Sprint one:

In this sprint the goal is to use Python code to scrape station and occupancy data from JCDecaus API. The requiring data are station information and real time bikes occupancy. After analysis json sample provides from the JCDecaus website we divided our task into three part in this sprint – (1) create instance RDS on AWS and python code for link to RDS database and create database environment, (2) python code to scrap data from API, (3) integrate python code and generate package of python code for keep scraping data every five minutes on EC2 instance and track the program run and database situation.

During Sprint:

Chen (1)

A little revise from following the lecture slide to create RDS instance. While create instance if select “not publicly accessible” will encounter cannot link to database even open inbound rule to all traffic. After revised this problem, able to successfully use pymsql package to link to database. In order to check linking status, add print out error message to tracking.

Based on the sample json file from the JCDecaus API, we decide to create two table – one for static station data, one for real-time data. Since in real-time data there are duplicate information can get from static station, we decide to not to keep this data in real-time table. We use MySQLWorkbench to generate empty table first then use python to insert data. Furthermore, since the data insert could double insert (ie, duplicate data) we use MySQL built in function to fool proof this possibility. That is, we use primary key on station number in table station and unique on “number (station number) + last\_update” which is no more one row of a certain station in certain last update time exists.

Wen-Ting (2)

The task is to periodically scrap data from JCDecaus API and insert data to corresponding table in database. The API provide two type of information: static station and real-time station data. We assume that the static data will not change in short time so we separate scraping python code into two modules – static.py and dynamic.py. The static one will only trigger by manually request. Since the dynamic will be run every five minutes and here don’t have to worry about whether every five minutes the data on API is updated or not (because we already guard this situation in database), the program here only need to check if the data is inserted successfully or not. Also, concerning there are no enough space in EC2 instance, we first store the scraping data into pickle file with filename include runtime and then insert data into instance, in case unsuccessfully insert due to unexpected reason. Moreover, in order to be able to track the error if happen we decided to add on more column in dynamic table – data insert time. We also find out that the update date data type that API provide is timestamp, so we keep the original timestamp information and add another column that is converted to date time in convenience for the following sprint.

Lan (3):

After setting up database environment and python code for scrapping data, the final step is to integrate into a main module for continuously run on EC2 instance. In order to be able to track different kind of error during the scraping stage and still can let the program constantly run on the instance without interrupt. In the main module, we add some exception detection and print out error message. While trying on local machine, it can clearly see the output message in shell and do certain handling strategy. However, we found out remote to EC2 instance it can not easily to keep trace the exception be thrown on the screen and also not easily to constantly keep eye on it in long run. Therefore, between choosing the error message record or keep the file that scrap from API before insert into database. We decide to keep the data on EC2 every time we do the json query from API. Even though we will not so clearly understand the exception is due to what reason, but we can mend the loss data after then.

Sprint one end review:

In this sprint we divided task into three parts, and one task can only start after the previous part is finished. Also, some functions are found out need to be revised after the other task is started. Then we need to go back and do some adjustment which take more time we expected to finish the first sprint. Therefore, in the next sprint we will try if can divide the task that can be work in parallel and do some testing in others’ code to avoid individual blind spot.

Sprint two:

In sprint two, our target is to use flask to build the server and use javascript to manipulate data get from server. In this stage the server is run on local machine and the website can show the station information on google map and the statistical chart on each station when the user click on the marker of the station. We divided the task into four sub-stage – (1) use bootstrap to build the web framework, (2) add google map on website, (3) add marker on map by using flask to return data to the web page, (4) click on marker and show the real-time information.

Chen:

Compare to previous sprint, in this sprint we work on the same sub-task together, one do the coding and the other two do the structure research and require material collection. After the sprint we find out this pattern is more efficient than the previous one and less revision to the previous take in each sub-stage. This sprint we spent more time on discussion of the whole structure could work. That is, how real-time data should response to the query from the web page. Is it better to return all information of all stations or better to response based on the station number that web page query by using POST/GET from the form. However, we found out it is hard to revised once we choose the certain structure. The way need to be stick to the end even could be less efficient, otherwise, need be started from the beginning.

Wen-Ting:

This sprint we revised the working pattern from sprint one. In sprint one, we spent more time on waiting previous sub-tasks to be finished and more go-back revision due to some issue we encounter on sub-tasks afterward. This sprint we have to work on some new technique that we never use before, such as flask, bootstrap and jquery. However, the whole process works more fluently than previous sprint. We work on the area that we can do more efficiently and give the feedback to each other. Also, we use cross validation to manually test the code to prevent from some issue that one people did not considered. The other things we found helpful in this stage is to use testing script before we start the actually coding. This can help us to more clearly what the outcome should perform when the sprint is over. Also, can more easily to consider more details while we create some functions.

Lan:

Since we adjust the working pattern in this stage, the most difficult part is not to integrate different codes together but to decide the structure that link the database, flask and javascript together. At the beginning of the stage, we need to decide the job of manipulating data should be done in flask or in javascript, and how the data query from database should forward to javascript. Also, we found it is hard to change the structure once we decide to use certain model. For example, we are choosing between whether to click on certain station and use http POST request to pass the station number to server and response to the web page based on this query, or dump the json string of whole week and let javascript to dynamic response to the event on the web page. Once we decide the second way, it is hard to change this pattern for the following sub-task.

Sprint review:

This sprint, we basically do all work together which found out is more efficient and less error than first sprint. However, in first sprint we have more efficient utilization of the function of GitHub. In this sprint less commit log during the development, which we found out the poor version control in this sprint. Sometime will encounter the situation that when require to go back to certain revision before, found out to be hard to remember what kind of change we made and where the code is changed.

Sprint three:

In final sprint we need to add statistical chart and show on web page once user click on the station marker and the weather information on the web page. After the flask web page is completed, we need to set the server on EC2 for publically access. Since the main structure had completed in Sprint two, we divided the tasks into three sub-task and work in parallel – (1) google chart for each station, (2) weather information, (3) server setup on Ec2

Chen:

Due to the AWS credit limitation, we store two weeks real time data in database (from March 20th to Arpil 4th). Therefore, the chart is summarized based on the data from these two weeks. However, there are an unexpected situation happened in this sprint. That is, the account is closed due to reach the limit grant. So we need to import data to database again. However, the import process require more than one days so we decide to leave data for only one week. It’s more faster to import to local machine so we import whole data to local machine as backup plane.

Wen-Ting:

In this sprint we need to add weather information on the backend webpages. The original idea is to show the hourly weather information under the google chart so user can see how the weather condition can influence the available bike stands. However, due to the open weather API only provide current time weather information and we didn’t do enough research at the first sprint so we have no enough data to do this function. Therefore, we delete this function revise to only show the current weather information on the side page.

Lan:

In this sprint, due to the limitation of the AWS grant we need to use the csv file that we backup from the previous database and import to the new created database. However, the import procedure take really long time to run on local machine by using MySQLWorkbench. Therefore, we try different way to fix this problem. Finally, we found out by using python code run insert query on EC2 can done in one hour. Even we can put all data back to new RDS database, we still left only one week for AWS usage saving.

Note:

Not properly use github

Not so efficient