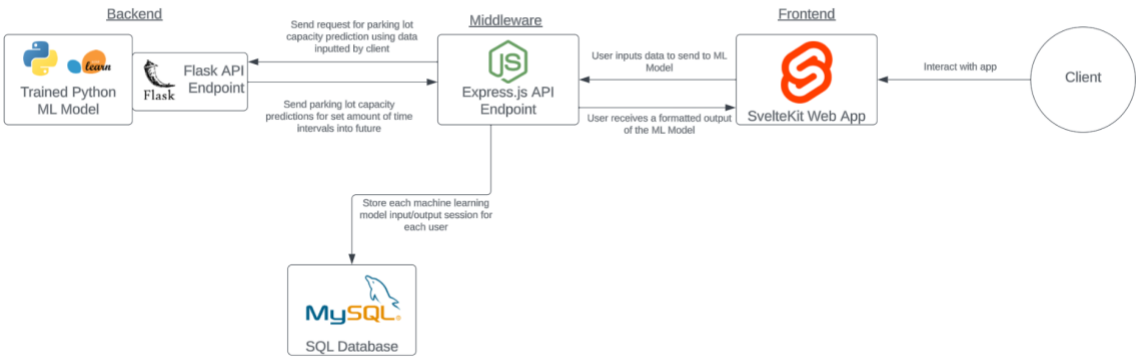
5. History and Analytics:

The application will have a history and analytics features. Users will be able to view their past parking requests and its correctness and also spot preferences. Additionally, it can highlight patterns, such as the times and places with the highest parking spot availability.

Scenario

In the proposed parking application, users have the convenience of easily entering their desired destination by inputting both the address and time into a search bar. Once this data is provided, the application immediately commences its spot prediction mechanism and offers relevant parking suggestions. To streamline the user's decision-making process, the application showcases an intuitive color-coded map: green indicating a high probability of available spots, yellow pointing to moderate availability, and red signaling low to no available spaces. This vibrant visual aid empowers users to swiftly gauge and choose the most suitable parking options, optimizing their overall experience.

Design



Front-End

The Svelte web application will have 3 key features: a map, a model performance page, and a user management page.

The routable map will be part of the user input page where the user will be able to input their departure time to send to the machine learning model along with an option to choose a preferred garage. The map itself will display the current predicted garage capacities if the user were to leave now; the predictions on the map will update regularly by sending requests to the machine learning model in set intervals. Once a user inputs and sends their departure time, the page resulting afterwards will display the predicted capacities for each garage at the user's time of expected arrival along with a garage recommendation.

The model performance page will feature usage statistics, usage history, and the model accuracy. The usage history will be retrieved from the database where the user sessions are stored for each use of the model.

Middleware

The middleware for our tech stack is run on Express.js. This middleware stores all the API endpoints that the frontend will utilize to make all of its requests: Calling the ML model, and storing the usage session.

Backend

31	32	33	34	35	40	45	50	55	60	65
286	257	279	221	305	-	-	-	-	-	-

Example of input and output data

The machine learning model is trained using time series forecasting with a sliding window. As seen in the figure above, the area shaded green would represent the sliding window, where it would take the capacity data of the past 5 minutes and train to learn the succeeding capacities in the future 30 minutes in 5-minute intervals. The expected usage of the model is as follows: it receives an arrival time and then produces predictions for 6 intervals in the future. The trained machine learning model will be directly accessible from a Flask endpoint which will be called by the Express.js middleware.

Details of the model:

```
model = Sequential([
    LSTM(50, activation='relu', input_shape=(X.shape[1], X.shape[2])),
    Dense(forecast_horizon)
])
```

- Long Short-Term Memory (LSTM) is a recurrent neural network (RNN) architecture. It's used in deep learning for processing, predicting, and classifying time-series data. LSTM

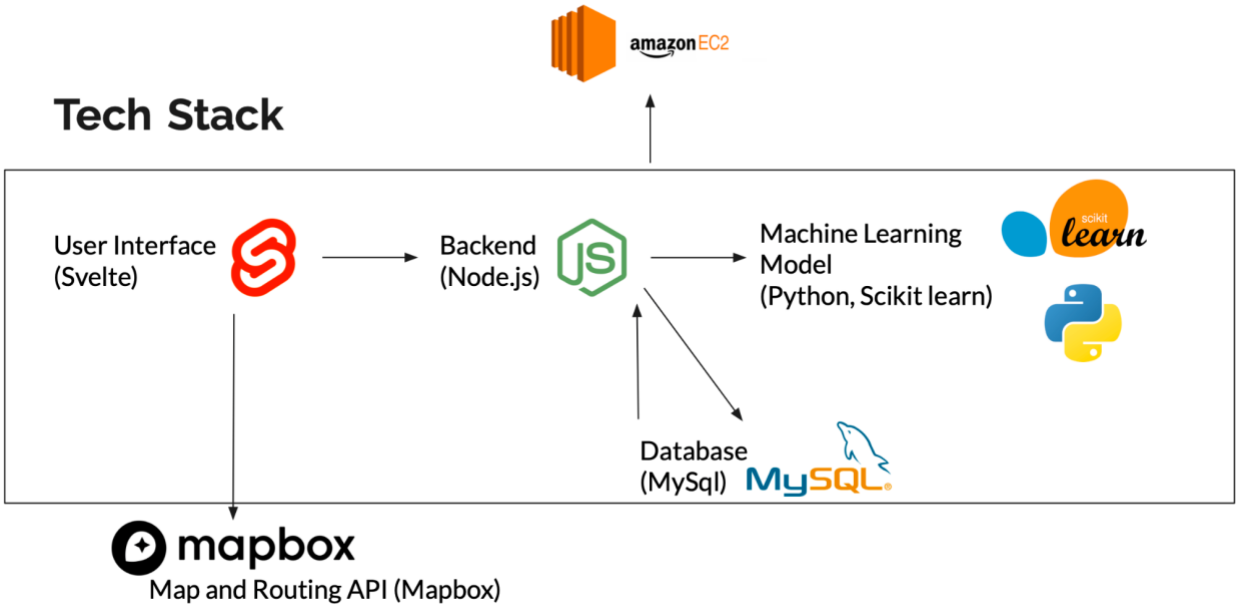
is good at capturing long-term dependencies, which makes it ideal for sequence prediction tasks.

- Sequential Model: The code initializes a Sequential mode, which allows stacking layers linearly.
- LSTM Layer: This layer will learn from sequences of data, capturing temporal patterns. The 'relu' activation function introduces non-linearity to the model, helping it learn more complex patterns.
- Dense Layer: This layer's purpose is to output the predicted values. Each neuron in this layer will correspond to a one-time step in the future that the model is predicting.
- The code uses TimeSeriesSplit from Scikit-learn to perform time-based cross-validation. This method is particularly useful for time series data, ensuring that the validation sets are always ahead in time compared to the training sets.
- The loss function is 'mean_absolute_error', which measures the average magnitude of errors in a set of predictions, without considering their direction.
- The LSTM model is designed to learn from historical sequences of data (sequence_length parameter) and predict the next few points in the series (forecast_horizon parameter). The sliding window approach (implemented via the step_size parameter) moves across the data, creating input-output pairs for the model to learn from.

```python
def interpolate_prediction(predictions, future_time):
    """
    Interpolate to get a prediction at a specific future time.

    Parameters:
    - predictions: A list of predicted values.
    - future_time: The time in the future (in minutes) for which the prediction is needed.

    Returns:
    - interpolated_value: The interpolated prediction value for the specified future time.
    """
    # Define the times corresponding to the predictions (e.g., 5, 10, 15, ... minutes into the future)
    future_times = [5, 10, 15, 20, 25, 30]

    # Ensure the predictions list is flattened (in case it's nested)
    flattened_predictions = [val for sublist in predictions for val in sublist]

    # Create the interpolator function
    interpolator = interp1d(
        future_times, flattened_predictions, kind='linear', fill_value='extrapolate')

    # Interpolate to get the prediction for the specified future time
    # Convert to native Python type
    interpolated_value = interpolator(future_time).item()

    return interpolated_value
```

- The function interpolate_prediction interpolates the predictions from the LSTM model to estimate the occupancy at a specific future time that does not align with the model's output timesteps.

Implementation

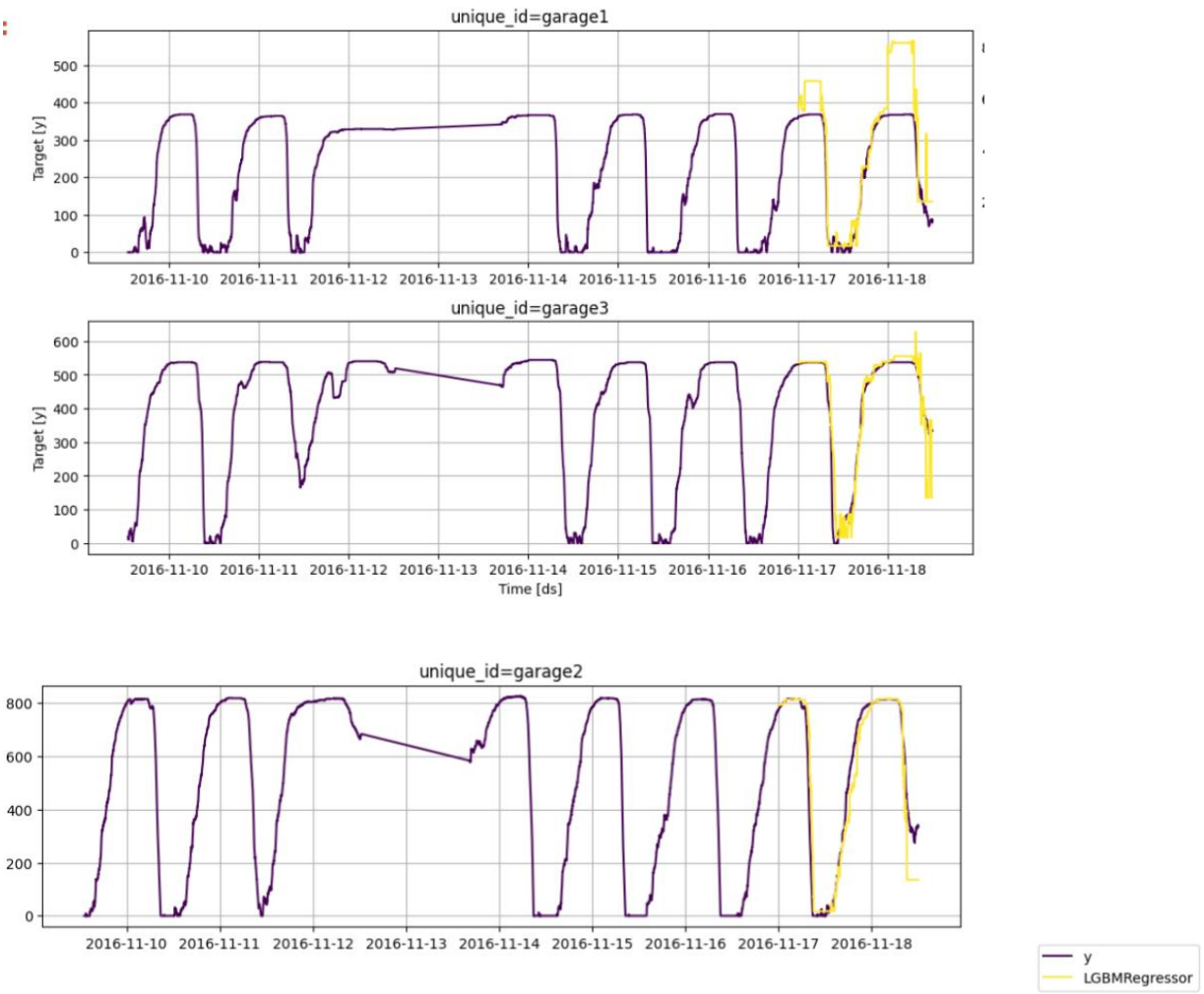


For the implementation of our project we focused on a microservice design. As seen from the design portion it can be seen that each part has its own specific function. The frontend, built with Svelte, is cleanly segregated into distinct features. These include an interactive map, a model performance dashboard, and user management page. A MySQL database handles the user credentials and sessions. The middleware is powered by Express.js and acts as the orchestration layer efficiently managing API calls for machine learning model interaction and session storage. This separated design enables the frontend to remain lightweight and focused on user experience. The backend is hosted on its own Flask server which operates independently aligning with our microserviced design. Overall we implemented our architecture in this way to ensure a clear division of responsibilities across the system. Ensuring each component focuses on a specific functionality and role. This helps to facilitate the concept of separation of concerns. This makes for better organization as it is easier to find a specific part of the application we would want to improve independent of other parts. Instead of going through a disorganized messy application. This helps for easier debugging, better scalability and updating, and aligns with modern web application development standards.

Testing/Evaluation/Assessment

First iteration using LGBMRegressor

```
models = [LGBMRegressor(random_state=0, n_estimators=100)]

model = MLForecast(models=models,
                   freq='T',
                   lags=[1440],
                   date_features=['dayofweek'],
                   )
```



In the first iteration, we used the LGBMRegressor model to predict future values. The model took in all of the data and only predicted values from the end of the dataset forward. Because this is not the input and output our client wanted, we did not continue forward with this iteration.

We then tried a few different approaches before ending up with our current model iteration.

The current iteration using LSTM

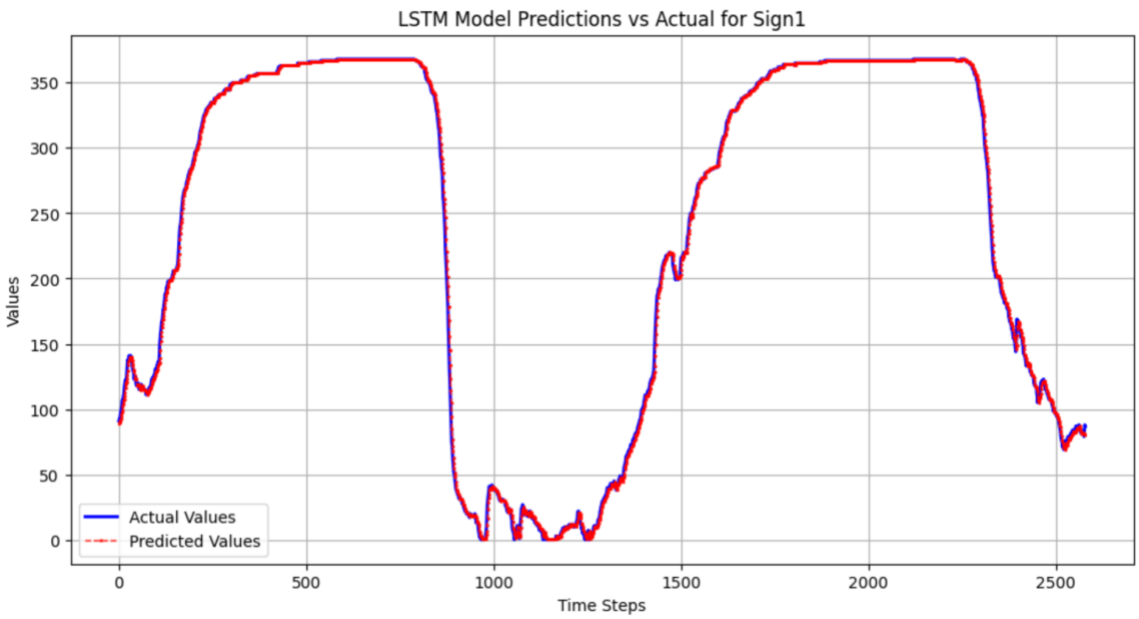
Data processing

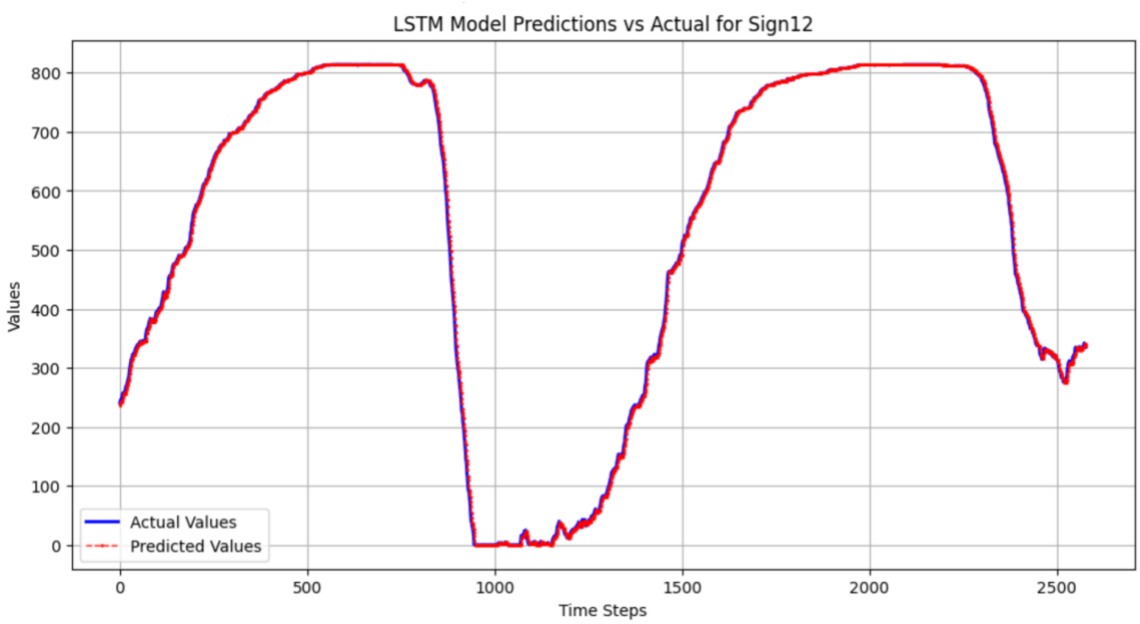
```
def preprocess_data(df, target_column, scaler,
sequence_length=5, forecast_horizon=6, step_size=5)
```

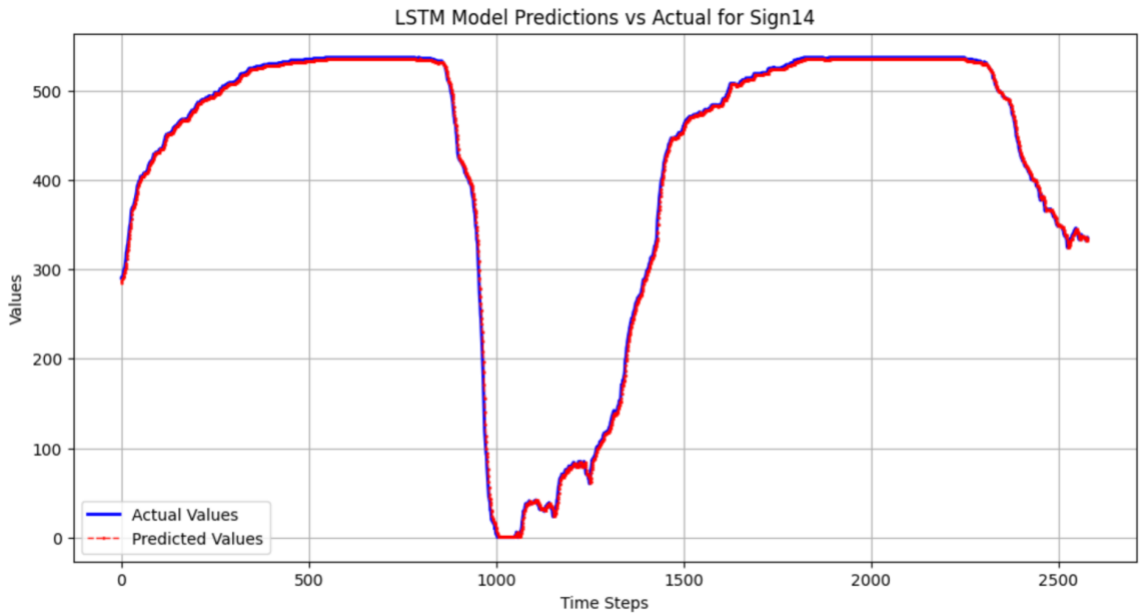
The preprocess_data function turns the time series dataset into a supervised learning problem, where the LSTM model can learn to predict the future values (y) based on sequences of past observations (X)

This is how the function works and how it transforms the data. The scaler turns data to values from 0 to 1 for easy processing using MinMax (instead of 0 to the maximum occupancy of the garage). It then extracts the day of the week and time of the day as features. Next, it creates sequences of past observations (5) and the corresponding future target values (6). Finally, the lists X and y are converted into numpy arrays to be used as inputs and targets for the LSTM model.

Testing/Evaluation/Assessment







- As you can see, the model works quite well in predicting future values. The following is the code used to test the model:

```
import pandas as pd

from sklearn.preprocessing import MinMaxScaler

from sklearn.model_selection import train_test_split

from keras.models import Sequential

from keras.layers import LSTM, Dense

import numpy as np

import matplotlib.pyplot as plt


def prepare_data(df, feature_column, look_back=1):

    # Convert timestamp to datetime and set as index

    df['ts'] = pd.to_datetime(df['ts'])

    df.set_index('ts', inplace=True)

    # Normalize the feature values
```

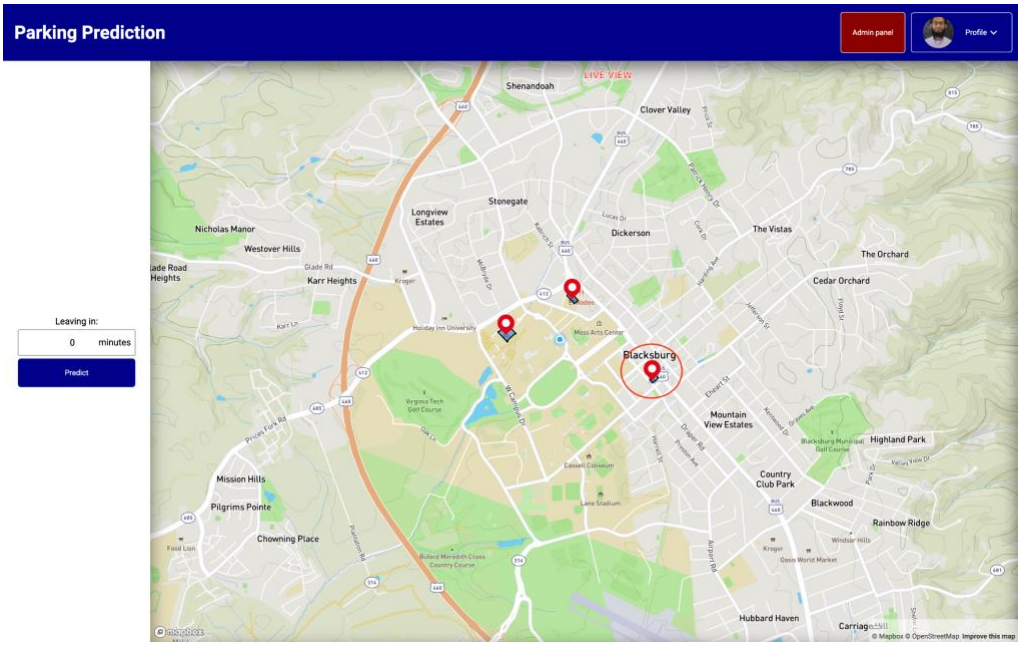
```
    scaler = MinMaxScaler(feature_range=(0, 1))
    df_scaled = scaler.fit_transform(df[[feature_column]])
    # Prepare the dataset for LSTM
    X, y = [], []
    for i in range(len(df_scaled) - look_back):
        X.append(df_scaled[i:(i + look_back), 0])
        y.append(df_scaled[i + look_back, 0])
    X = np.array(X)
    X = np.reshape(X, (X.shape[0], X.shape[1], 1))
    y = np.array(y)
    return X, y, scaler
# Example of preparing one of the datasets (you can loop this for all
datasets)
df = pd.read_csv('data/Sign14_full_fitted.csv') # Replace with actual file
path
feature_column = 'y14' # Replace with actual feature column name
X, y, scaler = prepare_data(df, feature_column, look_back=3)
# Splitting the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
shuffle=False)
# Building the LSTM model (this part should be run where TensorFlow is
installed)
model = Sequential()
model.add(LSTM(units=50, return_sequences=True,
```

```
input_shape=(X_train.shape[1], 1)))
model.add(LSTM(units=50))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mean_squared_error')
from keras.callbacks import EarlyStopping
# Early stopping
early_stop = EarlyStopping(monitor='val_loss', patience=5)
# Training the model with a larger batch size and fewer epochs
model.fit(X_train, y_train, validation_split=0.2, epochs=50, batch_size=64,
callbacks=[early_stop])
# Making predictions (to be continued in your environment)
predicted = model.predict(X_test)
predicted = scaler.inverse_transform(predicted) # To get back to original
scale
# You can then plot the results using matplotlib or any other suitable
library
def plot_predictions(actual, predicted, title):
    plt.figure(figsize=(12, 6))
    plt.plot(actual, label='Actual Values', color='blue', linewidth=2)
    plt.plot(predicted, label='Predicted Values', color='red', linestyle='-
-', linewidth=1, marker='o', markersize=1)
    plt.title(title)
    plt.xlabel('Time Steps')
    plt.ylabel('Values')
```

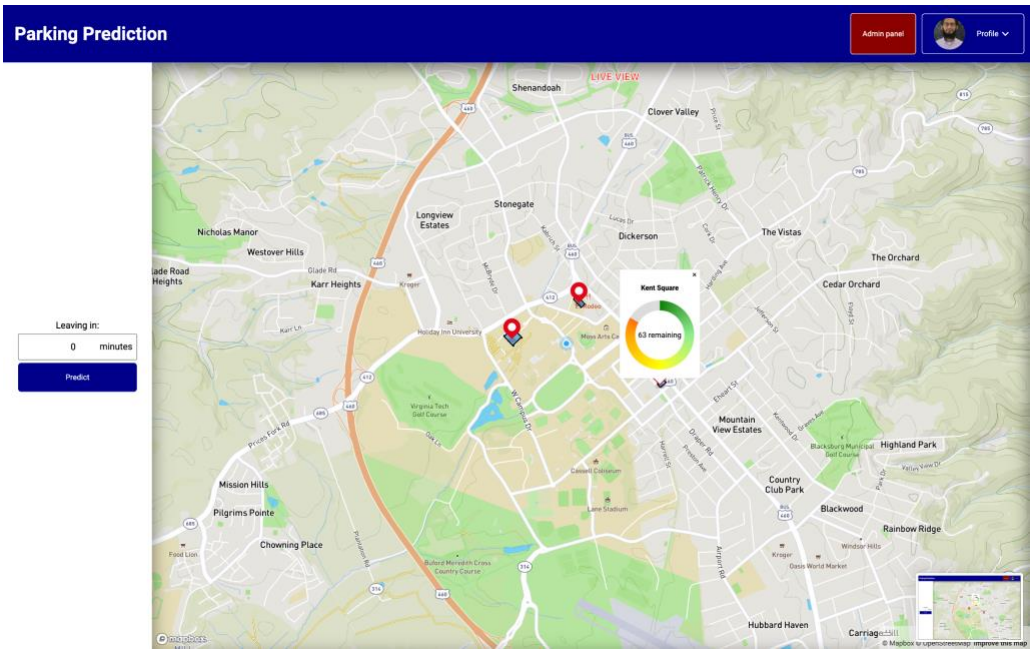
```
    plt.legend()
    plt.grid(True)  # Adding a grid for better readability
    plt.show()
# Assuming y_test_rescaled and predicted are your actual and predicted values
# y_test_rescaled should be the actual values from the test set, rescaled back to their original range
# predicted is the output from the model, also rescaled back to the original range
# Assuming y_test needs rescaling to compare with predictions
y_test_rescaled = scaler.inverse_transform(y_test.reshape(-1, 1))
plot_predictions(y_test_rescaled, predicted, 'LSTM Model Predictions vs Actual for Sign12')# replace with actual name
```

Users' Manual

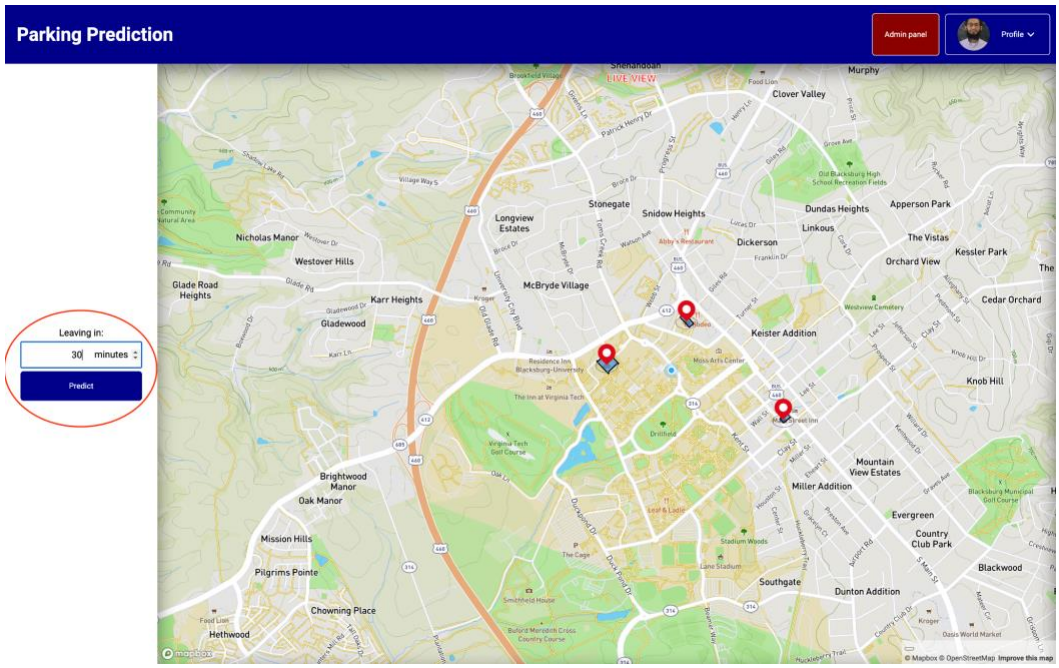
1. To preview the capacity of a garage at current time, click on a marker for the garage on the map.



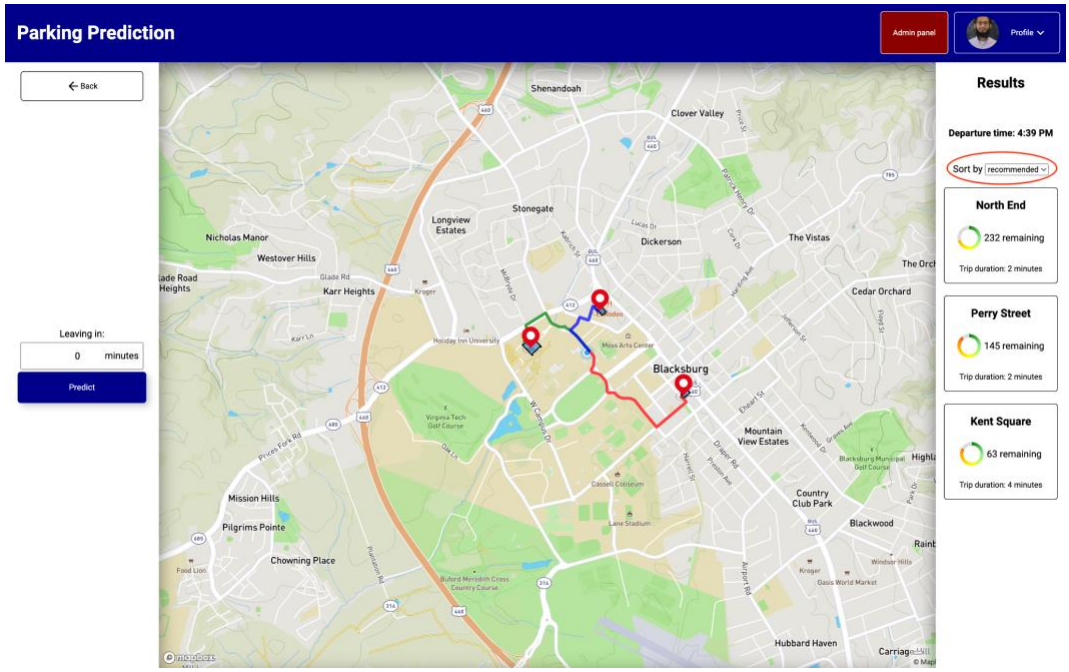
2. A popup will appear above the marker, displaying a circle that indicates capacity

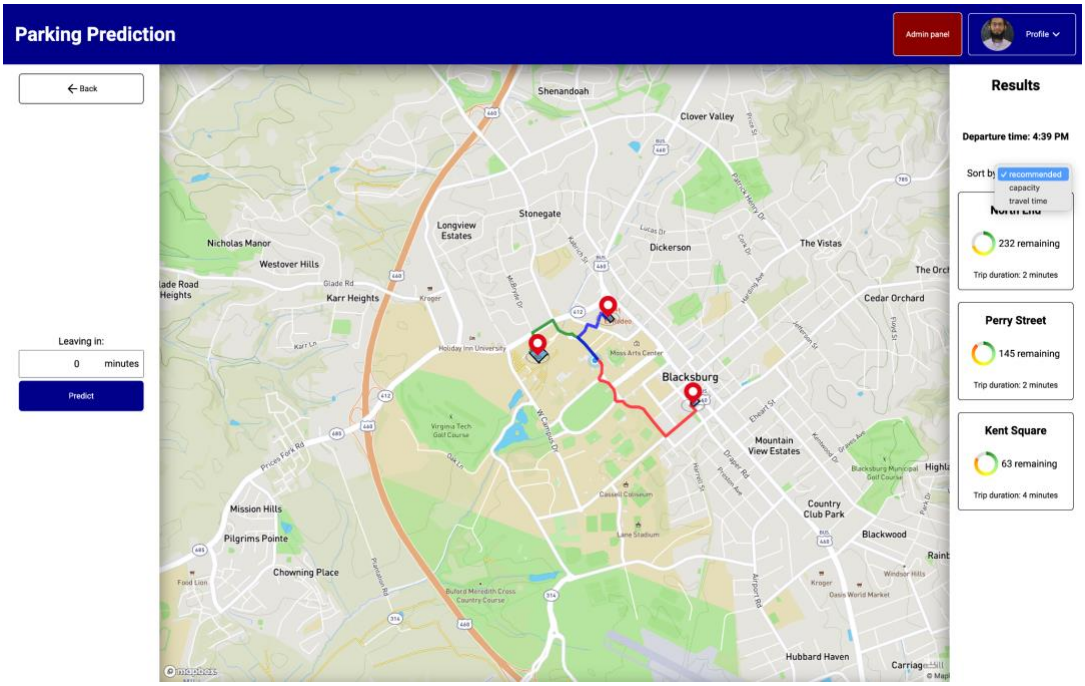


3. Click on the “Predict” button to show the capacities for all garages when leaving in a specified amount of minutes depending on the “minutes” input box.

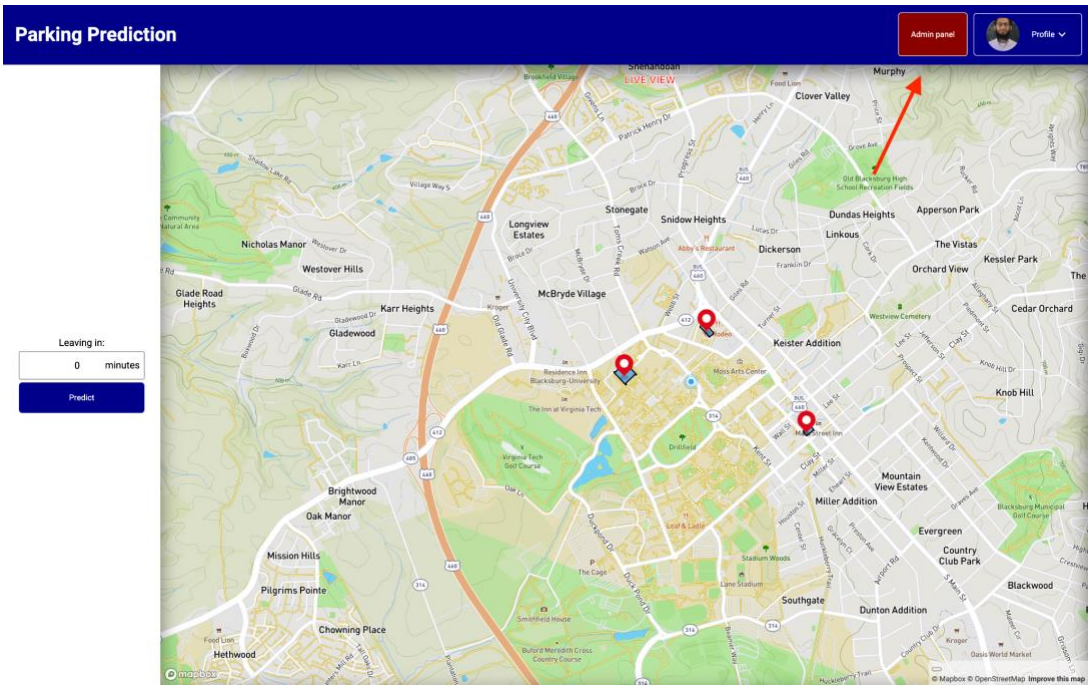


4. You can view the results from the machine learning model on the right side of the screen. Click on the dropdown indicated by “Sort by” near the top of the results to change the sorting order.

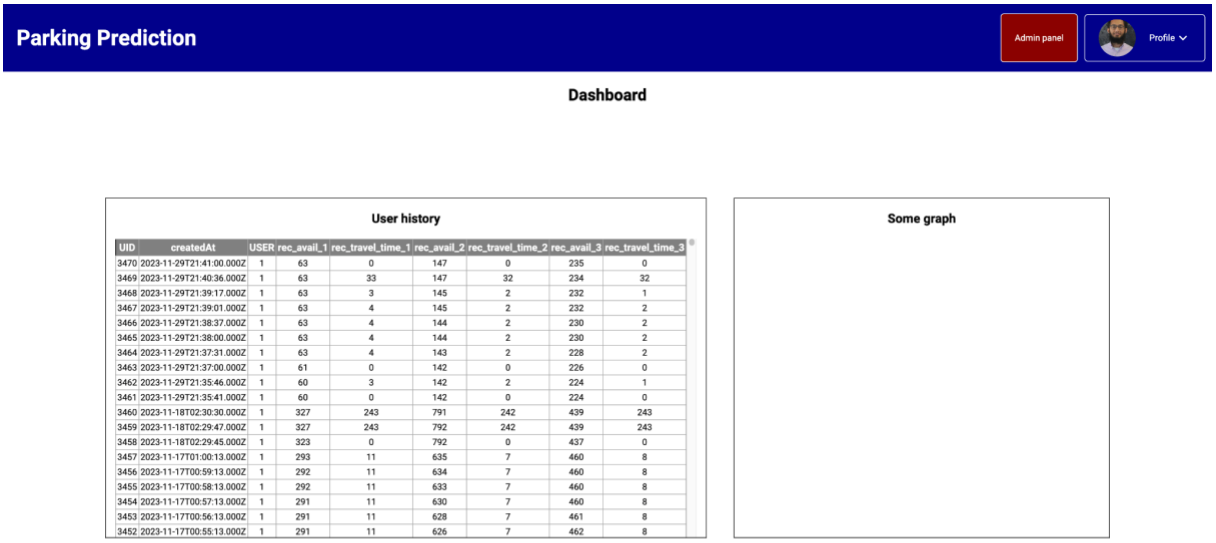




5. If you are an Admin, you will see a red button that says “Admin Panel” on the top right of the screen. Click it to navigate to the Admin Panel.



6. On the Admin Panel, you will see a table showing all the past user sessions.



UID	createdAt	USER	rec_avail_1	rec_travel_time_1	rec_avail_2	rec_travel_time_2	rec_avail_3	rec_travel_time_3
3470	2023-11-29T21:41:00.000Z	1	63	0	147	0	235	0
3469	2023-11-29T21:40:36.000Z	1	63	33	147	32	234	32
3468	2023-11-29T21:39:17.000Z	1	63	3	145	2	232	1
3467	2023-11-29T21:39:01.000Z	1	63	4	145	2	232	2
3466	2023-11-29T21:38:37.000Z	1	63	4	144	2	230	2
3465	2023-11-29T21:38:00.000Z	1	63	4	144	2	230	2
3464	2023-11-29T21:37:31.000Z	1	63	4	143	2	228	2
3463	2023-11-29T21:37:00.000Z	1	61	0	142	0	226	0
3462	2023-11-29T21:35:46.000Z	1	60	3	142	2	224	1
3461	2023-11-29T21:35:41.000Z	1	60	0	142	0	224	0
3460	2023-11-18T02:30:30.000Z	1	327	243	791	242	439	243
3459	2023-11-18T02:29:47.000Z	1	327	243	792	242	439	243
3458	2023-11-18T02:29:45.000Z	1	323	0	792	0	437	0
3457	2023-11-17T01:00:13.000Z	1	293	11	635	7	460	8
3456	2023-11-17T00:59:13.000Z	1	292	11	634	7	460	8
3455	2023-11-17T00:58:13.000Z	1	292	11	633	7	460	8
3454	2023-11-17T00:57:13.000Z	1	291	11	630	7	460	8
3453	2023-11-17T00:56:13.000Z	1	291	11	628	7	461	8
3452	2023-11-17T00:55:13.000Z	1	291	11	626	7	462	8

7. Click on the application title on the top left “Parking Prediction” to go back to the map view.

Developer's Manual

Important Information

The three components of the application are contained in the directories **front-end, ml-model,** and **persistence**.

To run the application locally the **front-end** and **persistence** directories are needed. The **ml-model** directory contains the python notebook to train and save ML models and create relevant scalers. The current models and scalers are already saved within the **persistence** directory so this can be ignored during deployment, unless any changes to the model need to be made.

To retrain models or to modify models, change `ml-model/PLAYGROUND/prediction.ipynb` as needed and rerun notebook. Then copy `ml-model/PLAYGROUND/model_data` to `persistence/ml-model/model_data` and copy `ml-model/PLAYGROUND/scaler_data` to `persistence/ml-model/scaler_data`.

Project currently does not use live data. Instead pseudo-live data is used. This involves providing the ml model with historical data in lieu of live data by matching the day of the week and time of day. When any request is made, the current day of the week and time of day is determined. Then, a data point in the data set with the same day of the week and time of the day is used as input for the model. In the future, real live data can be provided and the model should function as intended.

Running Project

Firstly, make sure that **Docker** (Docker Engine and Docker Compose), **Node.js,** and **Python** are installed.

Next clone repository locally:

https://github.com/eugene0628/parking-predictor

Next, change directory into persistence

```
cd persistence
```

Ensure there is a *.env* file with

```
DATABASE_URL="mysql://myuser:mypassword@localhost:3306/mydb"
```

Then, install all necessary node dependencies

```
npm i
```

Similarly, install all necessary flask dependencies

```
npm run i_flask
```

After ensuring Docker Daemon is running, build the docker image.

```
docker build
```

Then, start docker image

```
npm run persistence
```

Apply database schema to new mysql database

```
npm run migrate
```

To create default user for database run

```
npm run sample_user
```

In a separate CLI window, start the flask server

```
npm run flask
```

Finally, in a separate CLI window, start the express server

```
npm run serve
```

Next, build and deploy the **front-end**.

Change directory into **front-end**

```
cd front-end
```

Ensure there is a *.env* file with

```
SECRET_NODE_LINK="http://localhost:3000"
```

Then, install all necessary node dependencies

```
npm i –force
```

Run the front-end with

```
npm run dev
```

Important Technologies Used:

npm-liquibase database migration tool, used to create tables initially

express node.js server, used to serve front-end requests

npm-prisma js query builder, used within endpoints to perform CRUD on database.

Accessing Database

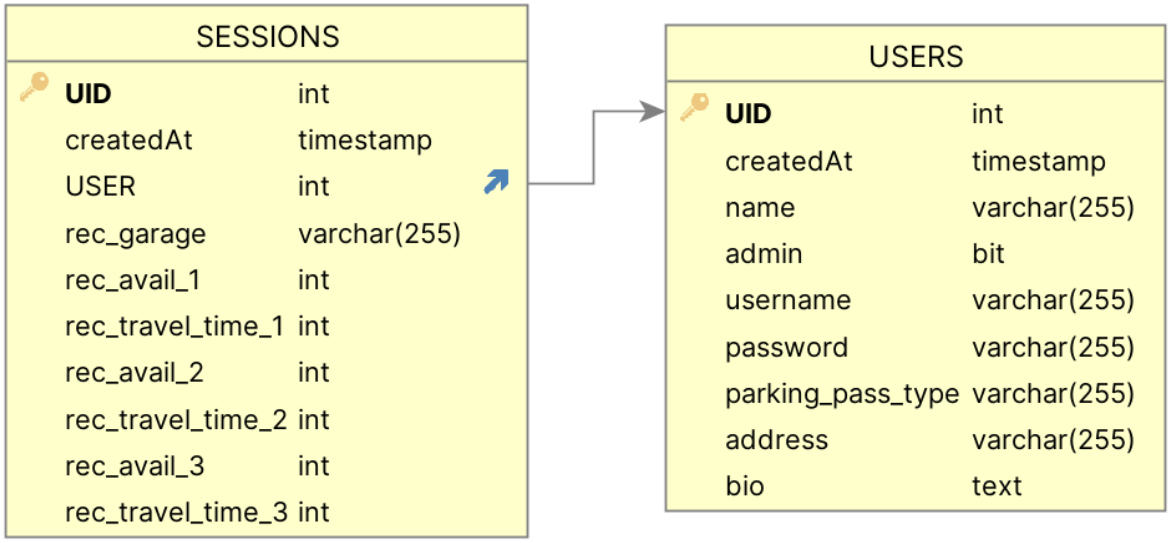
Using a database interfacing tool, the database can be viewed and modified for testing purposes.

DBVisualizer is the recommended tool. The connection specification is as follows.

Server Info
MySQL
MySQL 8
localhost
3306
mydb

The credentials are listed in *parking-predictor/persistence/docker-compose.yaml* under *MYSQL_USER* and *MYSQL_PASSWORD* and can be modified if needed. Similarly database ports and container ports can be specified under *ports*.

Database Schema



The project contains two main data sets: Users and Sessions. As users are created their info will be persisted in the database. Each user can create “sessions” that contain travel time to each of the garages and the ML predicted parking spot number at the time of arrival. The database

selected is the relational database **MySQL**.

If the schema needs to be modified, it should be modified in `parking-predictor/persistence/liquibase/changelog.yaml` according to Liquibase syntax. Then the `persistence/prisma/schema.prisma` should be updated as well. Run

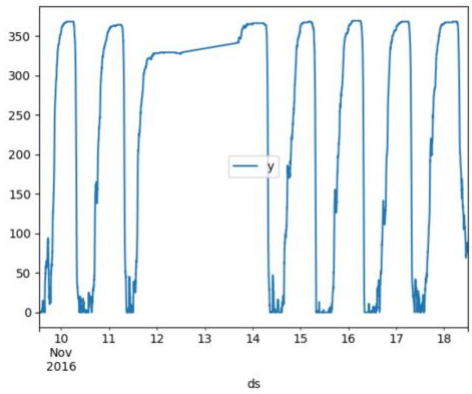
```
npx prisma generate
```

To generate new prisma schema.

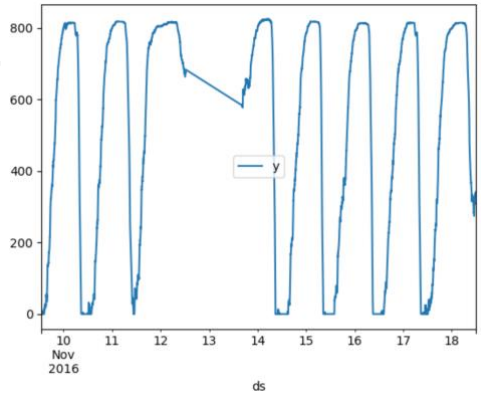
Data Provided

The client provided us with 3 .csv files. The data is from 3 different garages over the course of 10 days. It captures how many parking lot spaces are available each minute. Below is the data after it's been loaded.

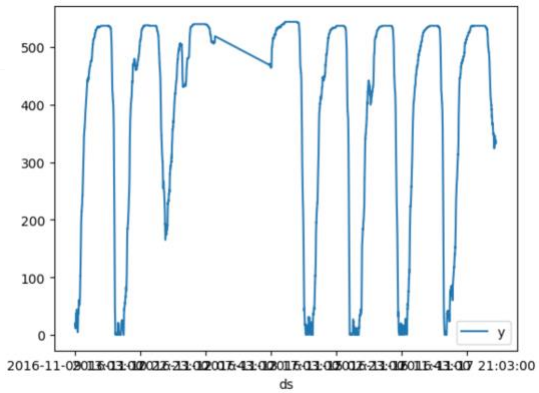
```
                      ds     y unique_id
0     2016-11-09 13:03:00   0.0   garage1
1     2016-11-09 13:04:00   0.0   garage1
2     2016-11-09 13:05:00   0.0   garage1
3     2016-11-09 13:06:00   0.0   garage1
4     2016-11-09 13:07:00   0.0   garage1
...                   ...   ...       ...
12892 2016-11-18 11:55:00  82.0   garage1
12893 2016-11-18 11:56:00  79.0   garage1
12894 2016-11-18 11:57:00  84.0   garage1
12895 2016-11-18 11:58:00  88.0   garage1
12896 2016-11-18 11:59:00  87.0   garage1

[12897 rows x 3 columns]
```



```
                     ds      y unique_id
0     2016-11-09 13:03:00    0.0   garage2
1     2016-11-09 13:04:00    0.0   garage2
2     2016-11-09 13:05:00    0.0   garage2
3     2016-11-09 13:06:00    0.0   garage2
4     2016-11-09 13:07:00    0.0   garage2
...                   ...    ...       ...
12892 2016-11-18 11:55:00  340.0   garage2
12893 2016-11-18 11:56:00  340.0   garage2
12894 2016-11-18 11:57:00  336.0   garage2
12895 2016-11-18 11:58:00  334.0   garage2
12896 2016-11-18 11:59:00  335.0   garage2

[12897 rows x 3 columns]
```




```
                     ds      y unique_id
0     2016-11-09 13:03:00   20.0   garage3
1     2016-11-09 13:04:00   20.0   garage3
2     2016-11-09 13:05:00   20.0   garage3
3     2016-11-09 13:06:00   16.0   garage3
4     2016-11-09 13:07:00   16.0   garage3
...                   ...    ...       ...
12892 2016-11-18 11:55:00  333.0   garage3
12893 2016-11-18 11:56:00  334.0   garage3
12894 2016-11-18 11:57:00  335.0   garage3
12895 2016-11-18 11:58:00  336.0   garage3
12896 2016-11-18 11:59:00  335.0   garage3

[12897 rows x 3 columns]
```



Libraries used for machine learning model

- pandas
- numpy
- matplotlib
- Sklearn
- keras

Lessons Learned

During the project our team gained valuable experience and insights that influenced the development of the project. Our team regularly held meetings throughout the semester with the client to ensure that the clients' needs were being met. The team also coordinated with each other frequently to discuss progress and address any challenges or problems that occurred during the development of the project. This process ensured that the group was able to address any problems encountered during the project frequently and find the best solution suited to the project's needs.

Timeline

Weekly Timeframe	Brief Task Overview
Weeks 1 & 2 (9/26/2023 - 10/10/2023)	Initial project set up including preliminary database, basic front end, express server, and jupyter notebook creation.
Weeks 3 & 4 (10/10/2023 - 10/24/2023)	Reorganization of data in jupyter notebook. Updates to database schema. Creation of server endpoints and integration of flask with express server. Created Mapbox view for frontend.
Weeks 5 & 6 (10/24/2023 - 11/07/2023)	Adjustments to predictions in ML model. Estimated travel time and progress bar displayed in frontend. Added authentication using passport and express session.
Weeks 8 & 9 (11/07/2023 - 11/21/2023)	Redesign of frontend and UI updates. Created sample predict function. Demo setup.
Final Week (11/21/2023 - 11/28/2023)	Demo walkthrough, documentation, final testing.

Problems & Solutions

Accuracy of ML model

Ensuring accurate prediction times was a concern throughout the development of the project. It was crucial that the model would have the capability to predict accurate times to the user. The group's solution in order to minimize any inaccuracies were to regularly update hyperparameters, reorganize data, and test different features to ensure the prediction capability of the model. This iterative approach helped us create an accurate prediction model and test features for the models capabilities.

Database Interface and Schema Design

The discussion between using liquidbase or prisma to interface with the database was an issue encountered during the development of the project. From a developmental approach the group wanted to ensure that the library used would be the most optimal choice for management of the database. The solution involved experimenting with both liquibase and prisma to determine which interface was more compatible with the database. After discussion between group members it was decided that prisma would be the best choice.

Future Work

Our future plans include…

- Adapting the project to use live data from each of the three parking garages.
- Creating a more advanced ML model that will ensure accurate predictions.
- Authentication and UI improvements.
- Implementing a feature that allows the school to add event data which could impact parking availability.

Acknowledgments

Client

- Dr. Mohamed Farag
    - Research associate in the Center for Sustainable Mobility
    - His research interests include intelligent transportation systems, connected/automated vehicles, C-V2X, machine learning, large-scale data analysis, large-scale system analysis and design, big data, and information retrieval.
- Weekly online meetings to update and approve progress
- Provides training data

References

Farag, M., Hilal, A., El-Tawab, S. (2019). Parking Occupancy Prediction and Traffic Assignment in a University Environment. In 2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC). https://doi.org/10.1109/CCWC.2019.8666493

Filho, Mario. "Multiple Time Series Forecasting with Lightgbm in Python." *Forecastegy (Alt + H)*, 28 Feb. 2023, forecastegy.com/posts/multiple-time-series-forecasting-with-lightgbm-in-python/.