DUBLIN BIKE MONITOR

UCD School of Computer Science

COMP30830 Software Engineering

Menghao Su-17210074

Renjie Fu-17211164

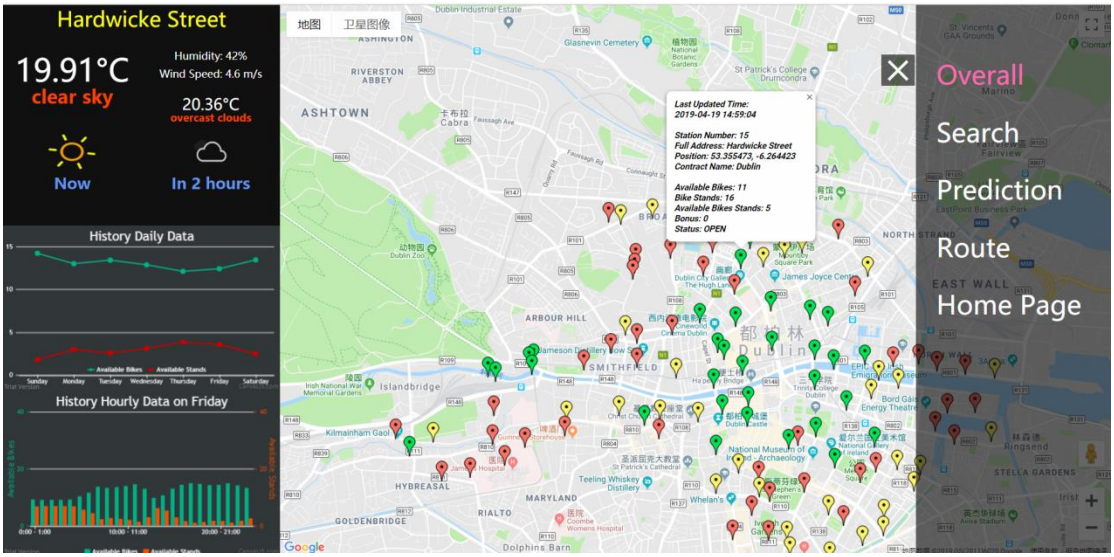
Anna Xuejiao Ge -18200158

April 18, 2019

1. Project Overview

1.1 Introduction

Dublin Bike is a public bicycle rental scheme which has operated in the city of Dublin since 2009. As a user-friendly software, Dublin Bike Monitor is mainly used to monitor, analyze and predict the Dublin Bike availability based on the historical data and give guidance for user on their travel plan.



All of the code/graph/notes are available on GITHUB at:
https://github.com/sedublinbike/SE
And the working EC2 address of the live website is:
http://35.160.137.228:5000
We designed this application with Windows 10.

The contribution of our group team member is:

Name	Contribution
Menghao Su	34%
Renjie Fu	33%
Anna Xuejiao Ge	33%

1.2 Objective

The main object of this software is to give users an overall understanding of the distribution and coverage of the Dublin Bike, and encourage more people to travel by bike, achieving ‘Green transportation’ as well as ‘Smart city’.

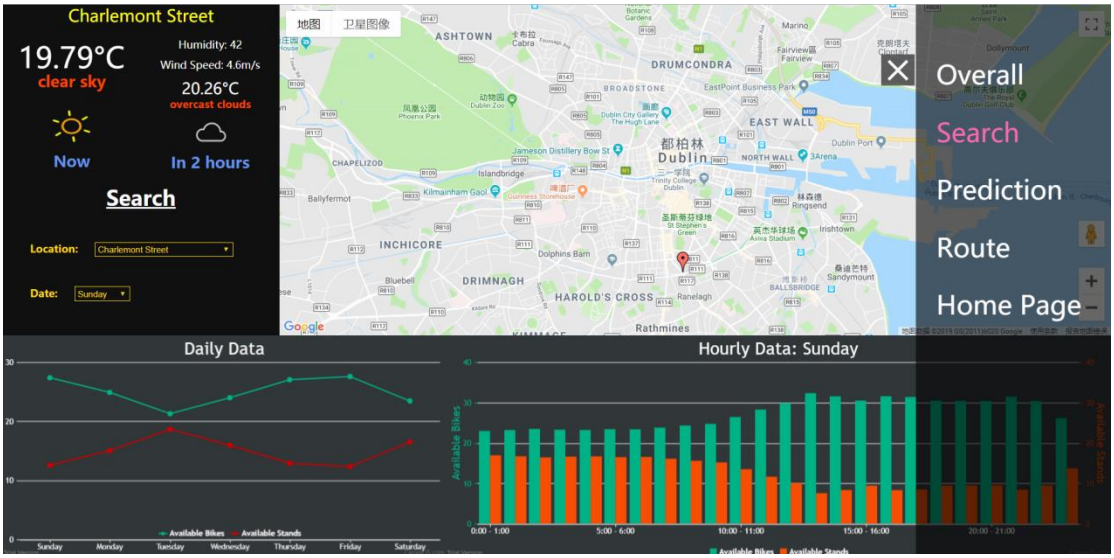
1.3 Target

This software helps users to:

1) Get a better look at the usage status and availability of Dublin Bike
2) Get some suggestion on how they can get a bike from a nearby station with sufficient bikes, as well as, in a nice weather.

1.4 Function

There are five main functions of this software:
1) Display the distribution of Dublin bike
2) Show the availability and weather condition of each station
3) Predict the future weather and availability of a specific station
4) Show the analytics tendency of each station hourly and daily
5) Schedule the route for users with a give start point and end point

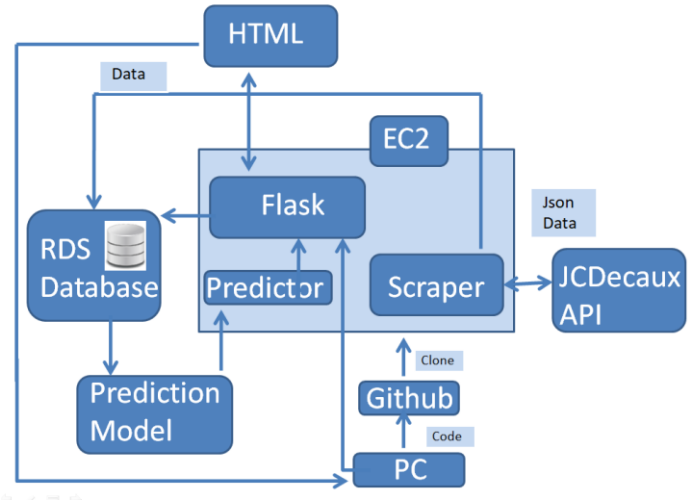


1.5 Feature

1) Cloud-based: This software deploys on the AWS RDS and EC2 running 24/7.
2) Chart-visualization: Chart is widely used in this software to give users a direct and clear picture of the availability hourly and daily. Different color of the markers is used to show the different availability of each station in the map.
3) Prediction model: We compared the effect of four different models after cleaning the data and comparing four different models.
4) User-friendly interface and dynamic weather icon:

1.6 Structure

The Scraper on EC2 is running continuously and gets data through JCDecaux API all the time before putting the data into AWS RDS. Programmers pulled the Flask template and front-end HTML pages into Github and cloned it into EC2. For the prediction feature, we trained four different models locally and used the Random Forest in predictor.py and deployed it in the cloud.



2. Team Organization

2.1 Teamwork

In this project team, there were three students cooperating together. At the beginning of the whole project, three of us were not quite familiar with each other. But we got a pretty high efficiency in discussion and teamwork through the project, and there were two main reasons for that:

First reason is that we set up a really great communication style and method to solve disagreement. For instance, when we design the interface of the website, three of us had completely different opinions. Su Menghao preferred to use four different pages as homepage, map_all, search_station, predict_future and route_scheduling separately. While Anna would like to put all functions into one single page and allow users to drop down and see their search results after clicking on submit button. Fu Renjie preferred to use single page but show the availability and prediction results in a popup. We tried all the possible methods and finally decided to use Su's suggestion, as we continually added in new features and this design model could make each feature relatively independent and cause less effect to the others features.

The second reason is the way we allotted our workload. We did not assign a big part such as back-end/front-end to a specific team member, as we felt it actually took longer to solve the problem in this way. Once someone met a particular technical difficulty, and others might not be able to know what was going on or provide suggestion to solve the problem. Instead, we always had a discussion during the sprint planning and divided every big part into several subtasks. Let's take an example of the prediction feature. After we discussed the data structure of dataset together, we split the prediction part into many sub tasks and then assigned the sub

tasks to each team member. Su was in charge of cleaning the data, Anna and Fu trained two predict models separately and then Fu deployed the model into the website, while Su would build the related UI HTML for that request. In this way, we could learn the needed skills together and help each other to improve. Besides, it helped us to fully understand what and how the other's part is look like so that we got a higher efficiency in debugging and communication.

Besides, we divided the whole project period into four sprints and made sure that every team member had an opportunity to be the sprint master of at least one sprint. Therefore, every member could learn Scrum/Agile and know how to set up daily stand up meeting and organize team discussion.

Here is an overview of four sprints:

Sprint	**Master**	**Main Tasks**	**Time estimated**	**Actual time**
One	Anna Xuejiao Ge	1) Set up the EC2 and RDS, 2) Set up group Github account 3) Learn SCRUM and Agile 4) Design data structure and get data through API	1) 3h 2) 0.5h 3) 2h*3 4) 3h	21h
Two	Anna Xuejiao Ge	1) Learn Flask basic knowledge 2) Achieve the back-end template using FLASK 3) Basic UI design 4) Connect Flask, RDS and HTML website	1) 30h 2) 20h 3) 20h 4) 40h	106h
Three	Menghao Su	1) Achieve analysis based on hourly and daily chart 2) Add in weather API 3) Beauty UI	1) 25h 2) 30h 3) 15h	72h
Four	Renjie Fu	1) Prediction of availability 2) Schedule the route 3) Beautify UI 4) Deploy the website on EC2 5) Project Report	1) 30h 2) 25h 3) 20h 4) 2h 5) 20h	111h

We all felt grateful to be in such a fantastic team and work together. Everyone in our team showed a humble attitude and everyone learnt quite a lot from each other. We agreed on and obeyed the way/mechanism to solve disagreement from the beginning to the end. When we had some different ideas within our team, such as looking for a weather API, we always tried multiple ways to achieve the goal, and tried to find a common/nice way to understand each other's thinking without causing misunderstanding, and then tested the ideas one by one to see whether it

made sense or not. If not, we tried the others, and finally we would find the most efficient solution.

All of our team members were willing to share our own opinion and we all enjoyed learning more and contributing more to our project. We all enjoyed sharing new ideas on what we just learned with other teammates and other students. And when we reviewed the whole process of the development, we felt it was just how a good team should be look like.

At the end of this project, we reviewed what we have achieved. Beyond learning the technical skills such as cloud service, Flask (back-end) and modeling, we realized that one of the best achievements of this project was learning and attempting to cooperate with others efficiently under advanced team cooperation methods: Agile and Scurm.

Huge thanks our lecturer Lawlor Aonghus and tutor Karl Roe (as product owner) for their nice and patient guidance and help. Lawlor and Karl both helped us to understand the whole project goal and get a big picture for it. Karl assisted us to split the project into four sprints and set goal/main tasks for each one as well as giving us meaningful suggestions while debugging. Lawlor helped us to understand the necessary knowledge on scrum, flask, could service, the structure, design pattern and real case, which are used in our project as well as pointing us a pathway about what we should continue to learn in the future.

2.2 Time Management

In order to utilize four sprints efficiently and achieve the goal on time, it was necessary for our team to have a good time management style. And we felt there were three basic areas in time management:

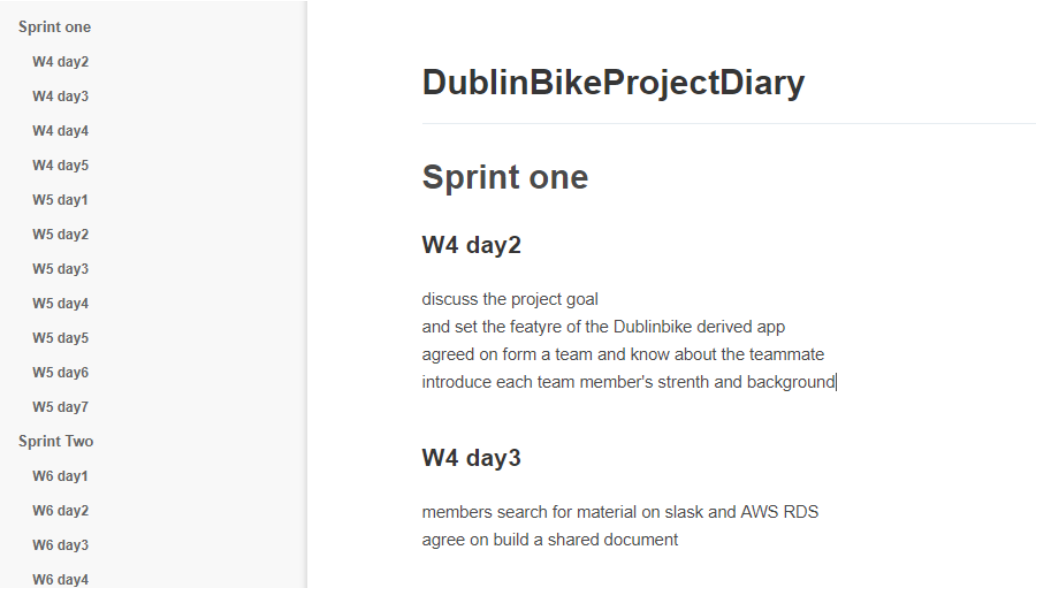
a) The stand up meeting to let every team member know how is the project going on.
b) The way to solve the clash on time with other modules (multi-task handling).
c) The way to estimate workload and make adjustment when debugging spent longer time than expected.

a) Stand up meeting and team discussion

As the school timetable scheduled, we have class from Monday to Friday, which means all of our team member could meet each other every workday. For Software Engineer module, we have two labs (on Tuesday and Thursday) and two lectures (on Monday and Thursday) each week. Therefore we could solve some difficulties we met during the lab and communicate with our tutor/product owner Karl.

For the first and second sprints, we insisted on stand up meeting every day and made daily log on Shimo, which is a cloud based text editor that allows user to edit together and generate directory on the left side automatically. For the third and fourth sprints, as the features became more and more complex, we needed more time to work together and discuss on some technical issues such as transferring the data from prediction model into HTML website. Therefore we agreed to add more hour-long team discussion in library study rooms. Besides, we kept a good record style and made fully usage the tool such as Trello and Shimo as well.

Our daily meeting record/log on the Shimo editor is available at:
https://shimo.im/docs/XEjbgB7qkwYufO6j



Additional team discussion in study room:

Time	Location
10am-12pm, 19th March	Group Study Room A, James Joyce Library
9am-10am, 29th March	Group Study Room 1, James Joyce Library
4pm-6pm, 9th April	Group Study Room A, James Joyce Library
12pm-1pm, 12th April	Group Study Room E, James Joyce Library
5pm-7pm, 15th April	Group Study Room 1, Health Center Library
5pm-7pm, 16th April	Group Study Room 2, Health Center Library
9am-10am, 17th April	Group Study Room 5, Health Center Library
10am-3pm, 19th April	Group Study Room F, James Joyce Library

(The group discussion draft and study room booking information are in the appendix session.)

b) Multi-task handling

Due to the fact that this project was not the only one we had during this semester, sometimes we found it was pretty difficult to keep on making contribution to our project every day. For instance, we had four project deadlines and two tests during week 6 and 7, and seven project deadlines plus one lab-test during week 10, 11 and 12.

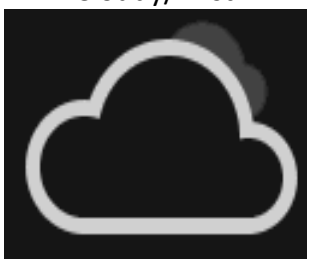
It pushed us to learn how to handle multi-tasks and improve our skill on time management. The solution we got is making an early overall plan about the whole stuff among all modules. As we could know the exam time and project deadline two to four weeks in advance, we took these dates into account when making the sprint planning. Besides, we agreed on arranging more self-study tasks such as basic understanding of Flask before mid-term during sprint two, and utilize one week during the reading week to work on our project together.

c) Estimation of workload

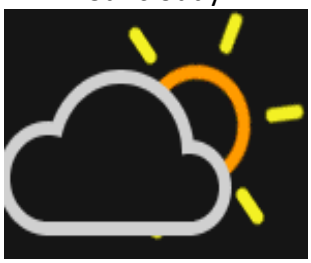
Our team found that it was a little hard to estimate time/workload when we never did it before. And we found some tasks/sub-tasks took longer time than we expected to be. Besides, it was a little tricky to predict the time needed to debug when our application became more and more complex as well. Therefore at the beginning of each sprint, we tried to arrange some extra time for each main task so that allocating enough time for debugging and learning new stuff.

For example, the dynamic weather icon is one of our best achievements in this project. In order to show weather condition dynamically, we generated seven different weather icon animation based on the real weather condition of Dublin that we generated from the database. Then we used gif tool to unify the eight icons' size. And we used the API called OpenWeatherMap to get weather information of each station, rather than the city, and showed the weather condition of both now and two hours later. At the beginning, we felt that weather was not a quite tough feature in our project. But later on we realized that it took a bit longer than we expected in creating gif files, selecting weather API, transferring parameters and so on.

Cloudy/Mist



Suncloudy



Snowy

Sunrainy(mixed)

Rainy

Lightning



Sunny

Although we spent longer time in weather feather, we felt it was worth to do so and we achieved a better effect than just using a simple weather plug-in. What's more, we felt that we learnt quite a lot while designing our own weather feature while presenting the weather condition of each station on website, and got a better understanding of how to connect front-end and back-end through JSON.

When we reviewed the whole project at the end, we found that we improved quite a lot on time management skill. In the first and second sprints we just achieved some basic tasks. While in the third and forth ones, we didn't only achieve some complex features, but with less time for each feature as well. It also proved that our efficiency improved quite a lot after this project.

3. Process

3.1 Overview

Scrum is an Agile framework for managing project, especially software development. In our project, three students worked together as a team on four sprints, each sprint was two weeks long, except sprint two for three weeks. Every team member got an opportunity to be sprint master and hold stand-up meeting, sprint planning and sprint review. And we had a project retrospective at the end of this project.

3.2 Sprint One

As we started to learn Scrum during sprint one, there is no sprint planning for sprint one and we started to hold stand up meeting since Tuesday in the first week of sprint one (Week 4).

3.2.1 Backlog

There are four main tasks in sprint one and they could be divided into ten subtasks. Here is a backlog chart:

Spring one Backlog

Sub Task	Responsible	4月1	4月2	4月3	4月4	4月5	5月1	5月2	5月3	5月4	5月5	Subtotal
1. Form a team and introduce each other	all		2									2
2. learning Scrum and Agile	all				2	2						4
3. understand the project goal and need	Anna Xuejiao Ge	1	1									2
4. Set up Github team account and add in all the team members	Fu Renjie							0.5				0.5
5. Buile Trello and Shimo shared account	all			0.5	0.5							1
6. Set up EC2 on AWS and share key with all team members	Su Menghao			1								1
7. Get data from JCDecaux API	Su Menghao						1					1
8. Design data structure	Fu Renjie						1		1	1		3
9. Set up AWS RDS and share key with all team members	Anna Xuejiao Ge							1	1			2
10. Deploy scrper.py on EC2 and link EC2 and AWS RDS	Fu & Su							1		1	1	3
11. Sprint Review	Anna Xuejiao Ge										1.5	1.5
	Subtotal	1	3	1.5	2.5	2	2	2.5	2	2	2.5	21

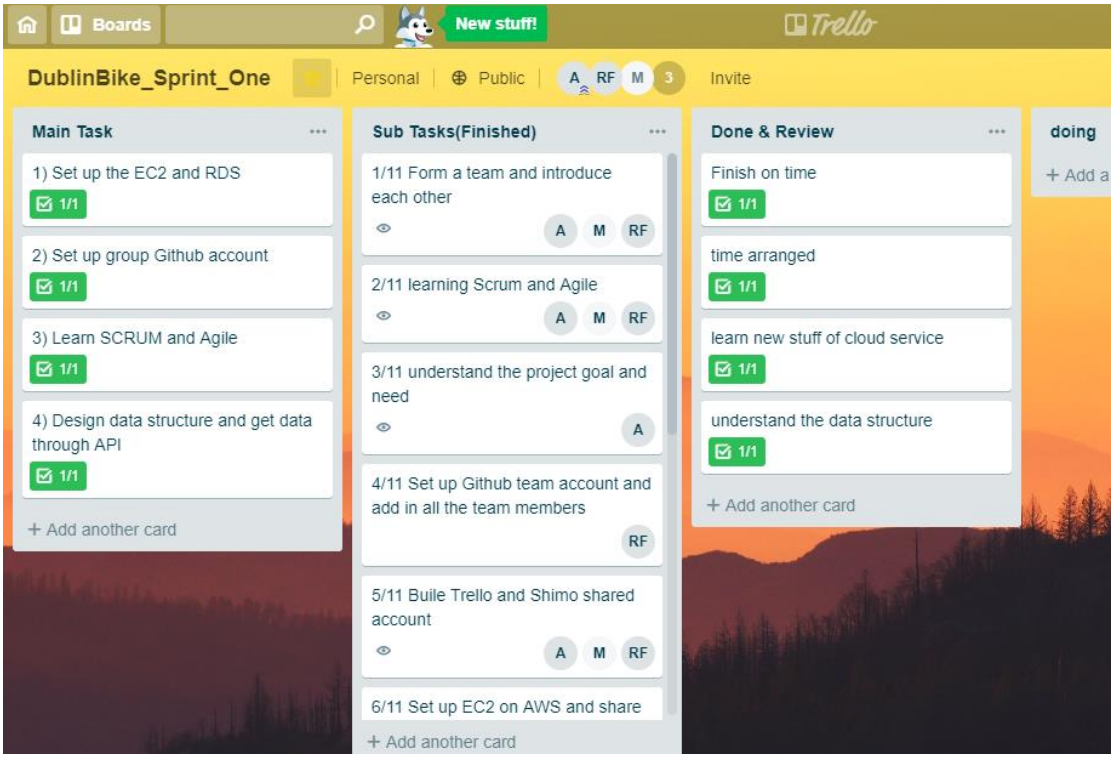
3.2.2 Problem encountered

When deploying scrper.py onto EC2, it failed multiple times due to lack of some packages. And the reason was that we didn't install all the needed packages (using pip command) when setting up/initializing EC2 instance. And the key is to find which package that we lacked is. Actually some packages relating to mysql and pymysql have parent-child relations and only after installing specific packages the needed one could be installed, just like how the Linux packages related with each other.

There are three ways to help us solve this problem. First is to reinstall all the needed packages in the vertical environment. Second is running the python file on local environment successfully and then upload the whole environment which containing all the needed packages for sure onto the virtual environment (EC2). And the final method is to use yml and check which package caused this problem in makeup language. Later on, we added in a line of code in the scrper.py that appointed to the MySQLdb package and then the file running successfully. The command is 'pymysql.install_as_MySQLdb()'.

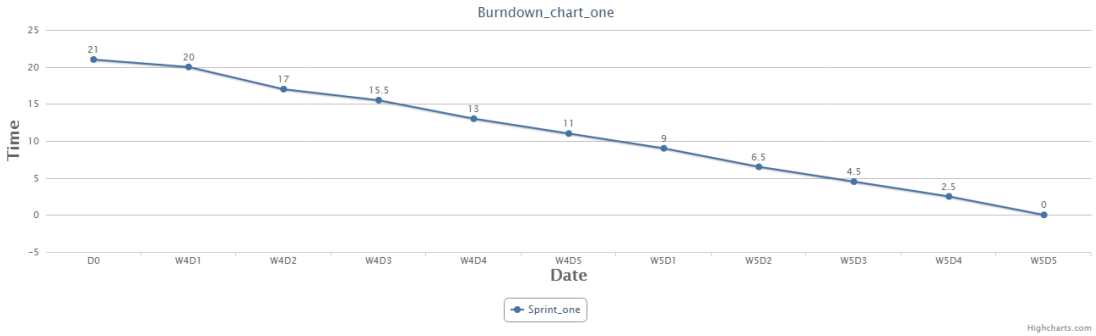
3.2.3 Sprint Review

In this sprint we achieved four main tasks and eleven subtasks:



Review: In this spring, we had a good start of the whole project. And we successfully completed all the planning tasks on time. We just set up AWS EC2 and RDS, and in the next sprint, we need to get more familiar with the cloud service, especially the VPC group. We did pretty well on understanding the data structure of the data we got from Dublin Bike through API and we hope it would help us to do a better job in the following sprints.

And the Burn-down chart is as followed:



From the chart we can find that we arrange all workload quite balancing, which represents that we did a well job on estimating workload at the sprint planning.

3.2.4 Sprint Retrospective

Consolidated list of factors that worked well	Consolidated list of factors were problematic

during the sprint just completed	during the sprint just completed
Good communication style and coordination among team members	We didn't fully understand the way of Scrum working
Using Trello to analyze tasks	Underestimated tasks and debugging
Even distribution of workload across two weeks	Slow in deploy EC2 and RDS as we were not familiar with them

3.3 Sprint two

3.3.1 Sprint Planning

At the beginning of sprint two, we knew that there would be many project deadlines and tests in the following two weeks and then we all agreed to arrange more tasks during the reading week while doing some self-learning tasks of Flask during week 6 and 7. Therefore, this sprint is three weeks long and we mainly focus on the understanding of basic Flask Routing/Decorator/Template during week 6 and 7, as well as getting Google Map through API. And then we built our own Flask frame and connect to RDS and HTML/JS in the spring holiday.

And we agreed to do four main tasks together in sprint two:

1） Learn Flask basic knowledge.
2） Achieve the back-end template using FLASK.
3） Basic UI design (drop-down menu etc.).
4） Connect Flask, RDS and HTML website.

Then we divided the main tasks into nine sub tasks and allotted them to three team members. Please see the following backlog for more details.

3.3.2 Backlog

There are ten subtasks in this sprint and here is a backlog chart below:

Spring Two Backlog

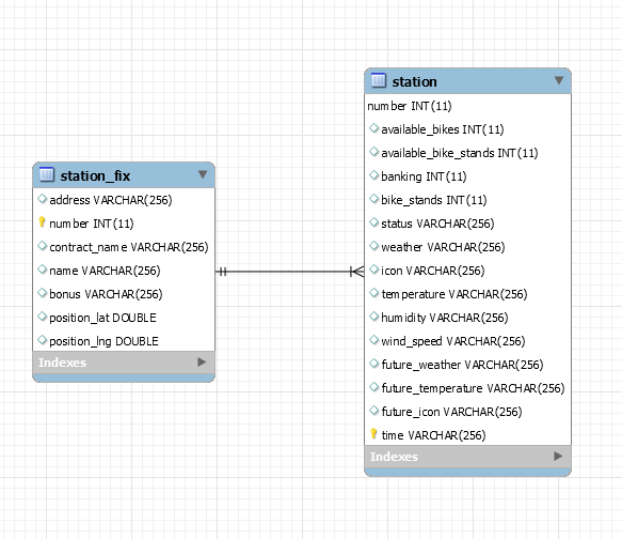
Sub Task	Responsible	W6D1	W6D2	W6D3	W6D4	W6D5	W7D1	W7D2	W7D3	W7D4	W7D5	H1	H2	H3	H4	H5	Sub total
1. Learn the basic knowledge of Flask	all	1			1			1				15	10	12			40
2. Set goal for sprint two	Anna Xuejiao Ge		2														2
3. Build Flask Template + rendering	Su Menghao							4		2							6
4. Build Flask Routing & Decorator	Fu Renjie			2					4	3							9
5. Build Template control flow/loop	Anna Xuejiao Ge						2	2									4
6. Get Google map through API	Su Menghao					2	2	1									5
7. JSON API to connect RDS and Flask and JS	Fu Renjie									2	4	2	5				13
8. Simple UI design(CSS)	Anna Xuejiao Ge						2						3	2	4	2	13
9. Create drop down monu for each station and show the availability	Su & Fu											2	4	4	2		12
10. Spring Review	Anna Xuejiao Ge															2	2
	Subtotal	1	2	2	1	2	6	8	4	7	4	19	22	18	6	4	106

3.3.3 Problem encountered

During this sprint we spent quite a lot time on searching for materials of Flask and Flask templates. It took us a period of time to understand the way of transferring parameters/data between database, Flask and HTML website. We discovered two popular ways to achieve that. First method is using Jinjia2. Jinjia2 is a full featured template engine for Python. We tried to use Jinja2 template to request for data and we also found Jiajia2 has a good function for debugging errors. The second way is getJSON based on JSON API. It could achieve a similar function as well. We built two layers of AJEX, one for the loading of all stations and another one for loading the information of a specific station. And the second is bedded in the first Ajax function. We used the callback function in AJAX to get respond as AJEX request is executed asynchronously.

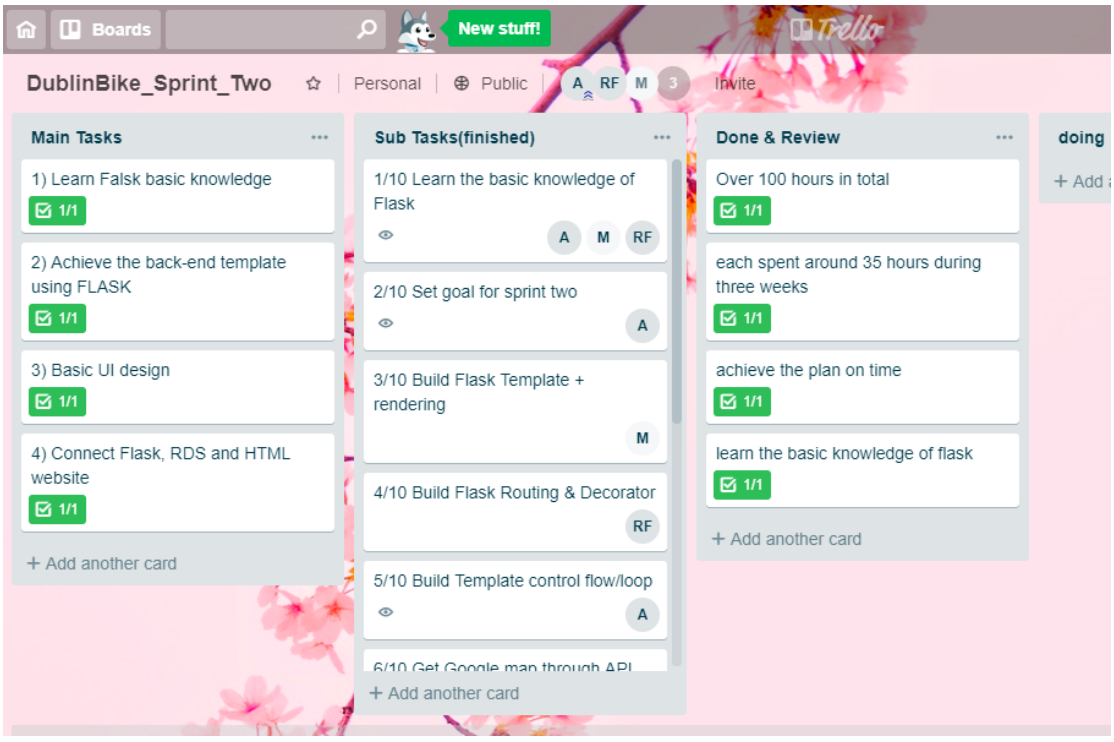
Besides, for the two table in our RDS database, at the beginning, we use the number as key of both of two tables. Then we found errors about ‘duplicate key’ when running the function. Then we realized that the number in two tables was not enough to distinguish a unique key. And then we set a combined key of number and time, while using datetime() to get the time on the moment.

Here is the Entity Relationship (ER) Diagram of our database:



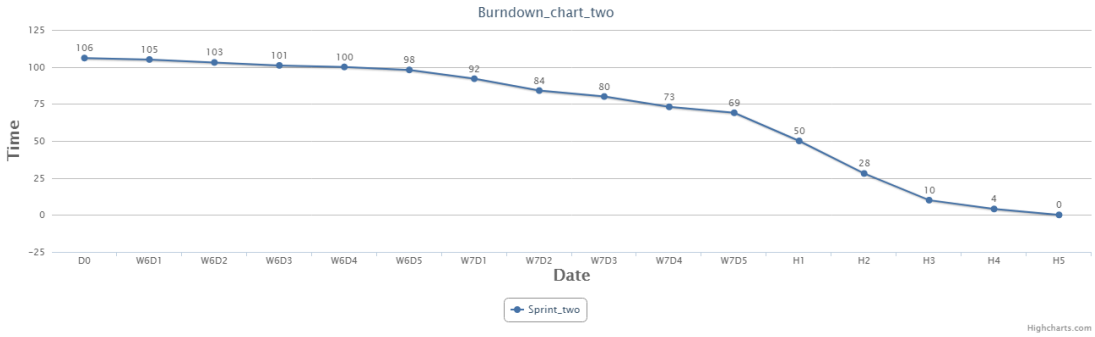
3.3.4 Sprint Review

In this sprint we achieved four main tasks and ten subtasks:



Review: In this sprint, we mainly focus on the basic set up of back-end and frond end. It took us much time to search, learn, connect and debug. In total, we spent over 100 hour over three weeks, including one week in holiday as scheduled in the sprint planning. And we also reviewed what we learnt during sprint one such as the data structure of database etc. We felt that there were still to be down on UI design and beautification. And in the following, while more features added in our application, we could do more in this part.

And the Burn-down chart is as followed:



From the burn-down chart you could find that the line is quite smooth during week 6 and 7 and it is sharper during the holiday, which means we did more in this period. We followed the sprint planning pretty well and finished all the tasks on time.

3.3.5 Sprint Retrospective

Consolidated list of factors that worked well during the sprint just completed	**Consolidated list of factors were problematic during the sprint just completed**
Efficient teamwork	Daily stand up meeting in week 6 and 7 were not efficient enough
Good sprint planning and time management	Underestimated tasks and debugging
High efficiency in self learning	Need more exercise on Flask and the connection between front-end and back-end

3.4 Sprint Three

3.4.1 Sprint Planning

In sprint three, we agreed to do three main tasks together:

a) Achieve analysis based on hourly and daily chart.
b) Add in weather API.
c) Beauty UI.

Then we divided the main tasks into ten sub tasks and allotted them to three team members. Please see the following blockage for more details.

3.4.2 Blockage

There are ten subtasks in this sprint and here is a backlog chart below:

Spring Three Backlog

Sub Task	Responsible	3月1	3月2	3月3	3月4	3月5	4月1	4月2	4月3	4月4	4月5	Subtotal
1. Set goal for sprint three	Su Menghao	1										1
2. Compare and select suitable weather API	Fu Renjie		2	2	3							7
3. Create dynamic weather icon	Anna Xuejiao Ge		2		2							4
4. Deploy the weather icon into frount-end	Fu Renjie			2	2							4
5. Connect weather information in RDS, JSON and frount-end	Su Menghao			2	1	4	5	4				16
6. Unit test	Su Menghao			1	1							2
7. Dessign the logic to build availability charts	Anna Xuejiao Ge						2	1				3
8. Beautify UI	Fu Renjie						2		4	3		9
9. Redisign the website structure and set seoerate pages for different features	Fu Renjie							2	11	7	5	25
10. Sprint Review	Su Menghao										1	1
	Subtotal	1	4	7	9	4	9	7	15	10	6	72

3.4.3 Problem encountered & design discussion

In this sprint, we put forward many different ideas of how to design the analysis feature and how the design the UI in HTML pages. For the analysis part, we discussed about whether we should use all the historical data to generate bar chart/line chart, or just use the latest one month's data to present. Then we decided to use all the historical data because we would like to avoid the effect of some recent event such like St. Patrick's Day, Easter and so on. We selected from many different APIs and finally chose the canvas API as it provide a dynamic generation process.

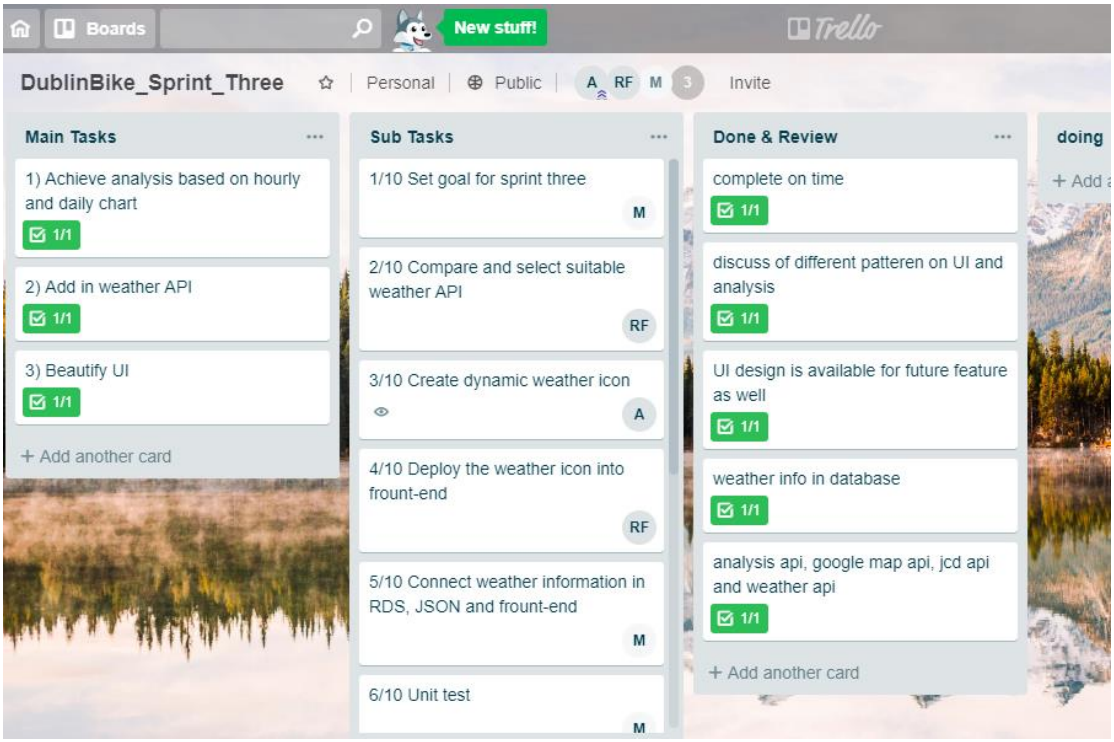
For the UI design, we decided a dominant tone of black and then designed a homepage as an entrance into our application. Then in the overall page, we allowed users to pick on the marker to see the specific information in a pop-up, and show the analysis on the left side.

For the weather issue, first we needed to add the weather information into our database. And then generate JSON file containing the following infomation:
--- "temperature"
--- "description"
--- "temperature_icon"
--- "humidity"
--- "wind_speed"

We also achieved the prediction of weather and temperature in two hour later.

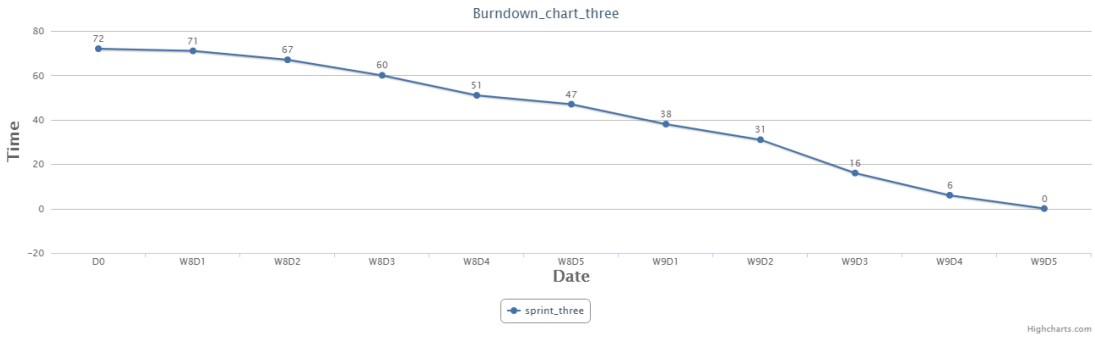
3.4.4 Sprint Review

In this sprint we achieved three main tasks and ten subtasks:



Review: In this sprint, we completed three main tasks on time. We discussed about the design of UI and analysis pattern. And what we did pretty well is review all the API that we used in the first three sprints and start to do the unit test. What we need to approve is to separate the CSS file from HTML file and achieve modularization.

And the Burn-down chart is as followed:



From the chart we can see that we distributed workload across two weeks evenly.

3.4.5 Sprint Retrospective

Consolidated list of factors that worked well during the sprint just completed	Consolidated list of factors were problematic during the sprint just completed
Efficient teamwork	Spent long time in small feature like weather
Distributed workload across two weeks evenly	Underestimated tasks and debugging

High efficiency in daily process	Not familiar with the connection between front-end and back-end and database

3.5 Sprint Four

3.5.1 Sprint Planning

In sprint four, we agreed to do five main tasks together:

1) Prediction of availability.
2) Schedule the route.
3) Beautify UI.
4) Deploy the website on EC2.
5) Project report.

Then we divided the main tasks into fourteen tasks and allotted them to three team members. Please see the following blockage for more details.

3.5.2 Blockage

There are fourteen subtasks in this sprint and here is a backlog chart below:

Spring Four Backlog

Sub Task	Responsible	Subtotal
1. Set goal for sprint four	Fu Renjie	2
2. Clean the dataset and select data from the whole dataset	Su Menghao	4
3. Train and test four different model	Anna & Fu	13
4. Validation and select the model with highest accuracy rate	Fu Renjie	7
5. Pridiction file	Su Menghao	8
6. Geberate JSON file and connect for the prediction	Anna Xuejiao Ge	17
7. Unit test	Su Menghao	2
8. Beautify UI	Anna Xuejiao Ge	15
9. Database Backup	Fu Renjie	1
10. Select Route API	Su Menghao	7
11. Generate Routing Planning	Anna Xuejiao Ge	4
12. Connect the Routing planning to frount end	Fu Renjie	7
13. Deploy the website on EC2	Anna Xuejiao Ge	5
14. Sprint Review and Project report	Fu & Su & Anna	19
	Subtotal	111

3.5.3 Problem encountered

This sprint was the trickiest one among all of the four sprints. For the prediction part, after cleaning and training, we deployed the predictor.py to the website and it would take around one minute to retain the model every day. Besides, when we generated the JSON file which would be requested by the HTML pages, we found the number of station in HTML is from 0 to 113, while in the training model it was from 2 to 19 and

from 21 to 115. And we reordered the number of station in the predictor.py to solve this problem. For the accuracy rate, our accuracy rate increased from 30% (linear regression) to 70% (random forest), but while updating new data, the rate decreased to around 30% again.

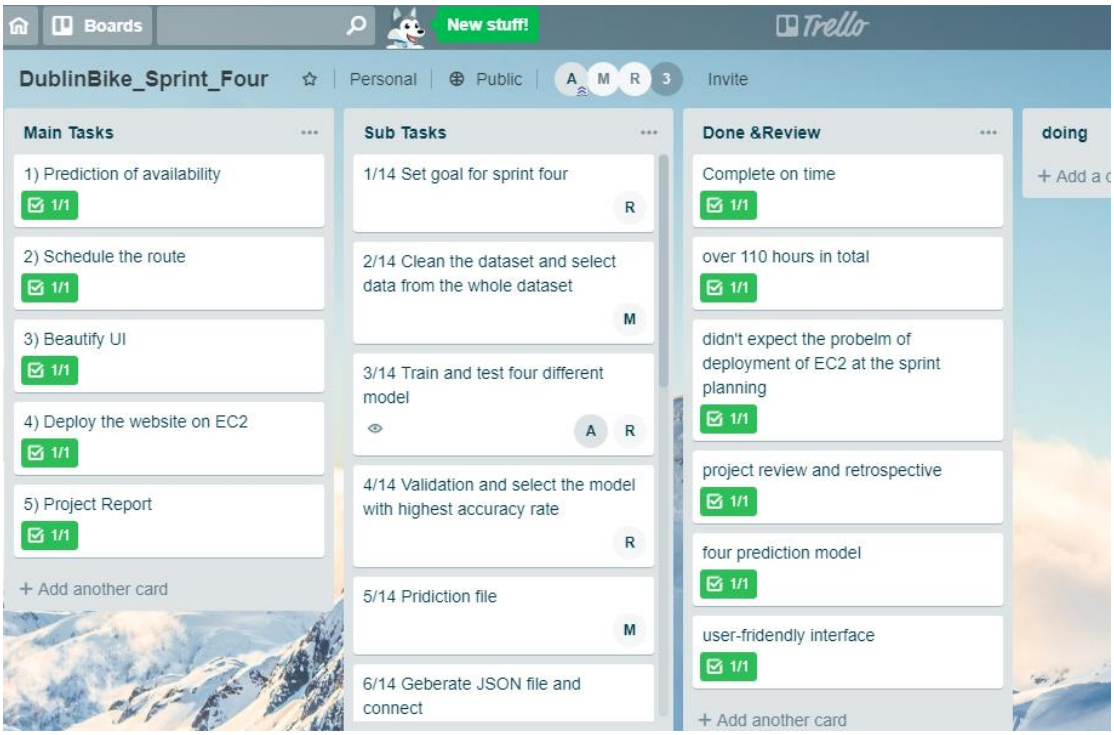
Because of some wrong operation one group damaged the database and lost all their data. Although this problem was not happened in our team but we found it should be paid attention to. And then we created a database backup with over 700,000 rows of information. We also considered about the code backup. While all the code was saved both on Github (with historical edition) and local machine, we thought it was safe enough.

For the test part, as we just learnt some basic unit python test this semester, thus we searched some materials on how to so unit test for Flask.

The biggest problem we encountered is the deployment on EC2. When we upload all the code files and trained models onto EC2, we found that the Google Map could not be shown properly. This was a problem we didn't even imagine previously. We asked our lecturer Lawlor for help and he told us that the proper way to do the website is keeping deploy the latest features onto EC2 all the time rather than upload it only once till the end as there might be some packages/VPC/parameter issues related to that. Thankfully after changed our VPC and variable we got our website shown properly on website.

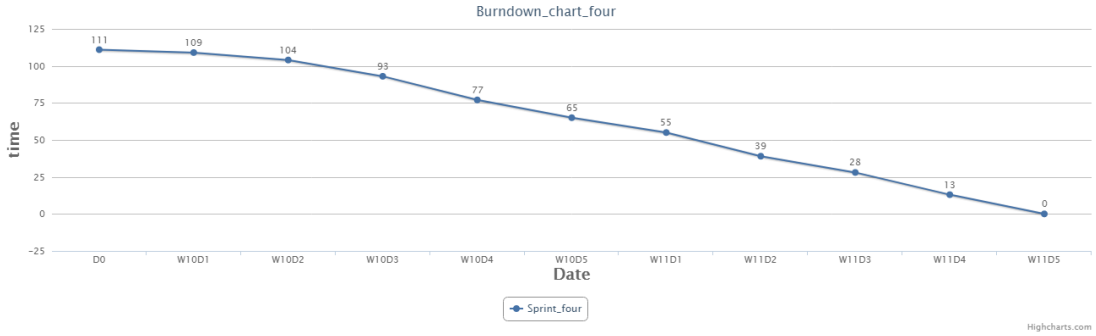
3.5.4 Sprint Review

In this sprint we achieved five main tasks and fourteen subtasks:



Review: In this sprint, we used over 110 hours in total to complete the five main tasks on time. We felt that we achieved quite a lot in this final sprint, including predicting, routing, UI and report. What we didn't expect is the difficulty/trouble we faced when deploying the application onto EC2. We did a good job on train four different models and picked random forest with the highest accuracy rate. And we built a cool UI interface with charts, map and hidden side bar. Another thing we did pretty well is we had kept to record our sprint planning, standup, burn down chart, backlog, and review all the time. There were many resources for us to select and enrich our group report.

And the Burn-down chart is as followed:



From the chart we can see that we distributed workload across two weeks evenly.

3.5.5 Sprint Retrospective

Consolidated list of factors that worked well during the sprint just completed	**Consolidated list of factors were problematic during the sprint just completed**
Efficient teamwork	Didn't deploy the application on EC2 at early stage
Distributed workload across two weeks evenly	Underestimated tasks and debugging
High efficiency in daily process	Prediction accuracy rate was not ideal enough
Became more familiar with the connection between front-end and back-end and database	Every day when the user opens the prediction page on the first time, it would take around one minute to retain the model before showing the prediction result.

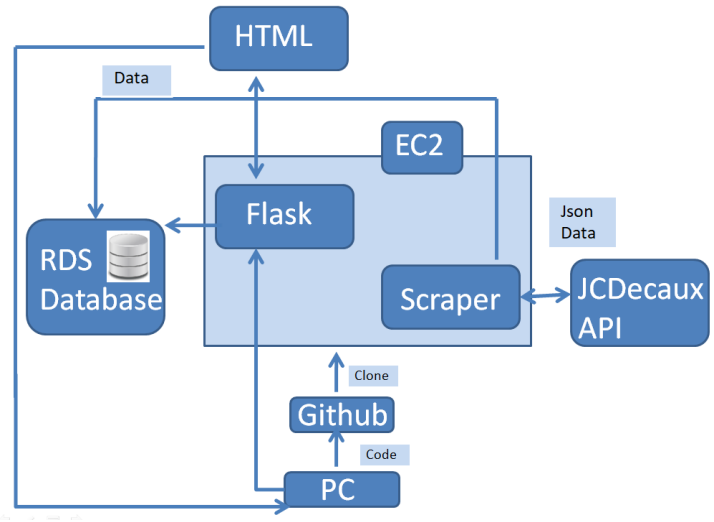
4. Project Retrospective

Something did well during the project	**Something to be improved & Something we learnt after the project**
Efficient teamwork & communication	Didn't deploy the application on EC2 at early

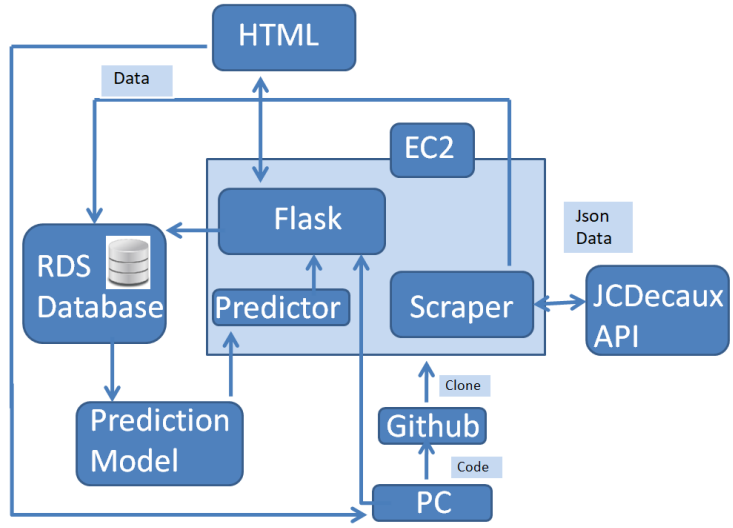
	stage
Distributed workload across period evenly	Underestimated tasks and debugging
High efficiency in daily process	Spent long time in learning new stuff
Kept record (meeting, discussion) all the time	Didn't distributed workload evenly across four sprints
Achieved the function to predict the available bike of a specific station in the following seven days.	Prediction accuracy rate was not ideal enough
	Every day when the user opens the prediction page on the first time, it would take around one minute to retain the model before showing the prediction result.

5. Architecture

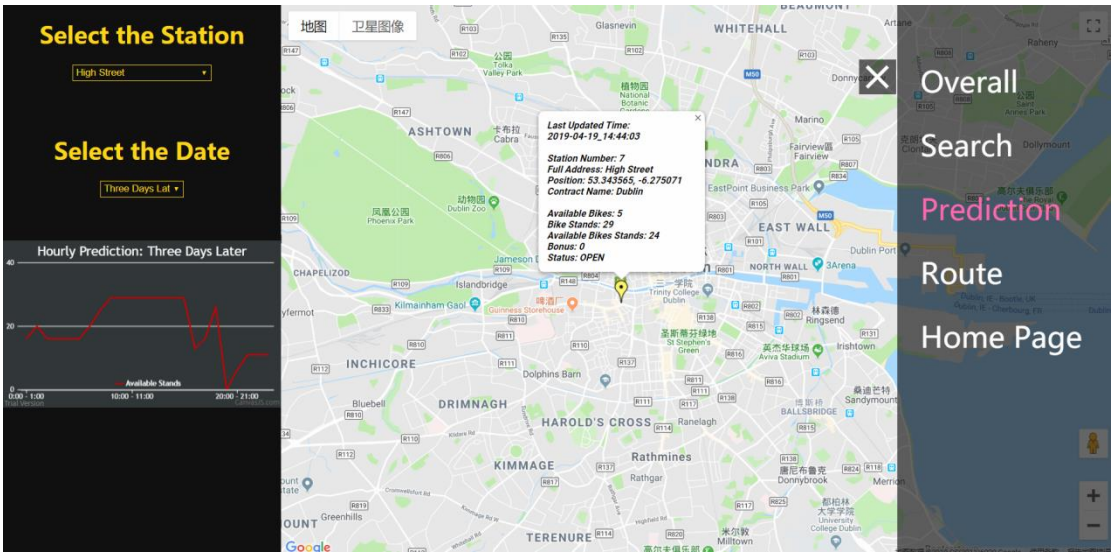
As this Dublin Bike Monitor is based on cloud service, it saved all the data in the AWS RDS and the program is running 24/7 on EC2. The back-end of this application is based on Flask, while front-end is based on HTML and JavaScript. And here is the first edition of our architecture.



As the application has more features such as prediction, the prediction model would read data from RDS and the model is deployed on EC2 and connected to Flask. Here is the second edition of architecture.



6. Analytics



When our team tried to train the model with data from RDS database, there were around 1,105,140 rows of information (from 15th March to 18th April). Then we decided to pick one row every hour instead of every five minutes, and minimized the dataset to around 92095 rows.

When splitting the training set and test set, we tried two ways: One is split by hand. We set the training set as data before 11th April and test set since 11th April till now. We also tried to use the train_test_split function from sklearn.model_selection.

We tried four models separately:

1) Decision Tree
2) KNN
3) Random Forest

4) Naïve Bayes

For the accuracy rate, we found that Decision Tree has the highest score around 73%. But when we retrain/test our Random Forest model in the final week, the accuracy rate was only 30%. We discussed of the reason for such decrease, and the most possible reason is we over-fitting.

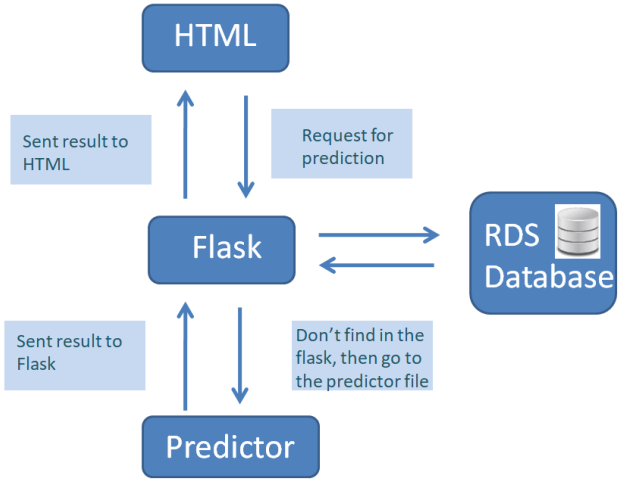
As mentioned above, we reordered the station number from 0 to 113, instead of 2 to 19 and from 21 to 115.

We created a predictor.py file which contains three functions:

1) predict_(df) is used to get the value.
2) predict_model(features, target) is used to create the model.
3) create_predict_set() is used to generate the dataset.

We achieved the function of predicting the available bike of a specific station in the following seven days.

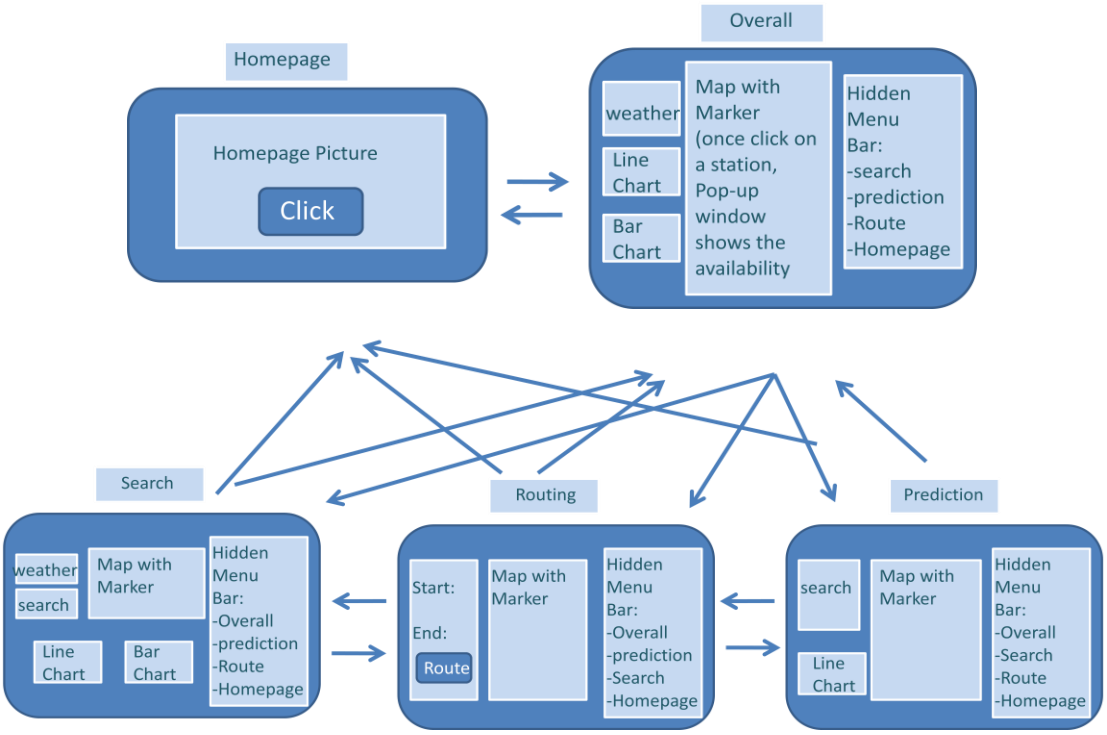
When we connected the predictor file with Flask and the architecture is shown below:



Every day when the user opens the prediction page on the first time, it would take around one minute to retain the model before showing the prediction result. And we felt this could be improved by added in a new scraper.py file in the future.

7. Design

Here is a workflow of the design of our HTML website.



Once opening the application, there is a **welcome homepage**. The user would click on the Button with 'View Stations' on it and then link to the overall page.

The overall page shows the Google Map with all stations with different colors of markers. The green one means that there are more than five available bikes at that station. The yellow one means that there are limited numbers of bikes (1-5) available at that station. And the red one means that there is no bike available at the station. The weather condition of the chosen station is shown at the top left corner. And the default value for it is the weather condition of station 30, which we use to represent the Dublin weather. It contains both the temperature/weather condition of the

moment and two hours later (prediction). And once click on a specific station, the analytic chart would show the availability of this station hourly (Bar chart) and daily (Line chart)

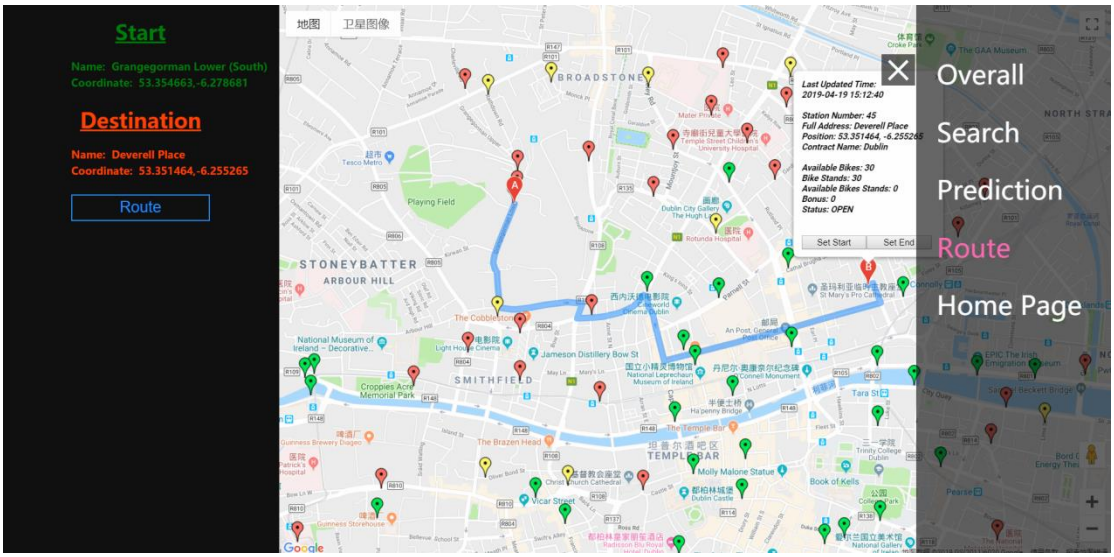
In order to provide a user-friendly interface, we design **a hidden menu bar** on the right sight with a '三' sign for it. And it could help the users to redirect to all the other sites.
When the user goes to **the search page**, the user could select a specific station from the drop down menu on the left side, or just pick the specific marker. And the position condition and availability status would come out in a popup. And the weather condition would be shown on the top left corner, while the bar chart and line chart of the tendency would be shown in the bottom.

For **the routing page**, the users could select a station as start point in the map as well as another station as end point. Then click on the 'Route' button. And then the route would be shown in the Map.

In the **prediction page**, the user could select a specific station from the map or from the drop down menu on the left side. And then the user would select a day in the following week to be predicted and click on the button. And then the line chart would show the availability of the station from 00:00 to 23:59.

8. Routing

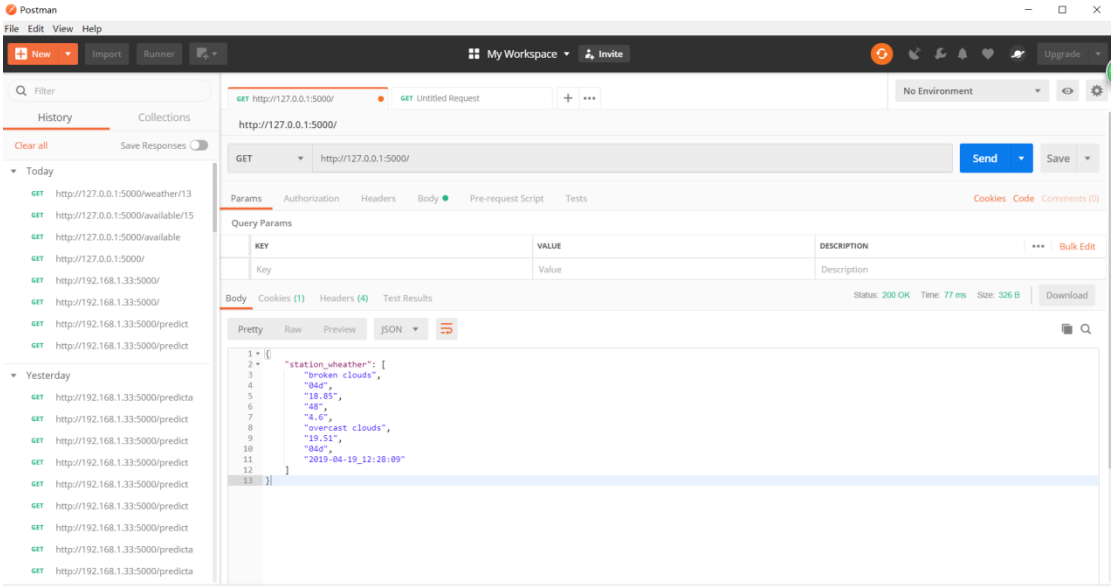
In order to show the route from one point to another, we used the Google Directions API. And the application requires the user to pick two points.



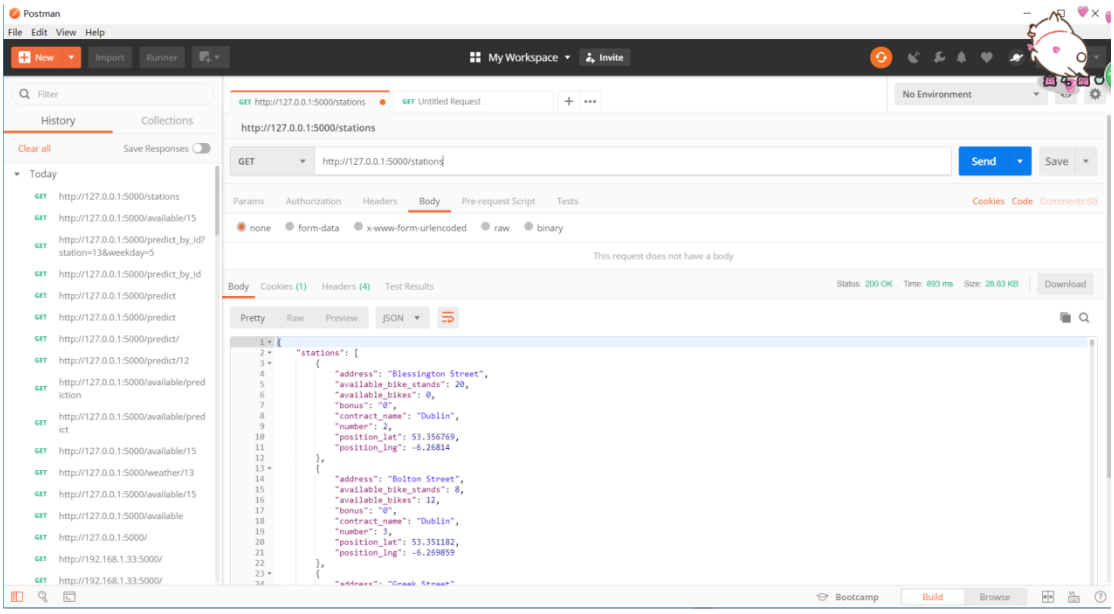
9. Unit Test

In this project, we did unit test in Postman and test the connect situation of back-end.

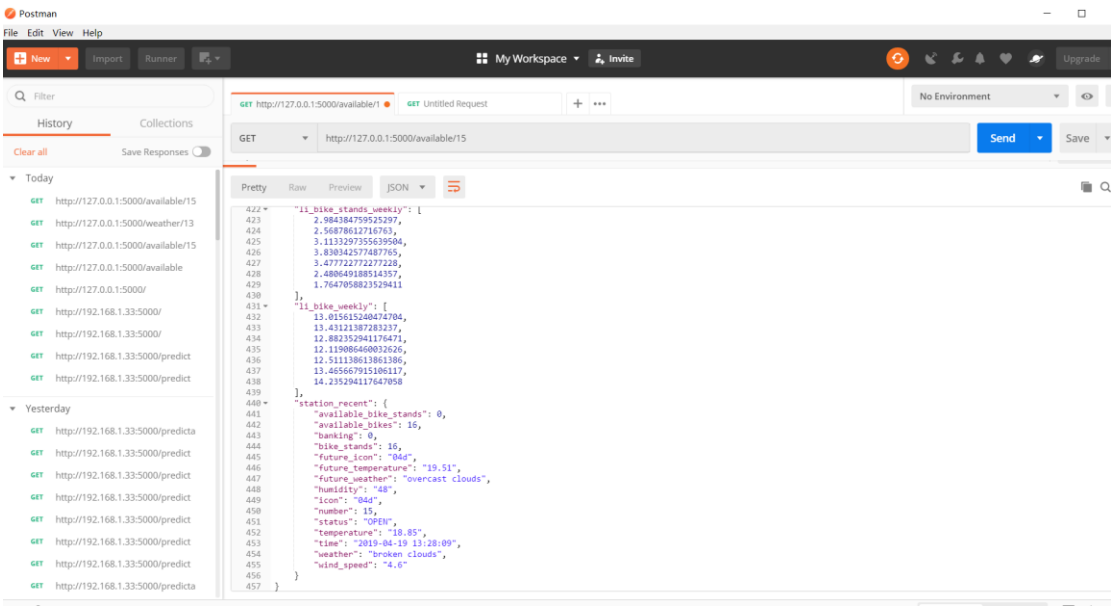
a) Homepage Test



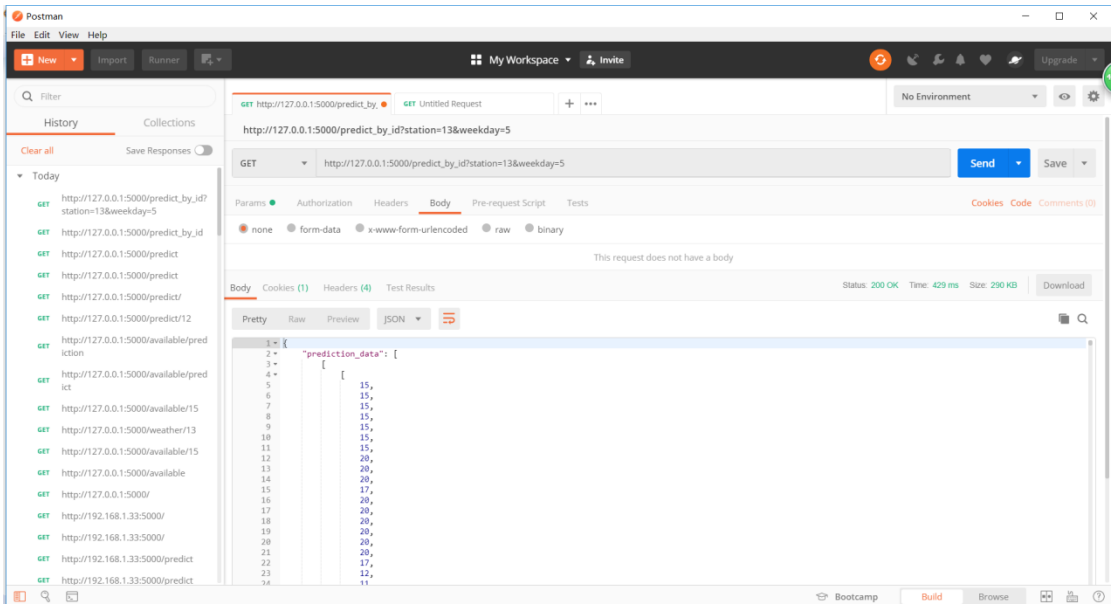
b) All station test



c) Station 15 test



d) Prediction test on weekday



10. Future Work

There are five main goals we would like to improve our application in the future:

(1) Less latency and quicker interaction

Till now the loading of a page containing a Google Map would need around two to three seconds. And this application's server side is not robust enough to serve large amount of users at the same time. Nginx accelerates content and application delivery, improves security, and facilitates availability and scalability for the busiest web sites on the Internet.

As our back-end is based on Flask frame in python and python is a High-level programming language, its performance is slower than the JavaScript/Node.js in high concurrency situation.

In the future we would like to use Node.js for back-end and add Nginx in the server.

(2) Prediction accuracy rate

We trained four different models during this project and we would like to improve the prediction accuracy rate with more available data and add in model calibration process by adjusting the model parameters and forcing within the margins of the uncertainties (Goodness-of-Fit or Cost Function).

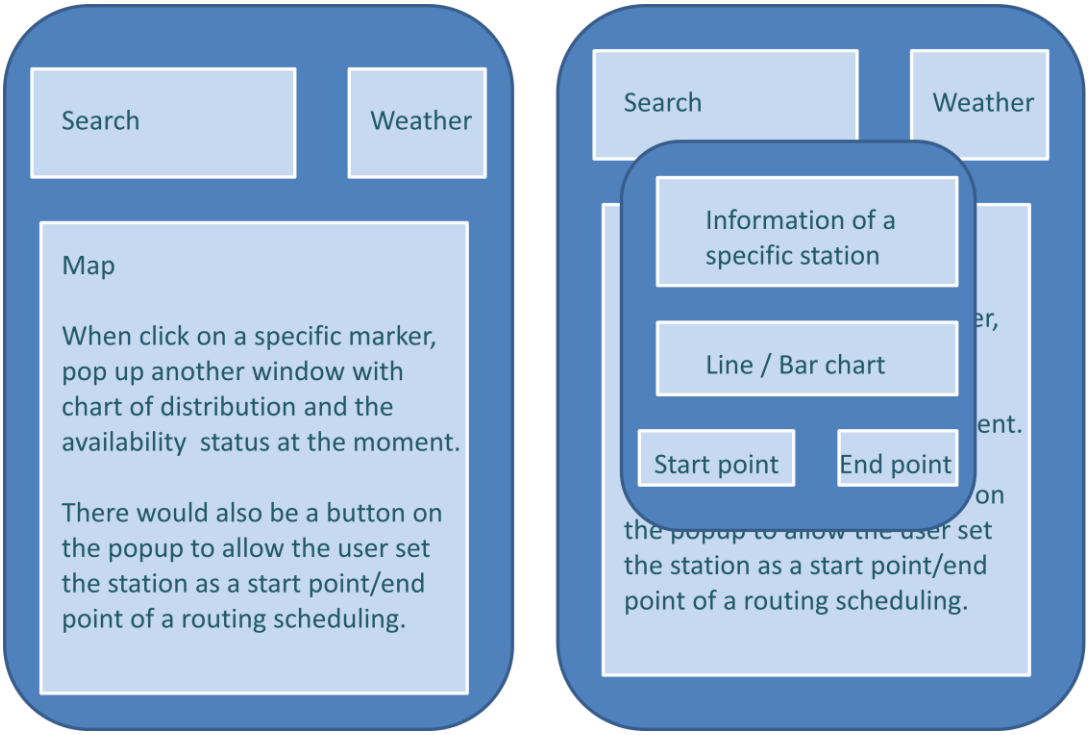
(3) Prediction auto-retraining

In this application the model would be retrained when the prediction page is opened for the first time every day and it would take around one minute to retrain the model and get the result.

We would like to improve the retraining mechanism by add in another update.py in EC2. The update.py would use a while true loop to check whether the last training date is same with current day and check once every two hours. And then the application would be retrained around 2am every day while causing less effect on user experience.

(4) Android edition

We would like to create an Android edition for our application in the future as well. And it would be look like this:



11. Reference

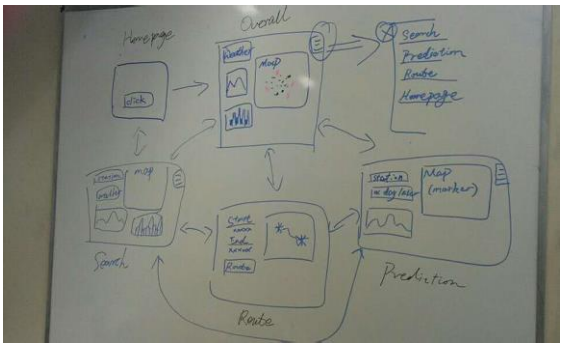
All of the code/graph/notes are available on GITHUB at:
https://github.com/sedublinbike/SE
And the working EC2 address of the live website is:
http://35.160.137.228:5000
Our daily meeting record/log on the Shimo editor is available at:
https://shimo.im/docs/XEjbgB7qkwYufO6j
Trello Board for Sprint One:
https://trello.com/b/1RgwAEv5/dublinbikesprintone
Trello Board for Sprint Two:
https://trello.com/b/GVzUAffv/dublinbikesprinttwo
Trello Board for Sprint Three:
https://trello.com/b/sFO2p7VT/dublinbikesprintthree
Trello Board for Sprint Four:
https://trello.com/b/XNk3WzjM/dublinbikesprintfour

JCDecaux API:
https://developer.jcdecaux.com/
OpenWeatherMap API:
https://openweathermap.org/api
Original weather icon:

https://openweathermap.org/weather-conditions
Beautiful HTML5 JavaScript Charts
https://canvasjs.com/
Google Map API:
https://developers.google.com/maps/documentation/javascript/tutorial?hl=zh-cn
Google Routing/Directions API
https://developers.google.com/maps/documentation/directions/start?hl=zh-cn
Postman
https://www.getpostman.com/

12. Appendix

(a) Group discussion and draft



Hi Anna Xuejiao,

The following has been confirmed:

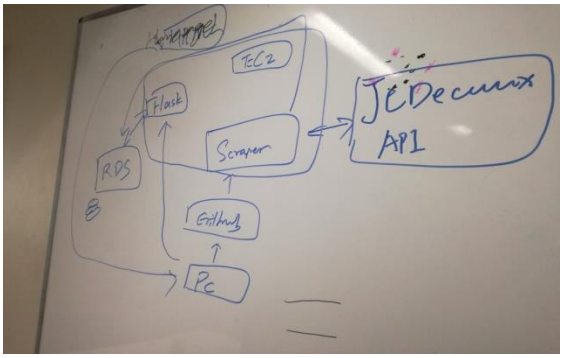
James Joyce Library:
Group Study Room F: 10:00am to 12:00pm Friday, 19th April 2019

Directions:
Group Study Room F: This room is located on Level 2 of the James Joyce Library

To cancel this booking visit: https://ucd.libcal.com/equipment/cancel?id=cs_jJXzxU4

Please ensure you keep your confirmation email as proof of your booking.

Thank you for using UCD Library Bookings and Training



Hi renjie,

The following has been confirmed:

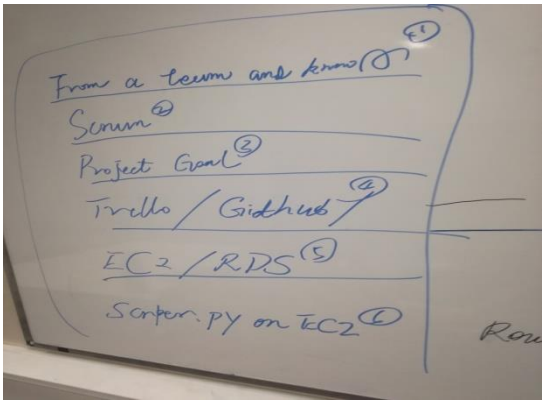
James Joyce Library:
Group Study Room F: 2:00pm to 4:00pm Friday, 19th April 2019

Directions:
Group Study Room F: This room is located on Level 2 of the James Joyce Library

To cancel this booking visit: https://ucd.libcal.com/equipment/cancel?id=cs_ERGx1co

Please ensure you keep your confirmation email as proof of your booking.

Thank you for using UCD Library Bookings and Training



Hi Menghao,

The following has been confirmed:

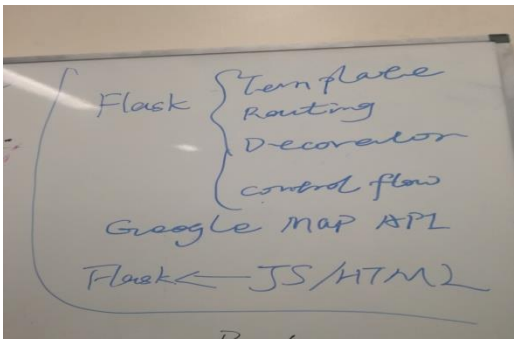
James Joyce Library:
Group Study Room F: 12:00pm to 2:00pm Friday, 19th April 2019

Directions:
Group Study Room F: This room is located on Level 2 of the James Joyce Library

To cancel this booking visit: https://ucd.libcal.com/equipment/cancel?id=cs_NdZxEUY

Please ensure you keep your confirmation email as proof of your booking.

Thank you for using UCD Library Bookings and Training



Hi Anna Xuejiao,

The following has been confirmed:

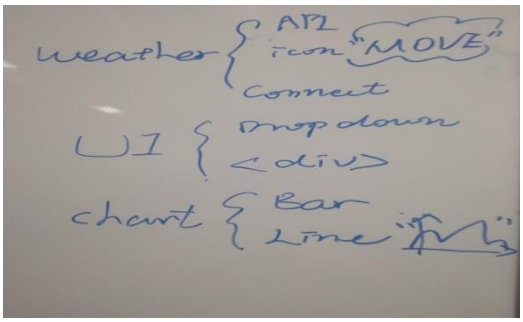
James Joyce Library:
Group Study Room A: 10:00am to 12:00pm Tuesday, 19th March 2019

Directions:
Group Study Room A: This room is located on Level 2 of the James Joyce Library

To cancel this booking visit: https://ucd.libcal.com/equipment/cancel?id=cs_odzj0lv

Please ensure you keep your confirmation email as proof of your booking.

Thank you for using UCD Library Bookings and Training



Hi Anna Xuejiao,

The following has been confirmed:

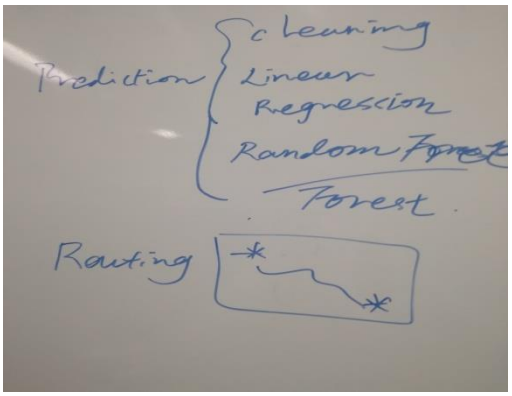
James Joyce Library:
Group Study Room 1: 9:00am to 10:00am Friday, 29th March 2019

Directions:
Group Study Room 1: This room is located on Level 1 in Hub 2 of the James Joyce Library

To cancel this booking visit: https://ucd.libcal.com/equipment/cancel?id=cs_PDpmAsA

Please ensure you keep your confirmation email as proof of your booking.

Thank you for using UCD Library Bookings and Training



Hi Anna Xuejiao,

The following has been confirmed:

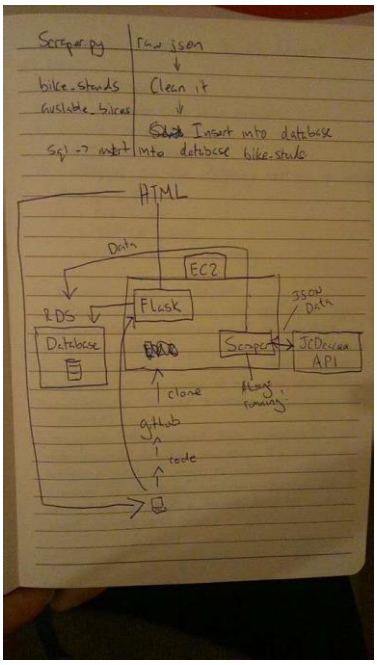
James Joyce Library:
Group Study Room A: 4:00pm to 6:00pm Tuesday, 9th April 2019

Directions:
Group Study Room A: This room is located on Level 2 of the James Joyce Library

To cancel this booking visit: https://ucd.libcal.com/equipment/cancel?id=cs_MepMAtk

Please ensure you keep your confirmation email as proof of your booking.

Thank you for using UCD Library Bookings and Training



Hi Anna Xuejiao,

The following has been confirmed:

James Joyce Library:
Group Study Room E: 12:00pm to 1:00pm Friday, 12th April 2019

Directions:
Group Study Room E: This room is located on Level 2 of the James Joyce Library

To cancel this booking visit: https://ucd.libcal.com/equipment/cancel?id=cs_O6RP3fA

Please ensure you keep your confirmation email as proof of your booking.

Thank you for using UCD Library Bookings and Training

(b) Jupyter Notebook

jupyter dbbikes_prediction_selection Last Checkpoint: Last Wednesday at 1:20 PM (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python [default]

Run Code

In [1]:
```
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from dateutil.parser import parse
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn import tree
from sklearn.neighbors import NearestNeighbors
from sklearn.feature_extraction import DictVectorizer
```

In [2]:
```
df = pd.read_csv('test1.csv',usecols = [1,2,3,4,5,6])
```

In [3]:
```
df.head(5)
```

Out[3]:

	number	available_bikes	available_bike_stands	weather	temperature	time
0	2	10	10	mist	11.40	2019-04-18_09:07:30
1	3	4	16	mist	11.40	2019-04-18_09:07:30
2	4	1	19	mist	11.40	2019-04-18_09:07:30
3	5	1	39	mist	11.39	2019-04-18_09:07:30
4	6	4	16	mist	10.85	2019-04-18_09:07:30

In [4]:
```
df['time'] = pd.to_datetime(df['time'],format="%Y-%m-%d_%H:%M:%S")
df['weekday'] = [i.weekday() for i in df['time']]
df['hour'] = [i.hour for i in df['time']]
df = df[df['weather'].isin(['broken clouds', 'clear sky', 'drizzle', 'few clouds', 'fog','heavy intensity rain', 'light intensity drizzle','lig
# df = df[df['weather'].isin(['broken clouds'])]

df = df.set_index('time',drop = False).groupby('number').resample('H').first().ffill().reset_index(drop=True).set_index('time',drop = True)
# df.drop_duplicates(inplace= True)
# df.to_csv('a.csv')
df
```

Out[4]:

time	number	available_bikes	available_bike_stands	weather	temperature	weekday	hour
2019-03-15 11:06:41	2	0	20	light intensity shower rain	9.92	4	11
2019-03-15 12:05:24	2	0	20	broken clouds	9.75	4	12
2019-03-15 13:06:31	2	1	19	broken clouds	10.28	4	13
2019-03-15 14:05:59	2	0	20	shower rain	9.88	4	14
2019-03-15 15:05:24	2	0	20	shower rain	10.04	4	15
2019-03-15 16:04:51	2	1	19	light intensity shower rain	9.35	4	16
2019-03-15 17:04:40	2	1	19	light intensity shower rain	8.61	4	17
2019-03-15 18:03:55	2	4	16	broken clouds	8.38	4	18
2019-03-15 19:03:08	2	12	8	light intensity shower rain	7.48	4	19
2019-03-15 20:02:33	2	13	7	light rain	6.82	4	20
2019-03-15 21:01:43	2	6	14	light rain	6.54	4	21

In [5]:
```
df.dtypes
```

Out[5]:
```
number                  int64
available_bikes         int64
available_bike_stands   int64
weather                object
temperature           float64
weekday                 int64
hour                    int64
dtype: object
```

In [6]:
```
np.unique(df['weather'].values)
```

Out[6]:
```
array(['broken clouds', 'clear sky', 'drizzle', 'few clouds', 'fog',
       'heavy intensity rain', 'light intensity drizzle',
       'light intensity drizzle rain', 'light intensity shower rain',
       'light rain', 'mist', 'moderate rain', 'overcast clouds',
       'scattered clouds', 'shower rain'], dtype=object)
```

In [7]:
```
for i in ['weather','hour','weekday','number']:
    df[i]=df[i].astype('category')
# df['weekday']=df['weekday'].astype('category')

df.dtypes
print(df.shape)
```

```
(92095, 7)
```

In [8]:
```
def split_set_by_hand(df):
    '''this function is designed to split the train/test set by hand'''
    test = df.truncate(after = '2019-04-08')
    train = df.truncate(before = '2019-04-11')
#     y_train = train['available_bike_stands'].reset_index(drop=True)
#     X_train = train[['number','weather','hour','temperature','weekday']].reset_index(drop=True)
#     y_test = test['available_bike_stands'].reset_index(drop=True)
#     X_test = test[['number','weather','hour','temperature','weekday']].reset_index(drop=True)
    y_train = train['available_bike_stands']
    X_train = train[['number','weather','hour','temperature','weekday']]
    y_test = test['available_bike_stands']
    X_test = test[['number','weather','hour','temperature','weekday']]
    return (X_train,X_test,y_train,y_test)
```

In [9]:
```
#aquire a special set with a fixed station and fixed weekday
df1 = df[(df['number']== 50) & (df['weekday'] == 1)]
df1
```

Out[9]:

time	number	available_bikes	available_bike_stands	weather	temperature	weekday	hour
2019-03-19 00:00:31	50	24	16	light intensity drizzle	7.97	1	0
2019-03-19 01:04:46	50	26	14	light intensity drizzle	7.74	1	1
2019-03-19 02:02:28	50	26	14	scattered clouds	7.23	1	2
2019-03-19 03:00:45	50	26	14	broken clouds	7.27	1	3
2019-03-19 04:05:34	50	26	14	broken clouds	6.79	1	4
2019-03-19 05:03:25	50	26	14	broken clouds	6.49	1	5

In [10]:
```
# split the special set into train/test set
X_train, X_test, y_train, y_test = split_set_by_hand(df1)
print("Training data:\n", pd.concat([X_train, y_train], axis=1))
print("\nTest data:\n", pd.concat([X_test, y_test], axis=1))
dicv = DictVectorizer(sparse = False)
X_train = dicv.fit_transform(X_train.to_dict(orient="records"))
X_test = dicv.transform(X_test.to_dict(orient="records"))
dicv.get_feature_names()
```

```
2019-04-02 19:00:22       1                  24
2019-04-02 20:06:40       1                  19
2019-04-02 21:05:43       1                  14
2019-04-02 22:04:49       1                  11
2019-04-02 23:03:50       1                   8

[72 rows x 6 columns]
```

Out[10]:
```
['hour',
 'number',
 'temperature',
 'weather=broken clouds',
 'weather=few clouds',
 'weather=light intensity drizzle',
 'weather=light intensity drizzle rain',
 'weather=light rain',
 'weather=mist',
 'weather=moderate rain',
 'weekday']
```

In [18]:
```
from sklearn.naive_bayes import MultinomialNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.decomposition import PCA

def predict_select( X_train, X_test, y_train, y_test):
    dicv = DictVectorizer(sparse = False)
#     pca = PCA(n_components=0.9)
    X_train = dicv.fit_transform(X_train.to_dict(orient="records"))
    X_test = dicv.transform(X_test.to_dict(orient="records"))
#     X_train = pca.fit_transform(X_train)
#     X_test = pca.fit_transform(X_test)

    dtr = tree.DecisionTreeClassifier()
    gc1 = GridSearchCV(dtr, param_grid={'max_depth':[5,15,25,35,45,55]}, cv=8)
    gc1.fit(X_train, y_train)
    print('the decision tree result:')
    print("the score of predicting the test sets is :", gc1.score(X_test, y_test))
    print("Mean cross-validated score is : ", gc1.best_score_)
    print("the best params is :", gc1.best_params_)
    print("the max of the mean_test_score is :",gc1.cv_results_['mean_test_score'].max())
    print()

    nbrs = KNeighborsClassifier()
    gc2 = GridSearchCV(nbrs, param_grid={'n_neighbors':[2,3,4,5,6]}, cv=8)
    gc2.fit(X_train, y_train)
    print('the K neighbors result:')
    print("the score of predicting the test sets is :", gc2.score(X_test, y_test))
    print("Mean cross-validated score is :", gc2.best_score_)
    print("the best params is :", gc2.best_params_)
    print("the max of the mean_test_score is :",gc2.cv_results_['mean_test_score'].max())
    print()
```

```
    rfc = RandomForestClassifier()
    gc3 = GridSearchCV(rfc, param_grid={"n_estimators": [25,100,150,200], "max_depth": [25,35,45,55]}, cv=8)
    gc3.fit(X_train, y_train)
    print('the random forest result:')
    print("the score of predicting the test sets is :", gc3.score(X_test, y_test))
    print("Mean cross-validated score is :", gc3.best_score_)
    print("the best params is :", gc3.best_params_)
    print("the max of the mean_test_score is :",gc3.cv_results_['mean_test_score'].max())
    print()

    mlt = MultinomialNB()
    gc4 = GridSearchCV(mlt, param_grid={"alpha": [0.01,0.05,0.1,0.5,1]}, cv=8)
    gc4.fit(X_train, y_train)
    print('the naive beyas result:')
    print("the score of predicting the test sets is :", gc4.score(X_test, y_test))
    print("Mean cross-validated score is :", gc4.best_score_)
    print("the best params is :", gc4.best_params_)
    print("the max of the mean_test_score is :",gc4.cv_results_['mean_test_score'].max())
    print()

    max_score_index = gc1
    for i in [gc2,gc3,gc4]:
        if i.cv_results_['mean_test_score'].max()>max_score_index.cv_results_['mean_test_score'].max():
            max_score_index = i
    print("the max mean test score result is :",max_score_index.cv_results_['mean_test_score'].max())
    print("the estimator is :",max_score_index.best_estimator_)

    return max_score_index.best_params_
```

In [19]:
```
df['temperature']=abs(df['temperature'])
df.reset_index(drop = True,inplace= True)
X_train, X_test, y_train, y_test = split_set_by_hand(df)
```

In [20]:
```
#train/test set is not random
predict_select(X_train, X_test, y_train, y_test)
```

```
the decision tree result:
the score of predicting the test sets is : 0.10029717682
Mean cross-validated score is :  0.038481329532
the best params is : {'max_depth': 5}
the max of the mean_test_score is : 0.038481329532

the K neighbors result:
the score of predicting the test sets is : 0.0591381872214
Mean cross-validated score is : 0.0333495584452
the best params is : {'n_neighbors': 6}
the max of the mean_test_score is : 0.0333495584452

the random forest result:
the score of predicting the test sets is : 0.0398216939079
Mean cross-validated score is : 0.034197900007
the best params is : {'max_depth': 25, 'n_estimators': 100}
the max of the mean_test_score is : 0.034197900007

the naive beyas result:
the score of predicting the test sets is : 0.0918276374443
Mean cross-validated score is : 0.0822195952994
the best params is : {'alpha': 0.5}
the max of the mean_test_score is : 0.0822195952994
the max mean test score result is : 0.0822195952994
the estimator is : MultinomialNB(alpha=0.5, class_prior=None, fit_prior=True)
```

Out[20]: {'alpha': 0.5}

In [21]:
```
# train/test set is random
y = df['available_bike_stands'].astype(str)
X = df[['number','weather','hour','temperature','weekday']]
print(X.dtypes)
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.25)
predict_select(X_train, X_test, y_train, y_test)
```

```
number          category
weather         category
hour            category
temperature      float64
weekday         category
dtype: object
the decision tree result:
the score of predicting the test sets is : 0.262552119527
Mean cross-validated score is :  0.25796644033
the best params is : {'max_depth': 35}
the max of the mean_test_score is : 0.25796644033

the K neighbors result:
the score of predicting the test sets is : 0.166000694927
Mean cross-validated score is : 0.159748664418
the best params is : {'n_neighbors': 6}
the max of the mean_test_score is : 0.159748664418
```

```
the random forest result:
the score of predicting the test sets is : 0.0815236275191
Mean cross-validated score is : 0.0774999638054
the best params is : {'max_depth': 25, 'n_estimators': 25}
the max of the mean_test_score is : 0.0774999638054

the naive beyas result:
the score of predicting the test sets is : 0.085041695622
Mean cross-validated score is : 0.0857089082249
the best params is : {'alpha': 0.5}
the max of the mean_test_score is : 0.0857089082249

the max mean test score result is : 0.25796644033
the estimator is : DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=35,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
```

Out[21]: {'max_depth': 35}

As shown above, it is not very clear that which the best estimator is , DecisionTreeClassifier estimator has the relatively high accuracy.so we choose DecisionTreeClassifier as our prediction function

In [ ]:
```
import joblib
def predict_model(features,target):
    dicv = DictVectorizer(sparse = False)
    X = dicv.fit_transform(features.to_dict(orient="records"))
    print(dicv.get_feature_names())
    y = target
    dtr = tree.DecisionTreeClassifier(max_depth = 15)
    dtr.fit(X,y)
    joblib.dump(dtr,'dbbikes_model.pkl')
```

In [ ]:
```
df.reset_index(drop=True)
y = df['available_bike_stands']
X = df[['number','weather','hour','temperature','weekday']]
print(X.dtypes)
predict_model(X,y)
```

In [ ]:
```
import time
import datetime
import os
def TimeStampToTime(timestamp):
    timeStruct = time.localtime(timestamp)
    return time.strftime('%Y-%m-%d',timeStruct)

def get_FileModifyTime(filePath):
    t = os.path.getmtime(filePath)
    return TimeStampToTime(t)

# aquire the 'dbbikes_model.pkl' last modified time and the current time,if they are the same day, system will not train the data again
file_modify_time =  get_FileModifyTime('dbbikes_model.pkl')
now = TimeStampToTime(time.time())
```

```
if now == file_modify_time:
    model = joblib.load("dbbikes_model.pkl")
else:
    df = df.reset_index(drop=True)
    y = df['available_bike_stands']
    X = df[['number','weather','hour','temperature','weekday']]
    predict_model(X,y)
    model = joblib.load("dbbikes_model.pkl")
```

In [ ]:
```
def create_predict_set(station,weekday):
    dict = {}
    dict['hour'] = [i for i in range(24)]
    dict['weekday'] = [weekday for i in range(24)]
    dict['number'] = [station for i in range(24)]
    dict['weather'] = ['broken clouds', 'clear sky', 'drizzle', 'few clouds','broken clouds','broken clouds', 'clear sky', 'drizzle', 'few clou
       'heavy intensity rain', 'light intensity drizzle',
       'light intensity drizzle rain', 'light intensity shower rain',
       'light rain', 'mist', 'moderate rain', 'overcast clouds',
       'scattered clouds', 'shower rain']
    dict['temperature'] =[6.00 for i in range(24)]
    df = pd.DataFrame(dict)
    for i in ['weather','hour','weekday','number']:
        df[i]=df[i].astype('category')
    dicv = DictVectorizer(sparse = False)
    df = dicv.fit_transform(df.to_dict(orient="records"))
    return df
df2 =create_predict_set(2,5)
model.predict(df2)
```

(c) Backlog

Spring one Backlog

Sub Task	Responsible	W4D1	W4D2	W4D3	W4D4	W4D5	W5D1	W5D2	W5D3	W5D4	W5D5	Subtotal
1. Form a team and introduce each other	all		2									2
2. learning Scrum and Agile	all				2	2						4
3. understand the project goal and need	Anna Xuejiao Ge	1	1									2
4. Set up Github team account and add in all the team members	Fu Renjie							0.5				0.5
5. Buile Trello and Shimo shared account	all			0.5	0.5							1
6. Set up EC2 on AWS and share key with all team members	Su Menghao			1								1
7. Get data from JCDecaux API	Su Menghao						1					1
8. Design data structure	Fu Renjie						1		1	1		3
9. Set up AWS RDS and share key with all team members	Anna Xuejiao Ge							1	1			2
10. Deploy scrper.py on EC2 and link EC2 and AWS RDS	Fu & Su							1		1	1	3
11. Sprint Review	Anna Xuejiao Ge										1.5	1.5
	Subtotal	1	3	1.5	2.5	2	2	2.5	2	2	2.5	21

Spring Two Backlog

Sub Task	Responsible	W6D1	W6D2	W6D3	W6D4	W6D5	W7D1	W7D2	W7D3	W7D4	W7D5	H1	H2	H3	H4	H5	Sub total
1. Learn the basic knowledge of Flask	all	1			1			1				15	10	12			40
2. Set goal for sprint two	Anna Xuejiao Ge		2														2
3. Build Flask Template + rendering	Su Menghao							4		2							6
4. Build Flask Routing & Decorator	Fu Renjie			2					4	3							9
5. Build Template control flow/loop	Anna Xuejiao Ge						2	2									4
6. Get Google map through	Su Menghao					2	2	1									5

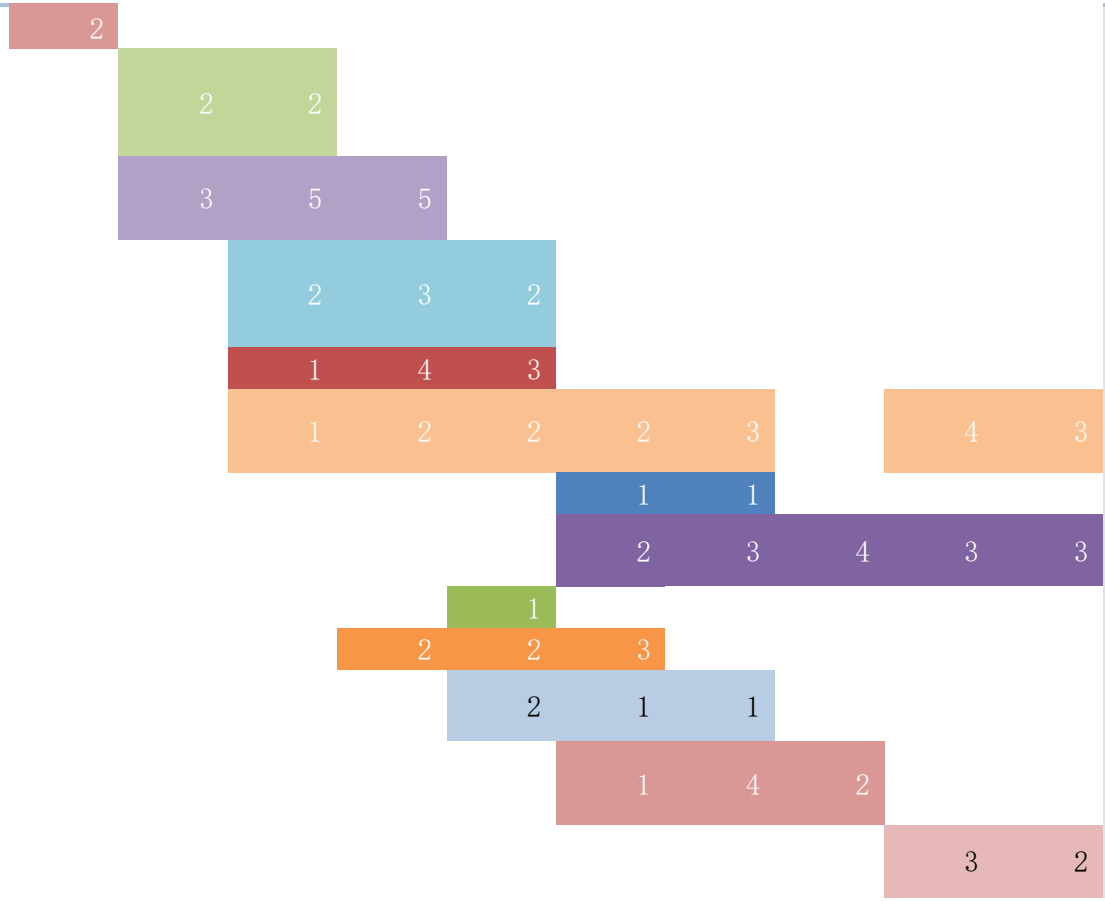
API																		
7. JSON API to connect RDS and Flask and JS	Fu Renjie										2	4	2	5				13
8. Simple UI design(CSS)	Anna Xuejiao Ge					2								3	2	4	2	13
9. Create drop down monu for each station and show the availability	Su & Fu												2	4	4	2		12
10. Spring Review	Anna Xuejiao Ge																2	2
	Subtotal	1	2	2	1	2	6	8	4	7	4	19	22	18	6	4		106

Spring Three Backlog

Sub Task	Responsible	W8D1	W8D2	W8D3	W8D4	W8D5	W9D1	W9D2	W9D3	W9D4	W9D5	Subtotal
1. Set goal for sprint three	Su Menghao	1										1
2. Compare and select suitable weather API	Fu Renjie		2	2	3							7
3. Create dynamic weather icon	Anna Xuejiao Ge		2		2							4
4. Deploy the weather icon into frount-end	Fu Renjie			2	2							4
5. Connect weather information in RDS, JSON and frount-end	Su Menghao			2	1	4	5	4				16
6. Unit test	Su Menghao			1	1							2
7. Dessign the logic to build availability charts	Anna Xuejiao Ge						2	1				3
8. Beautify UI	Fu Renjie						2		4	3		9
9. Redisign the website structure and set seoerate pages for different features	Fu Renjie							2	11	7	5	25
10. Sprint Review	Su Menghao										1	1
	Subtotal	1	4	7	9	4	9	7	15	10	6	72

Spring Four Backlog

Sub Task	Responsible	Subtotal
1. Set goal for sprint four	Fu Renjie	2
2. Clean the dataset and select data from the whole dataset	Su Menghao	4
3. Train and test four different model	Anna & Fu	13
4. Validation and select the model with highest accuracy rate	Fu Renjie	7
5. Pridiction file	Su Menghao	8
6. Geberate JSON file and connect for the prediction	Anna Xuejiao Ge	17
7. Unit test	Su Menghao	2
8. Beautify UI	Anna Xuejiao Ge	15
9. Database Backup	Fu Renjie	1
10. Select Route API	Su Menghao	7
11. Generate Routing Planning	Anna Xuejiao Ge	4
12. Connect the Routing planning to frount end	Fu Renjie	7
13. Deploy the website on EC2	Anna Xuejiao Ge	5



14. Sprint Review and Project report	Fu & Su & Anna							4	5	5	5	19
	Subtotal	2	5	11	16	12	10	16	11	15	13	111