San Francisco Rental Slackbot

W205 Final Paper

Team Members:

Amy Smessaert

Matthew Nelson

Joseph Kunkel

Qing Zhang

Instructor:

Arash Nourian

Introduction

The rental market in San Francisco is one of the fastest-moving in the country. It's also some of the most expensive real estate. Given these conditions, finding a place to live in the city can be especially difficult, and sometimes renters may need to branch out of their original housing parameters to less familiar areas. Our team sought to make this experience easier. We developed a Slackbot that would determine the quality-of-life and propose rental properties in San Francisco based on inputs from a user. We combined available data from DataSF (open source) and Craigslist into a tool that gives users quantitative neighborhood information to evaluate before making a big life decision: where to live.

As motivation for the project, one of the team members had previously gone through the experience of finding an apartment in San Francisco and, months later, stumbled upon this blog: https://www.dataquest.io/blog/apartment-finding-slackbot/. While the blog is a step forward and would have been useful to have during the apartment-finding process, the solution our team implemented is more detailed and precise. The original project was designed to collaborate with others on which Craigslist listings they thought were best, but contributing data was limited to that found in the Craigslist postings.

High-Level Technology Introduction

There were four key considerations that determined the technology we used:

1. The rental information needed to be comprehensive, free to access, and up-to-date. For these reasons, we opted to use Craigslist data. There are python packages that provide basic Craigslist scraping capabilities, which we customized to meet our parameters.
2. The rental information needed to be easily accessible, and our solution user-friendly. We

decided to use Slack as our communication instrument. Slack has become popular for inter-organizational communication, particularly in the Bay Area. We expect that many users are already familiar with the platform. In addition, Slack provides developer information for application customization.

3. We needed to store large diverse sets of data and access them quickly. To accommodate these needs, we decided to implement HDFS on AWS. The key data was cleaned, uploaded and stored in Amazon S3 bucket loading into HDFS via our data load script.

4. In order to be useful, our platform needed to rapidly respond to user input with the detailed query response. The bulk of the data was transformed into tables in Hive and queried according to user input with Spark. Implementing Spark via PySpark allowed for seamless integration with our custom framework.

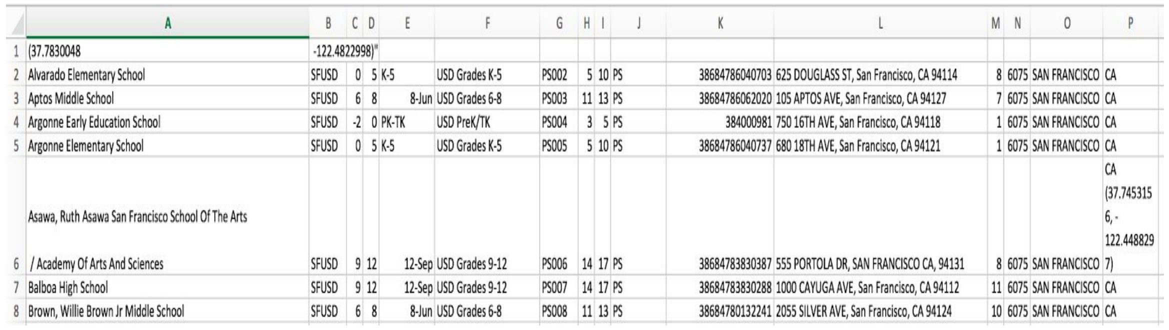
Data Sources

The data sources collected in this project include Craigslist rental listings (https://sfbay.craigslist.org/search/sfc/apa), and raw data and limited descriptive statistics detailing quantitative information about the city and county of San Francisco from Data SF (https://data.sfgov.org), and Vital Signs (http://www.vitalsigns.mtc.ca.gov/).

In our project prototype, we collected available data that described the following quality of life metrics, by zip code or neighborhood name: the number and detail of active businesses, bicycle share locations, bicycle parking, off-street automobile parking, incidents involving the San Francisco police department, incidents involving the San Francisco fire department, local school type and location, and tree coverage. Data detailing available rentals was obtained by a python script that collected listing information from Craigslist.org.

Data Collection & Processing

The data sets used for our project are made available as both JSON or CSV files. We use CSV files in the prototype bot, which necessitates simple data cleaning steps before loading. Hidden return characters were replaced with comma delimiters in bash, correct newlines inserted, and proper names (with extraneous commas) were removed with command line perl regex expressions according to individual needs (ex: > tr -d ‘\n’ <file, > sed -i 's/)"/)”\n/g’, > perl -npe 's/^"\w+,\s/"/' , etc.)



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	(37.7830048	-122.4822998)"														
2	Alvarado Elementary School	SFUSD	0	5	K-5	USD Grades K-5	PS002	5	10	PS		38684786040703	625 DOUGLASS ST, San Francisco, CA 94114	8	6075	SAN FRANCISCO	CA
3	Aptos Middle School	SFUSD	6	8	8-Jun	USD Grades 6-8	PS003	11	13	PS		38684786062020	105 APTOS AVE, San Francisco, CA 94127	7	6075	SAN FRANCISCO	CA
4	Argonne Early Education School	SFUSD	-2	0	PK-TK	USD PreK/TK	PS004	3	5	PS		384000981	750 16TH AVE, San Francisco, CA 94118	1	6075	SAN FRANCISCO	CA
5	Argonne Elementary School	SFUSD	0	5	K-5	USD Grades K-5	PS005	5	10	PS		38684786040737	680 18TH AVE, San Francisco, CA 94121	1	6075	SAN FRANCISCO	CA
6	Asawa, Ruth Asawa San Francisco School Of The Arts / Academy Of Arts And Sciences	SFUSD	9	12	12-Sep	USD Grades 9-12	PS006	14	17	PS		38684783830387	555 PORTOLA DR, SAN FRANCISCO CA, 94131	8	6075	SAN FRANCISCO	CA (37.745315 6, - 122.448829 7)
7	Balboa High School	SFUSD	9	12	12-Sep	USD Grades 9-12	PS007	14	17	PS		38684783830288	1000 CAYUGA AVE, San Francisco, CA 94112	11	6075	SAN FRANCISCO	CA
8	Brown, Willie Brown Jr Middle School	SFUSD	6	8	8-Jun	USD Grades 6-8	PS008	11	13	PS		38684780132241	2055 SILVER AVE, San Francisco, CA 94124	10	6075	SAN FRANCISCO	CA

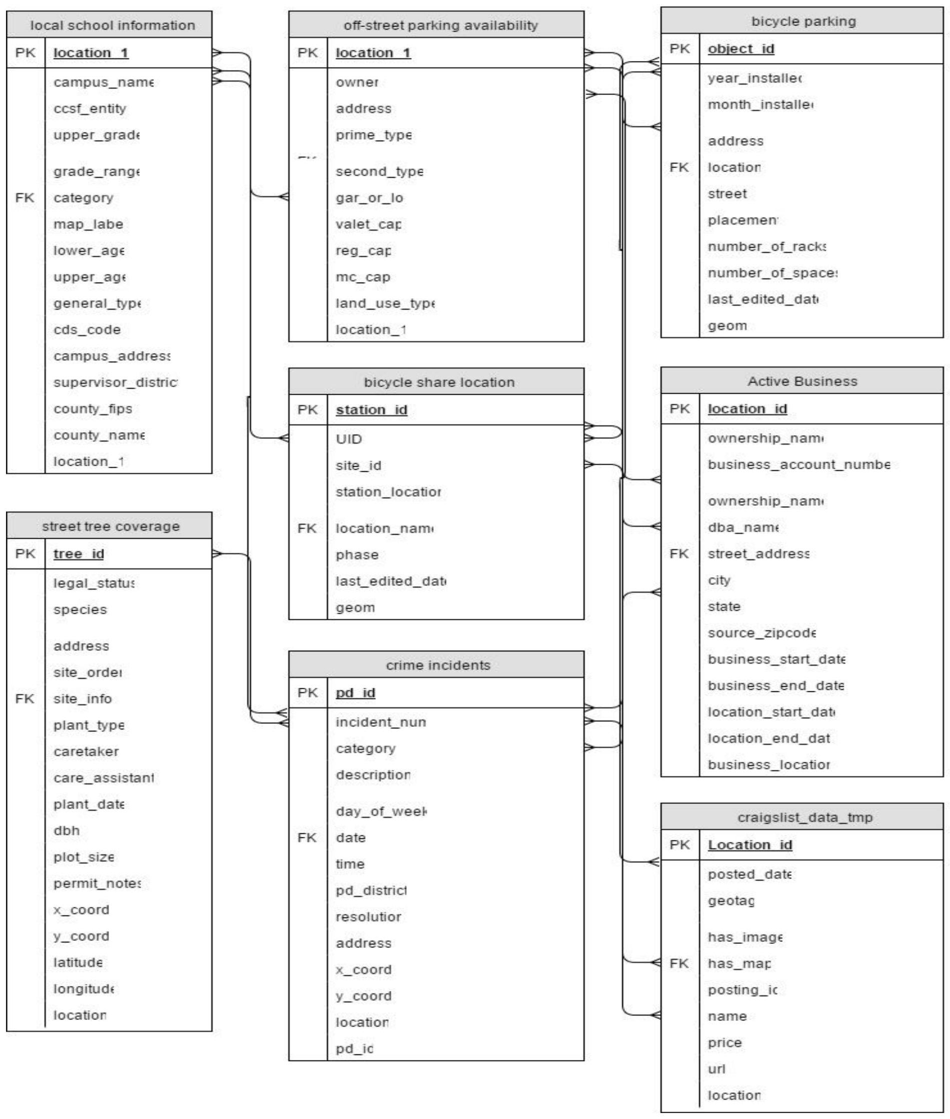
To avoid similar issues, we recommend manual file review for file-specific data-cleaning solutions. Cleaned data is stored in Amazon S3 to facilitate loading into our EC2 instance.

Our project solution is organized around zip codes. Some DataSF data sets contain address and zip code information, others contain geotag coordinates set in the center of a key building or facility. We use DataSF mapping information (https://data.sfgov.org/Geographic-Locations-and-Boundaries/Analysis-Neighborhoods/p5b7-5n3h) and https://data.sfgov.org/Geographic-Locations-and-Boundaries/San-Francisco-ZIP-Codes/srq6-hmpi) to project geotagged neighborhood information into predefined postal code regions. In the implementation, neighborhoods with irregular, non-rectangular boundaries are transformed into rectangular perimeters that encompass the most distant coordinates to facilitate mapping geotagged coordinates into our zip code framework.

Table construction in Hive

Once our static data was successfully cleaned and loaded into HDFS, we needed to turn it into tables so we could aggregate and query it. We wrote a DDL script to create the tables as the entity relationship diagram shows below. The tables are: Businesses, bike_parking, bike_station, schools, parking, SFPD and trees. Once the DDL script was executed, we performed additional manual checks of each table in Hive in order to ensure there weren't any missed data quality issues. Craigslist data is automatically scraped to a Hive table, so separate quality checks were implemented.

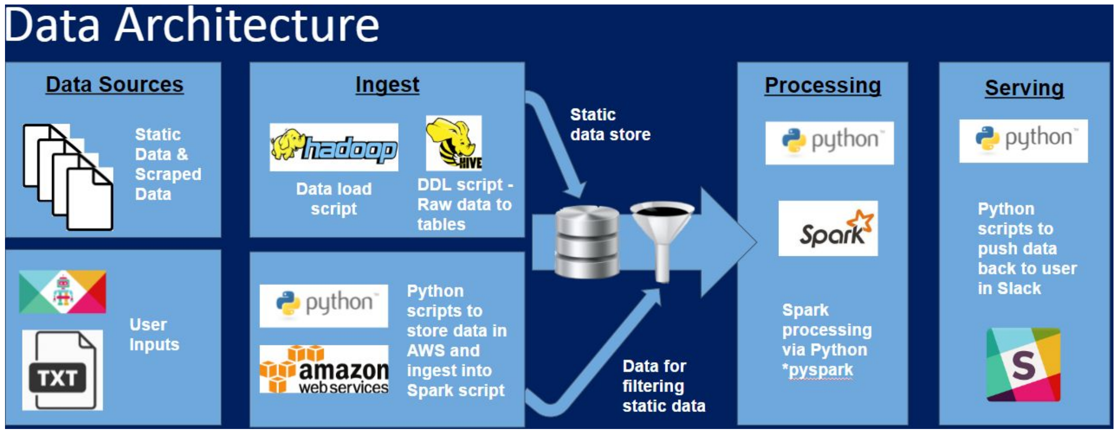
Entity Relationship Diagram



There are some instances in the load script and the DDL script where we mention data sets not contained in our prototype. This will be mentioned towards the end of the report in the "Future Roadmap" and "Potential Expansion" sections.

Implementation and Analysis

High Level Architecture


Data Architecture
Data Sources
Static Data & Scraped Data
User Inputs
TXT
Ingest
hadoop
HIVE
Data load script
DDL script - Raw data to tables
python
Python scripts to store data in AWS and ingest into Spark script
amazon webservices
Static data store
Data for filtering static data
Processing
python
Spark
Spark processing via Python *pyspark
Serving
python
Python scripts to push data back to user in Slack

Data Acquisition & Storage

Static data sets from DataSF and Vital Signs were downloaded locally in CSV format. The files Active_Business_Locations.csv, Map_of_Schools.csv, Schools_type.csv, Eviction_Notices.csv, Recreation___Park_Department_Park_Info_Dataset.csv, and Park_and_Open_Space_Map.csv required data cleaning before loading. In this prototype, we were not able to automate the data quality detection and cleaning. Improvements upon this are discussed in "Future Roadmap / Improvements." Cleaned data sets were loaded into Amazon storage in S3. This solution provided a central location for the data sets which assisted in the automation of the load process. Once the clean static data was in S3, a Load Data Lake script was written to create a file structure in HDFS, rename the files and put the respective files into their appropriate locations.

Python scraper code was written to scrape Craigslist data into a Hive table. A custom Slackbot script presents the user with options and asks them to rank their preferences, and stores their responses in a txt file. In order to implement the user interaction, our team needed to generate slack credentials, a unique channel, an APP, and generate an API token. A custom python script runs the app and stores the results of the interaction.

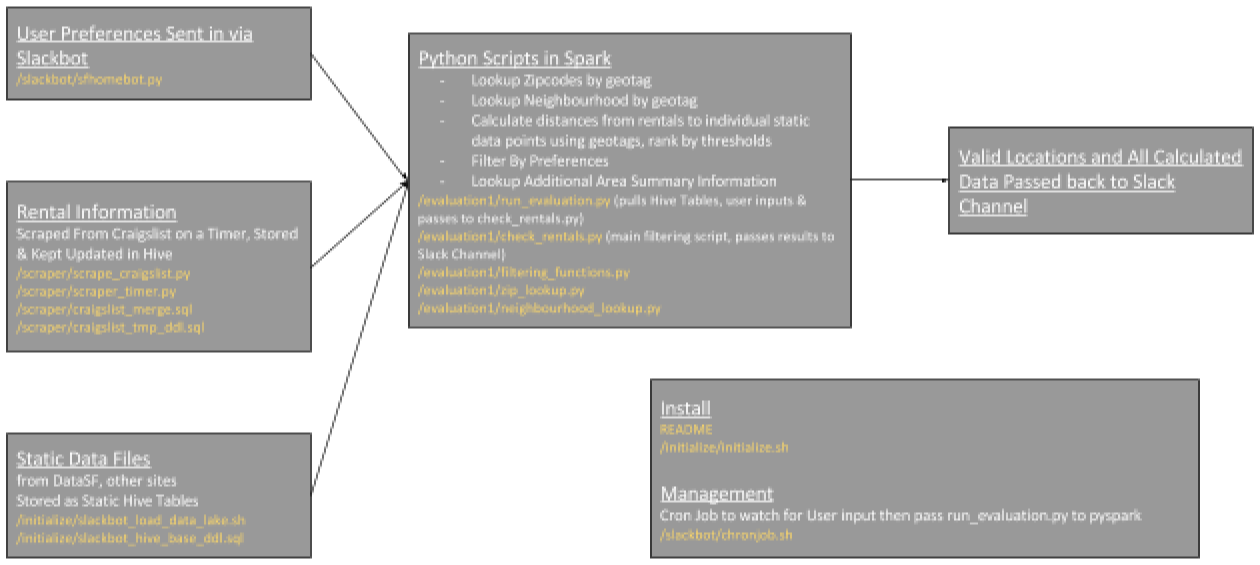
Data Ingest

Static data is transformed into Hive tables via DDL scripts. A .txt file from user input via the slackbot is loaded into the EC2 instance, and detected by a cron daemon. Once the .txt file is detected, the Spark analysis is executed and the data processed.

Data Processing and Serving

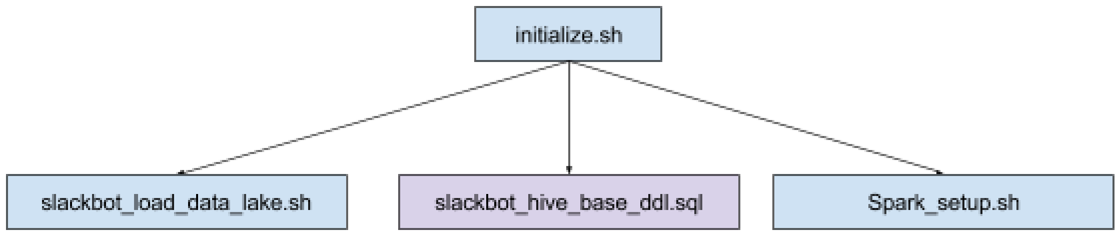
The PySpark evaluation script is executed, which queries and filters the static datastore using the specific user input recorded in the slackbot .txt file.
After filtration executes, a Python script aggregates the data and returns the solution to the user via the slack channel (retrieves the token information from the environment, selects, organizes and returns the display).

Implementation


User Preferences Sent in via Slackbot
/slackbot/sfhomebot.py
Rental Information
Scraped From Craigslist on a Timer, Stored & Kept Updated in Hive
/scraper/scrape_craigslist.py
/scraper/scraper_timer.py
/scraper/craigslist_merge.sql
/scraper/craigslist_tmp_ddl.sql
Static Data Files
from DataSF, other sites
Stored as Static Hive Tables
/initialize/slackbot_load_data_lake.sh
/initialize/slackbot_hive_base_ddl.sql
Python Scripts in Spark
- Lookup Zipcodes by geotag
- Lookup Neighbourhood by geotag
- Calculate distances from rentals to individual static data points using geotags, rank by thresholds
- Filter By Preferences
- Lookup Additional Area Summary Information
/evaluation1/run_evaluation.py (pulls Hive Tables, user inputs & passes to check_rentals.py)
/evaluation1/check_rentals.py (main filtering script, passes results to Slack Channel)
/evaluation1/filtering_functions.py
/evaluation1/zip_lookup.py
/evaluation1/neighbourhood_lookup.py
Valid Locations and All Calculated Data Passed back to Slack Channel
Install
README
/initialize/initialize.sh
Management
Cron Job to watch for User input then pass run_evaluation.py to pyspark
/slackbot/chronjob.sh

Environment Installation

The README in the repo describes the steps required to instantiate the project slackbot. The project was designed to be replicable on top of a base W205 Spring AWS AMI and utilizes installation scripts as provided throughout the w205 course to properly setup Hadoop, Hive, Postgres and Spark. Within the GitHub repository, the initialize.sh script installs many of the necessary packages needed for the slackbot to run.



<u>Initialization Script Details</u>

initialize.sh

- Starts Hadoop, Hive, sets up Spark, enables PATH variables
- Updates Python & PIP 2.6 to 2.7
- Installs required Python packages, python-craigslist and slackclient
- Downloads Static data from shared AWS S3 bin
- Runs slackbot_load_data_lake.sh
- Runs slackbot_hive_base_ddl.sql
- Changes permissions as needed

slackbot_load_data_lake.sh

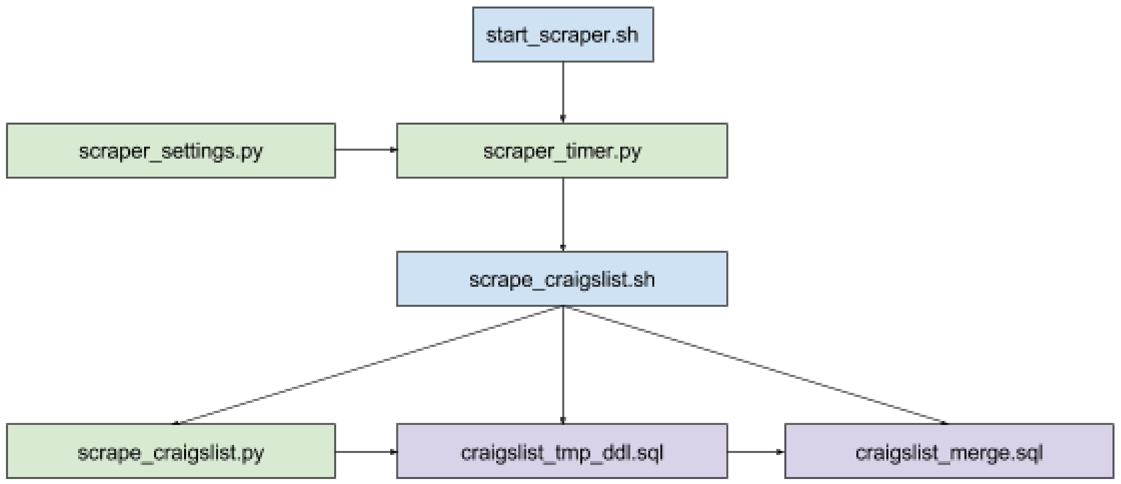
- downloads all Static Data files from S3
- Creates destination directories in Hadoop
- Moves static data files to Hadoop

slackbot_hive_base_ddl.sql

- Creates separate tables in Hive for each static dataset, as outlined in the ER diagram

Craigslist Scraper

The scraper was designed to utilize the functionality of the python-craigslist package, scrape craigslist with predefined settings, save the results as a .csv file, load this .csv file into Hive as a separate table, merge the current scraper results with an existing unique_rentals table in Hive to ensure no duplicates, delete all temporary files, and repeat this process on a timer. Our implementation currently grabs 50 available rentals randomly from the sfc.craigslist site, however a sort_by='newest' flag can be applied in scrape_craigslist.py in order to ensure that the most recent postings are always available to the users query. This would be most effective if the slackbot were left running and the corpus of available rentals grows.



<u>Scraper Script Details</u>

start_scraper.sh

- Sets Permissions
- Runs scraper_timer.py

scraper_timer.py

- Runs scraper_craigslist.sh once every _ seconds, as defined in scraper_settings.py

scrape_craigslist.sh

- Makes directories in Hadoop if they don't exist
- Deletes tmp files from last scraper
- runs scrape_craigslist.py
- Copies local tmp file to HDFS
- Runs craigslist_tmp_ddl.sql
- Runs craigslist_merge.sql

scrape_craigslist.py

- Scrapes craigslist (within site='sfbay', area="sfc", category='apa') and saves all scraped information to a local temporary csv file. Limit of postings per scrape is set to 50.

craigslist_tmp_ddl.sql

- Creates a table in Hive called craigslist_data_tmp containing the last scrapes rental information

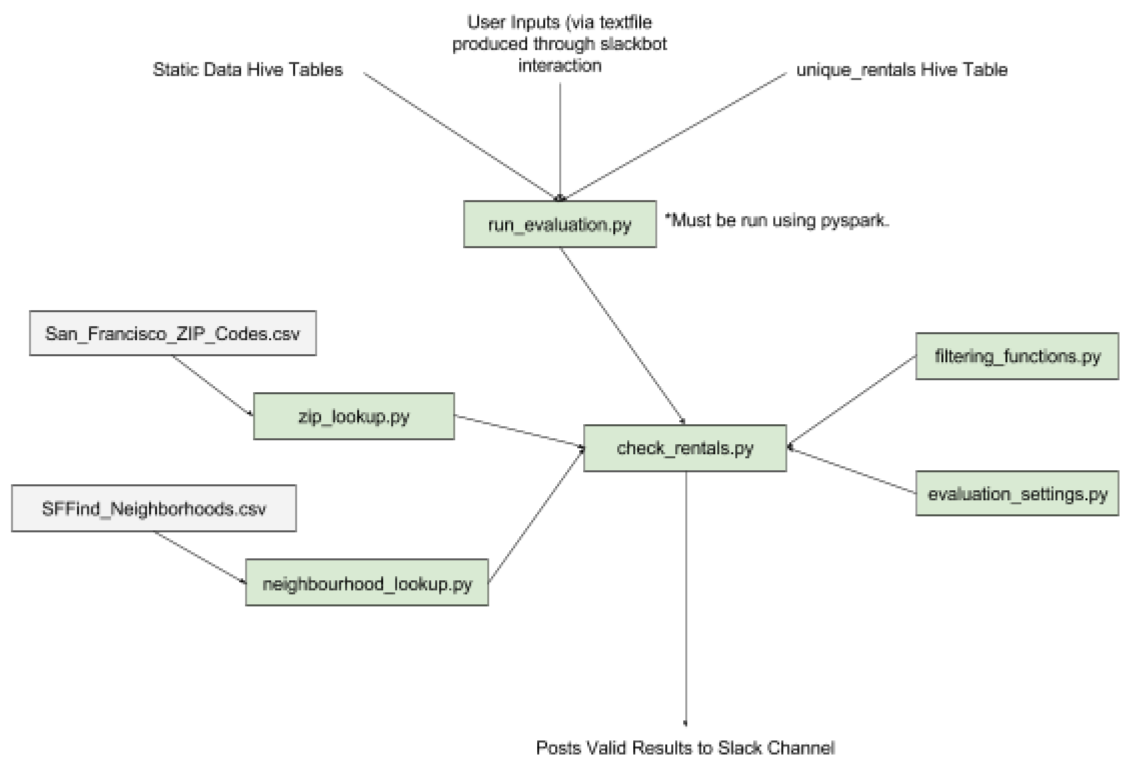
craigslist_merge.sql

- Merges craigslist_data_tmp with the unique_rentals table in Hive and saves only unique items

scraper_settings.py

- Contains adjustable settings for the Scraper such as timer interval & area

Rental Evaluation


User Inputs (via textfile produced through slackbot interaction
Static Data Hive Tables
unique_rentals Hive Table
run_evaluation.py
*Must be run using pyspark.
San_Francisco_ZIP_Codes.csv
zip_lookup.py
check_rentals.py
filtering_functions.py
evaluation_settings.py
SFFind_Neighborhoods.csv
neighbourhood_lookup.py
Posts Valid Results to Slack Channel

<u>Evaluation Script Details</u>

run_evaluation.py

- Opens User inputs from sfhomebot.txt and collects their preferences as variables
- Initiates a Spark Context
- Selects all information from unique_rentals Hive table & converts from a Spark dataframe to a python dictionary
- Queries zipcode, count(parks) from parks Hive table & converts from a Spark dataframe to a python dictionary
- Queries zipcode, count(bars) from businesses Hive table & converts from a Spark dataframe to a python dictionary
- Queries zipcode, count(restaurants) from businesses Hive table & converts from a Spark dataframe to a python dictionary
- Queries zipcode, count(businesses) from businesses Hive table & converts from a Spark dataframe to a python dictionary

- Queries location, geotag from bike_parking Hive table & converts from a Spark dataframe to a python dictionary
- Queries location, geotag from bike_stations Hive table & converts from a Spark dataframe to a python dictionary
- Queries location, geotag from schools Hive table & converts from a Spark dataframe to a python dictionary
- Queries location, geotag from trees Hive table & converts from a Spark dataframe to a python dictionary
- Queries location, geotag from sfpd Hive table & converts from a Spark dataframe to a python dictionary
- Queries location, geotag from parking Hive table & converts from a Spark dataframe to a python dictionary
- Runs check_rentals.py and passes in all collected data

check_rentals.py

- Runs zip_lookup.py if zipcode not present in listing
- Runs neighbourhood_lookup.py if neighbourhood name not present in listing
- Takes each rental and passes it through a number of filters as designed in filtering_functions.py
    - Max Price Filter
    - Min Price Filter
    - Zipcode Based Filters
        - # of Parks in Zipcode
        - # of Fires in Zipcode
        - # of Bars in Zipcode
        - # of Restaurants in Zipcode
        - # of Businesses in Zipcode
    - Distance Based Filters
        - Close to Bike Parking (Yes/No, Distance to)

        - Close to Bike Station (Yes/No, Distance to)
    - Density Style Filters
        - Density of Schools within certain radius of rental
        - Density of SFPD Events within certain radius of rental
        - Density of Vehicle Parking within certain radius of rental
        - Density of Trees within certain radius of rental
- If rental makes it through all of the filters it is then considered a valid rental and the result is passed to a Slack Channel for the User to View
- Results are formatted for increased readability and include many calculated fields to help facilitate informed decision making

filtering_functions.py

- Contains all functions used for filtering the rentals
- Functions used in this evaluation:
    - min_in_zip(zip_eval_dict): counts the maximum and minimum of items in a specific zipcode
    - points2distance(start,end): calculates the distance in km's between one set of lat/longs and another set of lat/longs
    - close_to_bike_parking(geotag, bike_parking_locations): calculates if a rental is close to bike parking & returns Yes/No, closest location name & distance. Maximum distance to a bike parking location = 0.5km.
    - close_to_bike_station(geotag, bike_station_locations): calculates if a rental is close to a bike share station & returns Yes/No, closest location name & distance. Maximum distance to a bike station = 1km.
    - parking_density(geotag, parking_locations): calculates if a rental is in a high/medium/low density area for parking, separated out by Private and Public lots & returns density and a count of locations for both Private and Public. 1km radius, low density threshold of 100, med density threshold of 500
    - school_density(geotag, school_locations): calculates if a rental is in a

    - high/medium/low density area for schools & returns density and a count of schools. 10km radius, low density threshold of 3, med density threshold of 10
    - sfpd_density(geotag, sfpd_locations): calculates if a rental is in a high/medium/low density area for SFPD incident & returns density and a count of incidents. 1km radius, low density threshold of 5000, med density threshold of 10000
    - tree_density(geotag, tree_locations): calculates if a rental is in a high/medium/low density area for trees & returns tree density and a count of the trees in the area. 1km radius, low density threshold of 1000, med density threshold of 10000

evaluation_settings.py

- Contains Slack Channel information and pulls in the Slack Token for posting to a channel from the set Environment Variables.

zip_lookup.py

- Imports San_Francisco_ZIP_Codes.csv, takes the given polygon coordinates for each zip code and approximates a square for each given ZIP. This was necessary in order to simplify some very complicated polygons, though it may result in some false-positives when doing the zip lookup.
- Defines functions to check whether a given geotag is located in a calculated zip_box and completes a lookup to determine a zipcode for any given geotag in the San Francisco area.

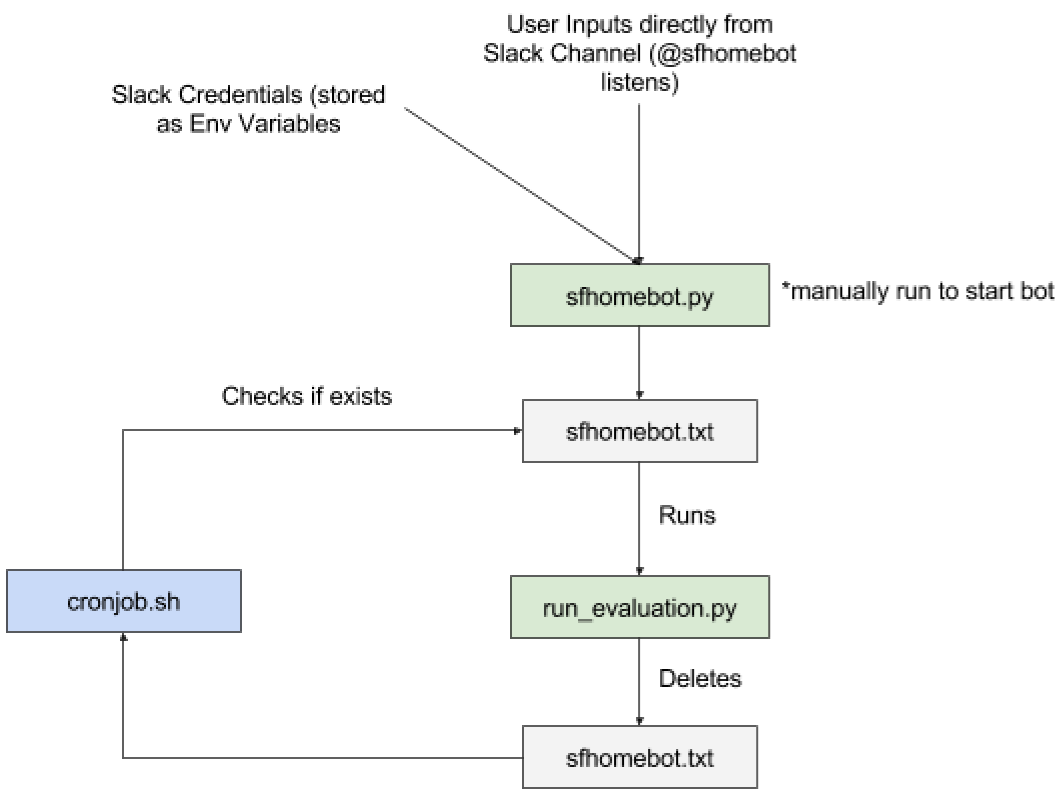
neighbourhood_lookup.py

- Imports SFFind_Neighborhoods.csv, takes the given polygon coordinates for each neighbourhood and approximates a square. This was necessary in order to simplify some very complicated neighbourhood polygons, though it may result in some false-positives when doing the neighbourhood lookup.
- Defines functions to check whether a given geotag is located in a calculated

neighbourhood_box and completes a lookup to determine a neighbourhood name for any given geotag in the San Francisco area.

SlackBot

The slackbot was designed to take input preferences from a user within a Slack channel, and pass this into the above Evaluation code. The implementation of the Bot is highly simplistic and has not been optimized for usability. Only one script had been created to interact with the user, and error handling regarding the user-inputs has not been implemented. The purposes of the SlackBot at this stage of development is simply to show the potential for interaction between a standard user in Slack and the back-end infrastructure / evaluation scripts on the AWS instance. In order to allow the slackbot to automatically begin an evaluation of a user's query, and to reset itself following a successful evaluation, a cron daemon was designed to look for the input .txt file in a specific folder and kick off the evaluation. Following a timer, it would then wipe the .txt file from memory and allow the user to re-submit a query from the top of the slackbot script. The implementation of the cron daemon did not work as intended, and it is recommended to pursue other options for slackbot interaction with our system.


User Inputs directly from Slack Channel (@sfhomebot listens)
Slack Credentials (stored as Env Variables
sfhomebot.py
*manually run to start bot
Checks if exists
sfhomebot.txt
Runs
cronjob.sh
run_evaluation.py
Deletes
sfhomebot.txt

<u>Slackbot Script Details</u>

sfhomebot.py

- Pulls SlackBot credentials from System Env Variables.
- Brings SlackBot online within Slack
- Listens for mentions @botname
- Asks questions of the user according to a specific script
- Parses and saves script replies to sfhomebot.txt

chronjob.sh

- checks if sfhomebot.txt exists
- if so, runs run_evaluation.py in Spark & deletes sfhomebot.txt
- repeats on a timer

Technical Challenges

Collaboration

We faced difficulties sharing the data and credentials. We initially planned to share snapshots of the EBS, but after trying a couple iterations, the snapshot wasn't rendering the loads accurately (unknown cause). Our solution was to use a shared Amazon AWS account with a persistent EBS volume. The credentials and .pem files were shared through different platforms, although we realize that this isn't an ideal solution and isn't the most secure. The overall build of the code for the project occurred within a github repository which ensured that the tables could easily be updated in our working instance simply by pulling the most recent repository and re-running the slackbot_load_data_lake.sh and slackbot_hive_base_ddl.sql scripts. This was executed successfully and team members could then separately work on the data cleaning, Hive DDL statements and main evaluation code separately without conflict.

Data Cleaning

The main issue we encountered was inconsistent formatting in a diverse set of static, open-source data. Data preparation and cleaning required limited, ad-hoc command line formatting changes.

Python version incompatibility

We use the Spring 2016 UCB AMI in our project instance. This AMI includes Python 2.6.6. As is common with code development, it is often easier to get pieces of code to work locally before bringing them into the shared branch. This was the case for our project. The default Python version on our machines is Python 3.5. Our team needed to write a series of scripts to update an AMI from 2.6 to 2.7, so that we could spin up separate instances on-demand.

The SlackClient wasn't compatible with Python 2.6 (update to Python 2.7 solved this incompatibility issue). We implemented a custom bot solution, but have learned of the existence of a slackbot package that may make integration simpler in the future.

New Technologies

There are a few technologies we did not have prior experience with required we manage steep learning curves:

Slack API. As mentioned above, we could have looked into using a bot package instead of the custom integration. This might have helped with connection/ disconnection from the SlackBot instead of a hard bot reset.

Craigslist Scraper. Due to our limited experience writing custom scrapers, we kept our prototype simple. Future versions might include images, and unstructured text.

Automation. Our team wanted the .txt script from the SlackBot to be automatically ingested into the Spark process. We selected a cron daemon to perform this function, which required significant time and energy investment.

Future Roadmap / Improvements

Our team opted to use technologies described in course exercises. We might have improved implementation by branching out of those technologies. We currently store information in Hive tables and write complex code to aggregate the static data. Had we selected a dynamic storage technology like MongoDB, the aggregation of the data would have been automated.

We opted to keep the user interface simple in our prototype. In order for the user to interact with and respond to our bot, they reference the bot with @sfhomebot in each response. This makes our code move through the question script more easily and also allows the parsing of the .txt script simple. In addition, our code doesn’t account for multi-line responses or include error-handling. We might improve upon UI in future implementations.

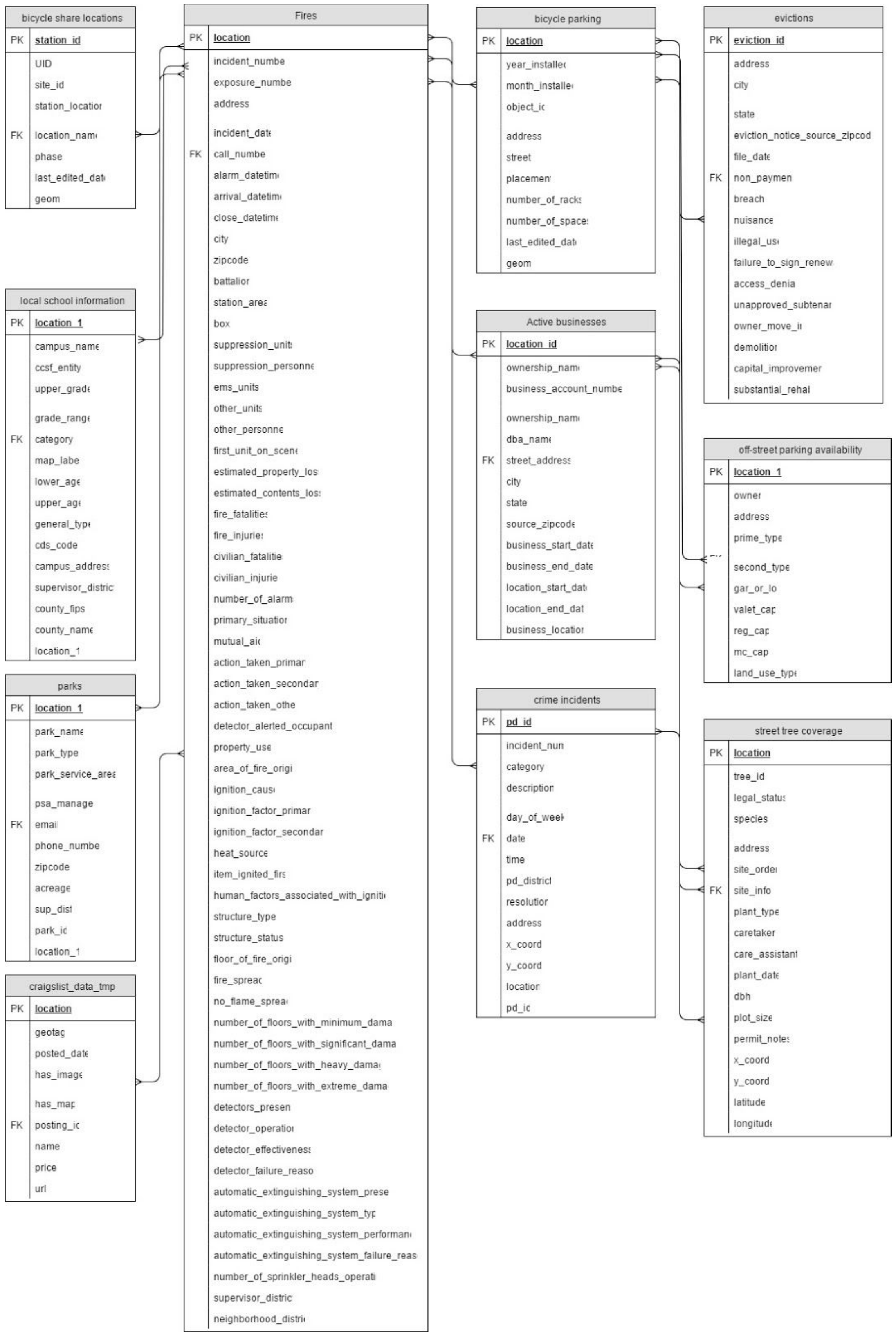
Slackbot disconnect / integration requires manual termination of the session in order to disconnect the SlackBot from the Slack channel. This issue will be solved with the bot package solution.

Our load and DDL scripts account for data that we were ultimately unable to incorporate due to time constraints / complexity issues. The ER diagram on the following page illustrates our intended data offering. With more development time, or a dynamic storage technology, we would have included additional data sets. However, they were not necessary to demonstrate the proof of concept usage of our SlackBot data application.

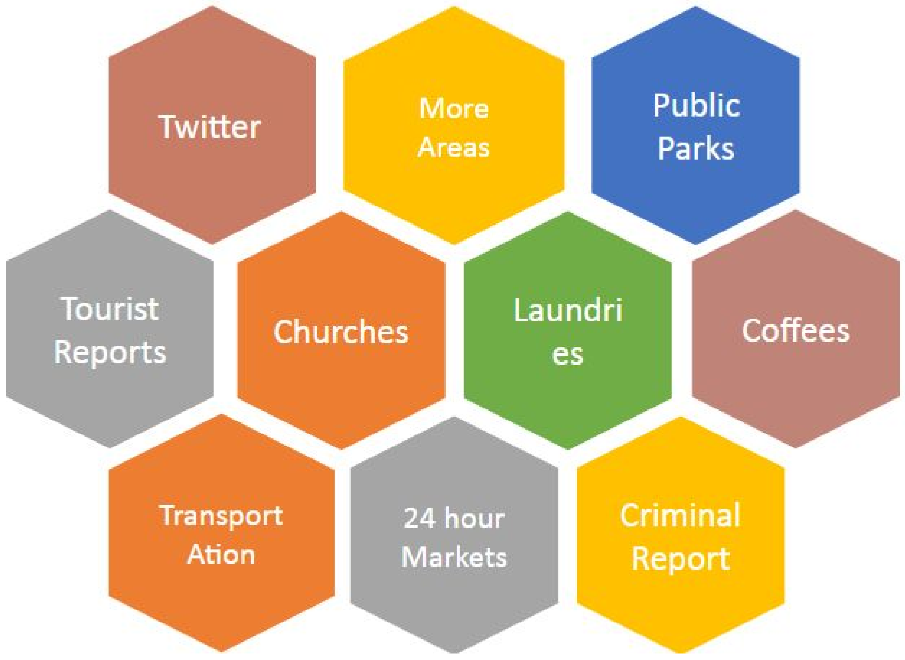
Potential Expansion

As the figure shows below, there are many other categories of information sources which we could include in an expanded database. These datasets might include local coffee shops, local churches, laundries, public transportations, supermarkets, etc. Providing daily utility information would make the dataset much more valuable to the user. We opted not to include streaming data, but in future iterations could include streaming traffic data, and up-to-minute parking availability. In addition, in our project we limit our focus to San Francisco city and county. We could expand data collection further, to include other areas in the Bay Area. As the open data model is adopted by more municipalities, expand to other metro areas altogether.

Future ERD Structure



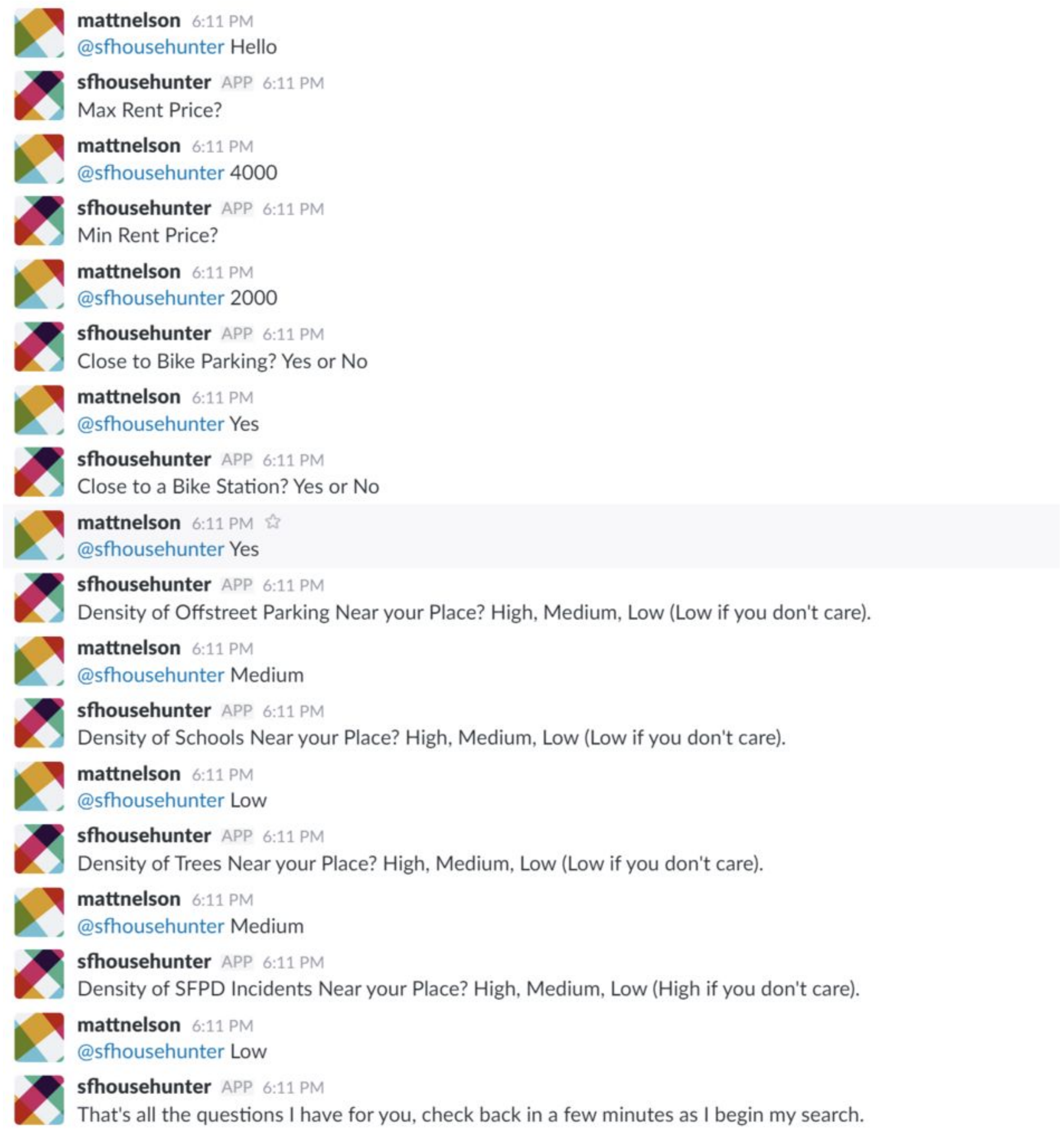
Potential future directions



Example Results

A working implementation of the User Interface and Slackbot Interaction is shown below:

Interaction between User and SlackBot



mattnelson 6:11 PM
@sfhousehunter Hello

sfhousehunter APP 6:11 PM
Max Rent Price?

mattnelson 6:11 PM
@sfhousehunter 4000

sfhousehunter APP 6:11 PM
Min Rent Price?

mattnelson 6:11 PM
@sfhousehunter 2000

sfhousehunter APP 6:11 PM
Close to Bike Parking? Yes or No

mattnelson 6:11 PM
@sfhousehunter Yes

sfhousehunter APP 6:11 PM
Close to a Bike Station? Yes or No

mattnelson 6:11 PM
@sfhousehunter Yes

sfhousehunter APP 6:11 PM
Density of Offstreet Parking Near your Place? High, Medium, Low (Low if you don't care).

mattnelson 6:11 PM
@sfhousehunter Medium

sfhousehunter APP 6:11 PM
Density of Schools Near your Place? High, Medium, Low (Low if you don't care).

mattnelson 6:11 PM
@sfhousehunter Low

sfhousehunter APP 6:11 PM
Density of Trees Near your Place? High, Medium, Low (Low if you don't care).

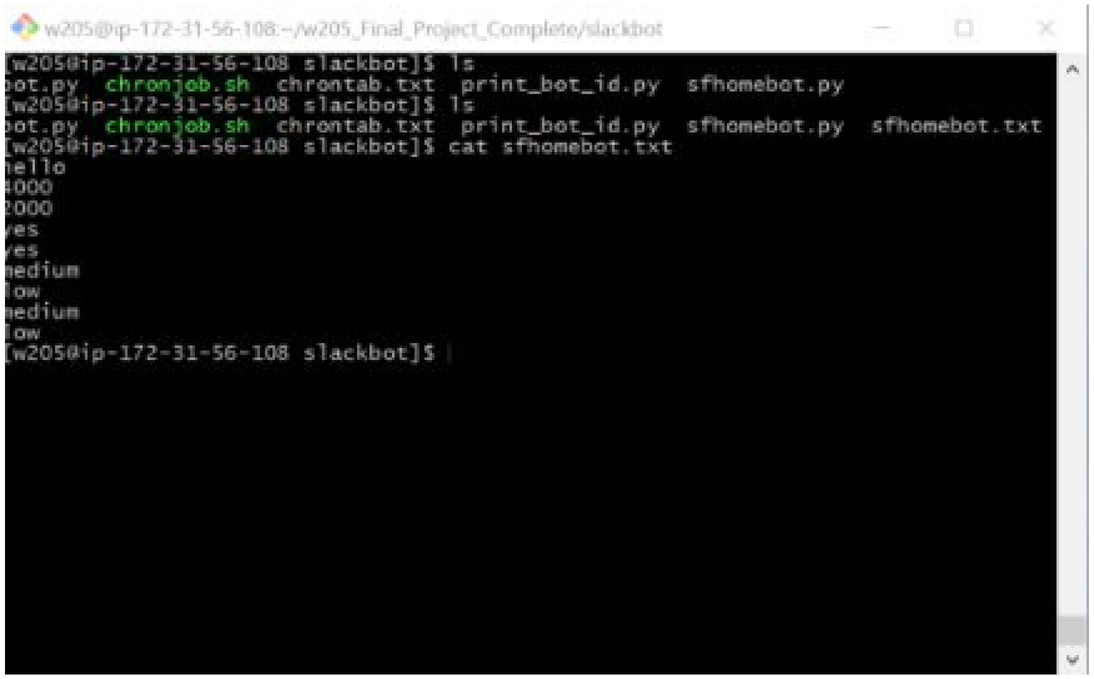
mattnelson 6:11 PM
@sfhousehunter Medium

sfhousehunter APP 6:11 PM
Density of SFPD Incidents Near your Place? High, Medium, Low (High if you don't care).

mattnelson 6:11 PM
@sfhousehunter Low

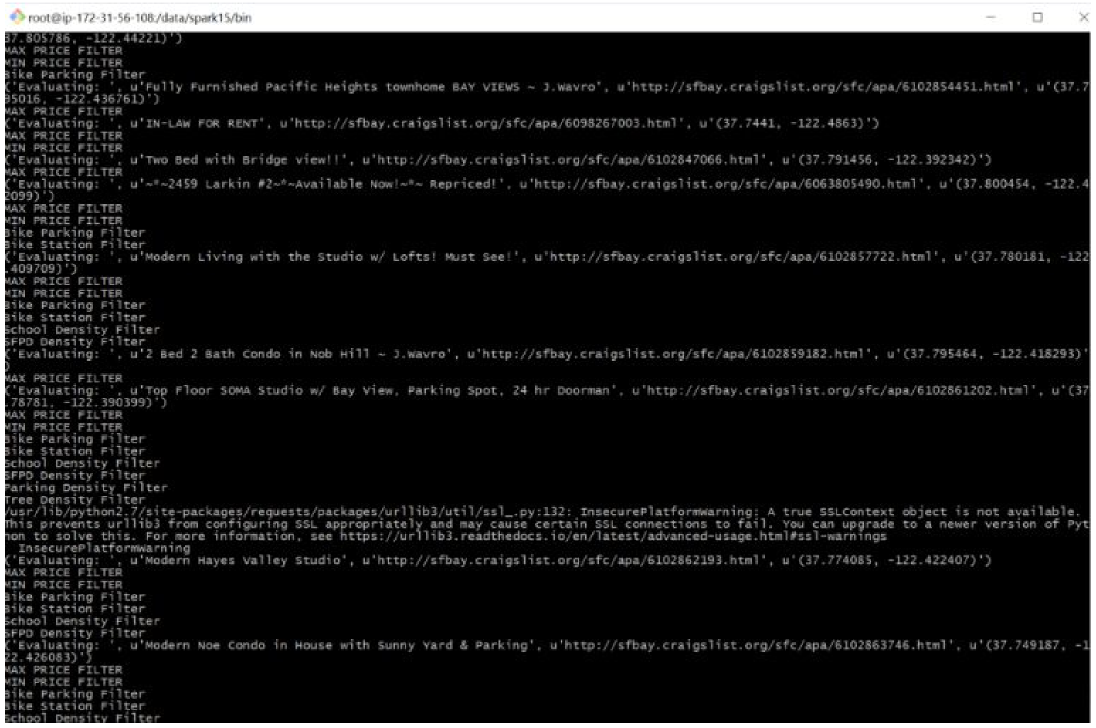
sfhousehunter APP 6:11 PM
That's all the questions I have for you, check back in a few minutes as I begin my search.

Example of User Output to a .txt file


w205@ip-172-31-56-108:~/w205_Final_Project_Complete/slackbot

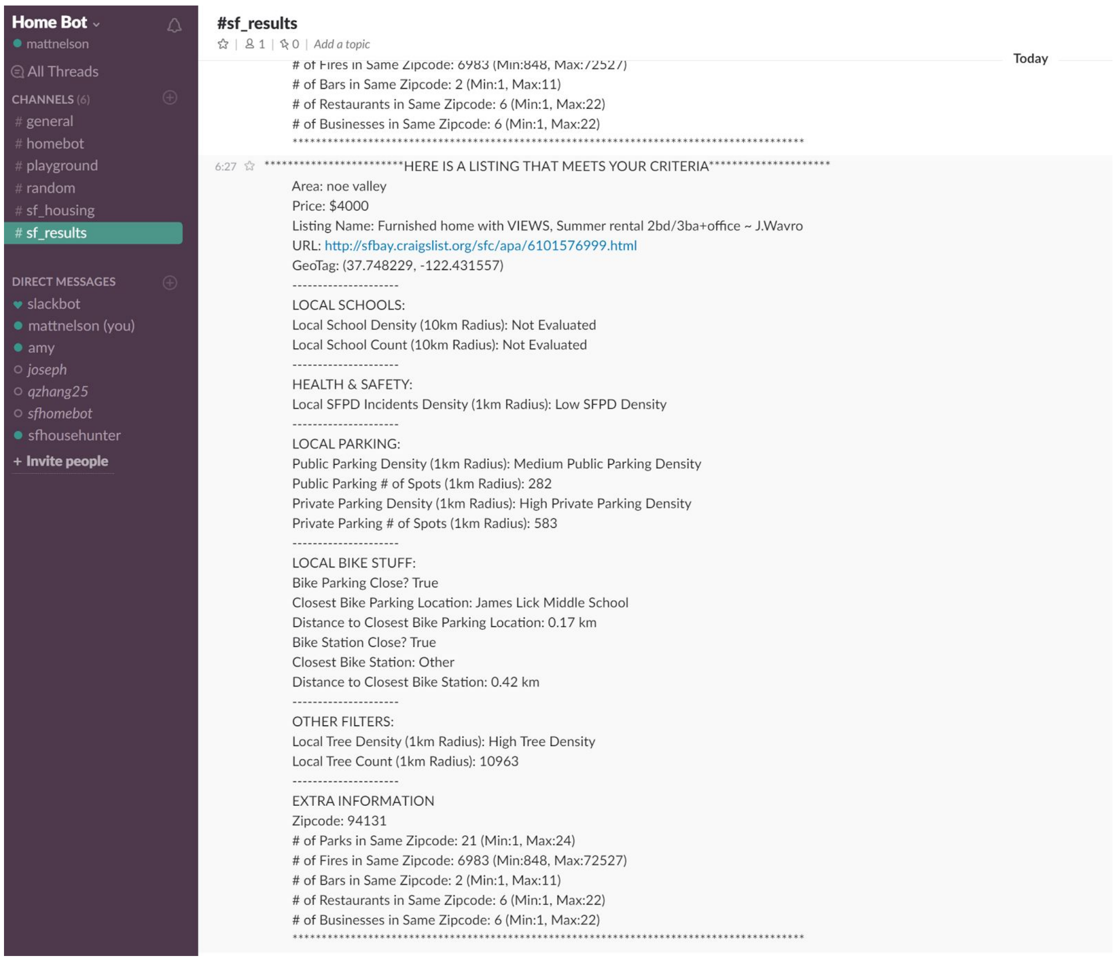
```
[w205@ip-172-31-56-108 slackbot]$ ls
bot.py  chronjob.sh  chrontab.txt  print_bot_id.py  sfhomebot.py
[w205@ip-172-31-56-108 slackbot]$ ls
bot.py  chronjob.sh  chrontab.txt  print_bot_id.py  sfhomebot.py  sfhomebot.txt
[w205@ip-172-31-56-108 slackbot]$ cat sfhomebot.txt
hello
4000
2000
yes
yes
medium
low
medium
low
[w205@ip-172-31-56-108 slackbot]$
```

Print statements to the screen while filtering scripts run in PySpark


root@ip-172-31-56-108:/data/spark15/bin

```
37.805786, -122.44221)')
MAX PRICE FILTER
MIN PRICE FILTER
Bike Parking Filter
('Evaluating: ', u'Fully Furnished Pacific Heights townhome BAY VIEWS ~ J.Wavro', u'http://sfbay.craigslist.org/sfc/apa/6102854451.html', u'(37.7
95016, -122.436761)')
MAX PRICE FILTER
('Evaluating: ', u'IN-LAW FOR RENT', u'http://sfbay.craigslist.org/sfc/apa/6098267003.html', u'(37.7441, -122.4863)')
MAX PRICE FILTER
MIN PRICE FILTER
('Evaluating: ', u'Two Bed with Bridge view!!', u'http://sfbay.craigslist.org/sfc/apa/6102847066.html', u'(37.791456, -122.392342)')
MAX PRICE FILTER
('Evaluating: ', u'~*~2459 Larkin #2~^~Available Now!~*~ Repriced!', u'http://sfbay.craigslist.org/sfc/apa/6063805490.html', u'(37.800454, -122.4
2099)')
MAX PRICE FILTER
MIN PRICE FILTER
Bike Parking Filter
Bike Station Filter
('Evaluating: ', u'Modern Living with the Studio w/ Lofts! Must See!', u'http://sfbay.craigslist.org/sfc/apa/6102857722.html', u'(37.780181, -122
.409709)')
MAX PRICE FILTER
MIN PRICE FILTER
Bike Parking Filter
Bike Station Filter
School Density Filter
SFPD Density Filter
('Evaluating: ', u'2 Bed 2 Bath Condo in Nob Hill ~ J.Wavro', u'http://sfbay.craigslist.org/sfc/apa/6102859182.html', u'(37.795464, -122.418293)'
)
MAX PRICE FILTER
('Evaluating: ', u'Top Floor SOMA Studio w/ Bay View, Parking Spot, 24 hr Doorman', u'http://sfbay.craigslist.org/sfc/apa/6102861202.html', u'(37
.78781, -122.390399)')
MAX PRICE FILTER
MIN PRICE FILTER
Bike Parking Filter
Bike Station Filter
School Density Filter
SFPD Density Filter
Parking Density Filter
Tree Density Filter
/usr/lib/python2.7/site-packages/requests/packages/urllib3/util/ssl_.py:132: InsecurePlatformWarning: A true SSLContext object is not available.
This prevents urllib3 from configuring SSL appropriately and may cause certain SSL connections to fail. You can upgrade to a newer version of Pyt
hon to solve this. For more information, see https://urllib3.readthedocs.io/en/latest/advanced-usage.html#ssl-warnings
  InsecurePlatformWarning
('Evaluating: ', u'Modern Hayes Valley Studio', u'http://sfbay.craigslist.org/sfc/apa/6102862193.html', u'(37.774085, -122.422407)')
MAX PRICE FILTER
MIN PRICE FILTER
Bike Parking Filter
Bike Station Filter
School Density Filter
SFPD Density Filter
('Evaluating: ', u'Modern Noe Condo in House with Sunny Yard & Parking', u'http://sfbay.craigslist.org/sfc/apa/6102863746.html', u'(37.749187, -1
22.426083)')
MAX PRICE FILTER
MIN PRICE FILTER
Bike Parking Filter
Bike Station Filter
School Density Filter
```

Responses pushed to Slack Channel



Home Bot
mattnelson
All Threads
CHANNELS (6)
# general
# homebot
# playground
# random
# sf_housing
# sf_results
DIRECT MESSAGES
slackbot
mattnelson (you)
amy
joseph
qzhang25
sfhomebot
sfhousehunter
+ Invite people

#sf_results
1 | 0 | Add a topic
Today

```
# of Fires in Same Zipcode: 6983 (Min:848, Max:72527)
# of Bars in Same Zipcode: 2 (Min:1, Max:11)
# of Restaurants in Same Zipcode: 6 (Min:1, Max:22)
# of Businesses in Same Zipcode: 6 (Min:1, Max:22)
************************************************************************************
```

6:27

```
***********************HERE IS A LISTING THAT MEETS YOUR CRITERIA********************
Area: noe valley
Price: $4000
Listing Name: Furnished home with VIEWS, Summer rental 2bd/3ba+office ~ J.Wavro
URL: http://sfbay.craigslist.org/sfc/apa/6101576999.html
GeoTag: (37.748229, -122.431557)
--------------------
LOCAL SCHOOLS:
Local School Density (10km Radius): Not Evaluated
Local School Count (10km Radius): Not Evaluated
--------------------
HEALTH & SAFETY:
Local SFPD Incidents Density (1km Radius): Low SFPD Density
--------------------
LOCAL PARKING:
Public Parking Density (1km Radius): Medium Public Parking Density
Public Parking # of Spots (1km Radius): 282
Private Parking Density (1km Radius): High Private Parking Density
Private Parking # of Spots (1km Radius): 583
--------------------
LOCAL BIKE STUFF:
Bike Parking Close? True
Closest Bike Parking Location: James Lick Middle School
Distance to Closest Bike Parking Location: 0.17 km
Bike Station Close? True
Closest Bike Station: Other
Distance to Closest Bike Station: 0.42 km
--------------------
OTHER FILTERS:
Local Tree Density (1km Radius): High Tree Density
Local Tree Count (1km Radius): 10963
--------------------
EXTRA INFORMATION
Zipcode: 94131
# of Parks in Same Zipcode: 21 (Min:1, Max:24)
# of Fires in Same Zipcode: 6983 (Min:848, Max:72527)
# of Bars in Same Zipcode: 2 (Min:1, Max:11)
# of Restaurants in Same Zipcode: 6 (Min:1, Max:22)
# of Businesses in Same Zipcode: 6 (Min:1, Max:22)
************************************************************************************
```

APPENDIX:

Resources

https://sfbay.craigslist.org/search/sfc/apa

https://data.sfgov.org

http://www.vitalsigns.mtc.ca.gov/

https://data.sfgov.org/Geographic-Locations-and-Boundaries/Analysis-Neighborhoods/p5b7-5n3h

https://data.sfgov.org/Geographic-Locations-and-Boundaries/San-Francisco-ZIP-Codes/srq6-hmpi

https://www.dataquest.io/blog/apartment-finding-slackbot/

GitHub Repository: Code

Setup System

https://s3.amazonaws.com/ucbdatasciencew205/setup_ucb_complete_plus_postgres.sh