Group 22

Dublin Bikes

2022/04/10

Contents

1. Project Overview

1.1 Introduction

During the COMP30830 module project, Team 22 created a website called DublinBikes to monitor the availability of bikes and estimate bike availability in using a machine learning model. The website was produced as part of the COMP30830 module project. As part of the project's deliverables, the website was launched on AWS EC2, and a summary of the findings was provided and summarized in this report.

Our Site's link: http://54.75.8.104:8080/

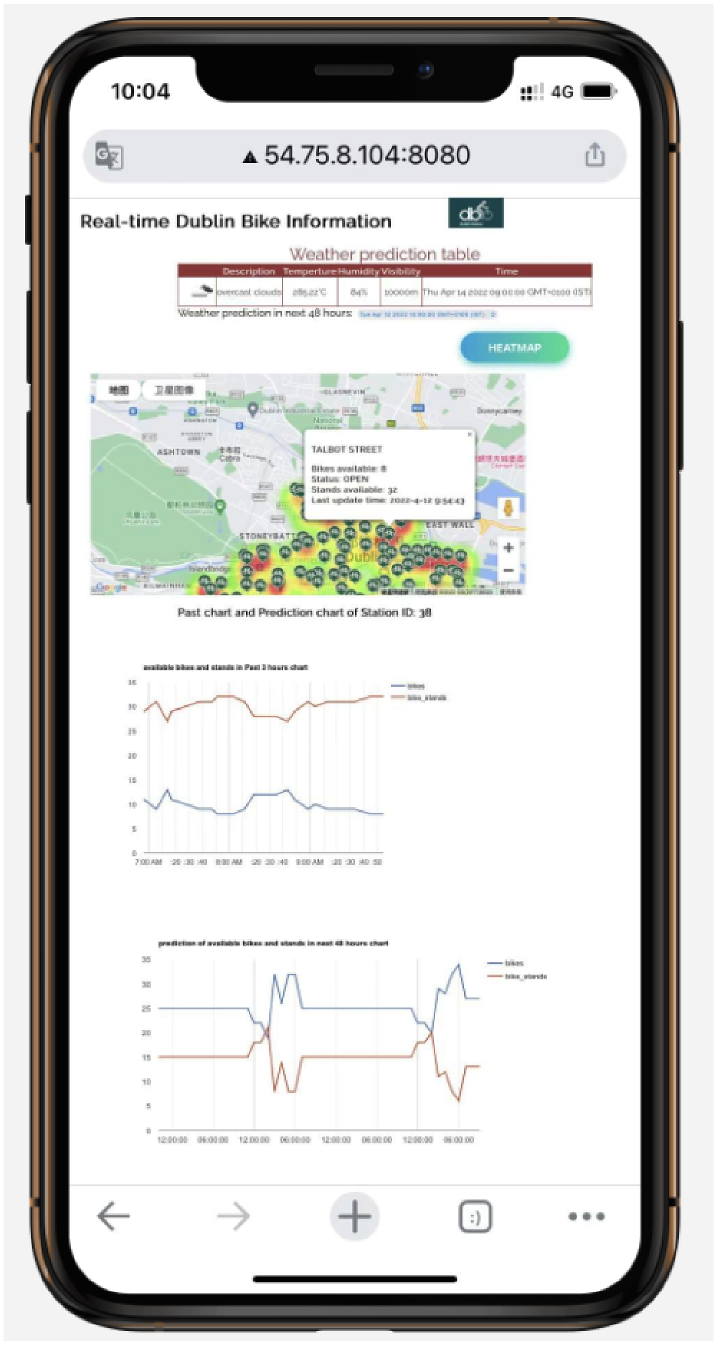
Our github's link: alexhuang19940623/comp30830-project (github.com)

Our members' username on Github: Wang Shengbin: Kuroko6668

Tianyu Huang: alexhuang19940623

Demonstrator：**MichelleDuggan**

Our App on iphone 11 Pro:



Real-time Dublin Bike Information

Weather prediction table

	Description	Temperture	Humidity	Visibility	Time
	overcast clouds	285.22℃	84%	10000m	Thu Apr 14 2022 09:00:00 GMT+0100 (IST)

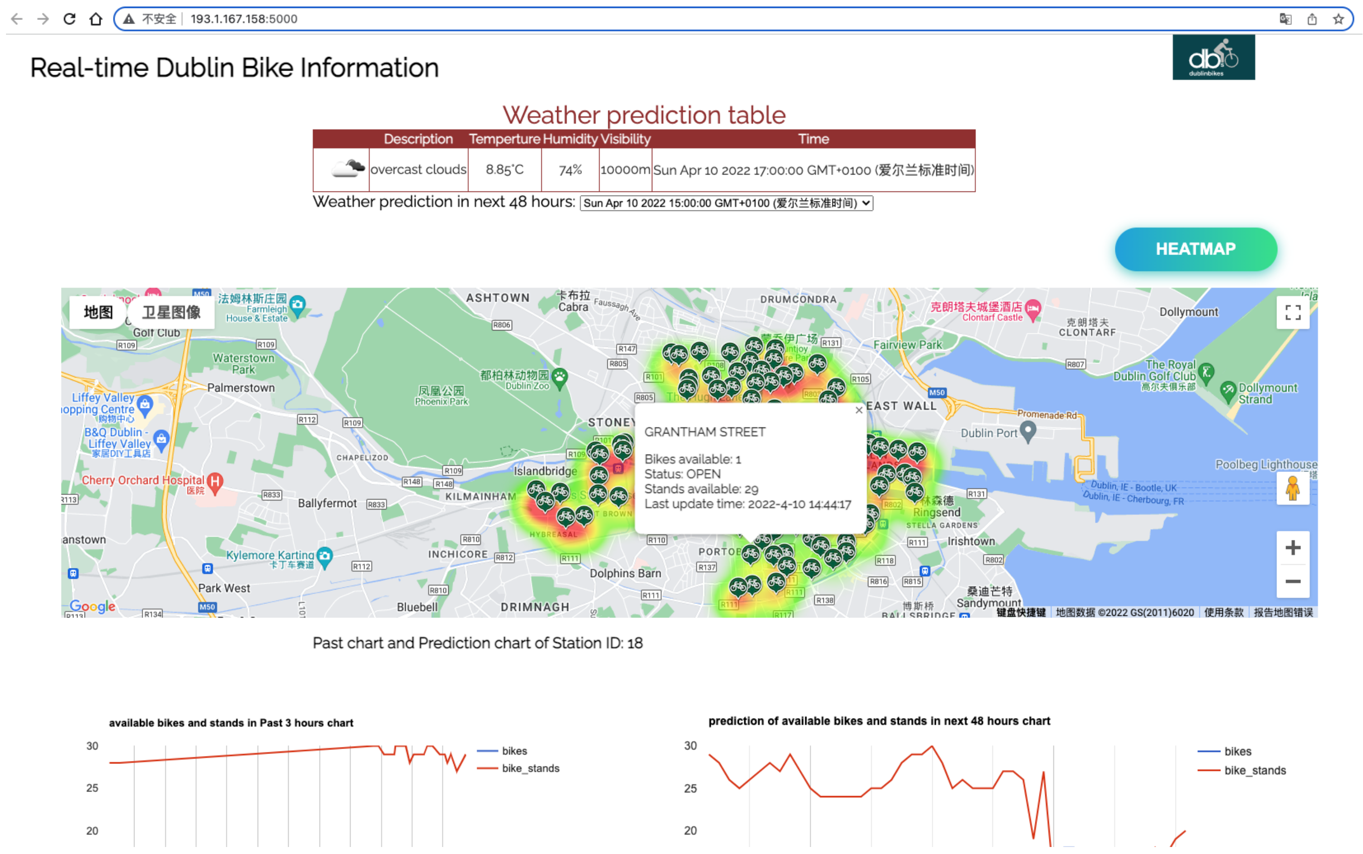
Weather prediction in next 48 hours: Tue Apr 12 2022 10:00:00 GMT+0100 (IST)

TALBOT STREET

Bikes available: 8
Status: OPEN
Stands available: 32
Last update time: 2022-4-12 9:54:43

Past chart and Prediction chart of Station ID: 38

Our application on web page:



Real-time Dublin Bike Information

Weather prediction table

	Description	Temperture	Humidity	Visibility	Time
	overcast clouds	8.85°C	74%	10000m	Sun Apr 10 2022 17:00:00 GMT+0100 (爱尔兰标准时间)

Weather prediction in next 48 hours: Sun Apr 10 2022 15:00:00 GMT+0100 (爱尔兰标准时间)

GRANTHAM STREET

Bikes available: 1
Status: OPEN
Stands available: 29
Last update time: 2022-4-10 14:44:17

Past chart and Prediction chart of Station ID: 18

available bikes and stands in Past 3 hours chart

prediction of available bikes and stands in next 48 hours chart

1.2 About this document

This document is intended to serve as our group’s learning journal for the development of this application development, with a particular emphasis on the following areas.

- The introduction of our efforts on design

- Our efforts on every technical problem

- Our solved problems

- Group members' contributions and areas for development.

- Others that were not included in the group's contribution.

1.3 Contributions

Shengbin Wang：Design, Front-End, JS, Flask, Deploye on EC2, Integration all Codes

Tianyu Huang: Design, Data Scraping Scripts, Data Analysis, Machine learning, Database

Hongpeng Zhang: Idea

Percentage of contribution

Shengbin Wang: 50%

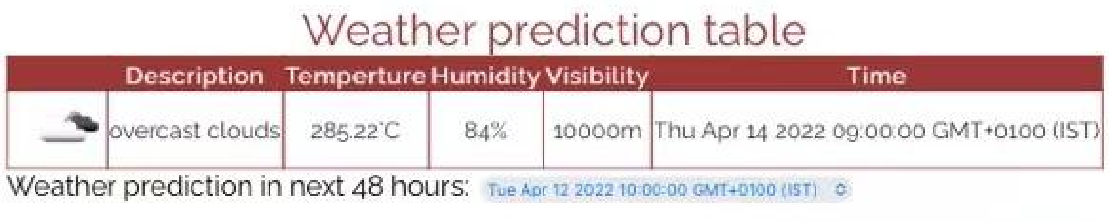
Tianyu Huang: 45%

Hongpeng Zhang: 5%

1.4 Structure

This app is composed of three main components.The first component is the weather display component, the second component is the map component and finally by the station information component.

1. **In the weather display component, We intend to use a table to show the current weather information for dublin and there is a time selection box that allows the user to choose to see the weather forecast for a certain time in the next 48 hours.**

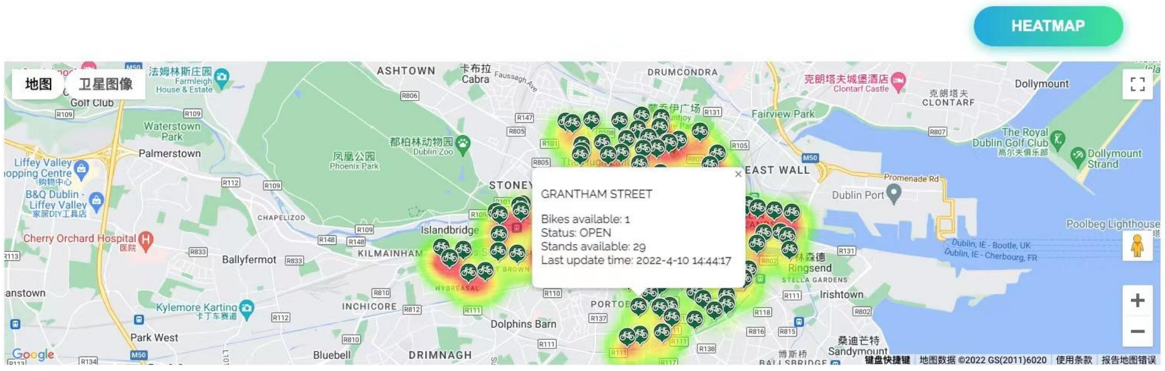


Weather prediction table

	Description	Temperture	Humidity	Visibility	Time
	overcast clouds	285.22°C	84%	10000m	Thu Apr 14 2022 09:00:00 GMT+0100 (IST)

Weather prediction in next 48 hours: Tue Apr 12 2022 10:00:00 GMT+0100 (IST)

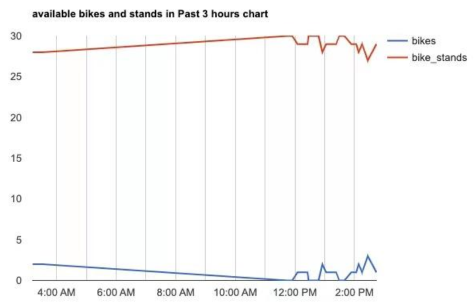
2. **In the map component, We have a Google map of Dublin with bike icons and heat maps for 110 bike stations. There is a green button at the top of the map which turns the heat map layer off or on.**

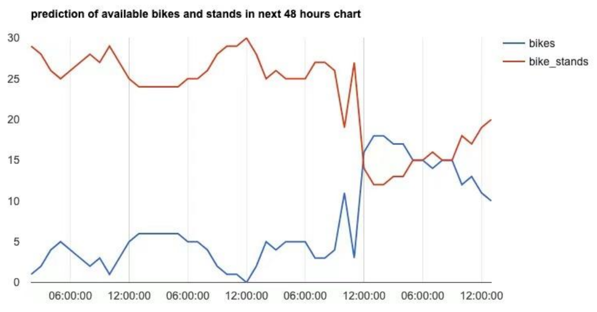


Past chart and Prediction chart of Station ID: 18

3. In the station information component, There is simply an indication message at first, inviting the user to click on the station marker. When the user clicks on one of the graphs, the station's id is presented, as well as a line graph indicating the number of bikes in the past three hours and a forecast of the number of bikes in the next 48 hours.

Past chart and Prediction chart of Station ID: 18

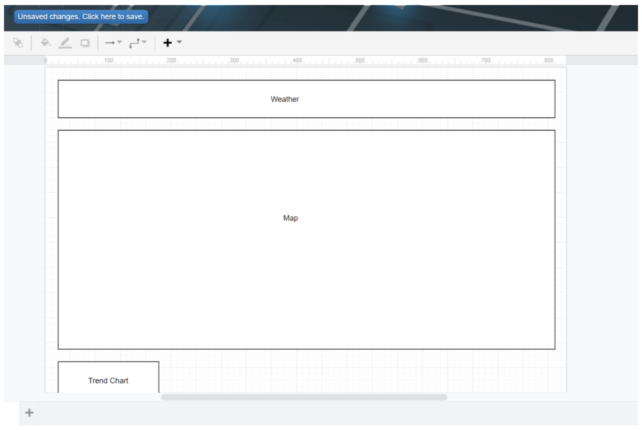




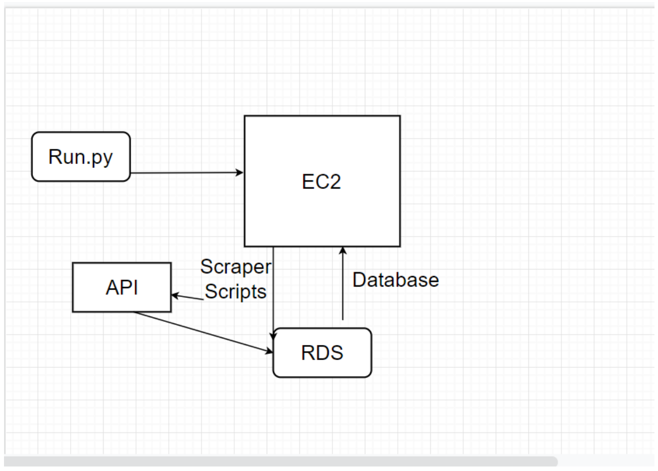
2. Architecture

2.1 Basic Architecture

2.1.1 Front-end architecture:

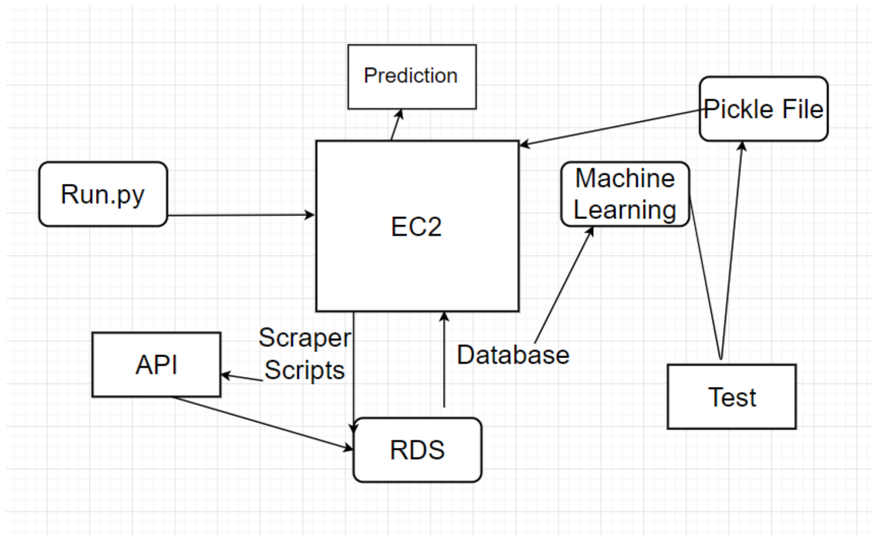


2.1.2 Back-end architecture:

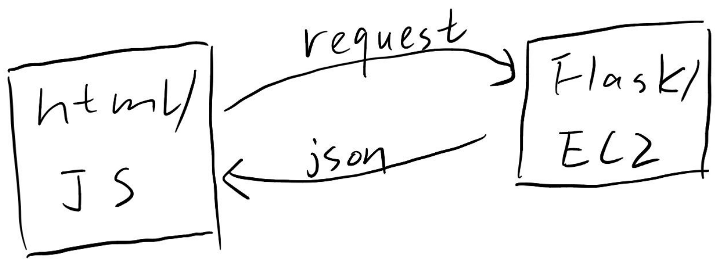


2.2 Full Architecture

back-end



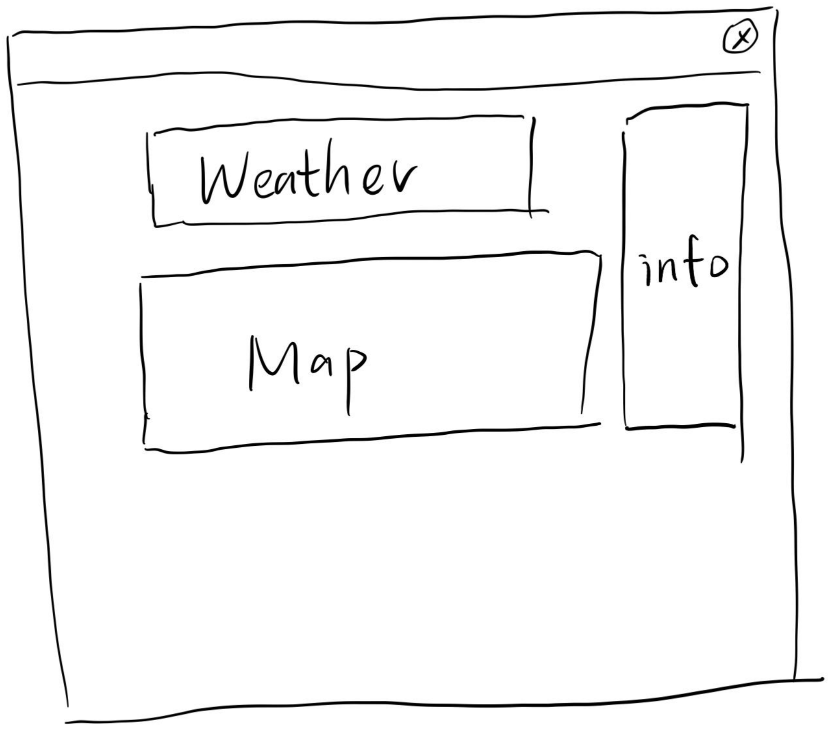
front-end architecture



3. Design

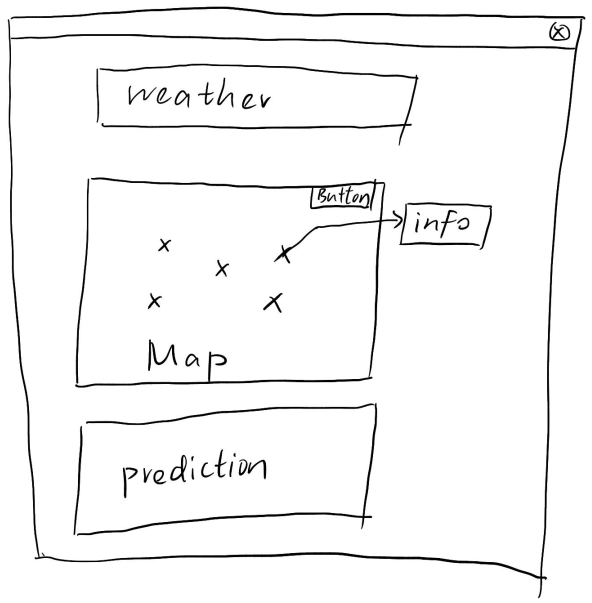
3.1 Web design

3.1.1 initail vision



According to the initial design, the website was split into three sections: a weather bar at the top, a map information portion in the center, and a section displaying model prediction trends information on the right side.

3.1.2 Final vision



However, we chose to abandon the inital design because the info box on the right might not be displayed completely due to the size of the user's window. We therefore chose to place the info box on the lower side so that it does not leave too much white space underneath the site, but also to better accommodate various window sizes.

3.1.3 Responsive design

To make websites responsive to user preferences and device capabilities, we choose to use responsive design. Because responsive design work in more situations than a fixed-width layout that only looks

right at one specific size. So our app can be adapted to the small size window of a mobile phone and the large size window of a website

3.2 Design of the weather component

Make use of data provided by the openweather(https://openweathermap.org/). This web provides a wide number of application programming interfaces (APIs) that may be utilized for our app. We have chosen the weather forecast api to get the weather forecast for the 48 hours after the user opens the app. And the current weather is stored every hour on ec2 for machine learning model training.

In the front-end we chose to present the current information in a table, as we believe this format is clearer and better suits the needs of the user. And with a select box interaction the user can see the weather forecast for his travel time.

3.3 Design of the map component

(1) Using google map api to display the map, markers, heatmap, infowindow and chart.

(2) The use of heat maps gives the user a better idea of the distribution of the number of bikes and stands.

(3) Using bike icon to replace the default marker of google map, So the user can more intuitively understand that this is a bike station marker .

(4) Using infoWindow to display the information about the station the user clicked, such as station name, bikes, stands, status, last update time.

(5) The system adopts a human-computer interaction approach with a beautiful and friendly interface design, and flexible, convenient and rapid information query.

(6) The ability to display the future number of bicycles at each location in each institution in an accurate, detailed and intuitive way.

(7) The system should be as easy to maintain and as easy to operate as possible.

3.4 Design of the Station information component

(1) Using text to remind users to click on the marker of the station they are interested in on the map.

(2) Sending a post request to flask, to get the prediction of bikes and stands.

(3) Using the Folding Line Chart to show the past and future trends of bikes, stands.

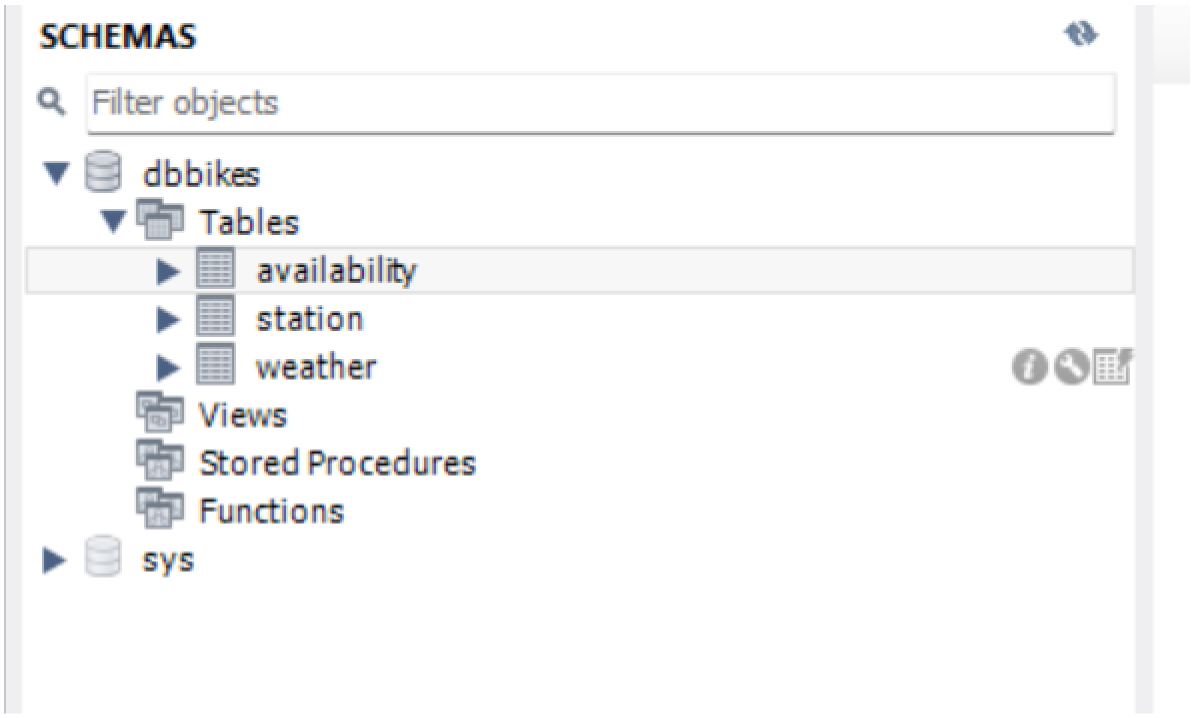
3.5 Data analysis section

For all tables deposited into the rds, we have to do data cleaning, the exact steps of which I will explain in the technical section, to remove such as the Nan value category to facilitate the convergence of machine learning.

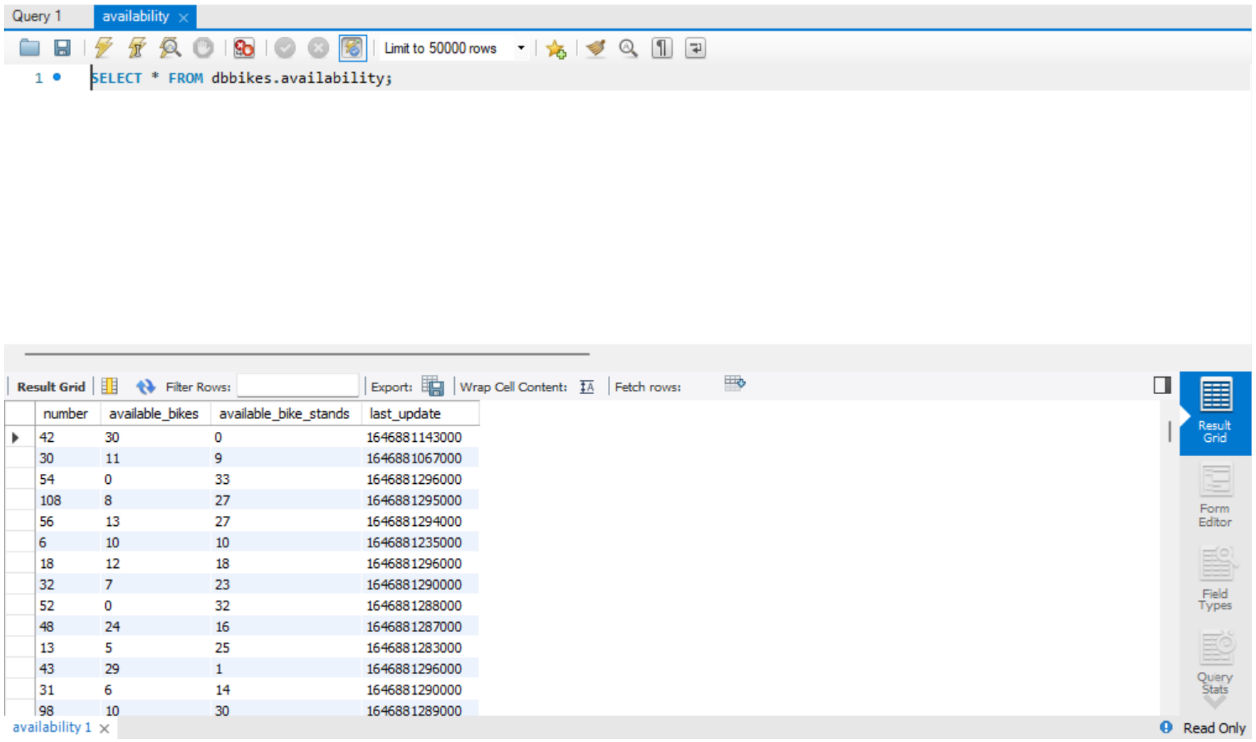
3.6 Data storage section

Create a script that continuously grabs data from the rds server and stores it, creating three schemas, one for availability, one for station, and one for weather. Each table has a fixed time interval between grabs, as explained in the technical section. The availability table consists of number, available_bikes, available_bikes_stands, and date. available_bikes, available_bikes_stands and date. The stations table contains addresses, specific coordinates and current status; and the weather table contains specific descriptions of the weather.

There are three tables in our databases.



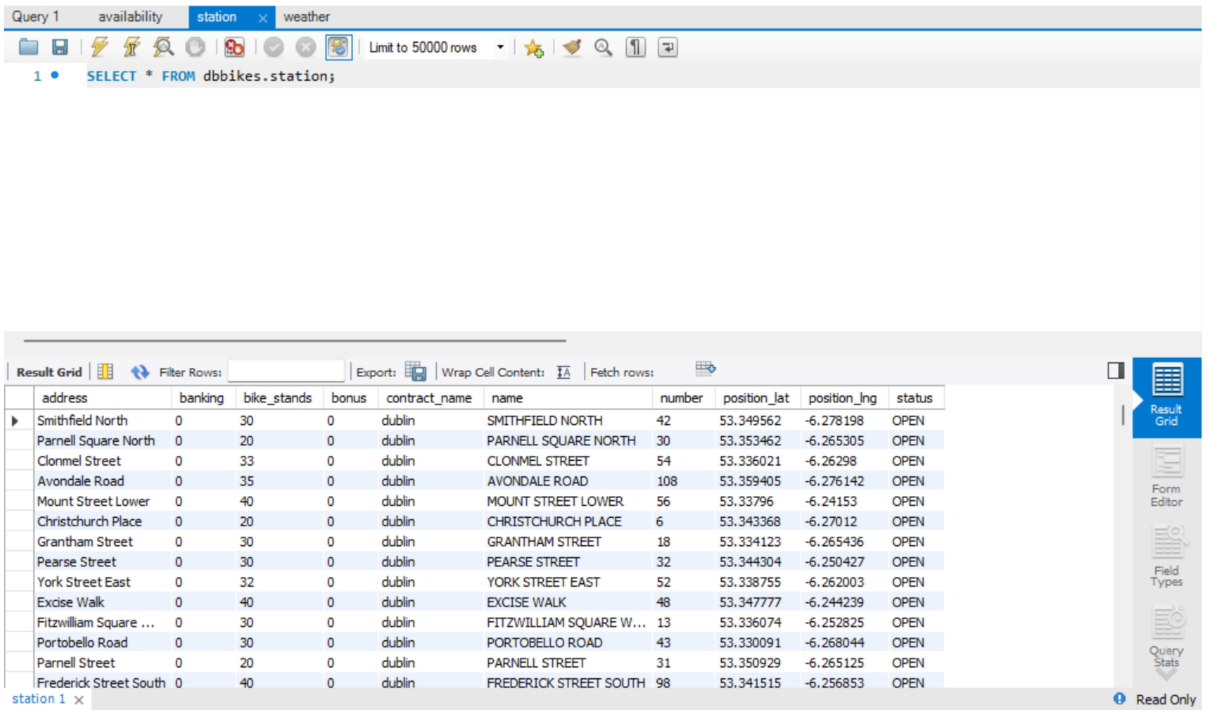
There are four colomns in availabilty table, number is station id, available_bikes is available bikes, available bikes stands is available_bikes_stands, last_update is the timestamp of the time when this information is updated.



```
SELECT * FROM dbbikes.availability;
```

number	available_bikes	available_bike_stands	last_update
42	30	0	1646881143000
30	11	9	1646881067000
54	0	33	1646881296000
108	8	27	1646881295000
56	13	27	1646881294000
6	10	10	1646881235000
18	12	18	1646881296000
32	7	23	1646881290000
52	0	32	1646881288000
48	24	16	1646881287000
13	5	25	1646881283000
43	29	1	1646881296000
31	6	14	1646881290000
98	10	30	1646881289000

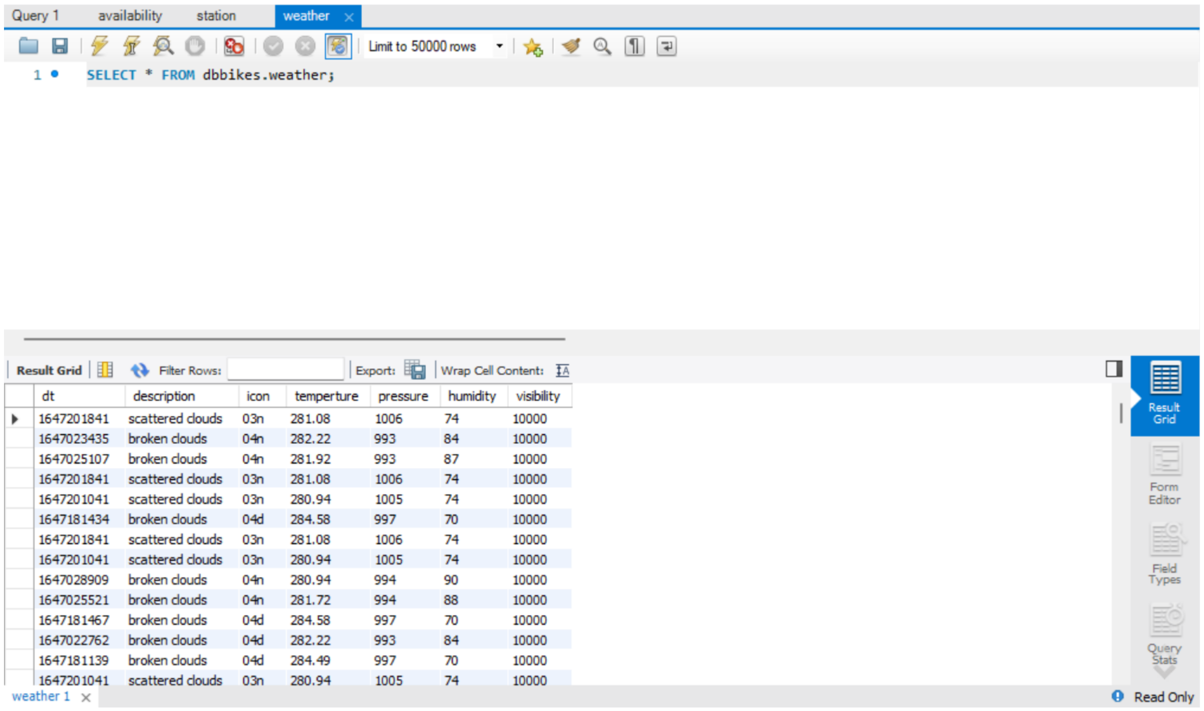
There are ten colomns in station table, address is station address, banking is banking, bike_stands is the total number of stands in this station. bonus is bonus, contract_name is dublin, number is station id, position_lat is latitude of this station, position_lng is longitude of this station. status is the status of this station.



```
SELECT * FROM dbbikes.station;
```

address	banking	bike_stands	bonus	contract_name	name	number	position_lat	position_lng	status
Smithfield North	0	30	0	dublin	SMITHFIELD NORTH	42	53.349562	-6.278198	OPEN
Parnell Square North	0	20	0	dublin	PARNELL SQUARE NORTH	30	53.353462	-6.265305	OPEN
Clonmel Street	0	33	0	dublin	CLONMEL STREET	54	53.336021	-6.26298	OPEN
Avondale Road	0	35	0	dublin	AVONDALE ROAD	108	53.359405	-6.276142	OPEN
Mount Street Lower	0	40	0	dublin	MOUNT STREET LOWER	56	53.33796	-6.24153	OPEN
Christchurch Place	0	20	0	dublin	CHRISTCHURCH PLACE	6	53.343368	-6.27012	OPEN
Grantham Street	0	30	0	dublin	GRANTHAM STREET	18	53.334123	-6.265436	OPEN
Pearse Street	0	30	0	dublin	PEARSE STREET	32	53.344304	-6.250427	OPEN
York Street East	0	32	0	dublin	YORK STREET EAST	52	53.338755	-6.262003	OPEN
Excise Walk	0	40	0	dublin	EXCISE WALK	48	53.347777	-6.244239	OPEN
Fitzwilliam Square ...	0	30	0	dublin	FITZWILLIAM SQUARE W...	13	53.336074	-6.252825	OPEN
Portobello Road	0	30	0	dublin	PORTOBELLO ROAD	43	53.330091	-6.268044	OPEN
Parnell Street	0	20	0	dublin	PARNELL STREET	31	53.350929	-6.265125	OPEN
Frederick Street South	0	40	0	dublin	FREDERICK STREET SOUTH	98	53.341515	-6.256853	OPEN

There are seven colomns in weather table, description is description of current weather of dublin, icon is weather icon, temperture is the current temperture. pressure is current pressure, humidity is humidity, visbility is current visbility, The dt is the timestamp of the time when this information is updated.



```sql
SELECT * FROM dbbikes.weather;
```

dt	description	icon	temperture	pressure	humidity	visibility
1647201841	scattered clouds	03n	281.08	1006	74	10000
1647023435	broken clouds	04n	282.22	993	84	10000
1647025107	broken clouds	04n	281.92	993	87	10000
1647201841	scattered clouds	03n	281.08	1006	74	10000
1647201041	scattered clouds	03n	280.94	1005	74	10000
1647181434	broken clouds	04d	284.58	997	70	10000
1647201841	scattered clouds	03n	281.08	1006	74	10000
1647201041	scattered clouds	03n	280.94	1005	74	10000
1647028909	broken clouds	04n	280.94	994	90	10000
1647025521	broken clouds	04n	281.72	994	88	10000
1647181467	broken clouds	04d	284.58	997	70	10000
1647022762	broken clouds	04d	282.22	993	84	10000
1647181139	broken clouds	04d	284.49	997	70	10000
1647201041	scattered clouds	03n	280.94	1005	74	10000

4.Techinical Part

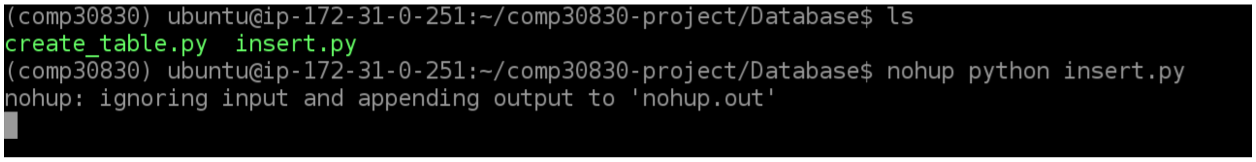
In this section we will describe in detail how we crawl the data, how we deposit the rds, and how we clean the data and build the model, the front-end part we will put in what we have learned.

4.1 Technical Explanation

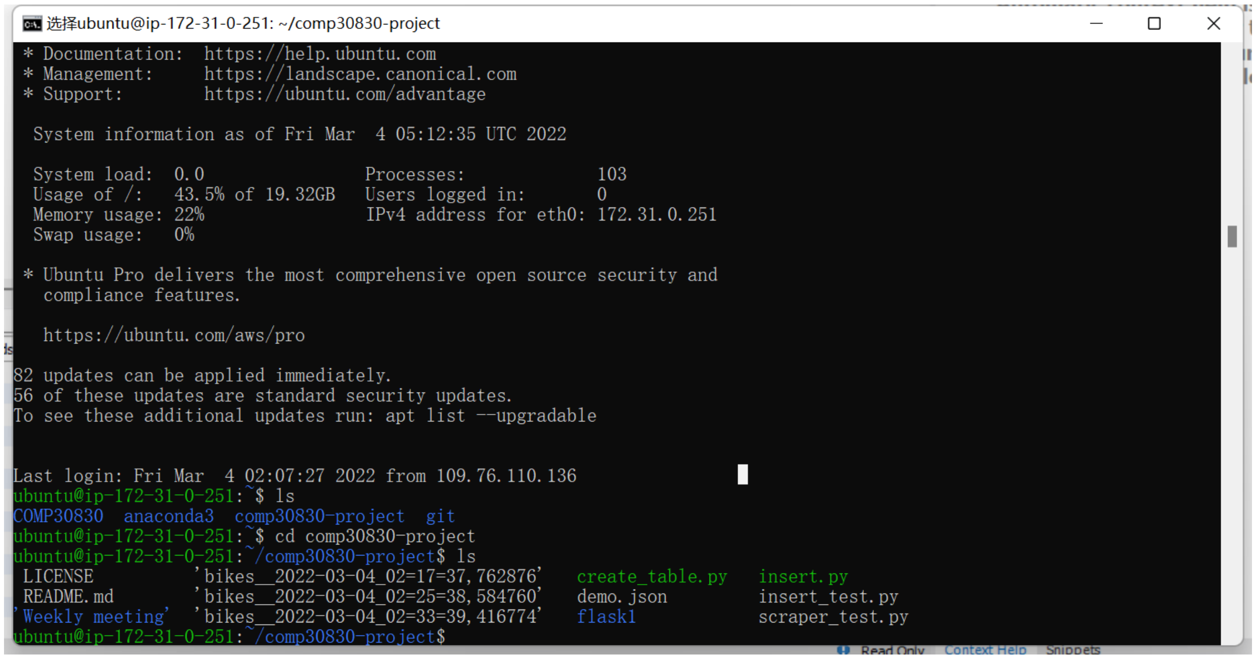
4.1.1 Data Crawl section

Keeping the server up and running

Once we have created the crawler, we have to upload it to the server and keep it running. Here I have chosen the nohup command: nohup python file.py



```
(comp30830) ubuntu@ip-172-31-0-251:~/comp30830-project/Database$ ls
create_table.py  insert.py
(comp30830) ubuntu@ip-172-31-0-251:~/comp30830-project/Database$ nohup python insert.py
nohup: ignoring input and appending output to 'nohup.out'
```



选择ubuntu@ip-172-31-0-251: ~/comp30830-project

```
 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Fri Mar  4 05:12:35 UTC 2022

  System load:  0.0               Processes:             103
  Usage of /:   43.5% of 19.32GB   Users logged in:       0
  Memory usage: 22%               IPv4 address for eth0: 172.31.0.251
  Swap usage:   0%

 * Ubuntu Pro delivers the most comprehensive open source security and
   compliance features.

   https://ubuntu.com/aws/pro

82 updates can be applied immediately.
56 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable


Last login: Fri Mar  4 02:07:27 2022 from 109.76.110.136
ubuntu@ip-172-31-0-251:~$ ls
COMP30830  anaconda3  comp30830-project  git
ubuntu@ip-172-31-0-251:~$ cd comp30830-project
ubuntu@ip-172-31-0-251:~/comp30830-project$ ls
 LICENSE          'bikes__2022-03-04_02=17=37,762876'   create_table.py   insert.py
 README.md        'bikes__2022-03-04_02=25=38,584760'   demo.json         insert_test.py
'Weekly meeting'  'bikes__2022-03-04_02=33=39,416774'   flask1            scraper_test.py
ubuntu@ip-172-31-0-251:~/comp30830-project$
```

The three database tables are in principle the same, we first have to create.

```
KEY = "50b3e4b1972f05f68760a2caaece786ed0bda969"
NAME = "Dublin"
STATIONS = "https://api.jcdecaux.com/vls/v1/stations"
URL="database-2.ceamwd5mbowc.eu-west-1.rds.amazonaws.com"
PASSWORD="11223344"
PORT="3306"
DB="dbikes"
USER ="TianyuHuang"
engine =create_engine("mysql+mysqldb://{}:{}@{}:{}/{}".format(USER,PASSWORD,URL,PORT,DB),echo=True)
sql = """
    CREATE DATABASE IF NOT EXISTS dbikes;
"""
engine.execute(sql)
```

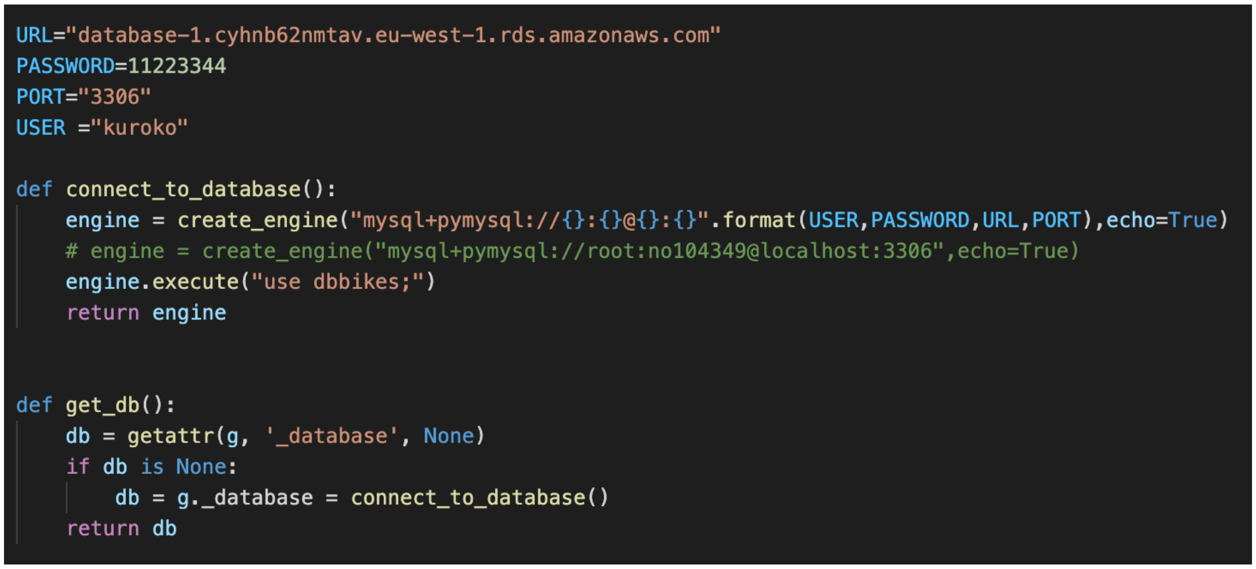
```
try:

    res = engine.execute(sql1)
    res = engine.execute("DROP TABLE IF EXISTS station")
    res = engine.execute(sql2)
    print(res.fetchall())
```

We then have to insert the data into the table.

```
def stations_to_db(text):
    stations = json.loads(text)
    print(type(stations), len(stations))
    for station in stations:
        print(station)
        vals = (
            station.get('address'), int(station.get('banking')), station.get('bike_stands'), int(station.get('bonus')),
            station.get('contract_name'), station.get('name'), station.get('number'),
            station.get('position').get('lat'), station.get('position').get('lng'), station.get('status')
        )
        engine.execute("insert into station values(%s, %s, %s, %s, %s, %s, %s, %s, %s, %s)",vals)
    return
```

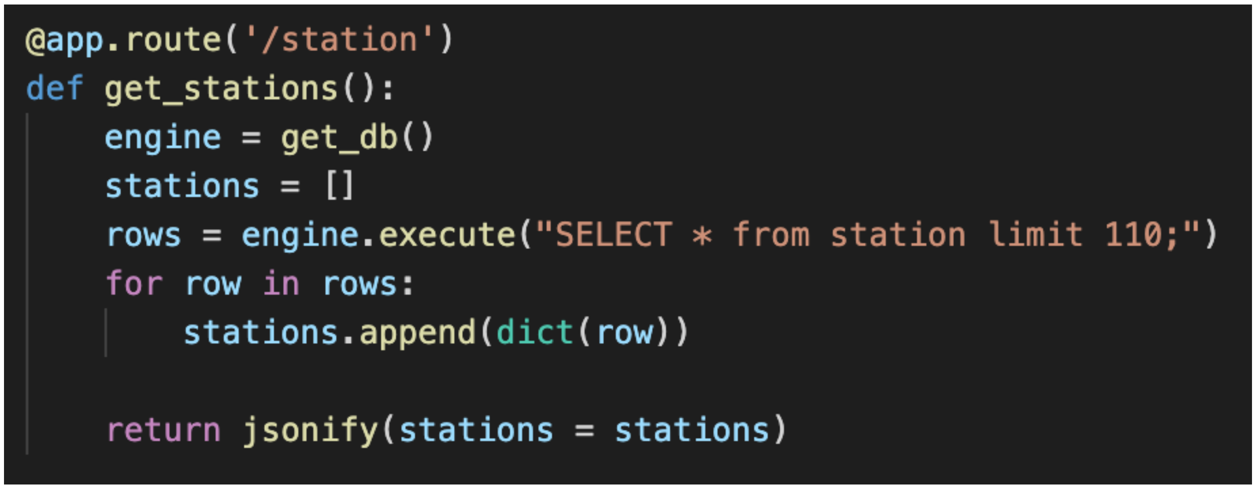
4.1.2 How To Get Json Files

We start by connecting to our database in the view.py file



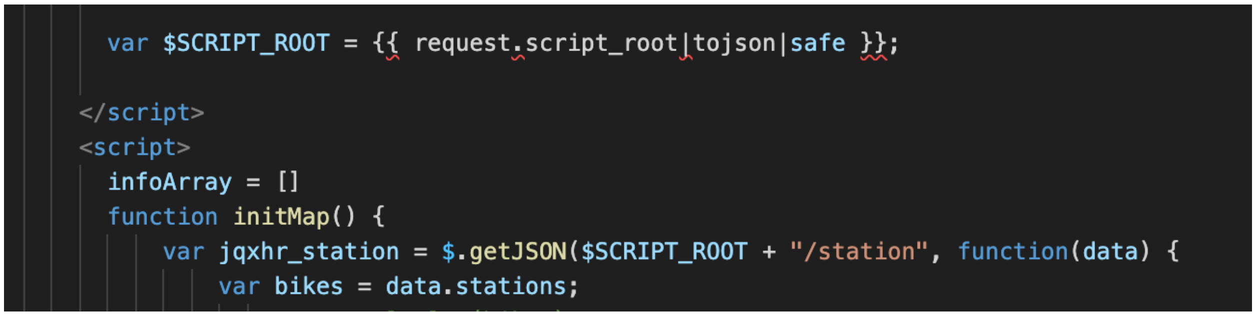
```
URL="database-1.cyhnb62nmtav.eu-west-1.rds.amazonaws.com"
PASSWORD=11223344
PORT="3306"
USER ="kuroko"

def connect_to_database():
    engine = create_engine("mysql+pymysql://{}:{}@{}:{}".format(USER,PASSWORD,URL,PORT),echo=True)
    # engine = create_engine("mysql+pymysql://root:no104349@localhost:3306",echo=True)
    engine.execute("use dbbikes;")
    return engine


def get_db():
    db = getattr(g, '_database', None)
    if db is None:
        db = g._database = connect_to_database()
    return db
```

Then Python decorator that Flask provides to assign "/station" URLs in our app to get_stations functions



```
@app.route('/station')
def get_stations():
    engine = get_db()
    stations = []
    rows = engine.execute("SELECT * from station limit 110;")
    for row in rows:
        stations.append(dict(row))

    return jsonify(stations = stations)
```

Use jQuery.getJSON() to *Load JSON-encoded data from the server using a GET HTTP request.*



```
    var $SCRIPT_ROOT = {{ request.script_root|tojson|safe }};

  </script>
  <script>
    infoArray = []
    function initMap() {
        var jqxhr_station = $.getJSON($SCRIPT_ROOT + "/station", function(data) {
            var bikes = data.stations;
```

4.2 Data Analytics Part

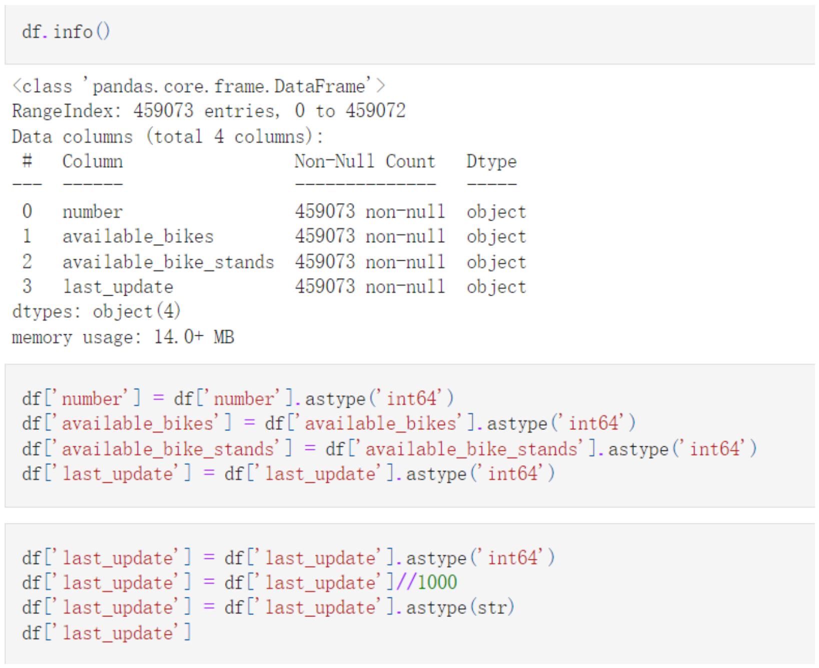
Choose the correct model

We identified two models at the beginning, linear regression and Auto Regression, and tested them separately.

After comparison, and logical problems, we finally chose the linear regression model, and I will describe the training process of the linear regression model below. Please refer to the ipynb file on github for the specific Auto Regression model training and results, and we finally chose the linear regression model.

1.Clean the data

a. The data we collect has many errors, so we have to clean the data, for the numbers we have to change to int64 format



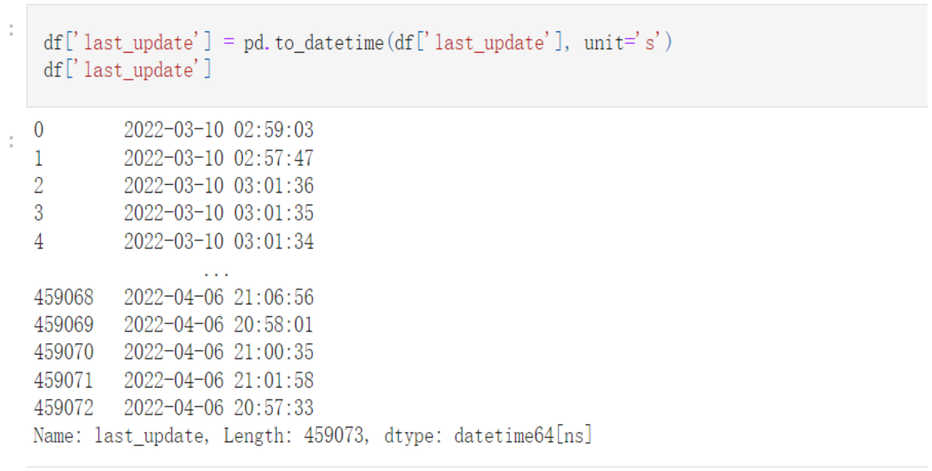
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 459073 entries, 0 to 459072
Data columns (total 4 columns):
 #   Column                Non-Null Count   Dtype
---  ------                --------------   -----
 0   number                459073 non-null  object
 1   available_bikes       459073 non-null  object
 2   available_bike_stands 459073 non-null  object
 3   last_update           459073 non-null  object
dtypes: object(4)
memory usage: 14.0+ MB
```

```
df['number'] = df['number'].astype('int64')
df['available_bikes'] = df['available_bikes'].astype('int64')
df['available_bike_stands'] = df['available_bike_stands'].astype('int64')
df['last_update'] = df['last_update'].astype('int64')
```

```
df['last_update'] = df['last_update'].astype('int64')
df['last_update'] = df['last_update']//1000
df['last_update'] = df['last_update'].astype(str)
df['last_update']
```

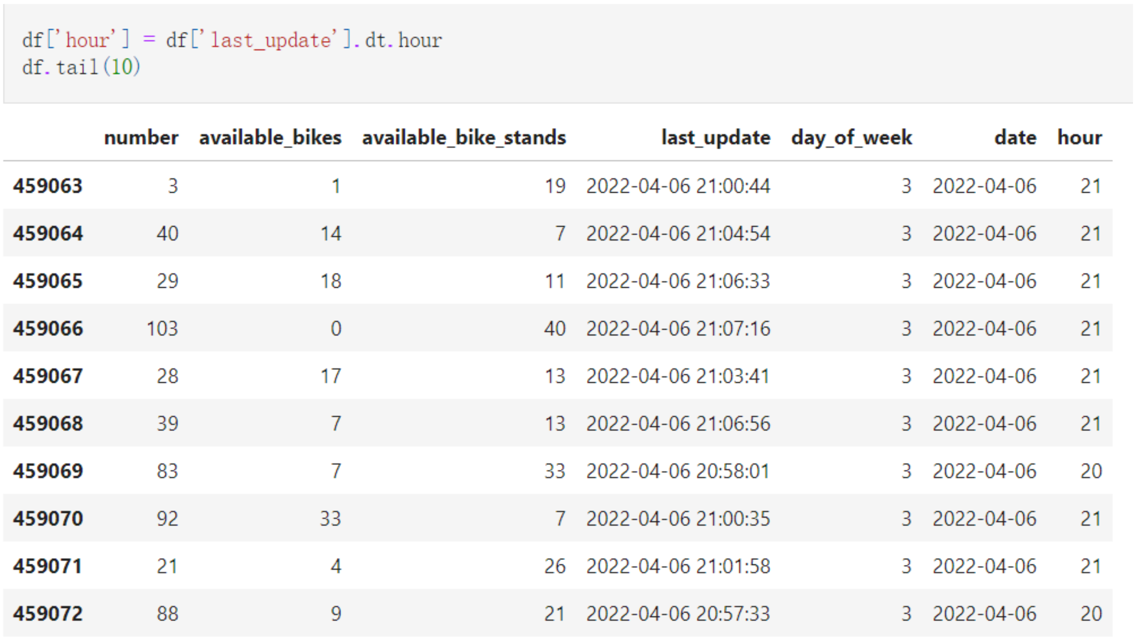
b. Change the last_update number we have collected in int64 format to date



```
df['last_update'] = pd.to_datetime(df['last_update'], unit='s')
df['last_update']
```

```
0        2022-03-10 02:59:03
1        2022-03-10 02:57:47
2        2022-03-10 03:01:36
3        2022-03-10 03:01:35
4        2022-03-10 03:01:34
                 ...
459068   2022-04-06 21:06:56
459069   2022-04-06 20:58:01
459070   2022-04-06 21:00:35
459071   2022-04-06 21:01:58
459072   2022-04-06 20:57:33
Name: last_update, Length: 459073, dtype: datetime64[ns]
```

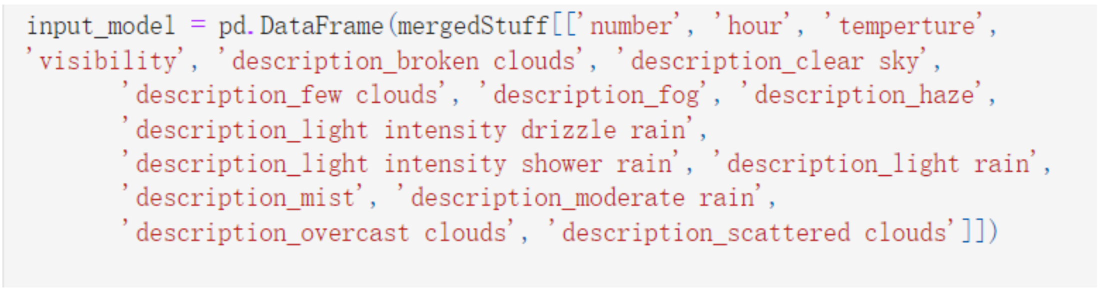
c. In order to be able to train the model and get the predicted weather and number of bikes at a specific hour, we have to add the attribute hour, so we get week and day along with it.



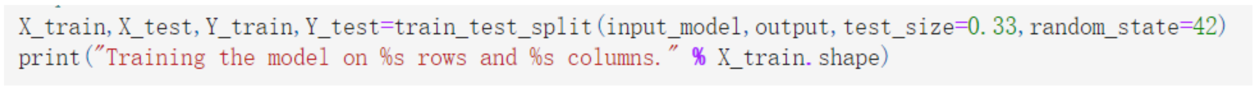
```
df['hour'] = df['last_update'].dt.hour
df.tail(10)
```

	number	available_bikes	available_bike_stands	last_update	day_of_week	date	hour
459063	3	1	19	2022-04-06 21:00:44	3	2022-04-06	21
459064	40	14	7	2022-04-06 21:04:54	3	2022-04-06	21
459065	29	18	11	2022-04-06 21:06:33	3	2022-04-06	21
459066	103	0	40	2022-04-06 21:07:16	3	2022-04-06	21
459067	28	17	13	2022-04-06 21:03:41	3	2022-04-06	21
459068	39	7	13	2022-04-06 21:06:56	3	2022-04-06	21
459069	83	7	33	2022-04-06 20:58:01	3	2022-04-06	20
459070	92	33	7	2022-04-06 21:00:35	3	2022-04-06	21
459071	21	4	26	2022-04-06 21:01:58	3	2022-04-06	21
459072	88	9	21	2022-04-06 20:57:33	3	2022-04-06	20

d. Select model and Set up tests



```
input_model = pd.DataFrame(mergedStuff[['number', 'hour', 'temperture',
'visibility', 'description_broken clouds', 'description_clear sky',
        'description_few clouds', 'description_fog', 'description_haze',
        'description_light intensity drizzle rain',
        'description_light intensity shower rain', 'description_light rain',
        'description_mist', 'description_moderate rain',
        'description_overcast clouds', 'description_scattered clouds']])
```

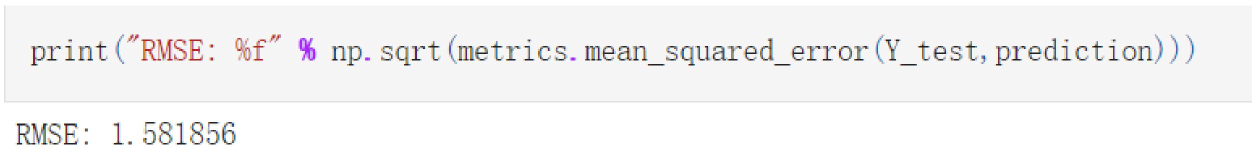


```
X_train, X_test, Y_train, Y_test=train_test_split(input_model, output, test_size=0.33, random_state=42)
print("Training the model on %s rows and %s columns." % X_train.shape)
```

e. Train the model

```
# Train the model
rf.fit(X_train, Y_train)

print("Testing the model on %s rows." % Y_test.shape[0])
LinearReg=LinearRegression()
LinearReg.fit(X_train,Y_train)
pred=LinearReg.predict(X_test)
pred1=pd.DataFrame(pred)
print (pred1)
print(X_test)
print(np.array(X_test))

# Get prediction for test cases
prediction = rf.predict(X_test)
```

After training the model, we get the rmse value



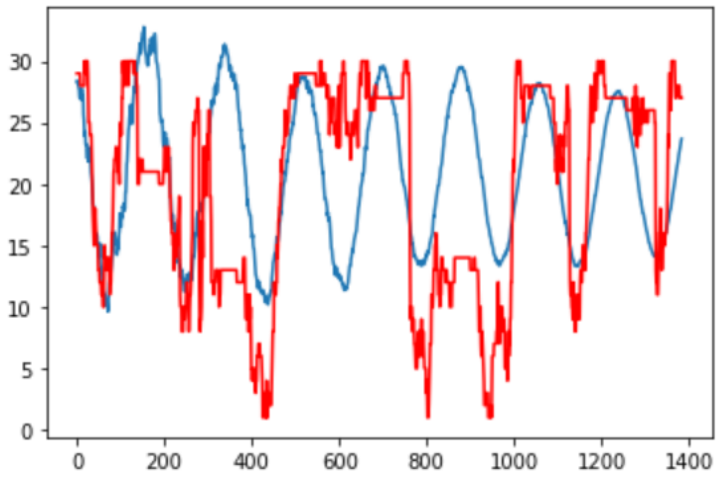
```
print("RMSE: %f" % np.sqrt(metrics.mean_squared_error(Y_test,prediction)))
```

```
RMSE: 1.581856
```

Auto Regression Result

```
model = AutoReg(train, lags = 500).fit()
logmodel = AutoReg(trainl, lags = 500).fit()
print(model.summary())
```

```
                            AutoReg Model Results
==============================================================================
Dep. Variable:                      y   No. Observations:                 2768
Model:                    AutoReg(500)  Log Likelihood               -3095.384
Method:                Conditional MLE  S.D. of innovations              0.947
Date:                 Wed, 06 Apr 2022  AIC                           7194.767
Time:                         22:15:22  BIC                          10069.547
Sample:                            500  HQIC                          8243.622
                                  2768
==============================================================================
```



The data was analyzed by machine learning, which discovered key traits that will aid your model. We do feature engineering since we know this is the data that will be passed to the machine learning model. However, feature engineering takes a long time, and we don't want to lose the feature engineered data after the laptop is turned off.

Pickle receives the feature-engineered data and saves it in binary format.

```
pickle.dump(rf, open('final_prediction.pickle', 'wb'))
```

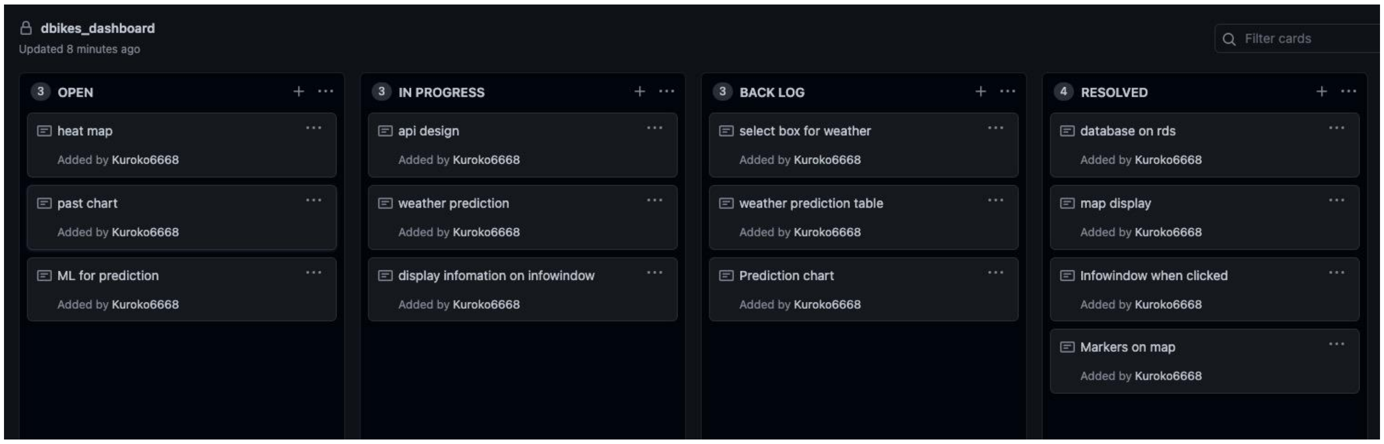
Then, in the view.py, reload the pickled data.

```
cur_dir = os.path.dirname(__file__)
random_forest = pickle.load(open(os.path.join(cur_dir,'final_prediction.pickle'),'rb'))
```

When the front-end send post request, the flask will put the station id, weather information, time into the ML module and response the front-end with prediction.

5. Team Collaboration

dashboard of Sprint:


dbikes_dashboard
Updated 8 minutes ago

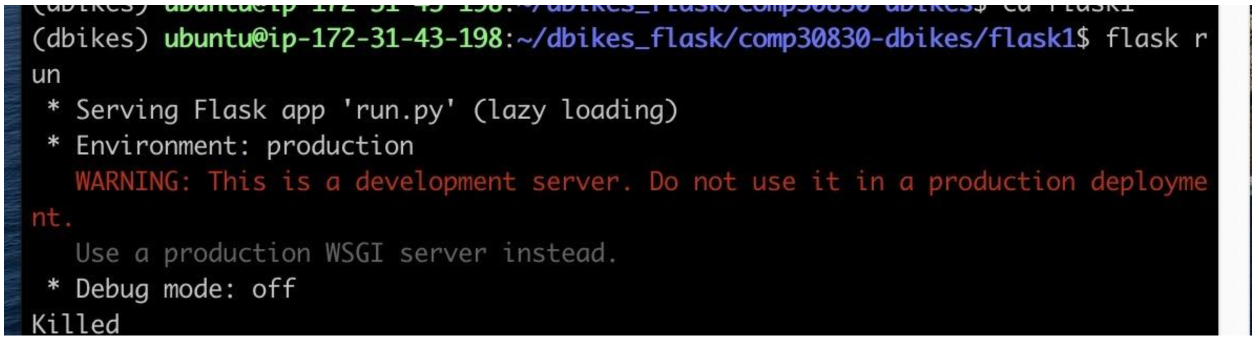
3 OPEN	3 IN PROGRESS	3 BACK LOG	4 RESOLVED
heat map Added by Kuroko6668	api design Added by Kuroko6668	select box for weather Added by Kuroko6668	database on rds Added by Kuroko6668
past chart Added by Kuroko6668	weather prediction Added by Kuroko6668	weather prediction table Added by Kuroko6668	map display Added by Kuroko6668
ML for prediction Added by Kuroko6668	display infomation on infowindow Added by Kuroko6668	Prediction chart Added by Kuroko6668	Infowindow when clicked Added by Kuroko6668
			Markers on map Added by Kuroko6668

5.1 Solved Problems

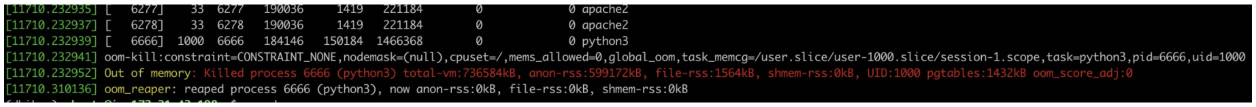
5.1.1 Machine learning models cannot be run because of the limited memory of ec2

Initially, after we uploaded the file to EC2 and ran it, we encountered an out-of-memory situation.

The decision tree for our model starts out at 100+ with 28 feature variables. When we passed this flask file to EC2 to run, the process was killed because of out of memory.



```
(dbikes) ubuntu@ip-172-31-43-198:~/dbikes_flask/comp30830-dbikes/flask1$ flask r
un
 * Serving Flask app 'run.py' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployme
nt.
   Use a production WSGI server instead.
 * Debug mode: off
Killed
```



```
[11710.232935] [   6277]    33  6277   190036     1419   221184        0             0 apache2
[11710.232937] [   6278]    33  6278   190036     1419   221184        0             0 apache2
[11710.232939] [   6666]  1000  6666   184146   150184  1466368        0             0 python3
[11710.232941] oom-kill:constraint=CONSTRAINT_NONE,nodemask=(null),cpuset=/,mems_allowed=0,global_oom,task_memcg=/user.slice/user-1000.slice/session-1.scope,task=python3,pid=6666,uid=1000
[11710.232952] Out of memory: Killed process 6666 (python3) total-vm:736584kB, anon-rss:599172kB, file-rss:1564kB, shmem-rss:0kB, UID:1000 pgtables:1432kB oom_score_adj:0
[11710.310136] oom_reaper: reaped process 6666 (python3), now anon-rss:0kB, file-rss:0kB, shmem-rss:0kB
```

To enable the machine learning model to run smoothly on ec2, we cut the accuracy of the model predictions. We reduced the number of input features and decision trees to allow the flask file to run smoothly on EC2, reducing the size of the generated pickle to a staggering 20MB.

5.1.2 Asynchronous requests and concurrent execution of mutiple requests cause display exceptions

The front end wants to get a prediction of future bikes and needs to send a post request to the back end. When I want to draw a line graph of bicycle forecasts for each of the next 48 hours, I need to send 48 requests to the back end to get an array of 48 in length.

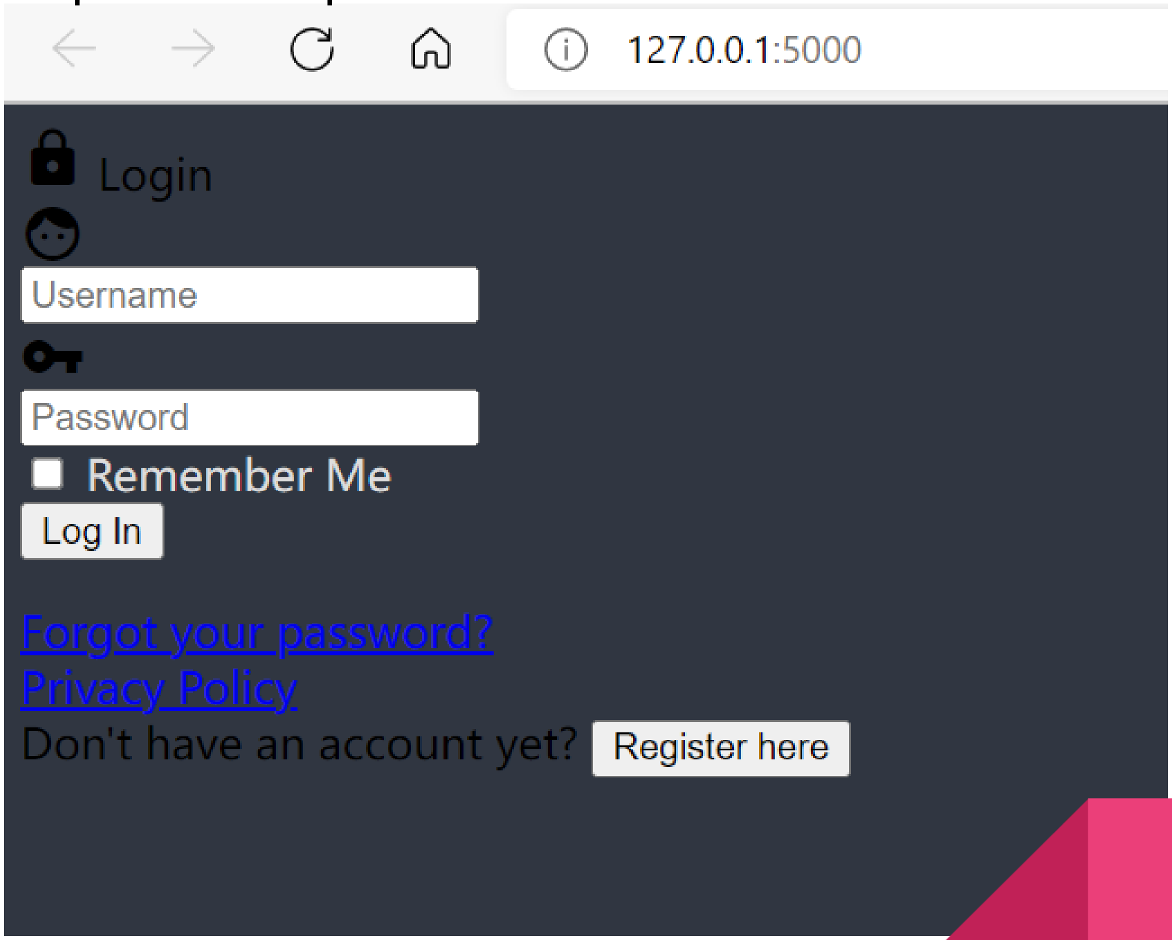
This creates an asynchronous request and concurrent execution of multiple requests, to solve this I use a variable that determines if the current return is equal to the array length of 48. Since the responses are returned at different times, I need to finally sort the forecast array to get the results in order.

5.2 Areas to Improve

5.2.1 Add a login page

As a complete website, we intend to integrate the registration page at a later date and the csv file generated by the registration is also stored in the rds database, but due to time constraints we have not completed this at the time the site went live, but we have made a separate login page that can be integrated at a later date.

For specific flask files please refer to our *GitHub*

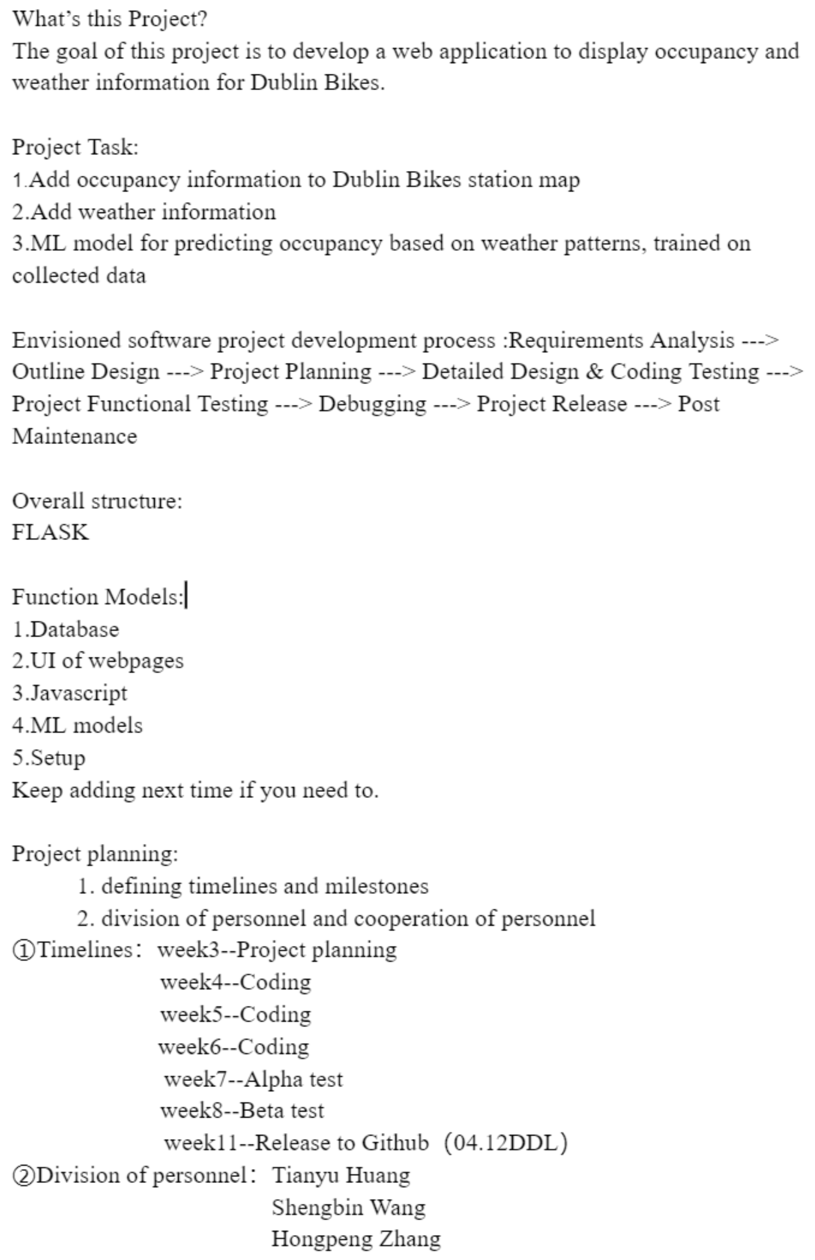


5.2.2 Use XGBoost Model

Again due to time constraints, later we would like to switch our prediction model to XGBoost and we have created a specific test file in github to test xgboost.

6. Meeting Minutes

At our first meeting in 12/02/2022, we had planned everything in detail.



What's this Project?
The goal of this project is to develop a web application to display occupancy and weather information for Dublin Bikes.

Project Task:
1.Add occupancy information to Dublin Bikes station map
2.Add weather information
3.ML model for predicting occupancy based on weather patterns, trained on collected data

Envisioned software project development process :Requirements Analysis ---> Outline Design ---> Project Planning ---> Detailed Design & Coding Testing ---> Project Functional Testing ---> Debugging ---> Project Release ---> Post Maintenance

Overall structure:
FLASK

Function Models:
1.Database
2.UI of webpages
3.Javascript
4.ML models
5.Setup
Keep adding next time if you need to.

Project planning:
1. defining timelines and milestones
2. division of personnel and cooperation of personnel

①Timelines： week3--Project planning
week4--Coding
week5--Coding
week6--Coding
week7--Alpha test
week8--Beta test
week11--Release to Github（04.12DDL）

②Division of personnel： Tianyu Huang
Shengbin Wang
Hongpeng Zhang

6.1 Meeting Logs

Demonstrator: MichelleDuggan

12/02/2022 – Zoom(Our first meeting)

Attendants: All

● The schedule for the project was determined and all the software and languages needed for the project were identified.

01/03/2022 - SE practical

Attendants: All

● Connect the scraper to the database - can do this locally

● Push this scraper to Github

● Test the script in the EC2 and RDS

● Find a way to keep the script running all the time

03/03/2022 - SE practical

Attendants: Tianyu Huang/ Shengbin Wang

● Discuss how the front-end code is implemented

● Try to make a simple page out of JavaScript and Api

● Learn how to analysis of the collected data

● The front-end is complete

<u>04/03/2022 - Discord</u>

<u>Attendants: Tianyu Huang/ MichelleDuggan</u>

<u>● We were behind schedule due to group member Hongpeng Zhang and I completed the script for the data collection and started the data collection.</u>

08/03/2022 - SE practical

Attendants: Tianyu Huang/ Shengbin Wang

● Exploration of the problem of non-convergence of data

● Test the Linear Regression Model with some data

10/03/2022 - SE practical

Attendants: Tianyu Huang/ Shengbin Wang

● Thinking about how to do data analysis

● Test the Auto Regression Model

29/03/2022 – SE practical

Attendants: Tianyu Huang/ Shengbin Wang

● After performing a complete data cleaning, share the results

● Tasks for sprint

● Determine the model and train it to get the training data

05/04/2022 – SE practical

Attendants: Tianyu Huang/ Shengbin Wang

● Try to set up the site to EC2 and try to do some finishing work