

IoT-AI weather station

With Raspberry Pi

지도교수 : 이상훈 교수님
20141203 박현승
20141217 정영관

CONTENTS

CONTENTS A

개발 도구

CONTENTS C

소스 설명

CONTENTS E

향후 계획

CONTENTS B

부품 설명

CONTENTS D

시연 & 테스트

CONTENTS F

Q & A

개발 환경 & 개발 도구



Raspberry Pi



Node.js



MongoDB



Libraries



VSCode



Colab

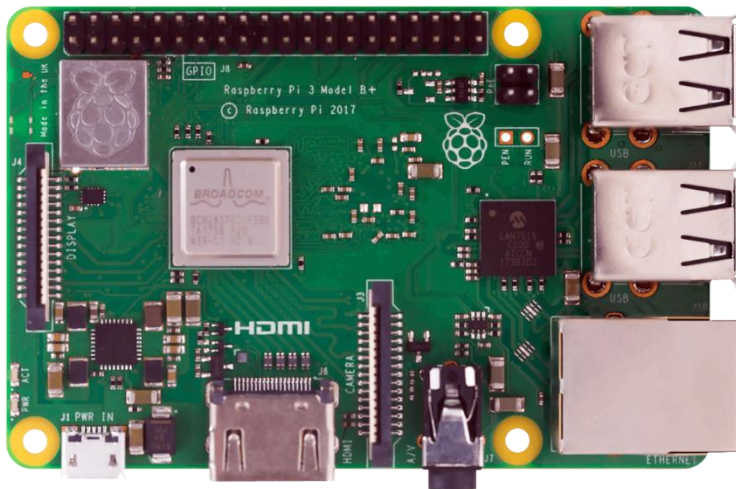


Python3



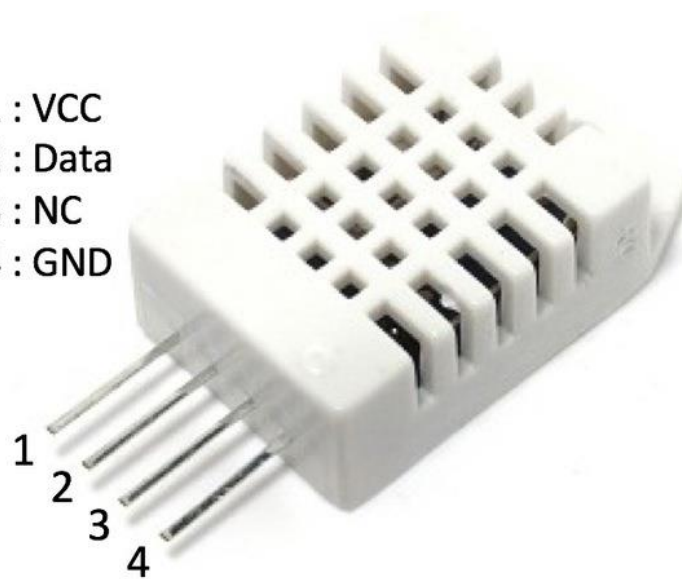
Plotly.js

부품 설명



- 초소형/초저가의 컴퓨터이다.
- OS로 리눅스(Raspbian)를 사용한다.
- 활용도가 무궁무진하다.
- 사용전압 : 5V 3A

- 1 : VCC
- 2 : Data
- 3 : NC
- 4 : GND



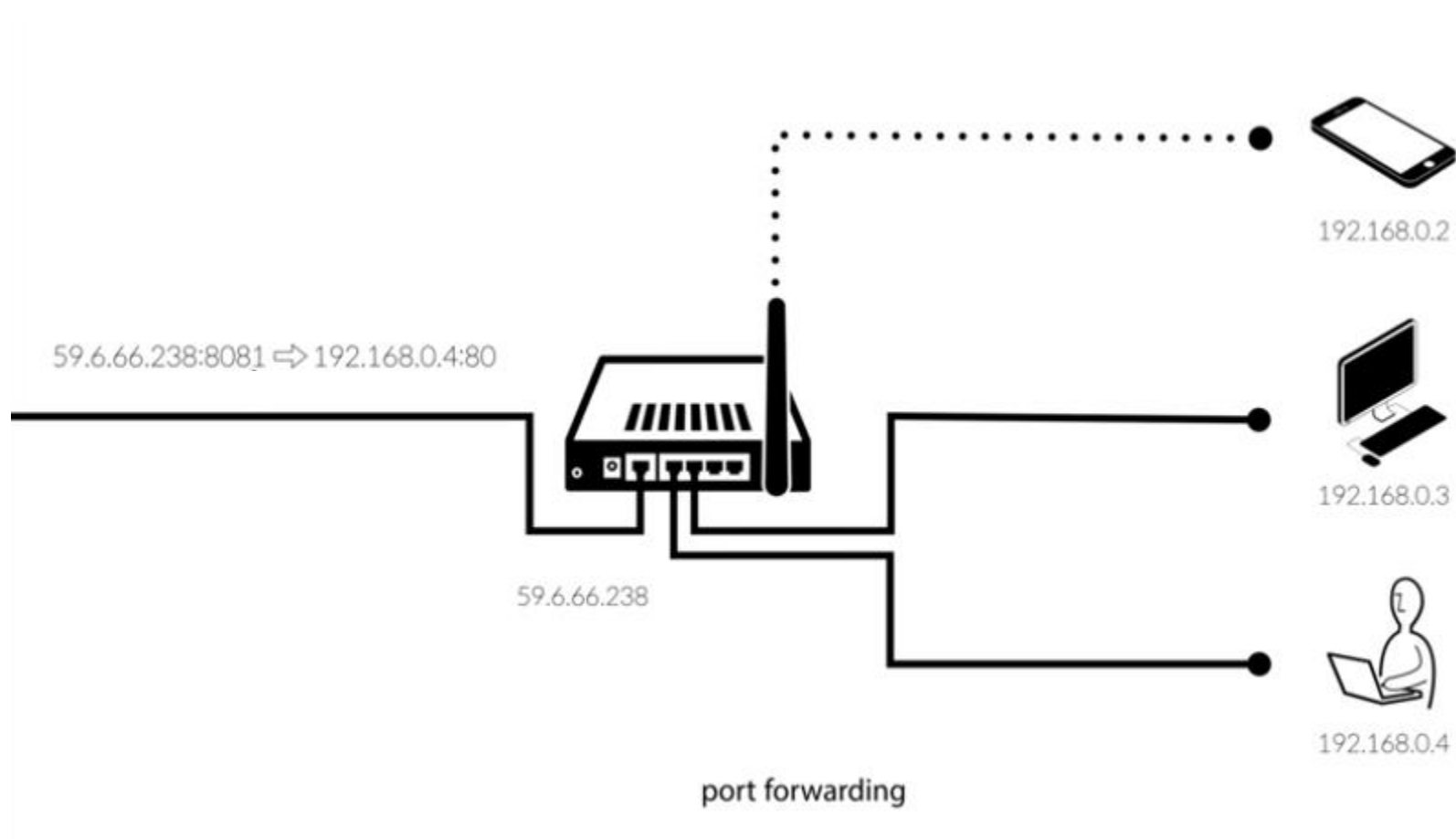
- 정전식 습도센서, NTC온도센서 사용
- 온도 측정 범위 -> $-40 \sim -80^{\circ}\text{C}$
습도 측정 범위 -> $0 \sim 100\% \text{ RH}$
- 온도오차 -> 0.5°C , 습도오차 -> 2%
- 사용전압 : 3.3~5V DC

Pin-Map

Raspberry Pi GPIO Header						
BCM	WiringPi	Name	Physical	Name	WiringPi	BCM
		3.3v	1	2	5v	
2	8	SDA.1	3	4	5V	
3	9	SCL.1	5	6	0v	
4	7	1-Wire	7	8	TxD	15 14
		0v	9	10	RxD	16 15
17	0	GPIO. 0	11	12	GPIO. 1	1 18
27	2	GPIO. 2	13	14	0v	
22	3	GPIO. 3	15	16	GPIO. 4	4 23
		3.3v	17	18	GPIO. 5	5 24
10	12	MOSI	19	20	0v	
9	13	MISO	21	22	GPIO. 6	6 25
11	14	SCLK	23	24	CE0	10 8
		0v	25	26	CE1	11 7
0	30	SDA.0	27	28	SCL.0	31 1
5	21	GPIO.21	29	30	0v	
6	22	GPIO.22	31	32	GPIO.26	26 12
13	23	GPIO.23	33	34	0v	
19	24	GPIO.24	35	36	GPIO.27	27 16
26	25	GPIO.25	37	38	GPIO.28	28 20
		0v	39	40	GPIO.29	29 21
BCM	WiringPi	Name	Physical	Name	WiringPi	BCM

- GPIO핀을 사용하여 센서제어
- GPIO핀을 사용할시 3.3V만 사용
- Physical : HW상의 물리적인 번호
WiringPi : C언어에서 사용
BCM : Python에서 사용

포트 포워딩이란?



| app.py

```
1  #!/usr/bin/python3
2  #sudo pigpiod 시작하기전 GPIO핀 활성화가 필요합니다.
3  from flask import Flask
4  import pigpio
5  import DHT22
6
7  app = Flask(__name__)
8
9  #Initiate GPIO for pigpio
10 pi = pigpio.pi()
11
12 #Setup the sensor
13 dht22 = DHT22.sensor(pi, 23) # use the actual GPIO pin name
14 dht22.trigger()
15
16 #동적 html 페이지의 소스코드입니다, format을 사용하여 데이터를 넣어주었습니다.
17 htmlCode = '''
18 <!DOCTYPE HTML>
19 <html>
20 <head><meta http-equiv="refresh" content="5" /></head>
21
22 <body>
23
24 <h1>RaspberryPi - Temperature and Humidity</h1>
25
26 <dl id="data">
27
28 <dt class="Celsius">
29 <h2>Temperature in Celsius : {cel} *C</h2>
30 </dt>
31
32 <dt class='Fahrenheit'>
33 <h2>Temperature in Fahrenheit : {fah} *F</h2>
34 </dt>
35
36 <dt class='Humidity'>
37 <h2>Humidity : {humi} %</h2>
38 </dt>
39 </dl>
40 </body>
41 </html>'''
```

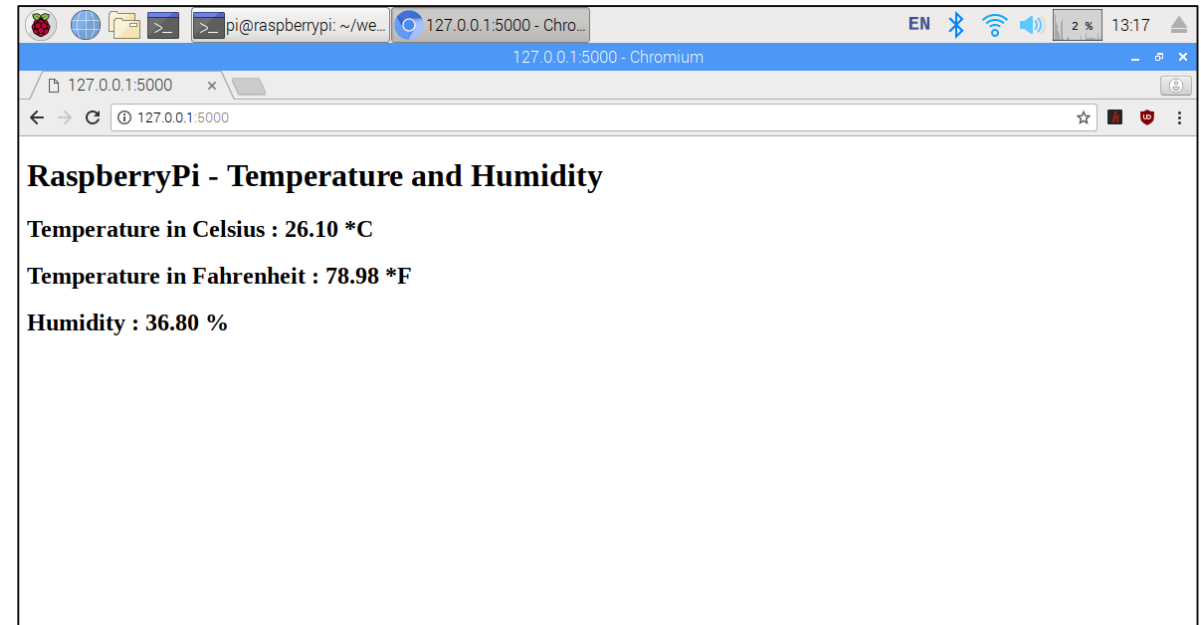
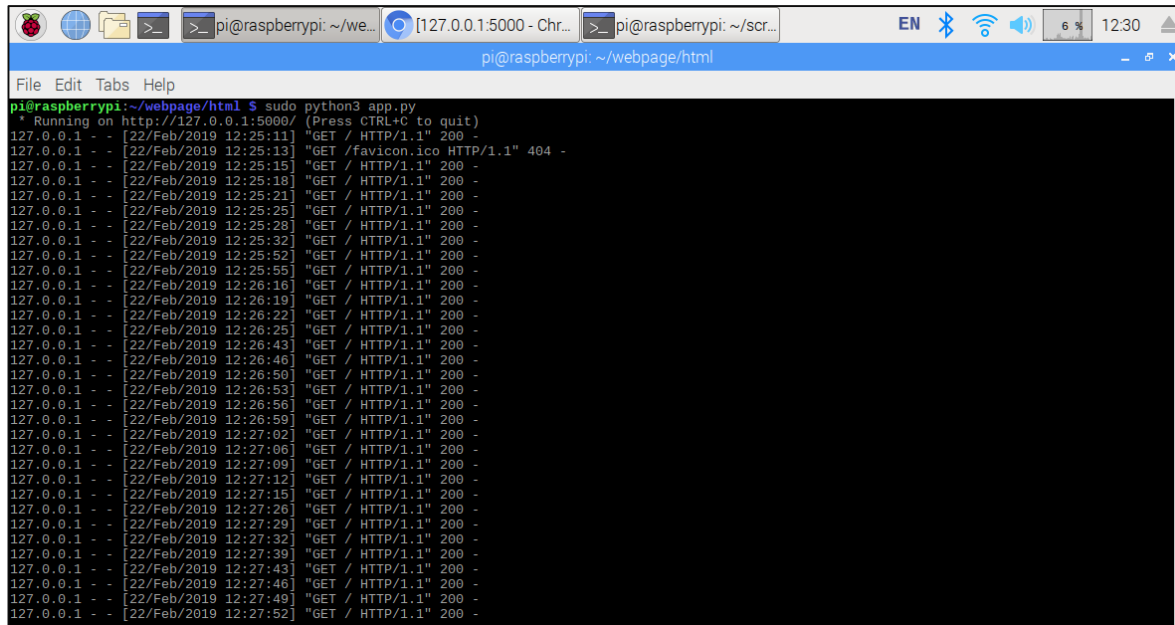
- Python flask 사용 동적 html 페이지 제작
- GPIO핀 제어를 위해 #sudo pigpiod 활성화
- GPIO 모듈 import받아 DHT22 제어
- GPIO BCM 22번 핀 사용
- 생성된 html 페이지는 5초마다 refresh.

| app.py

```
46
47 def readDHT22():
48     # Get a new reading
49     dht22.trigger()
50     # Save our values
51     humidity = '%.2f' % (dht22.humidity()) #humidity
52     temp = '%.2f' % (dht22.temperature()) #Celsius
53     fahr = '%.2f' % (dht22.temperature() * 1.8 + 32) #Fahrenheit
54     return (humidity, temp, fahr)
55
56 @app.route('/')
57 def dht_22():
58     humidity, temperature, fahr = readDHT22()
59
60     return htmlCode.format(cel=temperature, fah=fahr, humi=humidity)
61
62 if __name__ == '__main__':
63     app.run()
64
```

- ("/)로 들어가면 페이지를 볼 수 있고 port는 5000번으로 running 됨.

app.py(실행결과)



scrape

```
1 var cheerio = require('cheerio');
2 var request = require('request');
3 var io = require('socket.io').listen(3000, function (req, res) {
4   console.log('Listening on port 3000');
5 });
6
7 // MongoDB
8 var mongoose = require('mongoose');
9 var Schema = mongoose.Schema;
10 // MongoDB connection
11 mongoose.Promise = global.Promise;
12 mongoose.connect('mongodb://localhost:27017/iot',{
13   keepAlive: 300000,
14   connectTimeoutMS: 30000,
15 }, (err) => {
16   if (err) {
17     console.log('===> Error connecting to th');
18     console.log('Reason: th');
19   } else {
20     console.log('===> Succeeded in connecting to th');
21   }
22 }); // DB name
23
24 // Schema
25 var thSchema = new Schema({
26   date : String,
27   temperature : String,
28   humidity : String
29 });
30 // Display data on console in the case of saving data.
31 thSchema.methods.info = function () {
32   var thInfo = this.date
33   ? "date: " + this.date + ", Temp: " + this.temperature
34   + ", Humi: " + this.humidity
35   : "I don't have a date";
36   console.log("thInfo: " + thInfo);
37 };
38
39 var dht22data = [];
40 var temp_correlation = [];
41 var humi_correlation = [];
42 var dateStr = '';
43 var CelsiusData = '';
44 var HumidityData = '';
45 var Pearson_correlation = '0';
46
47 var TH = mongoose.model("th", thSchema); // sensor data model
48
```

- Module import
- Set socket running port 3000
- MongoDB connect
- MongoDB schema Definition
- Promise 설정

| scrape

```
49 var url = 'http://127.0.0.1:5000/'; // Flask address
50
51 var Celsius = '';
52
53 function getCelsius() {
54
55     request(url, function (err, res, body) {
56
57         var $ = cheerio.load(body);
58
59         $('#data .Celsius').each(function () {
60             Celsius = $(this).text().substring(26, 31);
61         });
62     });
63     return Celsius;
64 }
65
66 var Humidity = '';
67
68 function getHumidity() {
69
70     request(url, function (err, res, body) {
71
72         var $ = cheerio.load(body);
73
74         $('#data .Humidity').each(function () {
75             Humidity = $(this).text().substring(12, 17);
76         });
77     });
78     return Humidity;
79 }
80
81
82 function getDateString() {
83
84     var time = new Date().getTime();
85     var datestr = new Date(time + 32400000).toISOString().replace(/T/, ' ').replace(/Z/, '');
86
87     return datestr;
88 }
```

- Temperature, Humidity data scraping

| Correlation Coefficient

상관관계 계수	상관관계 정도	상관관계 계수	상관관계 정도
± 0.9 이상	매우 높다	$\pm 0.2 - \pm 0.4$ 미만	낮은 상관관계
$\pm 0.7 - \pm 0.9$ 미만	높은 상관관계	± 0.2 미만	상관관계가 거의 없음
$\pm 0.4 - 0.7$ 미만	다소 높은 상관관계		

scrape

```
90 function getPearsonCorrelation(x, y) {  
91  
92     var shortestArrayLength = 0;  
93  
94     if( x.length == y.length ) {  
95         shortestArrayLength = x.length;  
96     } else if(x.length > y.length) {  
97         shortestArrayLength = y.length;  
98         console.error('x has more items in it, the last ' + (x.length - shortestArrayLength) + ' item(s) will be ignored');  
99     } else {  
100         shortestArrayLength = x.length;  
101         console.error('y has more items in it, the last ' + (y.length - shortestArrayLength) + ' item(s) will be ignored');  
102     }  
103  
104     var xy = [];  
105     var x2 = [];  
106     var y2 = [];  
107  
108     for(var i = 0; i < shortestArrayLength; i++) {  
109         xy.push(x[i] * y[i]);  
110         x2.push(x[i] * x[i]);  
111         y2.push(y[i] * y[i]);  
112     }  
113  
114     var sum_x = 0;  
115     var sum_y = 0;  
116     var sum_xy = 0;  
117     var sum_x2 = 0;  
118     var sum_y2 = 0;  
119  
120     for(var j = 0; j < shortestArrayLength; j++) {  
121         sum_x += x[j];  
122         sum_y += y[j];  
123         sum_xy += xy[j];  
124         sum_x2 += x2[j];  
125         sum_y2 += y2[j];  
126     }  
127  
128     var step1 = (shortestArrayLength * sum_xy) - (sum_x * sum_y);  
129     var step2 = (shortestArrayLength * sum_x2) - (sum_x * sum_x);  
130     var step3 = (shortestArrayLength * sum_y2) - (sum_y * sum_y);  
131     var step4 = Math.sqrt(step2 * step3);  
132     var answer = ((step1 / step4).toFixed(2)) + "";  
133  
134     return answer;  
135 }  
136
```

- Get the correlation coefficient

scrape

```
137 function get_tensorflow(Celsius,Humidity){
138     // CR = W[0]*T + W[1]*H + b
139     tensorflow_cr = ((0.07446165) * Celsius) + (0.11316531 * Humidity) + (-2.8271704);
140
141     return tensorflow_cr + "";
142 }
143
144 function get_sklearn(Celsius,Humidity){
145     // CR = W[0]*T + W[1]*H + b
146     var sklearn_list = []
147
148     sklearn_list[0] = ((-0.09939661) * Celsius) + ((-0.25826953) * Humidity) + (12.35153308);
149     sklearn_list[1] = ((-0.01450908) * Celsius) + ((-0.30808969) * Humidity) + (12.12950693);
150     sklearn_list[2] = ((-0.13974128) * Celsius) + ((0.01788321) * Humidity) + (2.7670086);
151     sklearn_list[3] = ((0.11125783) * Celsius) + ((0.27725472) * Humidity) + (-13.42706412);
152     sklearn_list[4] = ((0.14238915) * Celsius) + ((0.27122129) * Humidity) + (-13.8209845);
153
154     var max = Math.max.apply(null, sklearn_list); // The largest value of the elements in the array.
155     var sklearn_cr = (sklearn_list.indexOf(max))+1; // index of array
156
157     return sklearn_cr + "";
158 }
159
160 function get_keras(Celsius,Humidity){
161     // CR = W[0]*T + W[1]*H + b
162
163     keras_cr = ((0.04672196) * Celsius) + (0.10079119 * Humidity) + (-1.5019808);
164
165     return keras_cr + "";
166 }
```

- ML모듈인 tensorflow, sklearn, keras을 사용하여 진행하였습니다.
- 각 모델의 weight와 bias를 구하여 쾌적도값 구함.
- $CR = W[0]*T + W[1]*H + b$
- Socket connect

scrape

```

164 setInterval( async (req, res) => {
165     dateStr = getDateString();
166     CelsiusData = getCelsius();
167     HumidityData = getHumidity();
168     tensorflowData = get_tensorflow(CelsiusData,HumidityData);
169     sklearnData = get_sklearn(CelsiusData,HumidityData);
170     kerasData = get_keras(CelsiusData,HumidityData);
171
172     dht22data[0] = dateStr; // Date
173     dht22data[1] = CelsiusData; // temperature data
174     dht22data[2] = HumidityData; // humidity data
175     dht22data[3] = Pearson_correlation; //correlation data
176     dht22data[4] = tensorflowData; // tensorflow cr data
177     dht22data[5] = sklearnData; // sklearn cr data
178     dht22data[6] = kerasData; // keras cr data
179
180     if(temp_Correlation.length < 10 && humi_Correlation.length < 10){
181         temp_Correlation.push((parseFloat(CelsiusData)) + (parseFloat(Math.random()/1000).toFixed(7)));
182         humi_Correlation.push(parseFloat(HumidityData));
183     }else{
184         Pearson_correlation = getPearsonCorrelation(temp_Correlation, humi_Correlation);
185         temp_Correlation.splice(0, 1);
186         temp_Correlation.push((parseFloat(CelsiusData)) + (parseFloat(Math.random()/1000).toFixed(7)));
187         humi_Correlation.splice(0, 1);
188         humi_Correlation.push(parseFloat(HumidityData));
189     }
190 }
191
192 var th = new TH({date:dateStr, temperature:CelsiusData, humidity:HumidityData});
193 // save iot data (document) to MongoDB
194 try {
195     let newTH = await th.save();
196 }catch(err) {
197     console.log(err);
198 }
199
200
201 io.sockets.emit('message', dht22data);
202 //console.log("Raspberry Pi," + dht22data);
203 }, 6000);
204

```

- setInterval 사용하여 6초마다 한번씩 실행
- MongoDB save를 Async/await 로 Asynchronous 처리
- date, temperature, humidity data를 mongoDB에 저장
- correlation coefficient 계산을 위해 온,습도 데이터에 노이즈 추가
- correlation coefficient 계산은 sliding Window 방식
- 3초마다 한번씩 socket으로 전송 및 MongoDB에 데이터 저장

Scrape(실행결과)

```
pi@raspberrypi: ~/scrape
File Edit Tabs Help
pi@raspberrypi:~/scrape $ sudo node index_scrape.c.js
(node:1083) DeprecationWarning: current URL string parser is deprecated, and will
be removed in a future version. To use the new parser, pass option { useNewUrlParser
Parser: true } to MongoClient.connect.
mongo db connection OK.
Raspberrypi,2019-04-18 16:10:50.336,,0
Raspberrypi,2019-04-18 16:10:53.336,24.10,44.50,0
Raspberrypi,2019-04-18 16:10:56.337,25.10,45.30,0
Raspberrypi,2019-04-18 16:10:59.340,25.10,45.70,0
Raspberrypi,2019-04-18 16:11:02.343,25.20,44.60,0
Raspberrypi,2019-04-18 16:11:05.344,25.20,45.80,0
Raspberrypi,2019-04-18 16:11:08.346,25.20,45.90,0
Raspberrypi,2019-04-18 16:11:11.349,25.20,45.80,0
Raspberrypi,2019-04-18 16:11:14.351,25.20,44.90,0
Raspberrypi,2019-04-18 16:11:17.351,25.20,46.30,0
Raspberrypi,2019-04-18 16:11:20.354,25.20,46.50,0
Raspberrypi,2019-04-18 16:11:23.355,25.20,46.50,NaN
Raspberrypi,2019-04-18 16:11:26.358,25.20,46.60,0.54
Raspberrypi,2019-04-18 16:11:29.373,25.20,45.50,0.19
Raspberrypi,2019-04-18 16:11:32.376,25.20,45.50,0.08
Raspberrypi,2019-04-18 16:11:35.377,25.20,45.60,-0.27
Raspberrypi,2019-04-18 16:11:38.378,25.20,45.60,-0.12
Raspberrypi,2019-04-18 16:11:41.379,25.20,45.20,-0.09
Raspberrypi,2019-04-18 16:11:44.379,25.20,45.20,-0.15
Raspberrypi,2019-04-18 16:11:47.382,25.20,45.20,-0.16
Raspberrypi,2019-04-18 16:11:50.382,25.20,45.00,-0.52
Raspberrypi,2019-04-18 16:11:53.383,25.20,45.30,-0.35
Raspberrypi,2019-04-18 16:11:56.386,25.20,46.00,-0.26
Raspberrypi,2019-04-18 16:11:59.387,25.20,44.90,-0.46
Raspberrypi,2019-04-18 16:12:02.391,25.30,45.10,-0.07
Raspberrypi,2019-04-18 16:12:05.394,25.20,45.40,0.22
Raspberrypi,2019-04-18 16:12:08.395,25.20,48.20,-0.22
Raspberrypi,2019-04-18 16:12:11.398,25.40,51.60,-0.21
Raspberrypi,2019-04-18 16:12:14.400,25.50,67.00,-0.16
Raspberrypi,2019-04-18 16:12:17.403,25.90,65.00,0.75
```

```
pi@raspberrypi: ~
File Edit Tabs Help
pi@raspberrypi:~ $ sudo /usr/bin/mongo localhost:27017
MongoDB shell version: 3.0.9
connecting to: localhost:27017/test
Server has startup warnings:
2019-02-27T00:52:05.057+0900 I CONTROL [initandlisten]
2019-02-27T00:52:05.057+0900 I CONTROL [initandlisten] ** NOTE: This is a 32 bit
MongoDB binary.
2019-02-27T00:52:05.057+0900 I CONTROL [initandlisten] ** 32 bit builds a
re limited to less than 2GB of data (or less with --journal).
2019-02-27T00:52:05.057+0900 I CONTROL [initandlisten] ** See http://doch
ub.mongodb.org/core/32bit
2019-02-27T00:52:05.062+0900 I CONTROL [initandlisten]
> show dbs
iot 0.078GB
local 0.078GB
> use iot
switched to db iot
> show collections
sensors
system.indexes
> db.sensors.find().pretty()
{
  "_id" : ObjectId("5c756458e0898e0c64a520e2"),
  "date" : "2019-02-27 01:07:52.714",
  "temperature" : "",
  "Fahrenheit" : "",
  "humidity" : "",
  "uv" : 0
}
{
  "_id" : ObjectId("5c75645be0898e0c64a520e3"),
  "date" : "2019-02-27 01:07:55.714",
  "temperature" : "23.00",
  "Fahrenheit" : "73.40",
  "humidity" : "55.00",
}
```


| Express-generator(www)

```
7  var app = require('../app');
8  var debug = require('debug')('myapp:server');
9  var http = require('http');
10
11  /**
12   * Get port from environment and store in Express.
13   */
14
15  var port = normalizePort(process.env.PORT || '3030');
16  app.set('port', port);
17
18  /**
19   * Create HTTP server.
20   */
21
22  var server = http.createServer(app);
23
24  /**
25   * Listen on provided port, on all network interfaces.
26   */
27
28  server.listen(port);
29  server.on('error', onError);
30  server.on('listening', onListening);
```

- Set express sever port 3030

| Express-generator(app.js)

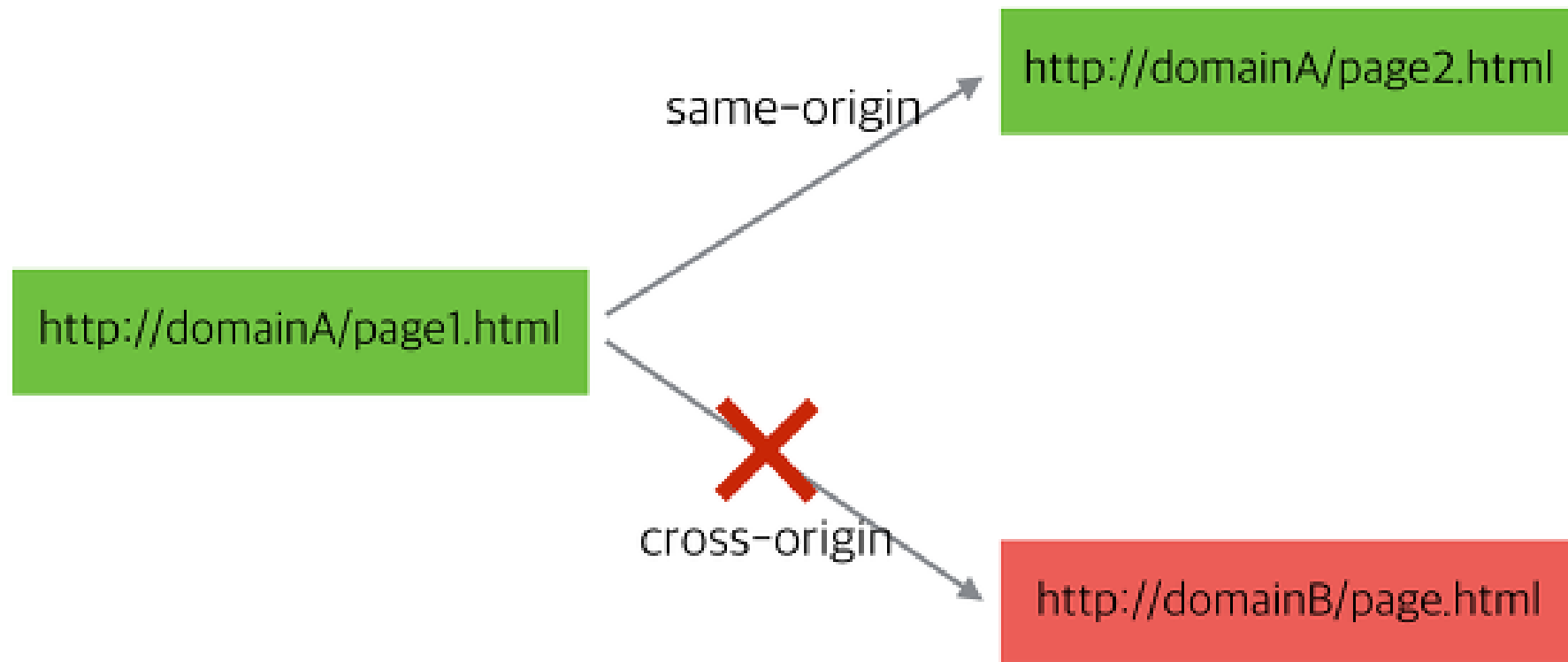
```
6  var cors = require('cors')();  
7  //var bodyParser = require('body-parser');  
8  var indexRouter_correlation = require('./routes/index_c');  
9  var indexRouter_CR = require('./routes/index_cr');
```

- Set CORS and router

```
26  app.use(express.static(path.join(__dirname, 'public')));  
27  
28  app.use('/cr_value', indexRouter_CR);  
29  app.use('/', indexRouter_correlation);
```

- Set router path

| Cors(Cross-Origin Resource Sharing) 란?



Routes(index_c.js)

```
1  var express = require('express');
2  var router = express.Router();
3
4  // MongoDB
5  var mongoose = require('mongoose');
6  var Schema = mongoose.Schema;
7  // MongoDB connection
8  mongoose.Promise = global.Promise;
9  mongoose.connect('mongodb://localhost:27017/iot',{
10     keepAlive: 300000,
11     connectTimeoutMS: 30000,
12 }, (err) => {
13     if (err) {
14         console.log('==> Error connecting to th');
15         console.log('Reason: th');
16     } else {
17         console.log('==> Succeeded in connecting to th');
18     }
19 }); // DB name
20
21 // Schema
22 var thSchema = new Schema({
23     date : String,
24     temperature : String,
25     humidity : String
26 });
27
28 var TH = mongoose.model("th", thSchema); // sensor data model
29
```

- MongoDB connect
- Created mongoDB object
- DB name is 'iot', collection name is 'th'
- Define schema

Routes(index_c.js)

```
30  /* GET home page. */
31  router.get('/', async (req, res, next) => {
32
33      res.render('index',{ title : 'RaspberryPi sensor'});
34  });
35
36  router.get('/th', async (req, res) => {
37
38      try {
39          let th = await TH.find({}).exec();
40          res.json(th);
41      } catch(err){
42          console.log(err);
43      }
44  });
45
46  // find data by id
47  router.get('/th/:id', async (req, res) => {
48
49      try {
50          let th = await TH.findById(req.params.id).exec();
51          res.json(th);
52      } catch(err){
53          console.log(err);
54      }
55  });
56
57  });
58
59  module.exports = router;
60
```

- 모든 mongoDB 로직을 Async/await 로 Asynchronous 처리
- ('/th')로 접속하면 mongoDB에 접근 -> th collections 에서 (date, temperature, humidity) data를 json형식으로 받음

Routes(index_cr.js)

```
1 var express = require('express');
2 var router = express.Router();
3
4 // MongoDB
5 var mongoose = require('mongoose');
6 var Schema = mongoose.Schema;
7 // MongoDB connection
8 mongoose.Promise = global.Promise;
9 mongoose.connect('mongodb://localhost:27017/iot',{
10   keepAlive: 300000,
11   connectTimeoutMS: 30000,
12 }, (err) => {
13   if (err) {
14     console.log('===> Error connecting to cr');
15     console.log('Reason: th');
16   } else {
17     console.log('===> Succeeded in connecting to cr');
18   }
19 }); // DB name
20 // Schema
21 var crSchema = new Schema({
22   date: String,
23   cr: String,
24   gender: String,
25   age : String
26 });
27 // Display data on console in the case of saving data.
28 crSchema.methods.info = function () {
29   var crInfo = this.date
30     ? "date: " + this.date + ", CR: " + this.cr
31     + ", gender: " + this.gender
32     + ", age: " + this.age
33     : "I don't have a date";
34   console.log("crInfo: " + crInfo);
35 };
36
37 var CR = mongoose.model("cr", crSchema); // sensor data model
38
```

- Set router
- MongoDB connect
- Created mongoDB object
- DB name is 'iot', collection name is 'cr'
- Define schema

Routes(index_cr.js)

```
47 router.get('/', async (req, res) => {
48
49     var cr = new CR({ date: getDateString(), cr: req.query.CR,
50                       gender: req.query.gender, age: req.query.age });
51     // save iot data (document) to MongoDB
52     try {
53         let newCR = await cr.save();
54         //console.log(newCR);
55     } catch(err) {
56         console.error(err);
57     }
58
59     res.send("You chose CR value is = " + req.query.CR + " Thank you^^ ");
60 });
61
62 router.get('/cr', async (req, res) => {
63
64     try {
65         let cr = await CR.find({}).exec();
66         res.json(cr);
67     } catch(err){
68         console.log(err);
69     }
70 }
71 });
72
73 router.get('/cr/:id', async (req, res) => {
74
75     try {
76         let cr = await CR.findById(req.params.id).exec();
77         res.json(cr);
78     } catch(err){
79         console.log(err);
80     }
81 }
82 });
83
84 module.exports = router;
```

- 모든 mongoDB 로직을 Async/await 로 synchronous 처리
- ('/')로 접속하면 평가페이지로 부터 받은 데이터를 queryString 으
로 받아서 mongoDB에 저장
- ('/cr')로 접속하면 mongoDB에 접근 -> cr collections 에서
data를 json형식으로 받음

Client Page(client_CR.html)

```

32 <form role="form" action="/cr_value" method="get">
33
34   <h2>쾌적도</h2>
35   <input type="radio" id="bnt1" style="width:30px; height:30px;" name="CR" value="5" checked>
36   <label for="bnt1" style="font-size:30px;"> 좋음</label><br>
37
38   <input type="radio" id="bnt2" style="width:30px; height:30px;" name="CR" value="4">
39   <label for="bnt2" style="font-size:30px;"> 조금 좋음</label><br>
40
41   <input type="radio" id="bnt3" style="width:30px; height:30px;" name="CR" value="3">
42   <label for="bnt3" style="font-size:30px;"> 보통</label><br>
43
44   <input type="radio" id="bnt4" style="width:30px; height:30px;" name="CR" value="2">
45   <label for="bnt4" style="font-size:30px;"> 조금 나쁨</label><br>
46
47   <input type="radio" id="bnt5" style="width:30px; height:30px;" name="CR" value="1">
48   <label for="bnt5" style="font-size:30px;"> 나쁨</label><br>
49   <br>
50
51   <h2>성별</h2>
52   <input type="radio" id="bnt6" style="width:30px; height:30px;" name="gender" value="male" checked>
53   <label for="bnt6" style="font-size:30px;"> 남성</label>
54
55   <input type="radio" id="bnt7" style="width:30px; height:30px;" name="gender" value="female">
56   <label for="bnt7" style="font-size:30px;"> 여성</label><br>
57   <br>
58
59   <h2>나이대</h2>
60   <input type="radio" id="bnt8" style="width:30px; height:30px;" name="age" value="10">
61   <label for="bnt8" style="font-size:30px;"> 10대</label>
62   <input type="radio" id="bnt9" style="width:30px; height:30px;" name="age" value="20" checked>
63   <label for="bnt9" style="font-size:30px;"> 20대</label>
64   <input type="radio" id="bnt10" style="width:30px; height:30px;" name="age" value="30">
65   <label for="bnt10" style="font-size:30px;"> 30대</label>
66   <input type="radio" id="bnt11" style="width:30px; height:30px;" name="age" value="40">
67   <label for="bnt11" style="font-size:30px;"> 40대</label>
68   <input type="radio" id="bnt12" style="width:30px; height:30px;" name="age" value="50">
69   <label for="bnt12" style="font-size:30px;"> 50대</label>
70   <input type="radio" id="bnt13" style="width:30px; height:30px;" name="age" value="60">
71   <label for="bnt13" style="font-size:30px;"> 60대</label><br>
72   <br>
73
74   <input type="submit" style="width:120px; height:60px; font-size:30px;">
75 </form>

```

- 사용자의 쾌적도 값을 입력 받을 수 있는 평가 페이지
- get method를 사용하여 form을 만듦
- 데이터는 쾌적도, 성별, 나이대로 구성됨
- 제출 버튼을 누르면 페이지가 ('/cr_value') 변경되면서 평가한 값이 get method 방식의 queryString으로 URL를 통해 전송됨.

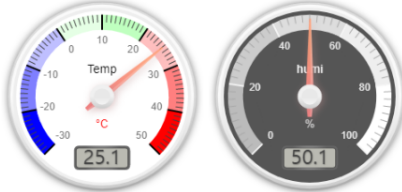
Client Page(client_CR.html)

```
76 <script>
77   var numPts = 50; // number of data points in x-axis
78   var dtda = []; // 1 x 2 array : [date, data1] from sensors
79   var preX = -1;
80   var preY = -1;
81   var initFlag = true;
82
83   var socket = io.connect('125.190.157.52:3000'); // port = 3000
84   socket.on('connect', function() {
85     socket.on('message', function(msg) {
86       // initial plot
87       if (msg[0] != '' && initFlag) {
88         dtda[0] = parseFloat(msg[1]); // temperature
89         dtda[1] = parseFloat(msg[2]); // humidity
90
91         initFlag = false;
92       }
93
94       dtda[0] = parseFloat(msg[1]);
95       dtda[1] = parseFloat(msg[2]);
96
97       // Only when any of temperature or Luminosity is different
98       // from the previous one, the screen is redrawn.
99       if (dtda[0] != preX || dtda[1] != preY) { // any change?
100
101         // when new data is coming, keep on streaming
102         gauge_temperature.setValue(dtda[0]); // temperature gauge
103         gauge_humidity.setValue(dtda[1]); // humidity gauge
104       }
105     });
106   });
107 }
```

- socket을 통해 실시간으로 전송되는 온,습도를 받음
- 사용자의 쾌적도 평가시 참고하기위해 실시간 온습도 게이지를 추가

Client Page(실행결과)

Comfort Requirement Check(쾌적한 정도)



쾌적도

☒ 좋음
☐ 조금 좋음
☐ 보통
☐ 조금 나쁨
☐ 나쁨

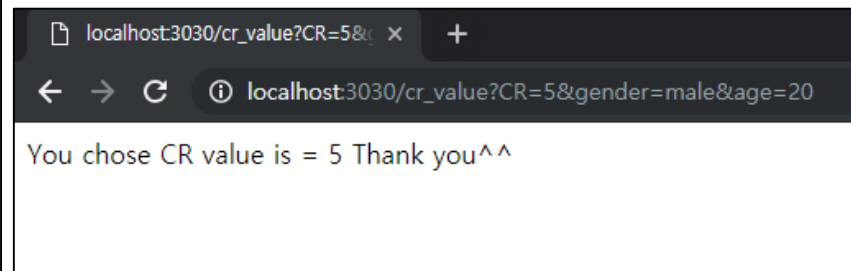
성별

☒ 남성 ☐ 여성

나이대

☐ 10대 ☒ 20대 ☐ 30대 ☐ 40대 ☐ 50대 ☐ 60대

제출



http://203.241.246.114:3030/client_CR.html

Client Page(plotly_c_ml.html)

```
44 <script>
45   var streamPlot = document.getElementById('myDiv');
46   var ctime = document.getElementById('time');
47   var tArray = [], // time of data arrival
48       y1Track = [], // value of sensor 1 : temperature
49       y2Track = [], // value of sensor 2 : humidity
50       numPts = 50, // number of data points in x-axis
51       dtda = [], // 1 x 7 array
52       preX = -1,
53       preY = -1,
54       preZ = -1,
55       preJ = -1,
56       preQ = -1,
57       preK = -1,
58       initFlag = true;
59
60   var socket = io.connect('http://127.0.0.1:3000'); // port = 3000
61   socket.on('connect', function() {
62     socket.on('message', function(msg) {
63       // initial plot
64       if (msg[0] != '' && initFlag) {
65         dtda[0] = msg[0];
66         dtda[1] = parseFloat(msg[1]); // temperature
67         dtda[2] = parseFloat(msg[2]); // humidity
68         dtda[3] = parseFloat(msg[3]); // correlation
69         dtda[5] = parseFloat(msg[5]); // sklearn cr data
70         dtda[4] = parseFloat(msg[4]); // tensorflow cr data
71         dtda[6] = parseFloat(msg[6]); // keras cr data
72         init();
73         initFlag = false;
74       }
75
76       dtda[0] = msg[0];
77       dtda[1] = parseFloat(msg[1]); // temperature
78       dtda[2] = parseFloat(msg[2]); // humidity
79       dtda[3] = parseFloat(msg[3]); // correlation
80       dtda[5] = parseFloat(msg[5]); // sklearn cr data
81       dtda[4] = parseFloat(msg[4]); // tensorflow cr data
82       dtda[6] = parseFloat(msg[6]); // keras cr data
83     });
  }
```

- 6개의 게이지와 온습도를 보여주는 그래프로 구성
- socket을 통해 실시간으로 전송되는 온,습도를 받음

Client Page(plotly_c_ml.html)

```
84 // Only when any of temperature or Luminosity is different
85 // from the previous one, the screen is redrawn.
86 if (dtda[1] != preX || dtda[2] != preY || dtda[3] != preZ || dtda[4] != preJ || dtda[5] != preQ || dtda[6] != preK) { // any change?
87     preX = dtda[1];
88     preY = dtda[2];
89     preZ = dtda[3];
90     preJ = dtda[4];
91     preQ = dtda[5];
92     preK = dtda[6];
93
94     // when new data is coming, keep on streaming
95     ctime.innerHTML = dtda[0];
96     gauge_temperature.setValue(dtda[1]) // temperature gauge
97     gauge_humidity.setValue(dtda[2]); // humidity gauge
98     gauge_correlation.setValue(dtda[3]); // correlation gauge
99     gauge_tensorflow.setValue(dtda[4]); // tensorflow cr data gauge
100    gauge_sklearn.setValue(dtda[5]); // sklearn cr data gauge
101    gauge_keras.setValue(dtda[6]); // keras cr data gauge
102
103    tArray = tArray.concat(dtda[0]);
104    tArray.splice(0, 1); // remove the oldest data
105    y1Track = y1Track.concat(dtda[1]);
106    y1Track.splice(0, 1); // remove the oldest data
107    y2Track = y2Track.concat(dtda[2]);
108    y2Track.splice(0, 1);
109
110    var update = {
111        x: [tArray, tArray],
112        y: [y1Track, y2Track]
113    }
114
115    Plotly.update(streamPlot, update);
116 }
117
118 });
119
120
121 function init() { // initial screen ()
122     for (i = 0; i < numPts; i++) {
123         tArray.push(dtda[0]); // date
124         y1Track.push(dtda[1]); // sensor 1 (temp)
125         y2Track.push(dtda[2]); // sensor 2 (humi)
126     }
127
128     Plotly.plot(streamPlot, data, layout);
129 }
```

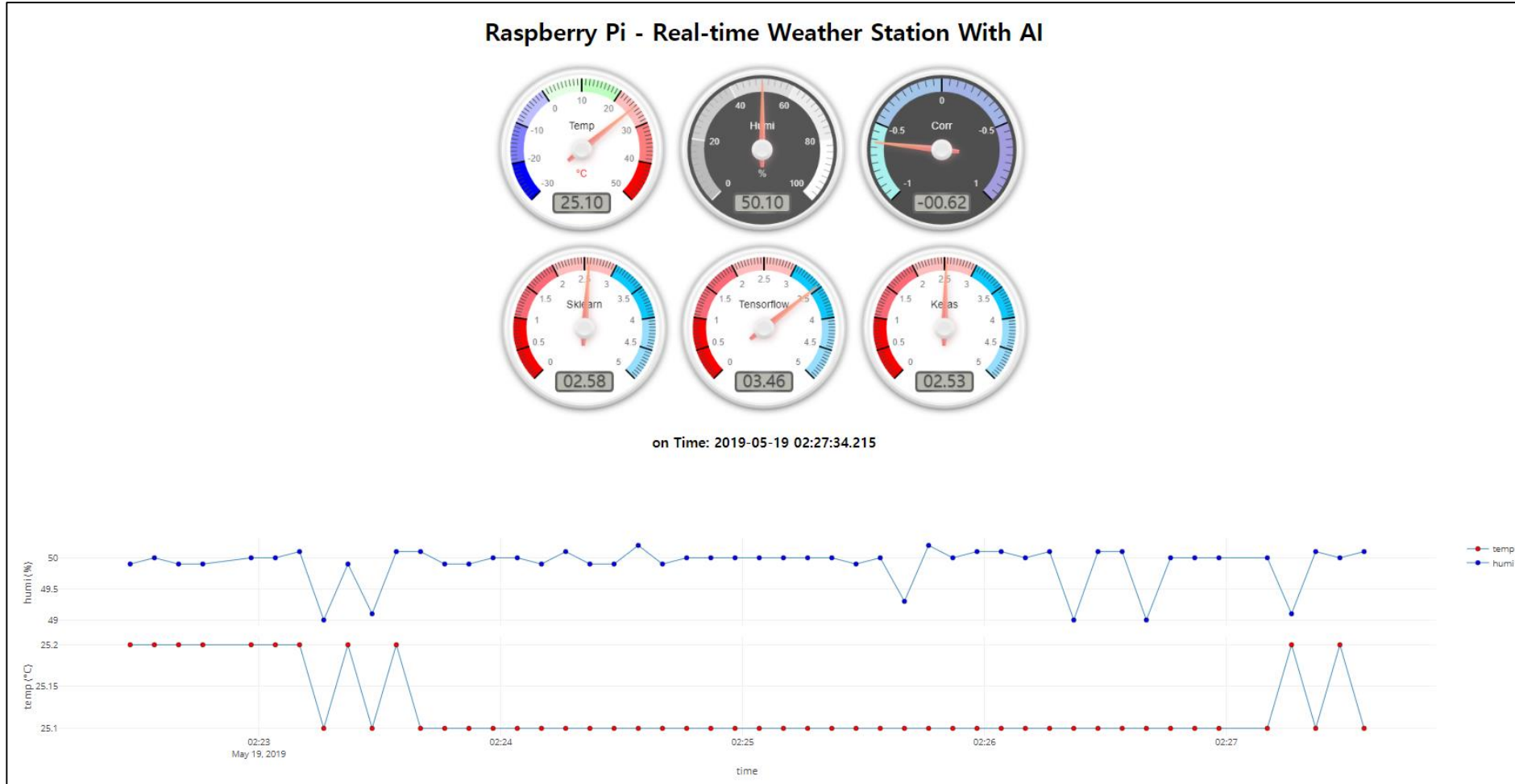
- 실시간으로 변하는 데이터를 처리하는 부분
- 게이지와 그래프의 value 값을 처리하는 부분
- 그래프의 초기 데이터를 설정하는 부분

Client Page(plotly_c_ml.html)

```
131 // data
132 var data = [{ ...
169
170 var layout = {
171   xaxis: {
172     title: 'time',
173     domain: [0, 1]
174   },
175   yaxis: {
176     title: 'temp (°C)',
177     domain: [0, 0.5],
178     range: [-30, 50]
179   },
180   xaxis2: {
181     title: '',
182     domain: [0, 1],
183     position: 0.35,
184     showticklabels: false
185   },
186   yaxis2: {
187     title: 'humi (%)',
188     domain: [0.55, 1],
189     range: [0, 100]
190   }
191 };
192
193 // gauge configuration
194 var gauge_temperature = new Gauge({ ...
264 gauge_temperature.draw();
265
266 var gauge_humidity = new Gauge({ ...
321 gauge_humidity.draw();
322
323 var gauge_correlation = new Gauge({ ...
373 gauge_correlation.draw();
374
375 var gauge_tensorflow = new Gauge({ ...
455 gauge_tensorflow.draw();
456
457 var gauge_sklearn = new Gauge({ ...
537 gauge_sklearn.draw();
538
539 var gauge_keras = new Gauge({ ...
619 gauge_keras.draw();
620 </script>
```

- 온습도를 구성하는 2개의 그래프
- 게이지는 순서대로 온도, 습도, 상관계수, sklearn, tensorflow, keras 로 구성되어있음.

Client Page(실행결과)



http://203.241.246.114:3030/plotly_c_ml.html

Client Page(client_DB_c.html)

```

17 <script>
18 <!-- JAVASCRIPT CODE GOES HERE -->
19
20 var cr_date = [];
21 var cr = [];
22 Plotly.d3.json("...", function (err, json) {
23
24     var jsonData = eval(JSON.stringify(json));
25
26     for (var i = 0; i < jsonData.length; i++) {
27         cr_date[i] = jsonData[i].date;
28         cr[i] = jsonData[i].cr;
29     }
30 });
31
32 Plotly.d3.json("...", function (err, json) {
33     //alert(json);
34     alert(JSON.stringify(json)); // It works!!!
35     //alert(JSON.parse(eval(json)));
36     if (err) throw err;
37
38     var data_range = 10
39
40     var date = [];
41     var temp = [];
42     var humi = [];
43     var date_avg = [];
44     var temp_avg = [];
45     var humi_avg = [];
46     var correlation = [];
47     var temp_c = [];
48     var humi_c = [];
49     var jsonData = eval(JSON.stringify(json));
50
51     for (var i = 0; i < jsonData.length; i++) {
52         date[i] = jsonData[i].date;
53         temp[i] = jsonData[i].temperature;
54         humi[i] = jsonData[i].humidity;
55     }
56
57     function getPearsonCorrelation(x, y) { --
104
105     for (var i = 0; i < jsonData.length; i+=data_range) { //
106         var temp_sum = 0;
107         var humi_sum = 0;
108
109         for (var j = i; j < i+data_range; j++) { // 0~9, 10~19, 20~29
110             temp_sum += parseFloat(temp[j]);
111             humi_sum += parseFloat(humi[j]);
112             temp_c.push(parseFloat(temp[j]))+(parseFloat(Math.random()/1000).toFixed(7)); // 0~10, 11~21, 22~32, 33~43
113             humi_c.push(parseFloat(humi[j]));
114         }
115         temp_avg.push(String(temp_sum/data_range));
116         humi_avg.push(String(humi_sum/data_range));
117
118         date_avg.push(date[i]); // 0, 10, 20, 30, 40, 50
119
120         correlation.push(getPearsonCorrelation(temp_c, humi_c));
121
122         temp_c.splice(0, data_range);
123         humi_c.splice(0, data_range);
124     }

```

- mongoDB에 저장되어있는 온습도를 볼 수 있는 페이지
- 각각 온습도와 쾌적도 값이 있는 mongoDB에 접근하여 데이터를 json형식으로 받아옴

Client Page(client_DB_c.html)

```
47 function getPearsonCorrelation(x, y) {
48
49     var shortestArrayLength = 0;
50
51     if (x.length == y.length) {
52         shortestArrayLength = x.length;
53     } else if (x.length > y.length) {
54         shortestArrayLength = y.length;
55         console.error('x has more items in it, the last ' + (x.length - shortestArrayLength) + ' item(s) will be ignored');
56     } else {
57         shortestArrayLength = x.length;
58         console.error('y has more items in it, the last ' + (y.length - shortestArrayLength) + ' item(s) will be ignored');
59     }
60
61     var xy = [];
62     var x2 = [];
63     var y2 = [];
64
65     for (var i = 0; i < shortestArrayLength; i++) {
66         xy.push(x[i] * y[i]);
67         x2.push(x[i] * x[i]);
68         y2.push(y[i] * y[i]);
69     }
70
71     var sum_x = 0;
72     var sum_y = 0;
73     var sum_xy = 0;
74     var sum_x2 = 0;
75     var sum_y2 = 0;
76
77     for (var j = 0; j < shortestArrayLength; j++) {
78         sum_x += x[j];
79         sum_y += y[j];
80         sum_xy += xy[j];
81         sum_x2 += x2[j];
82         sum_y2 += y2[j];
83     }
84
85     var step1 = (shortestArrayLength * sum_xy) - (sum_x * sum_y);
86     var step2 = (shortestArrayLength * sum_x2) - (sum_x * sum_x);
87     var step3 = (shortestArrayLength * sum_y2) - (sum_y * sum_y);
88     var step4 = Math.sqrt(step2 * step3);
89     var answer = ((step1 / step4).toFixed(2));
90
91     return answer;
92 }
```

- 상관계수도 같이 볼 수 있게 추가
- 상관계수는 온습도에대한 block 방식으로 구함

Client Page(client_DB_c.html)

```
126 var traces = [  
127   {  
128     type: "scatter",  
129     mode: "lines",  
130     name: 'Humidity',  
131     x: date_avg,  
132     y: humi_avg,  
133     yaxis : "y4",  
134     line: {  
135       color: '#3412fc'  
136     }  
137   },  
138   {  
139     type: "scatter",  
140     mode: "lines",  
141     name: 'Temperature',  
142     x: date_avg,  
143     y: temp_avg,  
144     yaxis : "y3",  
145     line: {  
146       color: '#fc1234'  
147     }  
148   },  
149   {  
150     type: "scatter",  
151     mode: "lines",  
152     name: 'Correlation',  
153     x: date_avg,  
154     y: correlation,  
155     yaxis : "y2",  
156     line: {  
157       color: '#34fc12'  
158     }  
159   },  
160   {  
161     type: "scatter",  
162     mode: "markers",  
163     name: 'Cr',  
164     x: cr_date,  
165     y: cr,  
166     yaxis : "y",  
167     line: {  
168       color: '#895400'  
169     }  
170   }  
];
```

- 온도, 습도, 상관계수, 쾌적도의 4개의 y축으로 구성

Client Page(client_DB_c.html)

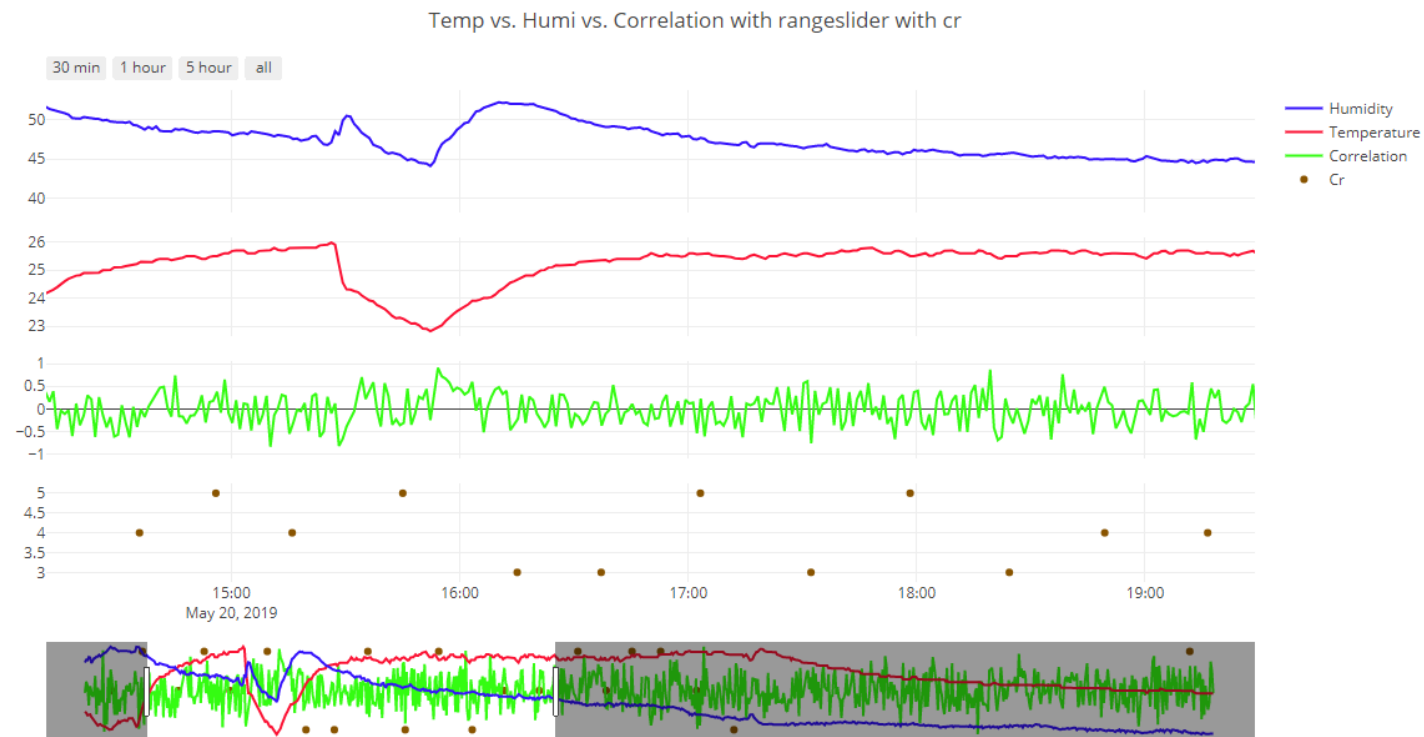
```
172 var layout = {  
173   title: 'Temp vs. Humi vs. Correlation with rangeslider with  
174   xaxis: {  
175     autorange: true,  
176     range: [date_avg[0], date_avg[date_avg.length - 1]],  
177     rangeselector: {  
178       buttons: [{  
179         count: 30,  
180         label: '30 min',  
181         step: 'minute',  
182         stepmode: 'backward'  
183       },  
184       {  
185         count: 1,  
186         label: '1 hour',  
187         step: 'hour',  
188         stepmode: 'backward'  
189       },  
190       {  
191         count: 5,  
192         label: '5 hour',  
193         step: 'hour',  
194         stepmode: 'backward'  
195       },  
196       {  
197         step: 'all'  
198       }  
199     ]  
200   },  
201   rangeslider: {  
202     range: [date_avg[0], date_avg[date_avg.length - 1]]  
203   },  
204   type: 'date'  
205 },  
206 yaxis: {  
207   autorange: true,  
208   range: [1, 5],  
209   type: 'linear',  
210   domain: [0, .2]  
211 },  
212 yaxis2: {  
213   autorange: true,  
214   range: [-1, 1],  
215   type: 'linear',  
216   domain: [.25, .45]  
217 },  
218 yaxis3: {  
219   autorange: true,  
220   range: [0, 100],  
221   type: 'linear',  
222   domain: [.50, .70]  
223 },  
224 yaxis4: {  
225   autorange: true,  
226   range: [0, 100],  
227   type: 'linear',  
228   domain: [.75, 1]  
229 }  
230 };  
231 Plotly.newPlot('myDiv', traces, layout);  
232 });  
233 </script>
```

- 구간별 시각화의 편의성을 위해 rangeslider와 button 추가

Client Page(실행결과)

MongoDB database visualization

Time series : Raspberry Pi Real-time Weather Station data



http://203.241.246.114:3030/client_DB_c_cr.html

Pandas(data-merge)

```

1 import pandas as pd
2 from pandas import DataFrame
3 import numpy as np
4
5
6 th = pd.read_json("http://203.241.246.114:3030/th")
7 cr = pd.read_json("http://203.241.246.114:3030/cr_value/cr")
8
9 iot_th = th[['date', 'temperature', 'humidity']]
10 iot_cr = cr[['date', 'cr']]
11
12 print(iot_th)
13
14 print(iot_cr)

```

- DB에 저장되어있는 데이터를 json형식으로 받아와서 dataframe을 구성
- 온습도와 쾌적도 데이터를 iot DB의 각각의 collections에서 받아온다.

```

1 '''
2 ths와 crs가 담긴 데이터 프레임들 date를 기준으로 outer방식으로 병합한다
3 '''
4 iot_data = pd.merge(iot_th, iot_cr, on='date', how="outer")
5 #iot_data.set_index('date',inplace=True) # date를 열로 설정한다
6
7 iot_data = iot_data.sort_values(['date'], ascending=[True]) # date를 기준으로 오름차순 정렬
8
9 # 온도 습도 값 계산을 위해 형변환
10 iot_data['temperature'] = pd.to_numeric(iot_data['temperature'])
11 iot_data['humidity'] = pd.to_numeric(iot_data['humidity'])
12 iot_data['cr'] = pd.to_numeric(iot_data['cr'])
13
14
15 idx = list(np.where(iot_data['cr'] > 0)[0]) # cr값이 0보다 큰값에 대한 인덱스 반환
16
17
18 length = len(idx)
19 for i in range(0,length):
20
21     idx_num = idx[i]
22
23     # 인덱스를 -1,1로 하면 연속으로 NaN이 있는 부분에서 에러발생. 3초마다 나오는 온도 습도 값의 차이가 거의 없으므로 -1,-2 인덱스로 계산
24     iot_data['temperature'].iloc[idx_num] = round((iot_data['temperature'].iloc[idx_num-1] + iot_data['temperature'].iloc[idx_num-2])/2,2)
25     iot_data['humidity'].iloc[idx_num] = round((iot_data['humidity'].iloc[idx_num-1] + iot_data['humidity'].iloc[idx_num-2])/2,2)
26
27 iot_data = iot_data.iloc[idx] # 전처리가 완료된 데이터 프레임으로 변경
28
29 del iot_data['date']
30
31 iot_data.to_csv("iot_data.csv", mode='w') # export csv
32
33 iot_data

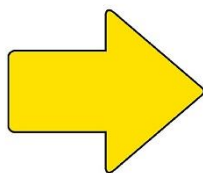
```

- 2개의 dataframe을 병합하여 ML에 사용할 dataframe을 제작한다.
- csv 파일 저장.

| Pandas(실행 결과)

	date	temperature	humidity
0	2019-05-20 13:22:12.509		
1	2019-05-20 13:22:18.511	23.30	51.80
2	2019-05-20 13:22:24.513	23.30	50.70
3	2019-05-20 13:22:30.514	23.60	50.70
4	2019-05-20 13:22:36.514	23.60	51.00
5	2019-05-20 13:22:42.515	23.60	51.40
6	2019-05-20 13:22:48.516	23.60	50.20
7	2019-05-20 13:22:54.518	23.60	51.30
8	2019-05-20 13:23:00.522	23.70	51.40
9	2019-05-20 13:23:06.523	23.60	50.00

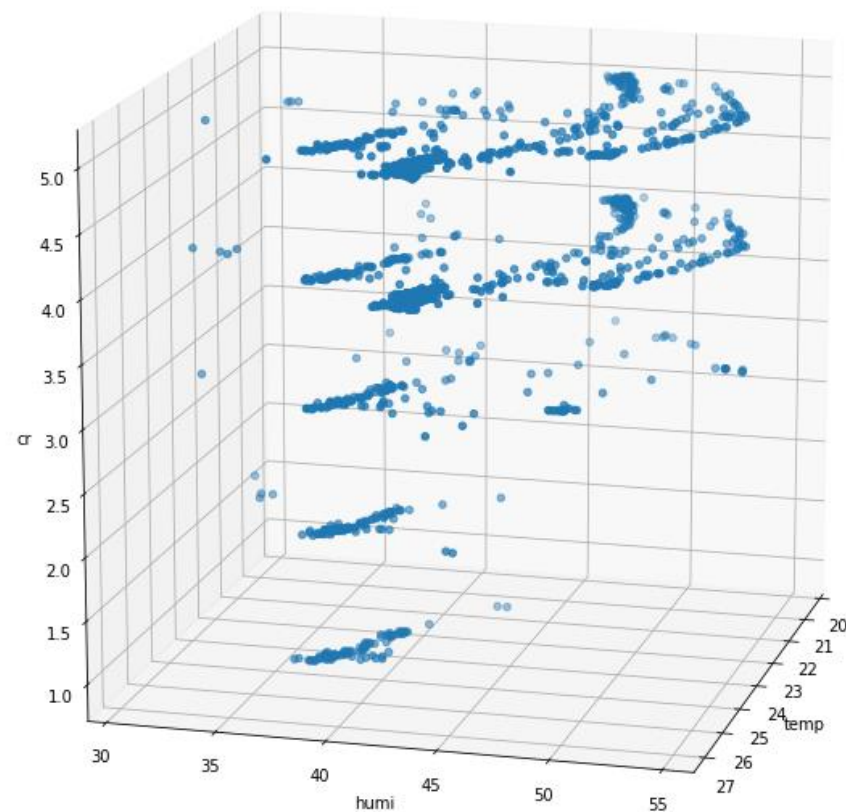
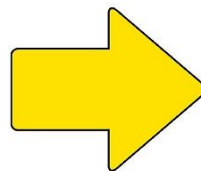
	date	cr
0	2019-05-20 13:43:45.118	4
1	2019-05-20 14:07:48.288	5
2	2019-05-20 14:35:51.499	4
3	2019-05-20 14:55:54.975	5
4	2019-05-20 15:15:57.875	4
5	2019-05-20 15:45:01.102	5
6	2019-05-20 16:15:04.446	3
7	2019-05-20 16:37:07.570	3
8	2019-05-20 17:03:10.937	5
9	2019-05-20 17:32:14.012	3



temperature	humidity	cr
21.40	52.20	4.0
22.55	50.60	5.0
22.50	50.50	4.0
21.90	53.45	5.0
21.90	53.50	5.0
21.90	53.60	5.0
21.90	53.10	5.0
21.90	53.25	5.0
21.90	53.25	5.0
21.90	53.52	5.0

3D plot(cr data)

```
11 import matplotlib.pyplot as plt
12 from mpl_toolkits.mplot3d import Axes3D
13 %matplotlib inline
14
15 data = read_csv('iot_data.csv', sep=',')
16
17 data = np.array(data, dtype=np.float32)
18
19 xs = np.array(data[:,1], dtype=np.float32)
20 ys = np.array(data[:,2], dtype=np.float32)
21 zs = np.array(data[:,3], dtype=np.float32)
22
23 fig = plt.figure(figsize=(12,12))
24 ax = fig.add_subplot(111, projection='3d')
25 ax.scatter(xs, ys, zs)
26 ax.set_xlabel('temp')
27 ax.set_ylabel('humi')
28 ax.set_zlabel('cr')
29 ax.view_init(15, 15)
30
31 plt.show()
```



ML(tensorflow)

```
1 import tensorflow as tf
2 import numpy as np
3 import matplotlib.pyplot as plt
4 from pandas.io.parsers import read_csv
5
6 model = tf.global_variables_initializer();
7
8
9 data = read_csv('iot_data.csv', sep=',')
10
11 # csv 파일안의 행 순서는 index_number, temp, humi, cr 입니다.
12
13 xy = np.array(data, dtype=np.float32)
14
15 x_data = xy[:, 1:-1] # temp, humi
16
17 y_data = xy[:, [-1]] # cr
```

- Read csv file
- X축과 y축 데이터 구성

ML(tensorflow)

```
24 # 행렬의 Shape을 None으로 지정하는 것은 N개의 데이터를 받겠다는 의미이다.(N x 2 행렬)
25
26 X = tf.placeholder(tf.float32, shape=[None, 2])
27
28 Y = tf.placeholder(tf.float32, shape=[None, 1])
29
30
31 # 각 변수를 array가 아닌 matrix로 나타냅니다.
32
33 W = tf.Variable(tf.random_normal([2, 1]), name="weight")
34
35 b = tf.Variable(tf.random_normal([1]), name="bias")
36
37
38 # matrix로 나타낸 변수를 행렬곱을 사용하여 가설을 세웁니다.
39
40 hypothesis = tf.matmul(X, W) + b
41
42
43 # 비용 함수를 설정합니다.
44
45 cost = tf.reduce_mean(tf.square(hypothesis - Y))
46
47
48 # 경사하강 라이브러리
49 # 최적화 함수를 설정합니다.
50
51 optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.0001)
52
53 train = optimizer.minimize(cost)
54
55
56
57 # 세션을 생성합니다.
58
59 sess = tf.Session()
60
61
62
63 # 글로벌 변수를 초기화합니다.
64
65 sess.run(tf.global_variables_initializer())
```

- MLR모델로 머신러닝을 합니다
- 2개의 input 변수를 1*2 matrix로 변경하고 가중치를 행렬 곱을 사용하여 설정함
- 오류를 측정하기위해 평균제곱오차를 사용했으며 최종적으로 비용이 가장 적을때를 구합니다.
- 최적화 알고리즘으로 GradientDescent를 사용하였습니다.

ML(tensorflow)

```

69 # 학습을 수행합니다.
70
71 for step in range(10001):
72     cost_, hypo_, _ = sess.run([cost, hypothesis, train], feed_dict={X: x_data, Y: y_data})
73
74     if step % 1000 == 0:
75         print("#", step, " 손실 비용: ", cost_)
76         print("- Weigth : ", sess.run(W[0]), sess.run(W[1]))
77         print("- Bias : ", sess.run(b))
78         print("- 꺾적도: ", hypo_[0])
79
80 saver = tf.train.Saver()
81
82 save_path = saver.save(sess, "saved.cpkt")
83
84 print('학습된 모델을 저장했습니다.')
```

- 10000번 학습하고 weight, bias 값을 출력함
- 학습된 모델을 저장합니다

```

# 0 손실 비용: 4066.1348
- Weigth : [0.45877138] [0.53624374]
- Bias : [0.34124586]
- 꺾적도: [74.224846]
# 1000 손실 비용: 1.1184845
- Weigth : [0.01319429] [0.0760383]
- Bias : [0.32479402]
- 꺾적도: [4.5760803]
```



```

# 9000 손실 비용: 1.1147795
- Weigth : [-0.00164946] [0.08453796]
- Bias : [0.3132561]
- 꺾적도: [4.6908393]
# 10000 손실 비용: 1.1147596
- Weigth : [-0.00161532] [0.08455066]
- Bias : [0.3118554]
- 꺾적도: [4.690832]
학습된 모델을 저장했습니다.
```

ML(tensorflow)

```
1 # 온,습도 변수의 값을 입력 받습니다.
2
3 temperature = float(input('온도: '))
4
5 humidity = float(input('습도: '))
6
7
8 with tf.Session() as sess:
9
10     sess.run(model)
11
12     # 저장된 학습 모델을 파일로부터 불러옵니다.
13
14     save_path = "saved.cpkt"
15
16     saver.restore(sess, save_path)
17
18
19
20
21     # 사용자의 입력 값을 이용해 배열을 만듭니다.
22
23     data = ((temperature, humidity), )
24
25     arr = np.array(data, dtype=np.float32)
26
27
28
29     # 예측을 수행한 뒤에 그 결과를 출력합니다.
30
31     x_data = arr[0:2]
32
33     dict = sess.run(hypothesis, feed_dict={X: x_data})
34
35     print(dict[0])
36
37     # Show the values of weights and bias
38     print('Weights:')
39     print(sess.run(w))
40     print('Bias:')
41     print(sess.run(b))
```

- 저장된 모델을 불러와서 임의의 온,습도를 넣어서 cr, weight, bias 값을 구합니다.

```
온도: 28
습도: 43
INFO:tensorflow:Restoring parameters from saved.cpkt
[3.902305]
Weights:
[[-0.00161532]
 [ 0.08455066]]
Bias:
[0.3118554]
```

ML(tensorflow) - 적용

```
137 function get_tensorflow(Celsius, Humidity){  
138     // CR = W[0]*T + W[1]*H + b  
139     tensorflow_cr = ((0.07446165) * Celsius) + (0.11316531 * Humidity) + (-2.8271704);  
140  
141     return tensorflow_cr + "";  
142 }
```

- ML으로 얻은 weight, bias 값을 가설 공식에 적용하여 cr값 구함
- $CR = W[0]*T + W[1]*H + b$

ML(scikit-learn)

```
1 import sklearn.linear_model
2 import numpy as np
3 from pandas.io.parsers import read_csv
4 # data split
5 from sklearn.model_selection import train_test_split
6
7 x_data = data[0:-1, 1:-1] # temp, humi
8
9 y_data = data[0:-1, [-1]] # cr
10
11 x_train, x_test, y_train, y_test = train_test_split(x_data, y_data,
12                                                    test_size=0.2, random_state=0)
13
14 print(x_train.shape)
15 print(y_train.shape)
16 print(x_test.shape)
17 print(y_test.shape)
```

```
(1192, 2)
(1192, 1)
(299, 2)
(299, 1)
```

- 온, 습도, 쾌적도 데이터를 x축 y축으로 구성하고 train_test_split 함수로 train data와 test data로 구분

ML(scikit-learn)

Logistic regression

```
[ ] 1 | from sklearn.linear_model import LogisticRegression
```

```
[ ] 1 | log_clf = LogisticRegression(solver='lbfgs', multi_class='auto', max_iter=1000)  
2 | log_clf.fit(x_train, y_train.ravel())
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,  
                    intercept_scaling=1, max_iter=1000, multi_class='auto',  
                    n_jobs=None, penalty='l2', random_state=None, solver='lbfgs',  
                    tol=0.0001, verbose=0, warm_start=False)
```

```
[ ] 1 | log_clf.score(x_test, y_test)
```

```
0.3511705685618729
```

- Sklearn의 logisticRegression모델 사용
- 최적화 알고리즘은 lbfgs 사용

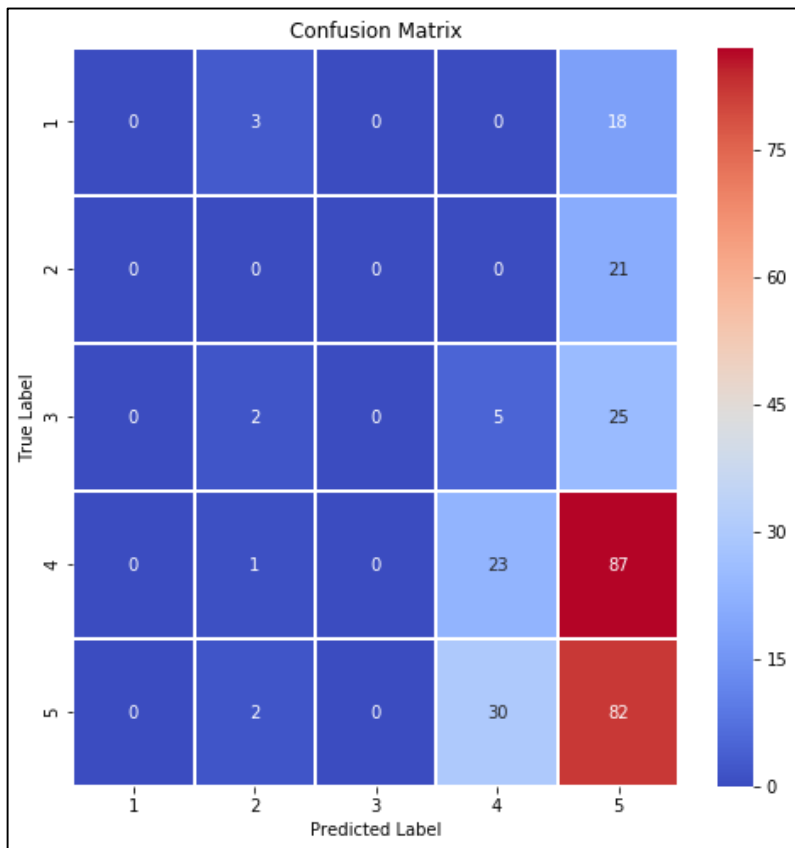
ML(scikit-learn)

```
[ ] 1 # 로지스틱 회귀 모델의 정확도(R^2)
    2
    3 print("트레인 세트의 정확도: {:.2f}".format(log_clf.score(x_train, y_train)))
    4 print("테스트 세트의 정확도: {:.2f}".format(log_clf.score(x_test, y_test)))
    5 print("weights: ", log_clf.coef_)
    6 print("bias : ", log_clf.intercept_)
```

```
↳ 트레인 세트의 정확도: 0.41
   테스트 세트의 정확도: 0.35
   weights: [[-0.09939661 -0.25826953]
             [-0.01450908 -0.30808969]
             [-0.13974128  0.01788321]
             [ 0.11125783  0.27725472]
             [ 0.14238915  0.27122129]]
   bias : [ 12.35153308 12.12950693  2.7670086 -13.42706412 -13.8209845 ]
```

- 정확도와 weight, bias 값을 구함

ML(scikit-learn) - 적용



```

144 function get_sklearn(Celsius, Humidity){
145     // CR = W[0]*T + W[1]*H + b
146     var sklearn_list = []
147
148     sklearn_list[0] = ((-0.09939661) * Celsius) + ((-0.25826953) * Humidity) + (12.35153308);
149     sklearn_list[1] = ((-0.01450908) * Celsius) + ((-0.30808969) * Humidity) + (12.12950693);
150     sklearn_list[2] = ((-0.13974128) * Celsius) + ((0.01788321) * Humidity) + (2.7670086);
151     sklearn_list[3] = ((0.11125783) * Celsius) + ((0.27725472) * Humidity) + (-13.42706412);
152     sklearn_list[4] = ((0.14238915) * Celsius) + ((0.27122129) * Humidity) + (-13.8209845);
153
154     var max = Math.max.apply(null, sklearn_list); // The largest value of the elements in the array.
155     var sklearn_cr = (sklearn_list.indexOf(max))+1; // index of array
156
157     return sklearn_cr + "";
158 }

```

- ML으로 얻은 weight, bias 값을 가설공식에 적용하여 cr값 구함
- $CR = W[0]*T + W[1]*H + b$

ML(keras)

```
1 import numpy as np
2 from pandas.io.parsers import read_csv
3 import keras
4 from keras.models import Sequential
5 from keras.layers.core import Dense
6 |
7 data = read_csv('iot_data.csv', sep=',')
8
9 data = np.array(data, dtype=np.float32)
```

- csv파일을 read하여 데이터를 받아옴

```
1 # data split
2 from sklearn.model_selection import train_test_split
3
4 x_data = data[0:-1, 1:-1] # temp, humi
5
6 y_data = data[0:-1, [-1]] # cr
7
8 x_train, x_test, y_train, y_test = train_test_split(x_data, y_data,
9                                                    test_size=0.2, random_state=0)
10
11 print(x_train.shape)
12 print(y_train.shape)
13 print(x_test.shape)
14 print(y_test.shape)
```

(1192, 2)
(1192, 1)
(299, 2)
(299, 1)

- 온,습도,쾌적도 데이터를 x축 y축으로 구성하고 train_test_split함수로 train data와 test data로 구분

| ML(keras)

```
[71] 1 model = Sequential()  
      2 model.add(Dense(1, input_shape=(2,), activation='relu'))  
      3 model.compile(loss='mean_squared_error', optimizer='adam', metrics=['accuracy'])  
      4 model.summary()
```



Layer (type)	Output Shape	Param #
dense_20 (Dense)	(None, 1)	3

Total params: 3
Trainable params: 3
Non-trainable params: 0

- Activation으로 sigmoid를 개선한 ReLU 사용
- Optimizer로 adam 사용

ML(keras)

```
1 hist = model.fit(x_train, y_train, epochs = 100, batch_size=10, verbose=1)
```

```
Epoch 1/100  
1192/1192 [=====] - 1s 434us/step - loss: 2411.8372 - acc: 0.0000e+00  
Epoch 2/100  
1192/1192 [=====] - 0s 98us/step - loss: 1706.9047 - acc: 0.0000e+00  
Epoch 3/100  
1192/1192 [=====] - 0s 96us/step - loss: 1172.5814 - acc: 0.0000e+00  
Epoch 4/100  
1192/1192 [=====] - 0s 96us/step - loss: 777.9529 - acc: 0.0000e+00  
Epoch 5/100  
1192/1192 [=====] - 0s 98us/step - loss: 497.1686 - acc: 0.0000e+00
```

⋮

```
Epoch 95/100  
1192/1192 [=====] - 0s 89us/step - loss: 1.1092 - acc: 0.3750  
Epoch 96/100  
1192/1192 [=====] - 0s 86us/step - loss: 1.1043 - acc: 0.3700  
Epoch 97/100  
1192/1192 [=====] - 0s 91us/step - loss: 1.1037 - acc: 0.3784  
Epoch 98/100  
1192/1192 [=====] - 0s 89us/step - loss: 1.1147 - acc: 0.3658  
Epoch 99/100  
1192/1192 [=====] - 0s 92us/step - loss: 1.1047 - acc: 0.3691  
Epoch 100/100  
1192/1192 [=====] - 0s 87us/step - loss: 1.1034 - acc: 0.3700
```

- 100번 학습중 loss, acc 값을 보여줌

ML(keras)

```
[74] 1 # 모델에 임의의 값을 넣어 test  
      2 #model.predict(x_train)  
      3 model.predict(np.array([23,43]).reshape(1,2))
```

```
↳ array([[3.9350345]], dtype=float32)
```

```
[75] 1 # 모델평가  
      2 test_loss, test_acc = model.evaluate(x_test, y_test)  
      3 print('정확도 : ', test_acc)
```

```
↳ 299/299 [=====] - 0s 588us/step  
   정확도 : 0.38795986661942905
```

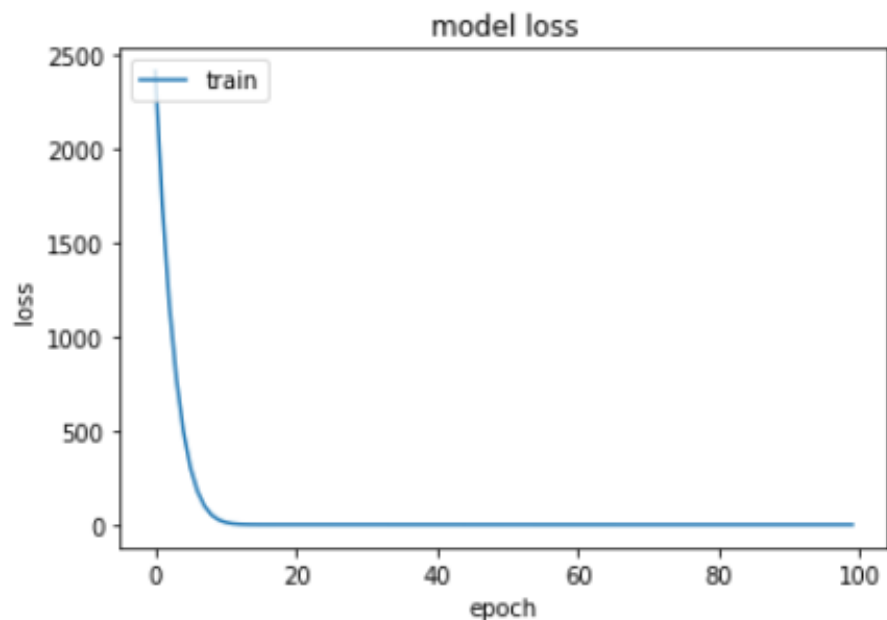
```
[76] 1 # get weight, bias value  
      2 W_, b_ = model.get_weights()  
      3 print(W_)  
      4 print('weight : ', W_[0],W_[1])  
      5 print('bias : ', b_)
```

```
↳ [[0.0301453 ]  
    [0.09299916]]  
   weight : [0.0301453] [0.09299916]  
   bias : [-0.757271]
```

- 정확도와 weight, bias 값을 구함

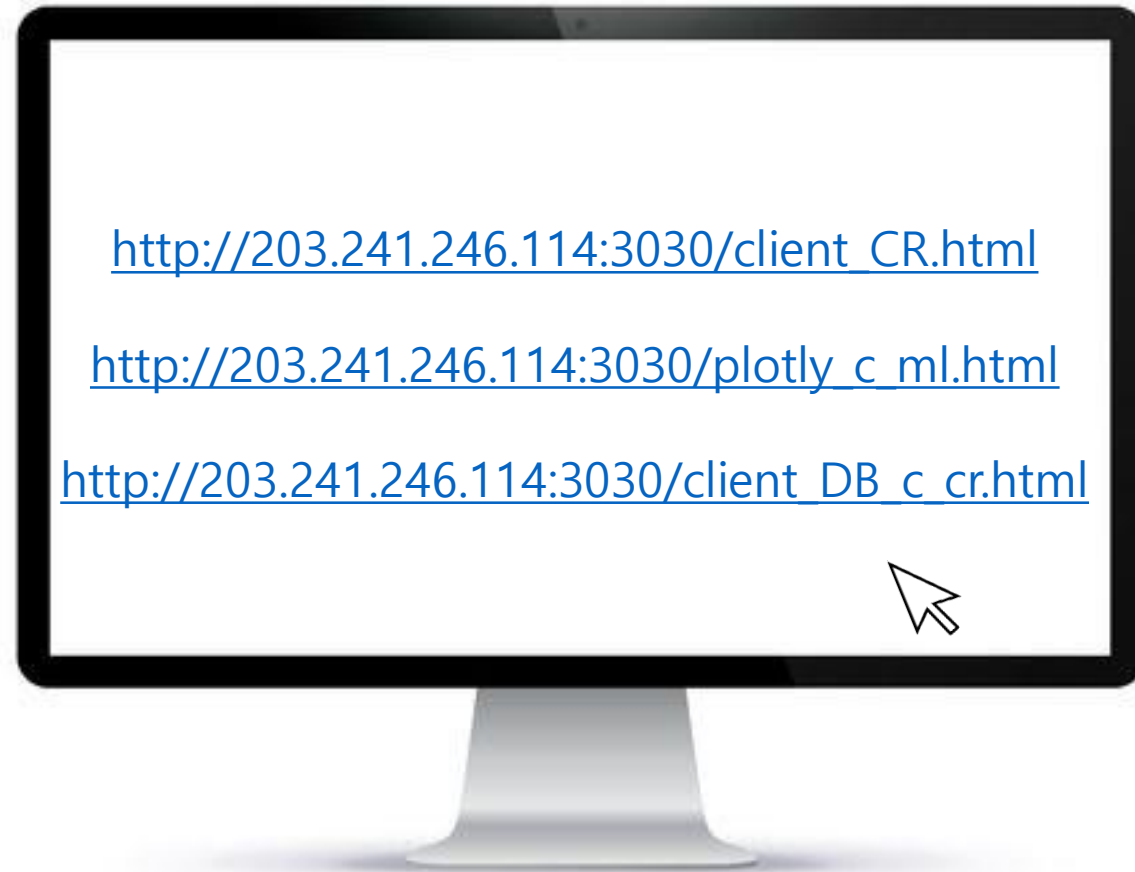
ML(keras) - 적용

```
1 # summarize history for loss
2 plt.plot(hist.history['loss'])
3 plt.title('model loss')
4 plt.xlabel('epoch')
5 plt.ylabel('loss')
6 plt.legend(['train', 'test'], loc='upper left')
7 plt.show()
```



```
160 function get_keras(Celsius, Humidity){
161     // CR = W[0]*T + W[1]*H + b
162
163     keras_cr = ((0.0301453) * Celsius) + (0.09299916 * Humidity) + (-0.757271);
164
165     return keras_cr + "";
166 }
```

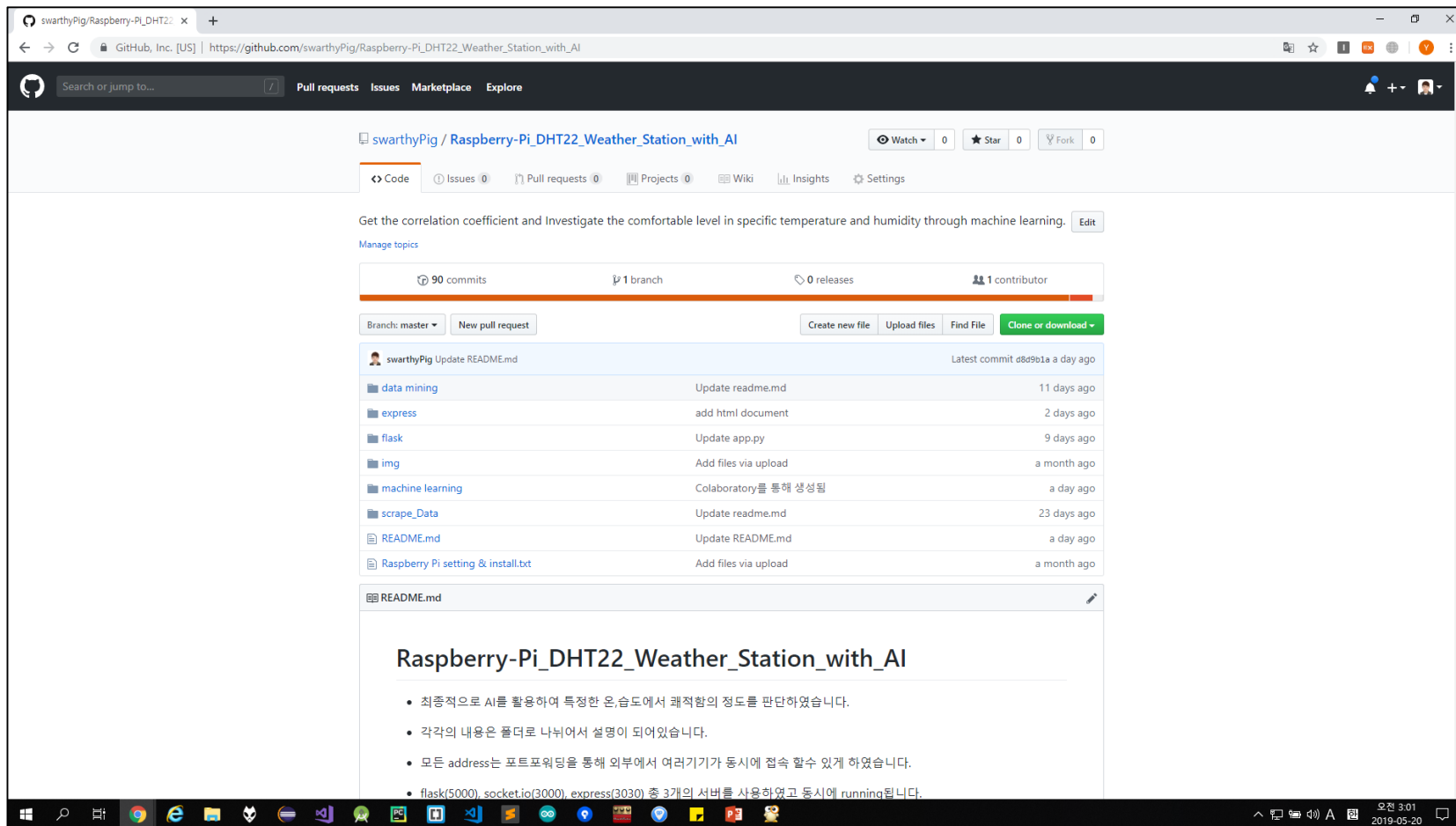
I 시연 & 테스트



I 향후 계획

| GitHub 오픈소스

github.com/swarthyPig/Raspberry-Pi_DHT22_Weather_Station_with_AI



swarthyPig / Raspberry-Pi_DHT22_Weather_Station_with_AI

Watch 0 Star 0 Fork 0

Code Issues 0 Pull requests 0 Projects 0 Wiki Insights Settings

Get the correlation coefficient and Investigate the comfortable level in specific temperature and humidity through machine learning. Edit

Manage topics

90 commits 1 branch 0 releases 1 contributor

Branch: master New pull request Create new file Upload files Find File Clone or download

swarthyPig	Update README.md	Latest commit d8d9b1a a day ago
data mining	Update readme.md	11 days ago
express	add html document	2 days ago
flask	Update app.py	9 days ago
img	Add files via upload	a month ago
machine learning	Colaboratory를 통해 생성됨	a day ago
scrape_Data	Update readme.md	23 days ago
README.md	Update README.md	a day ago
Raspberry Pi setting & install.txt	Add files via upload	a month ago

README.md

Raspberry-Pi_DHT22_Weather_Station_with_AI

- 최종적으로 AI를 활용하여 특정한 온도, 습도에서 쾌적함의 정도를 판단하였습니다.
- 각각의 내용은 폴더로 나뉘어서 설명이 되어있습니다.
- 모든 address는 포트포워딩을 통해 외부에서 여러기기가 동시에 접속 할수 있게 하였습니다.
- flask(5000), socket.io(3000), express(3030) 총 3개의 서버를 사용하였고 동시에 running됩니다.



Q & A



THANK YOU

(IoT-AI weather station)