SDOT Parking Maps

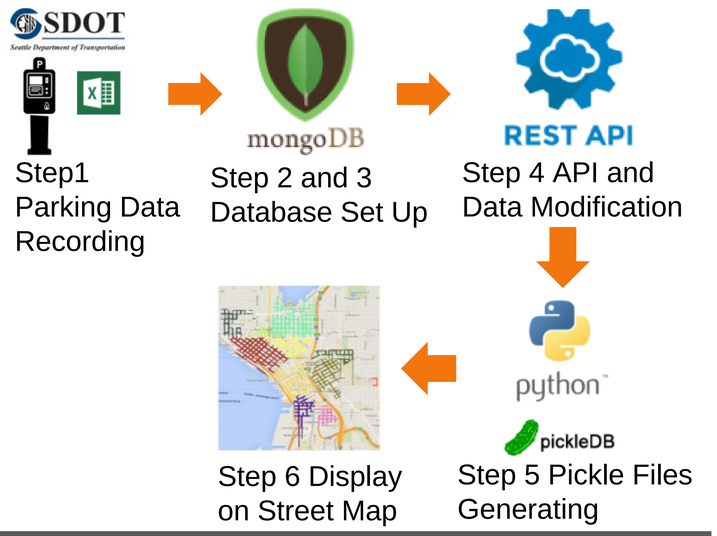
presented by UNIVERSITY OF WASHINGTON ELECTRICAL ENGINEERING & SDOT Seattle Department of Transportation

Robert Yody, Heng Zhou, Pyi Khine | Lily Ratliff | University of Washington

Project Overview

Seattle Department of Transportation tracks the details of each of the paid parking transactions which occur within the Seattle Metro area. Our goal was to harness this abundance of data and develop insightful analytics which could be utilized by SDOT, 3rd party companies, and Seattle residents to improve the paid parking experience. Our response was to create an API which analyzes the paid transactions which occur on a particular block for a particular day. The API then returns a value for each hourly interval of the day. These values can be used to forecast vacancy on future dates.

Project Procedure



Step1 Parking Data Recording

Step 2 and 3 Database Set Up

Step 4 API and Data Modification

Step 5 Pickle Files Generating

Step 6 Display on Street Map

Database

- **Step 1**
  1. Access SDOT parking data using Python code, DataAccess.py
  2. The code will produce a .csv file.
- **Step 2**
  1. Establish MongoDB for the SDOT project.
- **Step 3**
  1. Convert datetimes in the .csv from a string to a Gregorian value
  2. Upload the updated .csv file into MongoDB as a 'collection'.

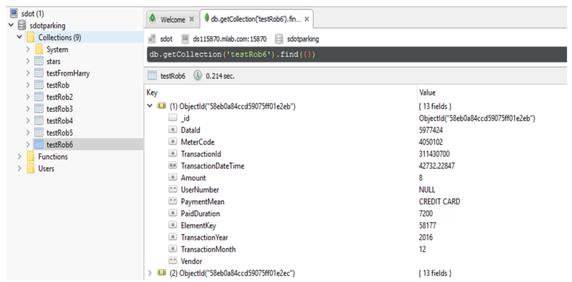


Figure 1. Parking Data on MongoDB in the collection

API and Calculation

- Application Programming Interface, or API, is a set of protocols and tools for building application software. API requests from system, receives response and runs the local software based on data received. Below are how we create API and correlative algorithm.
- **Step 4**
  1. Access MongoDB and run an API server using Python code, mongo.py
  2. Produce pickle files (.p) using Python code, gettingDataFromAPI.py

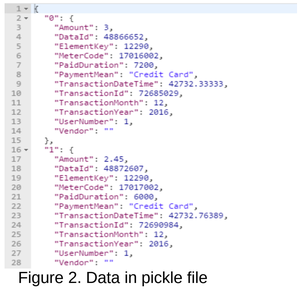


Figure 2. Data in pickle file

- **Step 5**
  1. Load .p files from PC with Python code, CalculationSampleV7_B.py
  2. Perform Boolean operations and calculations upon the .p files to produce fractions (part of CalculationSampleV7_B.py's functionality)

```
# The Hour of Interest in this example is 11:00AM - 12:00PM.
PDtotal = 0
# Starting with the FIRST HOUR BLOCK, the first iteration is:
    if  11:00 <= TransactionDateTime < 11:01
        if PaidDuration <= 60 mins
            PDtotal = PDtotal + PaidDuration
        elif PaidDuration > 60 mins
            PDtotal = PDtotal + 60 mins
        else
            PDtotal = PDtotal
    else
        PDtotal = PDtotal
# Second iteration (FIRST HOUR BLOCK)
    if 11:01 <= TransactionDateTime < 11:02
        if PaidDuration <= 59 mins
            PDtotal = PDtotal + PaidDuration
        elif PaidDuration > 59 mins
            PDtotal = PDtotal + 59 mins
        else
            PDtotal = PDtotal
    else
        PDtotal = PDtotal
# ... to 60 iterations
# Extending down to the THIRD HOUR BLOCK, The first iteration is:
    if 9:00 <= TransactionDateTime < 9:01
        if 120 mins <= PaidDuration <= 180 mins
            PDtotal = PDtotal + (PaidDuration - 120 mins)
        elif PaidDuration > 180 mins
            PDtotal = PDtotal + 60 mins
        else
            PDtotal = PDtotal
    else
        PDtotal = PDtotal
# Second iteration (THIRD HOUR BLOCK)
    if 9:01 <= TransactionDateTime < 9:02
        if 119  mins <= PaidDuration <= 179 mins
            PDtotal = PDtotal + (PaidDuration - 119 mins)
        elif PaidDuration > 179 mins
            PDtotal = PDtotal + 60 mins
        else
            PDtotal = PDtotal
    else
        PDtotal = PDtotal
# ... to 60 iterations
```

```
# Extending down to the SECOND HOUR BLOCK, The first iteration is:
    if 10:00 <= TransactionDateTime < 10:01
        if 60 mins <= PaidDuration <= 120 mins
            PDtotal = PDtotal + (PaidDuration - 60 mins)
        elif PaidDuration > 120 mins
            PDtotal = PDtotal + 60 mins
        else
            PDtotal = PDtotal
    else
        PDtotal = PDtotal
# Second iteration (SECOND HOUR BLOCK)
    if 10:01 <= TransactionDateTime < 10:02
        if 59 mins <= PaidDuration <= 119 mins
            PDtotal = PDtotal + (PaidDuration - 59 mins)
        elif PaidDuration > 119 mins
            PDtotal = PDtotal + 60 mins
        else
            PDtotal = PDtotal
    else
        PDtotal = PDtotal
# ... to 60 iterations

# After extending back 9 hours, ensuring that any transaction
# which may impact the Hour of Interest is accounted for:
print(PDtotal/ElementKeySupply)
PDtotal = 0 # So the program can do the whole thing over again for
            # each of the following Hours of Interest in the user
            # selected Day of Interest.
```

Figure 3. Pseudo Code for our Calculation Algorithm

Mapping

- Step 6

  To maximize our API, we delegated colors to represent different ranges of results, which we then overlaid on to a map of Seattle.

0-.12499 = 'gray'
.125-.2499 = 'lawngreen'
.25-.37499 = 'limegreen'
.375-.499 = 'yellow'
.5-.62499 = 'orange'
.625-.7499 = 'red'
.75-.87499 = 'maroon'
.875-1+ = 'black'

Figure 4. Color scheme relating calculation result and color display on map. Green color means more vacancy while red color means less vacancy.

- Demo 1 shows how a block's vacancy changes over a full day:

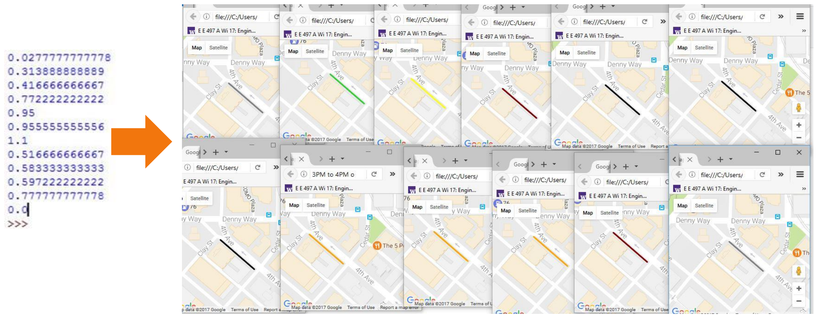


Figure 5. Results mapped for a single block over the course of Wednesday, 12/28/2016.

- Demo 2 shows the vacancy of multiple blocks during a given hour:

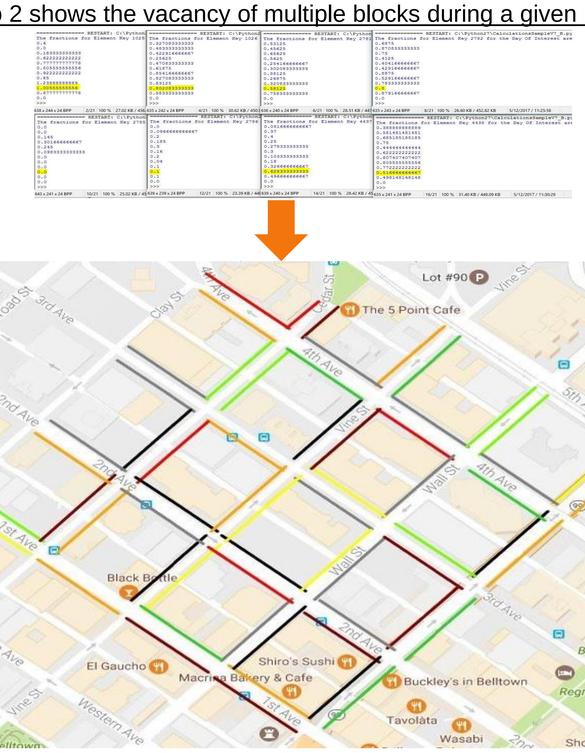


Figure 6. Results Mapped for multiple blocks from 5PM-6PM on Thursday, 12/29/2016.

Conclusion

- The API we have developed is very robust, accounting for a number of factors to produce highly accurate and useful parking vacancy forecasts for individual blocks within the Seattle Metro area. The demonstrations shown on this poster, which are raw results, represent just one of many ways that the insightful analytics produced by our API can be utilized. If a developer wants to determine average vacancy for specific weather conditions, near stadiums during sporting events, or during certain months or seasons, the results that our API produces make this incredibly easy to do. We hope that the thought, time, and effort which we have put forth on this project will be utilized by motivated professionals.

Works Cited

- Google Maps for their map of Seattle.
- Mlab.com for their MongoDB hosting platform.
- SDOT for access to their data.