Paid Parking Occupancy
Data Preparation

Created 3/22/2019

Introduction

The report gives an overview of the steps that the UW team took for data preparation to model parking occupancy at the blockface level. These steps included:

a. Defining business requirements and end goals, based on discussions with representatives from the City of Seattle IT and Seattle Department of Transportation
b. Identifying relevant data sources to answer business questions concerning rate-setting and better understanding parking behaviors
c. Loading the raw data sources into a database instance hosted in the cloud
d. Transforming the data to a format that is more suited for analytical processing
e. Integrating the data sources into a single view that can be read into Python for modelling and other analysis

1.1 Data Sources

The team's first three goals were to identify data sources that could address the many factors that might influence parking demand, focus on a single neighborhood with less variable parking behavior, and expand the statistical models employed in analysis.

The data sources identified were:

a. **Paid parking occupancy** [1] - Count of active transactions at 1 minute intervals throughout the day from mobile payment system and parking payment kiosks installed on each blockface around the city from 2017 and 2018. For our study, we pulled counts at 30 minute intervals.
b. **Annual Parking Study** [2] - Manual parking study details from citywide paid parking study, presented by blockface and date/hour of study from 2017 and 2018.
c. **Public garages/parking lots** [3] - Characteristics of off-street parking facilities (above-ground, below ground and surface lots) within the city of Seattle.
d. **Traffic Flow Map Volumes** [4] - Average annual weekday traffic volumes for selected streets used to produce the SDOT Traffic Flow Map for 2017 and 2018.
e. **Weather Seattle** [5] - Hourly weather measurements at WSU's Seattle weather station for each day of 2017 and 2018.
f. **Seattle Employment** [6] - Employment data for Seattle for each month in 2017 and 2018.
g. **Special Events Permits** [7] - Data about special events like concerts etc. that can have an effect on parking in the neighborhood.

h. **Businesses in Seattle** [8] - Business licenses in Seattle with addresses, to calculate proximity to streets on which parking might be affected during business hours.

1.2 Data Preparation

1.2.1 *Database Selection*

The team decided to host the data sources on *AWS Relational Database Service* instance. The database selected was *Postgres*. AWS RDS provides 20 GB of free storage which was considered adequate for the scope of the project, given the limited resources.

1.2.2 *Neighborhood and Timeframe Selection*

The first step to scope our business question was to limit our analysis to one specific neighborhood. The team initially planned to conduct exploratory data analysis on numerous neighborhoods to narrow down the analysis to a single one. However, due to enormous amount and format of the available data, the exploratory phase could not be conducted as it was computationally expensive. A full-scale analysis of parking data across Seattle would require greater time and resources than could be allotted for this project. However, it is reasonable for the city to consider neighborhoods individually because their parking characteristics tend to vary, according to the previous work done in this area.

The team decided to do rigorous literature review to understand the approach taken by previous researchers who conducted similar studies. In consideration of this study and Mary Catherine's notes, we selected *Belltown North* as the neighborhood of interest. The reasons we chose this were (not limited to) generous number of records, proximity to downtown, and relatively low level of unpaid parking, etc.

After choosing the neighborhood, the team selected 2017 and 2018 as the years of interest. This time frame gives the team two years of data to examine any seasonality effects and contrast the data from two years. Further, the records were extracted at 30 minute snapshot intervals. This helped the team control the size of data for loading into the database and also for analysis in Python.

1.2.3 *Loading the data in database*

The team loaded the data from different data sources into the database using various strategies as best suited for respective data sources.

- Paid parking occupancy 2017 [1] - The table was loaded using the copy command. Please refer to *Snippet 1* in the appendix for more details on how the command was configured.
- Paid parking occupancy 2018 [9] - The data was extracted by using the Socrata API offered by the Open Data Portal hosted by the City of Seattle. Please refer to *Snippet 2* in the appendix for more details on how the data was extracted using Python.
- Annual Parking Study [2] - The data was extracted from a csv file using the Python code in *Snippet 3.*

- Public garages/parking lots [3] - The data was extracted from a csv file using the Python code in *Snippet 4.*
- Traffic Flow Map Volumes [4] - The data was extracted from a csv file using the Python code in *Snippet 5.*
- Weather Seattle [5] - The data was extracted from a csv file using the Python code in *Snippet 6.*
- Seattle Employment [6] - The data was extracted from a csv file using the Python code in *Snippet 7.*
- Special Events Permits [7] - The data was extracted from csv file using the Python code in *Snippet 8.*
- Businesses in Seattle [8] - The data was extracted from a csv file using the Python code in *Snippet 9.*
- Blockface[10] - The data was extracted using the copy command.

1.2.4 *Data Cleaning*

After successfully loading the data in the database, the next phase was to transform the data in a way that would promote easy joining of the tables and feature engineering for later phases of the project.
The timestamp column was broken into separate date and time columns. These were then converted into *date* datatype for dates present in the data and *time without timezone* for all the timestamps in the data. Numerous features like day of the week, day, month, year, week of the year, hours, mins, etc., were extracted using SQL functions. Other numeric data was converted to the datatype *integer* (except latitude and longitude values, which are type *double precision*). Apart from these, missing values were replaced with nulls, and duplicate values were removed. An example of the SQL statements for the above can be found in *Snippet 10* in the appendix.

A couple of tables contained measurements for latitude and longitude. The team used PostGis extension to create a *geography* column which marks a point on the 2-D map. The results were visualized using geometry viewer in pgAdmin. The results can be viewed in *Figure 1* in the appendix, and the SQL statements can be found in *Snippet 11* in the appendix.

Business license data was used to calculate the number of businesses within a 200 meter (approximately 2 block) radius of a blockface and particular categories of business (restaurants, shopping, etc.) were manually identified using NAICS codes. SQL statements can be found in *Snippet 12* in the appendix.

1.3 Integrating Data sources

The next phase after cleaning the database was to integrate the data sources by joining the tables to each other based on certain keys and making a unified view that can be read into Python.

The team formulated an Entity Relationship Diagram that describes how different tables are connected to each other. The team used Visual Paradigm to reverse engineer the database instance to a data model. The ER diagram is described in Figure 2.

The SQL statements to achieve these joins are mentioned in *Snippet 13* in the appendix.

1.4 Resources to *City of Seattle*

This data pipeline documentation will appear in a package of resources delivered to City of Seattle upon the UW team's completion of this project. The City of Seattle can use these resources to achieve their overall business need to better understanding parking behaviors in Belltown North (during 2017 and 2018), which could influence rate-setting policies in the future. The City of Seattle can also benefit from having documentation of this project's design, in order to potentially replicate its successful outputs for other neighborhoods around the city. For example, many of the same data sources we identified could be extracted and transformed to prioritize analysis into another parking subarea. The City of Seattle can continue this work based on documentation of the processes and technical scripts executed to aggregate these data sources. We can also provide the database dump file so that the database (including schema, tables, views, grants, dependencies) can be restored to the current state easily without having to execute our scripts.

1.5 References

1. CityOfSeattleIT. (2019, March 5). *Open Data Platform*. Retrieved from data.seattle.gov: https://data.seattle.gov/Transportation/Paid-Parking-Occupancy-2017-Archived-/t96b-taja

2. CityOfSeattleIT. (2019, March 5). *Open Data Platform*. Retrieved from data.seattle.gov: https://data.seattle.gov/Transportation/Annual-Parking-Study-Data/7jzm-ucez

3. CityOfSeattleIT. (2019, March 5). *Open Data Platform*. Retrieved from data.seattle.gov: https://data.seattle.gov/Transportation/Public-Garages-or-Parking-Lots/xefx-khzm

4. CityOfSeattleIT. (2019, March 5). *Open Data Platform*. Retrieved from data.seattle.gov: https://data.seattle.gov/Transportation/Traffic-Flow-Map-Volumes/38vd-gytv

5. WSU. (2019, March). *AgWeatherNet*. Retrieved from weather.wsu.edu: http://weather.wsu.edu/index.php?p=92950

6. Bureau of Labor Statistics. (2019, March 15). *Local Area Unemployment Statistics: Seattle-Tacoma-Bellevue.* Retrieved from data.bls.gov: https://data.bls.gov/timeseries/LAUMT534266000000005?amp%253bdata_tool=XGtable&output_view=data&include_graphs=true

7. CityOfSeattleIT. (2019, March 5). *Open Data Platform*. Retrieved from data.seattle.gov: https://data.seattle.gov/Permitting/Special-Events-Permits/dm95-f8w5

8. CityOfSeattleIT. (2019, March 5). *Open Data Platform*. Retrieved from data.seattle.gov: https://data.seattle.gov/City-Business/Active-Business-License-Tax-Certificate/wnbq-64tb/data

9. CityOfSeattleIT. (2019, March 5). *Open Data Platform*. Retrieved from data.seattle.gov: https://data.seattle.gov/Transportation/2018-Paid-Parking-Occupancy-Year-to-date-/6yaw-2m8q

10. City of SeattleIT. (2019, March 5). *Open Data Platform.* Retrieved from https://data.seattle.gov/Transportation/Blockface/dmbn-t6rz

1.6 Appendix

1.6.1 *Figures*

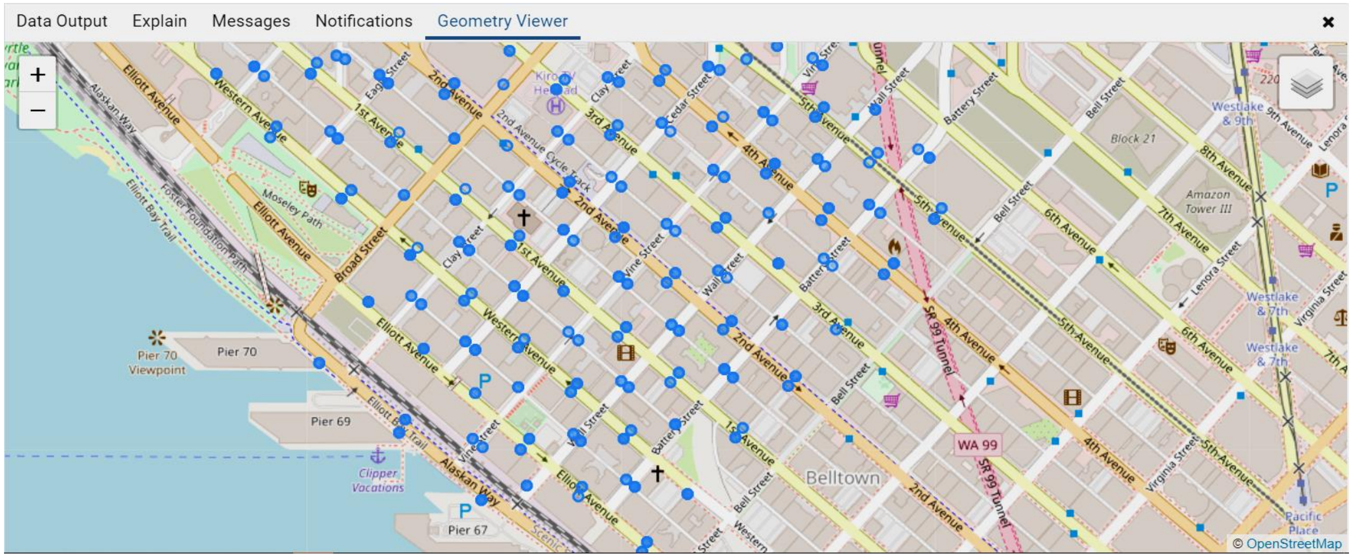

Figure 1: Parking transaction kiosks installed on Belltown North streets in Seattle

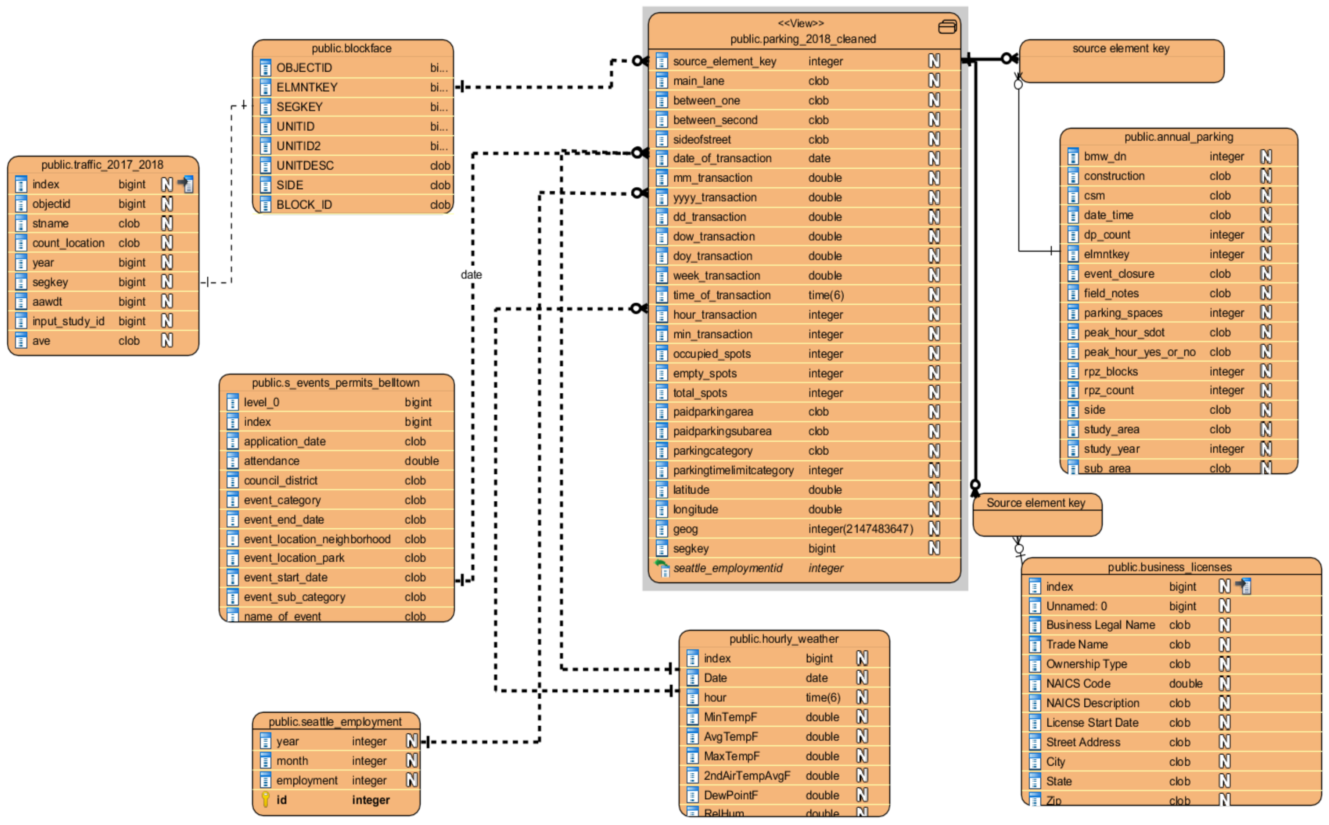

Figure 2: Entity Relationship Diagram of the Database Schema

1.6.2 *Code Snippets*

[1]

<u>Command Line</u>

```
zipgrep "Belltown" 2017.zip>>1.csv

grep "North" 1.csv>>2.csv

grep "00:00" 2.csv >>3_0.csv

grep "30:00" 2.csv >> 3_30.csv (edited)
psql -h rds-trafficjam.cpze98eqmmci.us-east-2.rds.amazonaws.com -U trafficjam -d dbtrafficjam -c "\copy parking_belltown_n_2017 from D:\3_30.csv DELIMITER ','"

psql -h rds-trafficjam.cpze98eqmmci.us-east-2.rds.amazonaws.com -U trafficjam -d dbtrafficjam -c "\copy parking_belltown_n_2017 from D:\3_0.csv DELIMITER ','"
```

<u>PGADMIN</u>

```sql
-- Table: public.parking_belltown_n_2017
-- DROP TABLE public.parking_belltown_n_2017;
CREATE TABLE public.parking_belltown_n_2017 ( occupancydatetime text COLLATE pg_catalog."DEFAULT", paidoccupancy text COLLATE pg_catalog."DEFAULT", blockfacename text COLLATE pg_catalog."DEFAULT", sideofstreet text COLLATE pg_catalog."DEFAULT", sourceelementkey text COLLATE pg_catalog."DEFAULT", parkingtimelimitcategory text COLLATE
pg_catalog."DEFAULT", parkingspacecount text COLLATE pg_catalog."DEFAULT", paidparkingarea text COLLATE
pg_catalog."DEFAULT", paidparkingsubarea text COLLATE pg_catalog."DEFAULT", parkingrate text COLLATE
pg_catalog."DEFAULT", parkingcategory text COLLATE pg_catalog."DEFAULT", lat text COLLATE pg_catalog."DEFAULT", lon text
COLLATE pg_catalog."DEFAULT" ) WITH ( oids = FALSE ) tablespace pg_default;
ALTER TABLE public.parking_belltown_n_2017 owner TO trafficjam;
```

[2]

```
# pip install sodapy

import pandas as pd
from sodapy import Socrata

import pyodbc
import psycopg2

import pandas.io.sql as pdsql

from sqlalchemy import create_engine

# Unauthenticated client only works with public data sets. Note 'None'
# in place of application token, and no username or password:
client = Socrata("data.seattle.gov", None)

conn = psycopg2.connect(host="rds-trafficjam.cpze98eqmmci.us-east-
2.rds.amazonaws.com",
                       database="dbtrafficjam", user="****", password="****")

cur = conn.cursor()


# ## 2018 Paid Parking Occupancy Data

def getLat(loc):
   lat = loc['coordinates'][0]
   return lat


def getLon(loc):
   lon = loc['coordinates'][1]
   return lon


for i in range(0, 11):
   count = i*100000
   query = ("https://data.seattle.gov/resource/6yaw-
2m8q.json?$limit=100000&$offset="+str(count) +
            "&paidparkingarea=Belltown"
            "&paidparkingsubarea=North"
            "&$where=date_extract_mm(occupancydatetime)%20in%20(0,30)"
            "&$order=:id"
            "&$select=:*,*")
   raw_data = pd.read_json(query)
   raw_data['lat'] = raw_data.location.apply(getLat)
   raw_data['lon'] = raw_data.location.apply(getLon)
   raw_data.drop(['location'], axis=1, inplace=True)
   raw_data['index'] = i
   raw_data = raw_data[['index', ':id', 'blockfacename', 'lat', 'lon',
'occupancydatetime', 'paidoccupancy',
                         'paidparkingarea', 'paidparkingsubarea', 'parkingcategory',
                         'parkingspacecount', 'parkingtimelimitcategory',
'sideofstreet',
                         'sourceelementkey']]
   for col in raw_data.columns:
       if col != 'index':
           raw_data[col].fillna(" ", inplace=True)
   #tuples = [tuple(map(str,x)) for x in raw_data.values]
   tuples = [tuple(x) for x in raw_data.values]
   dataText = ','.join(cur.mogrify(
       '(%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s)', row).decode('utf-8') for row in
tuples)
   cur.execute(
       'insert into paid_parking_2018_belltown_north_30 values ' + dataText)
   conn.commit()
   print(count, flush=True)
```

[3]

```
## need to change parameters - where clause

#results = client.get("fdax-a9ur", limit=1000000, where="study_year>2017")
results = client.get("fdax-a9ur", limit=2000)

# Convert to pandas DataFrame
results_df = pd.DataFrame.from_records(results)

conn = psycopg2.connect(host="rds-trafficjam.***.amazonaws.com",
                        database="***", user="***", password="****")

cur = conn.cursor()

engine = create_engine('postgresql://trafficjam***.amazonaws.com:5432/dbtrafficjam')

results_df.to_sql('annual_parking', con=engine, if_exists='replace', index=True, index_label=None, chunksize=None,
dtype=None)
```

[4]

```
csv = pd.read_csv('Public_Garages_or_Parking_Lots.csv')
import_to_sql_csv('public_garages_lots',csv)
```

[5]

```
csv = pd.read_csv('Traffic_Flow_Map_Volumes.csv')
import_to_sql_csv('traffic_2017_2018', csv)
```

[6]

```
csv = pd.read_csv('Hourly Data  AgWeatherNet at Washington State University.csv')
import_to_sql_csv('hourly_weather', csv)
```

[7]

```
conn = psycopg2.connect(host="rds-trafficjam.***.amazonaws.com",database="***********", user="******",
password="********")
cur = conn.cursor()
with open('employment.csv', 'r') as f:
  cur.copy_from(f, 'seattle_employment', sep=',')
```

```
conn.commit()
conn.close()
```

[8]

```
results = client.get("dm95-f8w5", limit=3000)
import_to_sql('special_events_permits',results)
```

[9]

```
import numpy as np
import pandas as pd
# censusgeocode is a wrapper for the US Census geocoder API
import censusgeocode as cg

# read in CSV downloaded from Open Data Portal
df = pd.read_csv('Active_Business_License_Tax_Certificate.csv')


belltown = df[df['Zip'].str[:5] == '98121']


NAICS = pd.DataFrame(belltown['NAICS Description'].unique())


NAICS.to_csv('BelltownNAICS.csv')

# Pull just addresses to use for US Census geocoder in required format
belltownaddresses = pd.DataFrame(belltown.loc[:, 'Street Address':'Zip'])

# Batch US Census geocoder accepts only 1000 addresses at a time
belltownaddresses[:1000].to_csv('belltownaddresses1.csv', header=False)

results1 = cg.addressbatch('belltownaddresses1.csv', returntype = 'locations')

results1df = pd.DataFrame(results1)

# Create second batch for geocoder
belltownaddresses[1000:].to_csv('belltownaddresses2.csv', header=False)

results2 = cg.addressbatch('belltownaddresses2.csv', returntype = 'locations')

results2df = pd.DataFrame(results2)

results = pd.concat([results1df, results2df])

results['id'] = pd.to_numeric(results['id'], errors = 'coerce', downcast = 'integer')

results.set_index('id', inplace = True)

final = pd.merge(belltown, results, left_index = True, right_index = True, how = 'left')

final.drop(columns = ['address_x', 'address_y'], inplace = True)

final.drop(final.tail(1).index,inplace=True)

len(final[final['match'] == False])/len(final)

final.to_csv('business_licenses.csv')
```

[10]

```
CREATE OR REPLACE VIEW public.parking_2018_cleaned AS
 SELECT parking_belltown_n_2018.sourceelementkey AS source_element_key,
split_part(parking_belltown_n_2018.blockfacename, ' BETWEEN'::text, 1) AS main_lane,
    split_part(split_part(parking_belltown_n_2018.blockfacename, 'BETWEEN '::text, 2), ' AND'::text, 1) AS between_one,
    split_part(split_part(parking_belltown_n_2018.blockfacename, 'BETWEEN '::text, 2), 'AND '::text, 2) AS between_second,
    parking_belltown_n_2018.sideofstreet,
    to_date("substring"(parking_belltown_n_2018.occupancydatetime, 0, 11), 'YYYY-MM-DD'::text) AS date_of_transaction,
    date_part('month'::text, to_date("substring"(parking_belltown_n_2018.occupancydatetime, 0, 11), 'YYYY-MM-DD'::text)) AS
mm_transaction,
    date_part('year'::text, to_date("substring"(parking_belltown_n_2018.occupancydatetime, 0, 11), 'YYYY-MM-DD'::text)) AS
yyyy_transaction,
    date_part('day'::text, to_date("substring"(parking_belltown_n_2018.occupancydatetime, 0, 11), 'YYYY-MM-DD'::text)) AS
dd_transaction,
    date_part('dow'::text, to_date("substring"(parking_belltown_n_2018.occupancydatetime, 0, 11), 'YYYY-MM-DD'::text)) AS
dow_transaction,
    date_part('doy'::text, to_date("substring"(parking_belltown_n_2018.occupancydatetime, 0, 11), 'YYYY-MM-DD'::text)) AS
doy_transaction,
    date_part('week'::text, to_date("substring"(parking_belltown_n_2018.occupancydatetime, 0, 11), 'YYYY-MM-DD'::text)) AS
week_transaction,
    to_timestamp("substring"(parking_belltown_n_2018.occupancydatetime, 12, 8), 'HH24:MI:SS'::text)::time without time zone
AS time_of_transaction,
    "substring"(parking_belltown_n_2018.occupancydatetime, 12, 2)::integer AS hour_transaction,
    "substring"(parking_belltown_n_2018.occupancydatetime, 15, 2)::integer AS min_transaction
   FROM parking_belltown_n_2018
```

[11]

```
ALTER TABLE parking_belltown_n_2018 ALTER COLUMN lat TYPE double precision USING lat::DOUBLE PRECISION;
ALTER TABLE parking_belltown_n_2018 ALTER COLUMN lon TYPE double precision USING lon::DOUBLE PRECISION;
CREATE extension postgis;
ALTER TABLE parking_belltown_n_2018 ADD COLUMN geog geography(point,4326);
UPDATE parking_belltown_n_2018
SET geog = St_makepoint(lon, lat);
```

[12]

```
SELECT l.source_element_key
            ,l.latitude
            ,l.longitude
            ,l.geog
            ,count(b.geog) AS license_count
            ,count(CASE WHEN b."NAICS Code" = 812930 THEN 1 END) AS parking_lots_garages_count
```

```
        ,count(CASE WHEN b."NAICS Code" IN (522110, 55111, 525990, 522120) THEN 1 END) AS banks_count
        ,count(CASE WHEN b."NAICS Code" IN (713120, 445291, 312120, 722514, 722410, 722511, 722513, 722330, 722515,
312130) THEN 1 END) AS bars_restaurants_count
        ,count(CASE WHEN b."NAICS Code" IN (711120, 711190, 711219, 711310, 711110) THEN 1 END) AS
event_venues_count
        ,count(CASE WHEN b."NAICS Code" IN (445120, 445220, 445210, 445110) THEN 1 END) AS grocery_market_count
        ,count(CASE WHEN b."NAICS Code" IN (713940) THEN 1 END) AS gyms_count
        ,count(CASE WHEN b."NAICS Code" IN (721191, 721110) THEN 1 END) AS hotels_count
        ,count(CASE WHEN b."NAICS Code" IN (621498, 621399, 621310, 621210, 621330, 621320, 621340, 621111, 621112)
THEN 1 END) AS medical_offices_count
        ,count(CASE WHEN b."NAICS Code" IN (712110) THEN 1 END) AS museums_count
        ,count(CASE WHEN b."NAICS Code" IN (812111, 812112, 812113) THEN 1 END) AS salons_count
        ,count(CASE WHEN b."NAICS Code" IN (452990, 446199, 442299, 453998, 323117, 448130, 446120, 443142, 448140,
446191, 423210, 442110, 453220, 451120, 448310, 448320, 448110, 446130, 448190, 454390, 453910, 446110, 451130, 451110,
453991, 453310, 424330) THEN 1 END) AS shopping_count
FROM location AS l
LEFT JOIN business_licenses AS b
ON ST_Dwithin(l.geog, b.geog, 200)
GROUP BY l.source_element_key
        ,l.latitude
        ,l.longitude
        ,l.geog
```

[13]

```
SELECT    a.source_element_key,
     a.main_lane,
     a.between_one,
     a.between_second,
     a.sideofstreet,
     a.date_of_transaction,
     a.mm_transaction,
     a.yyyy_transaction,
     a.dd_transaction,
     a.dow_transaction,
     a.doy_transaction,
     a.week_transaction,
     a.time_of_transaction,
     a.hour_transaction,
     a.min_transaction,
     a.occupied_spots,
     a.empty_spots,
     a.total_spots,
     a.paidparkingarea,
     a.paidparkingsubarea,
     a.parkingcategory,
     a.parkingtimelimitcategory,
     a.latitude,
     a.longitude,
     a.geog,
     a.segkey,
     b.parking_spaces      AS total_spots_survey,
     b.total_vehicle_count AS occupied_spots_survey,
     b.study_time          AS time_of_survey,
     w.hour                AS weather_at_hour,
     w."AvgTempF"          AS average_temp,
     w."TotPrecin"         AS rain_inches,
     w."RelHum"            AS humidity,
     p.name_of_event       AS event,
     e.employment,
     t.aawdt AS avg_traffic_density,
     CASE
         WHEN t.aawdt IS NOT NULL THEN 1
         ELSE 0
     END            AS main_city_street,
     l.license_count AS business_license_count,
     l.parking_lots_garages_count,
     l.banks_count,
     l.bars_restaurants_count,
     l.event_venues_count,
     l.grocery_market_count,
     l.gyms_count,
     l.hotels_count,
     l.medical_offices_count,
     l.museums_count,
     l.salons_count,
     l.shopping_count
FROM    parking_2018_cleaned a
LEFT JOIN annual_parking b
ON      a.source_element_key = b.elmntkey
AND     a.date_of_transaction = b.study_date
AND     a.time_of_transaction >= (b.study_time - '00:30:00'::interval)
AND     a.time_of_transaction <= (b.study_time + '00:30:00'::interval)
LEFT JOIN hourly_weather w
ON      a.date_of_transaction = w."Date"
AND     a.hour_transaction::DOUBLE PRECISION = date_part('hour'::text, w.hour)
LEFT JOIN s_events_permits_belltown p
```

```
ON      p.event_date = a.date_of_transaction
LEFT JOIN seattle_employment e
ON      e.year::DOUBLE PRECISION = a.yyyy_transaction
AND      e.month::DOUBLE PRECISION = a.mm_transaction
LEFT JOIN traffic_2017_2018 t
ON      a.segkey = t.segkey
AND      a.yyyy_transaction = t.year::DOUBLE PRECISION
LEFT JOIN business_license_counts l
ON     a.source_element_key = l.source_element_key;
```