Weather Forecast Predictions

Sun Hyoung Kim

Introduction:

This report focuses on analyzing weather forecast predictions' accuracy and creating graphical data visualization. I will compare the past predictions of weather forecasts to the actual weathers, and see how accurate the weather forecast predictions were.

In particular aspects, I'm interested in comparing predictions and actual data of temperature, humidity, and probability of precipitation.

Specifically with the time, I will do data analysis of

March (1~31) and **April (1~25),**
with times **8:00 AM** (morning) / **12:00 PM** (lunch time) / **16:00 PM** (afternoon) respectively.

So I will do six different analysis and have six different plots
(March – humidity , March – temperature , March– probability of precipitation
April – humidity , April – temperature , April – probability of precipitation)

Because I did six different analysis, I had six different .py files. However, their codes were pretty much the same with each other, and so it will be not great to have all of the codes in this report (taking too much space!). So this report will mainly show **March – temperature** comparison as an example.

Below is a table of content, showing the steps of what I will be doing in this report:

Table of Contents

Step 1: Downloading Data..... 2
B) Materials for all sections:..... 2
B) Methods for Step 1:..... 2
Step 2: Parsing the websites (.html) and making lists for the content 3
B) Methods for Step 2:..... 3
C) Results (output) on Step 2: 5
Step 3: Getting lists of temperature, humidity, and probability of precipitation:..... 5
B) Methods for Step 3:..... 5
C) Results (output) on Step 3 (March, probability of precipitation): 6
C) Results (output) on Step 3 (March, humidity):..... 6
C) Results (output) on Step 3 (March, temperature):..... 6
Step 4: Getting data from JSON files:..... 6
B) Methods for Step 4:..... 7
C) Results (output) on Step 4 (getting JSON files):..... 7
C) Results (output) on Step 4 (getting temperature elements from JSON files): 9
C) Results (output) on Step 4 (modifying lists):..... 10
C) Results (output) on Step 4 (modifying lists again):..... 11
Step 5: Error between real data and predictions: 12
B) Methods for Step 5:..... 12
C) Results (output) and Discussions on Step 5:..... 12

Step 6: Plots of real data and predictions:.....13
B) Methods for Step 6:.....13
C) Results (output) and Discussions on Step 6:.....14
Step 7. Conclusion:.....20

Step 1: Downloading Data

I download all the data I need from Dr. Ringland's website first.

B) Materials for all sections:

Buffalo Google Forcast Weatherpages collected by Dr. Ringland - http://copper.math.buffalo.edu/463/weatherpages/

Specifically, ones with **json format** (predictions)
Example: google_forecast_2015_05_01_06_01_19.json

AND

ones with **KBUF.html.html** (actual data)
Example: 2015-04-30_05-02-05_w1.weather.gov_data_obhistory_KBUF.html.html

B) Methods for Step 1:

Since there are too many websites that I have to download on Dr. Ringland's weatherpage website, I used code below to download all datasets I need.

Code:

```
import re
import urllib2
import os

response = urllib2.urlopen('http://blue.math.buffalo.edu/463/weatherpages/')
html = response.read()

myre = '<a href="(.+.json)">'
results = re.findall(myre,html)

for result in results:
       name = 'wget http://blue.math.buffalo.edu/463/weatherpages/'+result
       os.system(name)

myre = '<a href="(.+.KBUF.html.html)">'
results = re.findall(myre,html)

for result in results:
       name = 'wget http://blue.math.buffalo.edu/463/weatherpages/'+result
       os.system(name)
```

The code above allowed me to download all the JSON data files and KBUF.html.html files, so I saved my time.
For KBUF.html.html files, each website shows the past three days of the website page's date.

For example, if I look at 2015-04-18_17-02-05..._KBUF.html.html, I can see information of dates April 18, 17, 16 and 15. So if I see 2015-03-01_05-02-04..._KBUF.html.html, it gives me information of February 28, 27, and 26 that I do not want to have (I only want March (1-31) and April (1-25) datasets, otherwise those February ones will mess up my datasets badly in the future). So I **manually deleted the first three dates of March and April** KBUF.html.html ones that contain previous month's information (so I have 2015-03-04 ~ 2015-03-31...KBUF.html.html and 2015-04-04 ~ 2015-04-25...KBUF.html.html, that will give me actual data of March (1-31) and April (1-25)).

Step 2: Parsing the websites (.html) and making lists for the content

Unfortunately, the KBUF.html.html datasets are not in JSON format which is very easy to read, so I had to parse all KBUF.html.html datasets.

B) Methods for Step 2:

I go to “View Page Source” on the KBUF.html.html website to see what I have to extract and delete, and get rid of all unnecessary ones like “</td>” thing from the source. When getting rid of unnecessary ones, I use **split()** method many times by using for loops.
Next, I make lists and put information I need into the lists. Since I'm only doing analysis of the times 8:00 AM, 12:00 PM, and 4:00 PM, I look for the contents with those times only. (the KBUF.html.html files show times in 3:54 AM, 4:54 AM... instead of on the hours. So I looked for 07:54, 11:54, and 15:54. (8:00 AM, 12:00 PM, and 4:00 PM)

Below code is an example of **March, temperature** comprison.

Code:

```
import glob
import urllib2
import simplejson as json
from pylab import*
from numpy import*

real_set = []
for link in glob.glob('2015-03*_KBUF.html.html'): #march only

      url = 'http://blue.math.buffalo.edu/463/weatherpages/'+link
      s = urllib2.urlopen(url).read()
      x = '<tr align="center" bgcolor="#b0c4de"><th width="26">Max.</th><th
width="26">Min.</th></tr><tr align="center" valign="top" bgcolor="#eeeeee">'
      y = '<td></td><td></td><td></td></tr><tr align="center" bgcolor="#b0c4de"><th
rowspan="3">'
      html = s.split(x)[1].split(y)[0]
      total_list = []
      row_list = []
      for row in html.split('</td>'):
            for element in row.split('<td>'):
                  for other in element.split('<td align="right">'):
                        for another in other.split('<td align="left">'):
                              if another!='' and '\r\n' not in another:
                                    if another=='</tr><tr align="center" valign="top"
bgcolor="#f5f5f5">' or another=='</tr><tr align="center" valign="top" bgcolor="#eeeeee">':
                                          total_list.append(row_list)
                                          row_list=[]
```

```
                                        else:
                                              row_list.append(another)

      for ind_list in total_list:
            if ind_list[1]== '07:54' or ind_list[1]== '11:54' or ind_list[1]== '15:54':
                  real_set.append(ind_list)
```

each individual list looks like this:
['05', '07:54', 'NW 9', '10.00', 'Overcast', 'FEW030 OVC180', '10', '-1', '19', '10', '61%', '-3', 'NA', '30.25', '1025.9']

I make a big list to contain each individual March list. After putting my each individual March list into a big list, I get rid of all duplicated lists (since each website shows three past dates' information, there are a lot of duplicated lists)

(continued) Code:

```
import itertools

real_set.sort()
modified_set = sorted(list(real_set for real_set,_ in itertools.groupby(real_set)))
```

This allowed me to reduce a lot of duplicated lists in the big list. However, there were some lists with same dates and times, but different content in other elements, so they were not removed from the code above, which only removes exactly the same duplicated lists. Also, there was a list with time 11:52 instead of 11:54, which disobeys the websites' format. So I had to print them on my terminal first, and put the lists into my .py file manually and deleted/added from my big list.

(continued) Code:

```
#11:52 - bad time!
missing_list = ['03', '11:52','SE 6', '2.00', 'Light Snow Fog/Mist', 'FEW017 BKN025 OVC032',
'20', '16', '85%', '12', 'NA', '30.09', '1020.3', '0.02']

duplicate_list = [ ['05', '07:54', 'NW 9', '10.00', 'Overcast', 'FEW030 OVC180', '10', '-1',
'19', '10', '61%', '-3', 'NA', '30.25', '1025.9'], ['05', '15:54', 'W 9', '10.00', 'Mostly
Cloudy', 'BKN033', '12', '-0', '58%', '-0', 'NA', '30.32', '1028.2'], ['05', '11:54', 'Vrbl
6', '9.00', ' Light Snow', 'SCT021 OVC028', '11', '-0', '61%', '1', 'NA', '30.29',
'1027.1'], ['06', '07:54', 'S 8', '10.00', 'Fair', 'CLR', '-0', '-5', '5', '-1', '79%', '-
14', 'NA', '30.49', '1034.5'], ['06', '11:54', 'SW 17', '10.00', 'A Few Clouds', 'FEW250',
'14', '-1', '51%', '-2', 'NA', '30.47', '1033.5'], ['06', '15:54', 'SW 20 G 28', '10.00',
'Mostly Cloudy', 'FEW033 FEW220 BKN260', '19', '6', '57%', '3', 'NA', '30.36', '1029.8'],
['07', '07:54', 'SW 15 G 22', '10.00', 'Overcast', 'OVC030', '22', '12', '23', '19', '66%',
'9', 'NA', '30.10', '1020.6'], ['07', '11:54', 'SW 16 G 28', '7.00', ' Light Snow', 'BKN014
BKN027 OVC050', '25', '18', '75%', '12', 'NA', '30.04', '1018.3'], ['07', '15:54', 'SW 17',
'10.00', 'Overcast', 'BKN032 OVC045', '30', '19', '64%', '18', 'NA', '29.96', '1015.6']
]

modified_set.append(missing_list)

for i in modified_set:
      if i in duplicate_list:
            modified_set.remove(i)
```

C) Results (output) on Step 2:

(* Please note that the code above doesn't include “print modified_set” though, as the main things I would like to see are not those. In this report, I am showing what each step is doing.)

Now I get a valid list:

```
[['01', '07:54', 'Calm', '2.50', ' Light Snow Fog/Mist', 'FEW031 OVC045', '17', '14', '88%',
'NA', 'NA', '30.42', '1031.6'], ['01', '11:54', 'W 3', '3.00', ' Light Snow', 'BKN017 BKN027
OVC070', '25', '19', '78%', 'NA', 'NA', '30.31', '1028.0'], ['01', '15:54', 'SW 6', '0.75',
' Light Snow Fog/Mist', 'FEW009 FEW017 OVC020', '25', '22', '88%', '18', 'NA', '30.22',
'1024.6', '0.02', '0.02'], ['02', '07:54', 'W 16 G 28', '5.00', ' Light Snow', 'FEW014
OVC030', '20', '15', '81%', '6', 'NA', '30.22', '1024.5'], ['02', '11:54', 'W 22 G 30',
'7.00', ' Light Snow and Breezy', 'BKN023', '25', '17', '72%', '10', 'NA', '30.30',
'1027.3'], ['02', '15:54', 'W 23 G 32', '10.00', 'Mostly Cloudy and Breezy', 'FEW034
BKN042', '26', '11', '53%', '11', 'NA', '30.35', '1029.0'],

…
…

['30', '11:54', 'SW 20 G 28', '10.00', ' Light Snow Rain', 'FEW013 BKN034 OVC050', '37',
'31', '79%', '27', 'NA', '29.79', '1009.4'], ['30', '15:54', 'W 15', '0.25', ' Heavy Snow
Fog', 'OVC004', '33', '31', '92%', '23', 'NA', '29.81', '1010.3', '0.08'], ['31', '07:54',
'S 5', '10.00', 'Overcast', 'FEW065 BKN090 OVC200', '30', '24', '32', '26', '79%', '25',
'NA', '29.84', '1011.2'], ['31', '11:54', 'Calm', '10.00', 'Overcast', 'FEW065 OVC085',
'35', '24', '64%', 'NA', 'NA', '29.79', '1009.5'], ['31', '15:54', 'N 3', '10.00', 'Mostly
Cloudy', 'FEW036 BKN045 BKN250', '39', '23', '53%', 'NA', 'NA', '29.80', '1009.8']]
```

I successfully parsed and got the March data!

Step 3: Getting lists of temperature, humidity, and probability of precipitation:

Based on the parsed datasets (lists) I made in Step 2, I am modifying the lists again into ones I would like to analyze (numerical values of temperature, humidity, and probability of precipitation).

Since I'm taking **March_temperature** comparison as an example in this report, I will mainly focus on writing the report based on my codes in a March_temperature comparison .py file, but I will also show how to get data of humidity and probability of precipitation in this step.

B) Methods for Step 3:

If it either rained or snowed, the list must contain “Rain” or “Snow” in one of their elements.

If it either rained or snowed, I give a number '98' (making things easier in the future when making a plot), and if it did not, I give a number '2'.

Code (precipitation or probability):

```
conditionlist = []
for eachlist in sorted(modified_set):
	if 'Rain' in eachlist[4] or 'Snow' in eachlist[4]:
		conditionlist.append('98')
	else:
		conditionlist.append('2')
```

C) Results (output) on Step 3 (March, probability of precipitation):

```
['98', '98', '98', '98', '98', '2', '2', '98', '2', '2', '2', '2', '98', '98', '2', '2',
'2', '98', '2', '2', '2', '2', '2', '2', '2', '2', '2', '2', '2', '2', '2', '2', '2', '2',
'2', '2', '2', '2', '2', '2', '98', '2', '2', '2', '2', '2', '2', '2', '98', '2', '2', '2',
'2', '2', '2', '2', '2', '98', '2', '98', '2', '2', '98', '2', '2', '2', '2', '2', '2', '2',
'2', '2', '98', '98', '2', '98', '98', '2', '98', '98', '98', '2', '2', '2', '2', '2', '2',
'98', '98', '2', '2', '2']
```

Code (humidity):

In elements in each list, only humidities have “%” in the end, so I apply this fact when adding humidities into a humidity_list.

```
humidity_list = []
for eachlist in sorted(modified_set):
      for element in eachlist:
            if element[-1]=='%':
                  humidity_list.append(element[:-1])
```

C) Results (output) on Step 3 (March, humidity):

```
['88', '78', '88', '81', '72', '53', '74', '82', '67', '69', '76', '64', '67', '64', '79',
'49', '57', '71', '69', '69', '85', '59', '64', '85', '73', '65', '72', '58', '58', '97',
'89', '92', '75', '35', '27', '51', '27', '37', '89', '83', '96', '75', '70', '64', '78',
'68', '83', '86', '70', '34', '60', '40', '45', '81', '34', '31', '42', '76', '60', '86',
'89', '68', '85', '55', '57', '56', '43', '38', '70', '43', '37', '52', '63', '89', '93',
'96', '92', '82', '85', '72', '70', '57', '46', '74', '52', '48', '79', '79', '92', '79',
'64', '53']
```

Code (temperature):

```
temperaturelist = []
for eachlist in sorted(modified_set):
      temperaturelist.append(eachlist[6]) #temperature is the 7th element in each list
```

C) Results (output) on Step 3 (March, temperature):

```
['17', '25', '25', '20', '25', '26', '13', '29', '31', '32', '31', '9', '10', '13', '1',
'16', '20', '22', '25', '30', '28', '32', '33', '27', '34', '38', '31', '43', '46', '34',
'36', '33', '29', '34', '37', '26', '44', '48', '41', '44', '37', '33', '34', '36', '33',
'44', '38', '40', '33', '35', '23', '31', '33', '18', '30', '36', '35', '39', '44', '38',
'36', '40', '20', '22', '24', '17', '20', '24', '17', '29', '38', '33', '43', '44', '37',
'33', '34', '31', '29', '28', '17', '20', '26', '20', '32', '36', '36', '37', '33', '30',
'35', '39']
```

(*Please note that these lists above are ordered respectively):
(March 1st 8:00 AM, March 1st 12:00 PM, March 1st 4:00PM, March 2nd 8:00 AM … March 31st 12:00 PM, March 31st 4:00 PM)

Step 4: Getting data from JSON files:

There are too many JSON files on the website since they were updated in every single hour. Therefore I am only taking each day's JSON files. Just like the KBUF.html.html files, there were some missing data and duplicated JSON files when I was dragging the data by using a

for loop. So I manually added a “dummies” list in order to get rid of the duplicated data, and “missinglists” in order to add them to the data lists I want to have.

Please note that when I am comparing the predictions to real data, I get six days before for each date. So for example, if I am doing analysis of March 23rd, the prediction dates are March 17th ~ March 22nd. So I needed a list containing (February 22nd ~ February 28th {February 27th missing}), (February 23rd ~ March 1st), (February 24th ~ March 2nd), … (March 24th ~ March 29th), (March 25th ~ March 30th) in order to compare with actual datasets of March 1st ~ March 31st.

B) Methods for Step 4:

I firstly make a list having all JSON files I need for March.

Code:

```
dummies=['google_forecast_2015_02_20_22_01_43.json',
'google_forecast_2015_02_21_22_01_19.json',
'google_forecast_2015_03_22_01_01_19.json','google_forecast_2015_03_20_03_01_20.json',
'google_forecast_2015_03_21_03_01_20.json',
'google_forecast_2015_03_22_03_01_19.json', 'google_forecast_2015_03_23_03_01_20.json',
'google_forecast_2015_03_24_03_01_19.json', 'google_forecast_2015_03_25_03_01_22.json',
'google_forecast_2015_04_03_01_01_19.json']

totallist = []
missinglists =
['google_forecast_2015_03_12_21_01_20.json','google_forecast_2015_03_17_21_01_20.json']



for file1 in sorted( glob.glob('google_forecast_2015_02_2*_22_01_*.json')
+sorted(glob.glob('google_forecast_2015_03*22_01*.json'))+missinglists):
      if file1 not in dummies:
            totallist.append(file1)

datelist=[]
sequencelist=[]

for i in range(len(totallist)-6):
      datelist = totallist[i:i+6]
      sequencelist.append(datelist)
```

C) Results (output) on Step 4 (getting JSON files):

```
[['google_forecast_2015_02_22_22_01_19.json', 'google_forecast_2015_02_23_22_01_19.json',
'google_forecast_2015_02_24_22_01_19.json', 'google_forecast_2015_02_25_22_01_43.json',
'google_forecast_2015_02_26_22_01_19.json', 'google_forecast_2015_02_28_22_01_08.json'],
['google_forecast_2015_02_23_22_01_19.json', 'google_forecast_2015_02_24_22_01_19.json',
'google_forecast_2015_02_25_22_01_43.json', 'google_forecast_2015_02_26_22_01_19.json',
'google_forecast_2015_02_28_22_01_08.json', 'google_forecast_2015_03_01_22_01_07.json'],
…
…

 ['google_forecast_2015_03_24_22_01_43.json', 'google_forecast_2015_03_25_22_01_20.json',
'google_forecast_2015_03_26_22_01_19.json', 'google_forecast_2015_03_27_22_01_20.json',
'google_forecast_2015_03_28_22_01_20.json', 'google_forecast_2015_03_29_22_01_20.json'],
['google_forecast_2015_03_25_22_01_20.json', 'google_forecast_2015_03_26_22_01_19.json',
```

```
'google_forecast_2015_03_27_22_01_20.json', 'google_forecast_2015_03_28_22_01_20.json',
'google_forecast_2015_03_29_22_01_20.json', 'google_forecast_2015_03_30_22_01_19.json']]
```

These things in the list will be used well soon.

Next, I get temperature elements from the JSON files above respectively.

In the JSON files, days of the week with times are presented instead of dates.
March 1st was on Sunday, so I start with finding Sunday data for analyzing March 1st one.
For March 2nd, I find Monday data, and so on.

Also, the JSON files take predictions ahead around two weeks, so there are duplicated Sunday data. To prevent me from getting unwanted data, I use a break in each time I use a for loop right after getting the data I need.

(Continued) Code:

```
c=0
temperature_list = []
all_tlist = []
for datelist in sequencelist:
	c+=1
	for date in datelist:
		jsonfile = json.loads(open(date).read() )
		if c==8:
			c=1

		if c==1:
			for lists in jsonfile['data']:
				if lists['dts']=='Sunday 8:00 AM' or lists['dts']=='Sunday 12:00
PM':
					temperature_list.append(lists['t'])
				elif lists['dts']=='Sunday 4:00 PM':
					temperature_list.append(lists['t'])
					all_tlist.append(temperature_list)
					temperature_list = []
					break

		if c==2:
			for lists in jsonfile['data']:
				if lists['dts']=='Monday 8:00 AM' or lists['dts']=='Monday 12:00
PM':
					temperature_list.append(lists['t'])
				elif lists['dts']=='Monday 4:00 PM':
					temperature_list.append(lists['t'])
					all_tlist.append(temperature_list)
					temperature_list = []
					break
		if c==3:
			for lists in jsonfile['data']:
				if lists['dts']=='Tuesday 8:00 AM' or lists['dts']=='Tuesday
12:00 PM':
					temperature_list.append(lists['t'])
				elif lists['dts']=='Tuesday 4:00 PM':
					temperature_list.append(lists['t'])
					all_tlist.append(temperature_list)
					temperature_list = []
					break
```

```
            if c==4:
                  for lists in jsonfile['data']:
                        if lists['dts']=='Wednesday 8:00 AM' or lists['dts']=='Wednesday
12:00 PM':
                              temperature_list.append(lists['t'])
                        elif lists['dts']=='Wednesday 4:00 PM':
                              temperature_list.append(lists['t'])
                              all_tlist.append(temperature_list)
                              temperature_list = []
                              break
            if c==5:
                  for lists in jsonfile['data']:
                        if lists['dts']=='Thursday 8:00 AM' or lists['dts']=='Thursday
12:00 PM':
                              temperature_list.append(lists['t'])
                        elif lists['dts']=='Thursday 4:00 PM':
                              temperature_list.append(lists['t'])
                              all_tlist.append(temperature_list)
                              temperature_list = []
                              break
            if c==6:
                  for lists in jsonfile['data']:
                        if lists['dts']=='Friday 8:00 AM' or lists['dts']=='Friday 12:00
PM':
                              temperature_list.append(lists['t'])
                        elif lists['dts']=='Friday 4:00 PM':
                              temperature_list.append(lists['t'])
                              all_tlist.append(temperature_list)
                              temperature_list = []
                              break
            if c==7:
                  for lists in jsonfile['data']:
                        if lists['dts']=='Saturday 8:00 AM' or lists['dts']=='Saturday
12:00 PM':
                              temperature_list.append(lists['t'])
                        elif lists['dts']=='Saturday 4:00 PM':
                              temperature_list.append(lists['t'])
                              all_tlist.append(temperature_list)
                              temperature_list = []
                              break
```

C) Results (output) on Step 4 (getting temperature elements from JSON files):

```
[[19, 32, 35], [18, 30, 34], [12, 25, 27], [13, 25, 28], [13, 24, 25], [22, 32, 28], [28,
30, 28], [20, 27, 27], [23, 28, 26], [24, 29, 27], [21, 26, 27], [24, 27, 25], [18, 29, 30],
[13, 24, 25], [16, 28, 31], [15, 33, 28], [18, 24, 23], [14, 26, 29], [27, 28, 27], [35, 38,
37],
…
…
[36, 41, 42], [36, 42, 41], [36, 41, 41], [37, 40, 42], [37, 39, 40], [38, 41, 41], [31, 39,
42], [30, 41, 42], [29, 38, 41], [28, 36, 38], [28, 38, 40], [29, 37, 39]]
```

Just reminding, when I am making a plot, the orders are March 1st 8:00 AM ~ March 31st 4:00 PM, respectively in time order like below:

(March 1st 8:00 AM, March 1st 12:00 PM, March 1st 4:00PM, March 2nd 8:00 AM, March 2nd 12:00 PM, March 2nd 4:00PM, … March 31st 8:00 AM, March 31st 12:00 PM, March 31st 4:00 PM))

So In order to compare March 1st 8:00 AM data to the predictions, I need the lists to be

ordered like this:

February 22nd ~ February 28th (all 8:00 AM), (so total 6 of these at first)
February 22nd ~ February 28th (all 12:00 PM),
February 22nd ~ February 28th (all 4:00 PM),
February 23rd ~ March 1st (all 8:00 AM),
…
March 25th ~ March 30th (all 4:00 PM)

The output above was in order like February 22nd, 8:00 AM –> February 22nd, 12:00 PM … Unfortunately, this is not the order I want, so I had to modify the list again in order to have it in a way I want.

(Continued) Code:

```
big_list=[]
six_bundle=[]
c=0
for each_list in all_tlist:
	c+=1
	six_bundle.append(each_list)
	if c==6:
		c=0
		big_list.append(six_bundle)
		six_bundle = []


morning = []
lunch = []
afternoon = []

another_big_list=[]
for bundle in big_list:
	for each_list in bundle:
		morning.append(each_list[0])
		lunch.append(each_list[1])
		afternoon.append(each_list[2])

	another_big_list.append(morning)
	morning = []

	another_big_list.append(lunch)
	lunch = []

	another_big_list.append(afternoon)
	afternoon = []
```

C) Results (output) on Step 4 (modifying lists):

```
[[19, 18, 12, 13, 13, 22], [32, 30, 25, 25, 24, 32], [35, 34, 27, 28, 25, 28], [28, 20, 23,
24, 21, 24], [30, 27, 28, 29, 26, 27], [28, 27, 26, 27, 27, 25], [18, 13, 16, 15, 18, 14],
[29, 24, 28, 33, 24, 26], [30, 25, 31, 28, 23, 29], [27, 35, 37, 33, 33, 31], [28, 38, 40,
34, 32, 31], [27, 37, 36, 30, 31, 30], [20, 17, 13, 10, 10, 10], [24, 15, 19, 15, 14, 13],

…
…

[25, 20, 18, 17, 16, 16], [37, 35, 33, 32, 32, 32], [41, 39, 38, 37, 38, 38], [36, 36, 36,
```

```
37, 37, 38], [41, 42, 41, 40, 39, 41], [42, 41, 41, 42, 40, 41], [31, 30, 29, 28, 28, 29],
[39, 41, 38, 36, 38, 37], [42, 42, 41, 38, 40, 39]]
```

Now the lists are well ordered in a way I want:

February 22nd ~ February 28th (all 8:00 AM),
February 22nd ~ February 28th (all 12:00 PM),
February 22nd ~ February 28th (all 4:00 PM),
February 23rd ~ March 1st (all 8:00 AM),
...
March 25th ~ March 30th (all 4:00 PM)

To make things easier in the future when I will be making plots, I modified the list again. All individual lists above consist of the predictions with six days ago ones, five days ago ones, … two days ago ones, and one days ago ones. So I made six different lists consisting of six days ago ones only, five days ago ones only, … two days ago ones only, and one days ago ones only.

(Continued) Code:

```
six_ago=[]
five_ago=[]
four_ago=[]
three_ago=[]
two_ago=[]
one_ago=[]

for each_list in another_big_list:
       six_ago.append(each_list[0])
       five_ago.append(each_list[1])
       four_ago.append(each_list[2])
       three_ago.append(each_list[3])
       two_ago.append(each_list[4])
       one_ago.append(each_list[5])
```

C) Results (output) on Step 4 (modifying lists again):

```
[19, 32, 35, 28, 30, 28, 18, 29, 30, 27, 28, 27, 20, 24, 22, 10, 24, 28, 25, 34, 36, 19, 28,
31, 27, 34, 35, 29, 37, 41, 31, 37, 40, 23, 31, 33, 25, 33, 38, 31, 38, 38, 31, 37, 37, 32,
41, 44, 31, 33, 32, 25, 30, 34, 29, 34, 36, 24, 36, 38, 28, 36, 39, 21, 27, 27, 19, 27, 27,
22, 35, 37, 34, 47, 51, 40, 41, 40, 27, 30, 31, 21, 28, 31, 25, 37, 41, 36, 41, 42, 31, 39,
42] #six_ago
…
[22, 32, 28, 24, 27, 25, 14, 26, 29, 31, 31, 30, 10, 13, 14, 4, 17, 20, 21, 31, 33, 28, 30,
31, 28, 35, 37, 32, 43, 44, 37, 40, 41, 24, 34, 36, 25, 42, 49, 41, 44, 42, 35, 37, 40, 36,
46, 45, 38, 38, 37, 21, 30, 33, 19, 32, 35, 31, 41, 44, 39, 42, 43, 20, 24, 28, 16, 22, 26,
20, 32, 39, 31, 47, 48, 37, 39, 37, 30, 36, 35, 17, 22, 27, 16, 32, 38, 38, 41, 41, 29, 37,
39] # one_ago
```

Finally all the data are well ordered and organized!
Now I just need to do actual data analysis with only few simple code lines in easy ways!

Step 5: Error between real data and predictions:

Now I'd like to see numerical errors between real data and predictions.

B) Methods for Step 5:

In lists, I convert the elements into “int” versions first, so that I can subtract them (lists by lists). Next, I subtract the elements with absolute values ([actual data list] - [prediction lists] { six days ago, five days ago … one day ago}).

And then I divide them by length of the lists, and round them up to two decimal points.

Code:

```
int_temperaturelist= [int(i) for i in temperaturelist]

int_six_ago= [int(i) for i in six_ago]
int_five_ago= [int(i) for i in five_ago]
int_four_ago= [int(i) for i in four_ago]
int_three_ago= [int(i) for i in three_ago]
int_two_ago= [int(i) for i in two_ago]
int_one_ago= [int(i) for i in one_ago]

err_six_ago = [abs(i - j) for i, j in zip(int_temperaturelist , int_six_ago)]
err_five_ago = [abs(i - j) for i, j in zip(int_temperaturelist , int_five_ago)]
err_four_ago = [abs(i - j) for i, j in zip(int_temperaturelist , int_four_ago)]
err_three_ago = [abs(i - j) for i, j in zip(int_temperaturelist , int_three_ago)]
err_two_ago = [abs(i - j) for i, j in zip(int_temperaturelist , int_two_ago)]
err_one_ago = [abs(i - j) for i, j in zip(int_temperaturelist , int_one_ago)]


print 'March_temperature errors:'
print '_____________________________________________________________'
print 'average error of six days ago predictions:',
round(float(sum(err_six_ago))/len(err_six_ago),2)
print 'average error of five days ago predictions:',
round(float(sum(err_five_ago))/len(err_five_ago),2)
print 'average error of four days ago predictions:',
round(float(sum(err_four_ago))/len(err_four_ago),2)
print 'average error of three days ago
predictions:',round(float(sum(err_three_ago))/len(err_three_ago),2)

print 'average error of two days ago predictions:',
round(float(sum(err_two_ago))/len(err_two_ago),2)
print 'average error of one days ago predictions:',
round(float(sum(err_one_ago))/len(err_one_ago),2)
```

C) Results (output) and Discussions on Step 5:

March_temperature comparison:

```
March_temperature errors:
_____________________________________________________________
average error of six days ago predictions: 4.66
average error of five days ago predictions: 3.57
average error of four days ago predictions: 3.33
average error of three days ago predictions: 2.81
average error of two days ago predictions: 2.58
average error of one days ago predictions: 2.38
```

Since the codes are pretty much the same, I will put the results of other comparisons without presenting the codes.

April_temperature comparison:

```
April_temperature errors:
_____________________________________________________________
average error of six days ago predictions: 5.39
average error of five days ago predictions: 4.93
average error of four days ago predictions: 4.43
average error of three days ago predictions: 4.16
average error of two days ago predictions: 3.68
average error of one days ago predictions: 3.25
```

March_humidity comparison:

```
March_humidity errors:
_____________________________________________________________
average error of six days ago predictions: 13.11
average error of five days ago predictions: 12.35
average error of four days ago predictions: 10.84
average error of three days ago predictions: 9.92
average error of two days ago predictions: 9.11
average error of one days ago predictions: 8.84
```

April_humidity comparison:

```
April_humidity errors:
_____________________________________________________________
average error of six days ago predictions: 15.21
average error of five days ago predictions: 12.45
average error of four days ago predictions: 12.03
average error of three days ago predictions: 11.33
average error of two days ago predictions: 11.31
average error of one days ago predictions: 10.72
```

If we look at the results, both temperature and humidity predictions in March were more accuare than those in April. Also, temperature predictions have lesser errors than humidity predictions'. Lastly, the more recent predictions, the more accurate predictions.

Step 6: Plots of real data and predictions:

In this step I will make plots showing all the results of actual data and predictions.

B) Methods for Step 6:

Since I put my the data I need into the lists, I just have to make plots by inserting those lists.

Code:

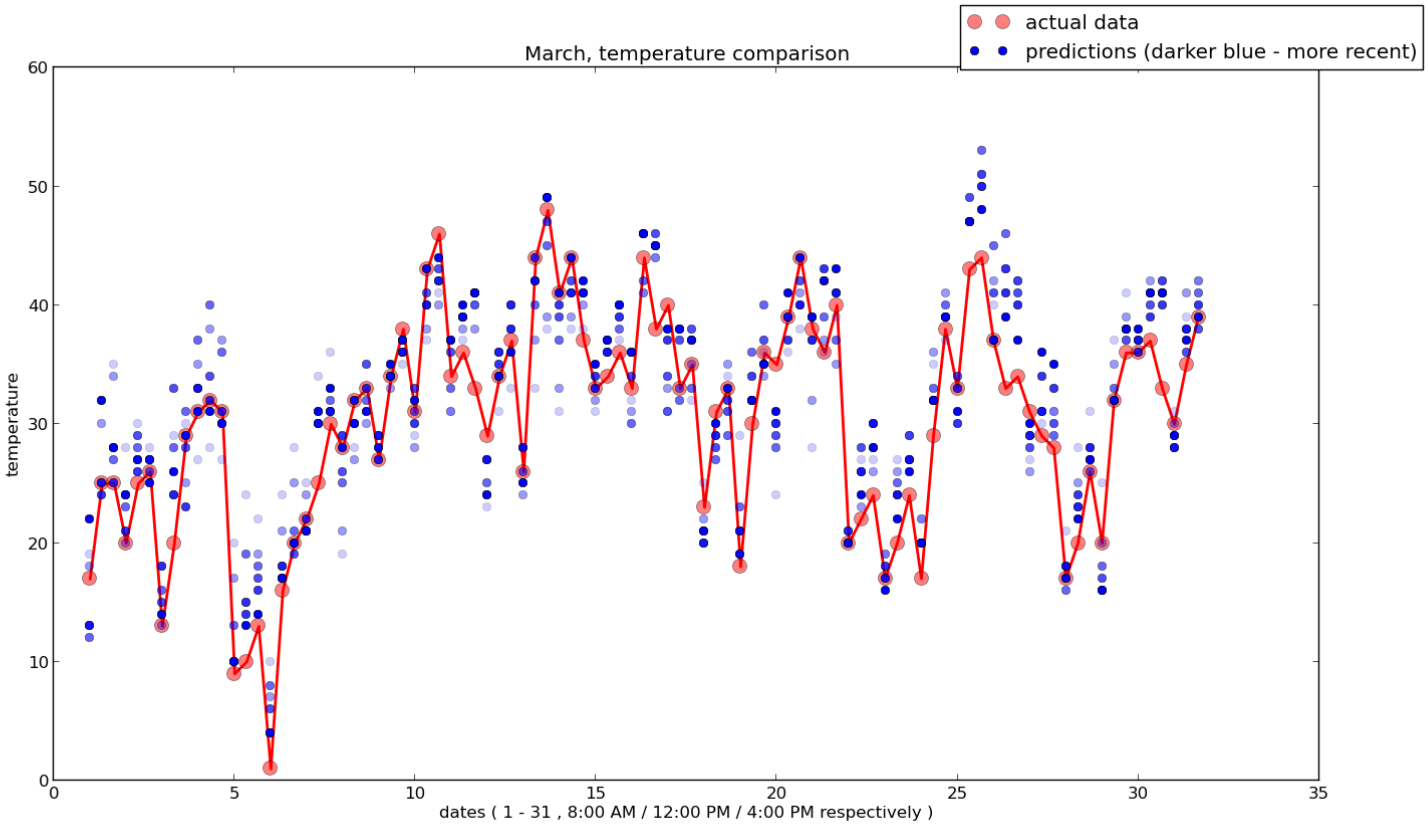
```
dates=[]
date=1
```

```
for date_count in (temperaturelist):
      dates.append(date)
      date+=1./3  # for each day, 8 AM, 12 PM, 4 PM – 1 day total

plot(dates,temperaturelist,'r-',linewidth=2)
plot(dates,temperaturelist,'ro', markersize=10, alpha=0.5,label='actual data')
xlabel('dates ( 1 - 31 , 8:00 AM / 12:00 PM / 4:00 PM respectively )')
ylabel('temperature')
title('March, temperature comparison')

plot(dates,six_ago,'bo',alpha=0.2)
plot(dates,five_ago,'bo',alpha=0.4)
plot(dates,four_ago,'bo',alpha=0.6)
plot(dates,three_ago,'bo',alpha=0.7)
plot(dates,two_ago,'bo',alpha=0.8)
plot(dates,one_ago,'bo',alpha=2.0, label='predictions (darker blue - more recent)')
legend(loc=9, bbox_to_anchor=(0.9, 1.1))
show()
```

C) Results (output) and Discussions on Step 6:

Besides the result of the code above (March_temperature comparison plot), I will also put the plots of other comparisons.

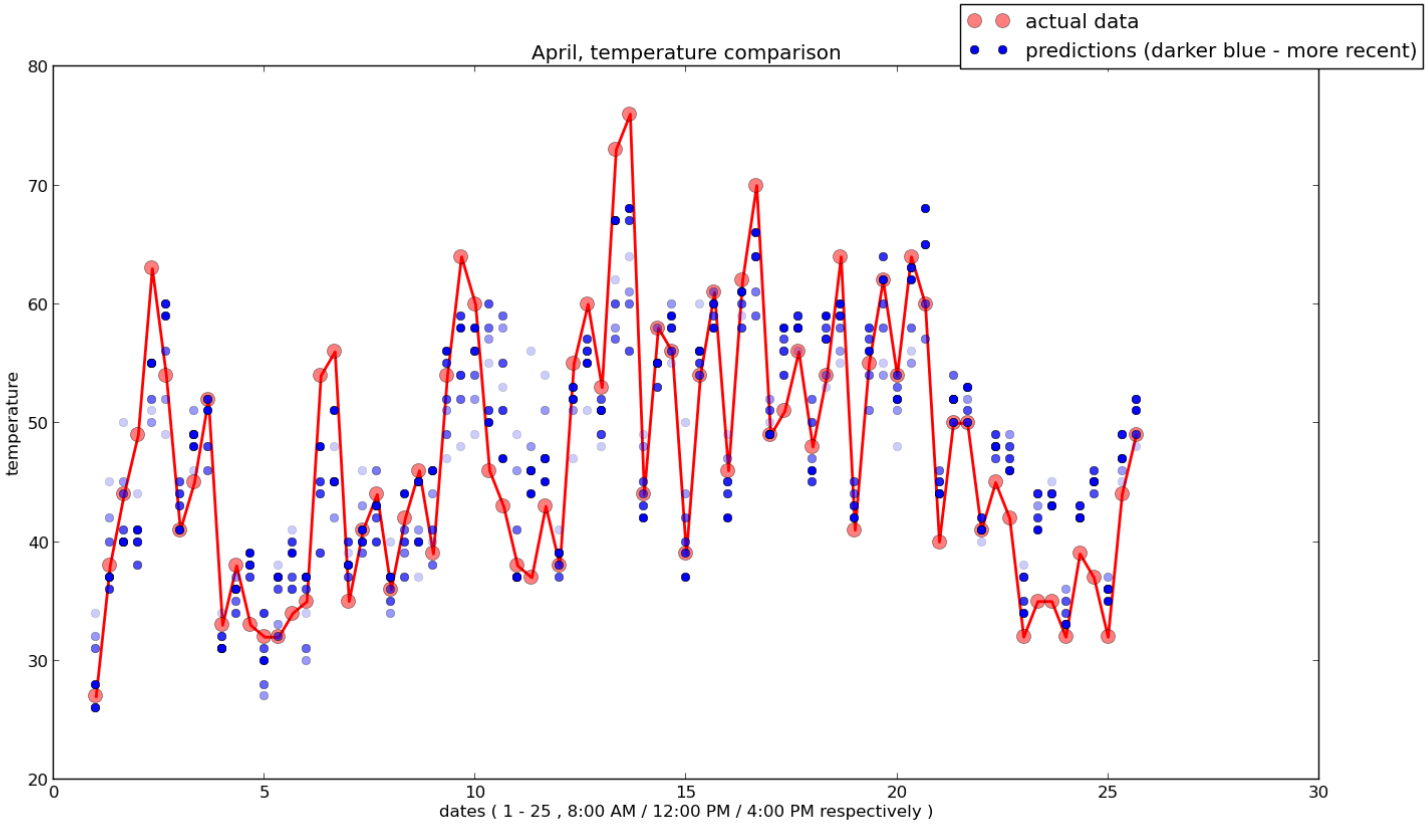
March_temperature comparison:



Blue dots are the predictions, and red dots are the real data. For each blue dot, the darker, the more recent predictions. So the light blue dots are the predictions of 5~6 days ago from

the actual data. The dark blue dots are the recent predictions like 1~2 days ago. By looking at the graph above, blue dots are likely to be located higher than the actual data, so it can be assumed that in March, the weather predictions usually expect temperature warmer than the actual ones.

April_temperature comparison:



It will be great to know whether the temperature predictions usually expect warmer conditions (higher temperature) than the actual conditions. Numerical results are presented soon. For April, it looks like blue dots are located quite equally though (both higher than or lower than the real data equally)

So I wrote codes showing average numercal values of the ones with higher average temperature than expected (warmer), and the ones with lower average temperature than expected (colder).

Code:
(*please note that this time when I'm doing subtraction:
[predictions] - [actual data] instead of [actual data] - [predictions],
so that positive numbers represent warmer expectations than actual temperature, and negative numbers represent colder expectations than actual temperature)

```
err_six_ago = [(i - j) for i, j in zip(int_six_ago , int_temperaturelist )]
err_five_ago = [(i - j) for i, j in zip(int_five_ago, int_temperaturelist)]
err_four_ago = [(i - j) for i, j in zip(int_four_ago, int_temperaturelist)]
```

```
err_three_ago = [(i - j) for i, j in zip(int_three_ago, int_temperaturelist)]
err_two_ago = [(i - j) for i, j in zip(int_two_ago, int_temperaturelist)]
err_one_ago = [(i - j) for i, j in zip(int_one_ago, int_temperaturelist)]

warmer_error=[]
colder_error=[]
for error in err_six_ago:
	if error>0:
		warmer_error.append(error)
	elif error<0:
		colder_error.append(error)
print 'April - warmer? colder?'
print '_____________________________________________'
print 'average warmer error of six days ago predictions:',
round(float(sum(warmer_error))/len(warmer_error),2)
print 'average colder error of six days ago predictions:',
round(float(sum(colder_error))/len(colder_error),2)

warmer_error=[]
colder_error=[]
for error in err_five_ago:
	if error>0:
		warmer_error.append(error)
	elif error<0:
		colder_error.append(error)
print 'average warmer error of five days ago predictions:',
round(float(sum(warmer_error))/len(warmer_error),2)
print 'average colder error of five days ago predictions:',
round(float(sum(colder_error))/len(colder_error),2)

warmer_error=[]
colder_error=[]
for error in err_four_ago:
	if error>0:
		warmer_error.append(error)
	elif error<0:
		colder_error.append(error)
print 'average warmer error of four days ago predictions:',
round(float(sum(warmer_error))/len(warmer_error),2)
print 'average colder error of four days ago predictions:',
round(float(sum(colder_error))/len(colder_error),2)

warmer_error=[]
colder_error=[]
for error in err_three_ago:
	if error>0:
		warmer_error.append(error)
	elif error<0:
		colder_error.append(error)
print 'average warmer error of three days ago predictions:',
round(float(sum(warmer_error))/len(warmer_error),2)
print 'average colder error of three days ago predictions:',
round(float(sum(colder_error))/len(colder_error),2)

warmer_error=[]
colder_error=[]
for error in err_two_ago:
	if error>0:
		warmer_error.append(error)
	elif error<0:
		colder_error.append(error)
print 'average warmer error of two days ago predictions:',
round(float(sum(warmer_error))/len(warmer_error),2)
print 'average colder error of two days ago predictions:',
```

```
round(float(sum(colder_error))/len(colder_error),2)

warmer_error=[]
colder_error=[]
for error in err_one_ago:
      if error>0:
            warmer_error.append(error)
      elif error<0:
            colder_error.append(error)
print 'average warmer error of one day ago predictions:',
round(float(sum(warmer_error))/len(warmer_error),2)
print 'average colder error of one day ago predictions:',
round(float(sum(colder_error))/len(colder_error),2)
```

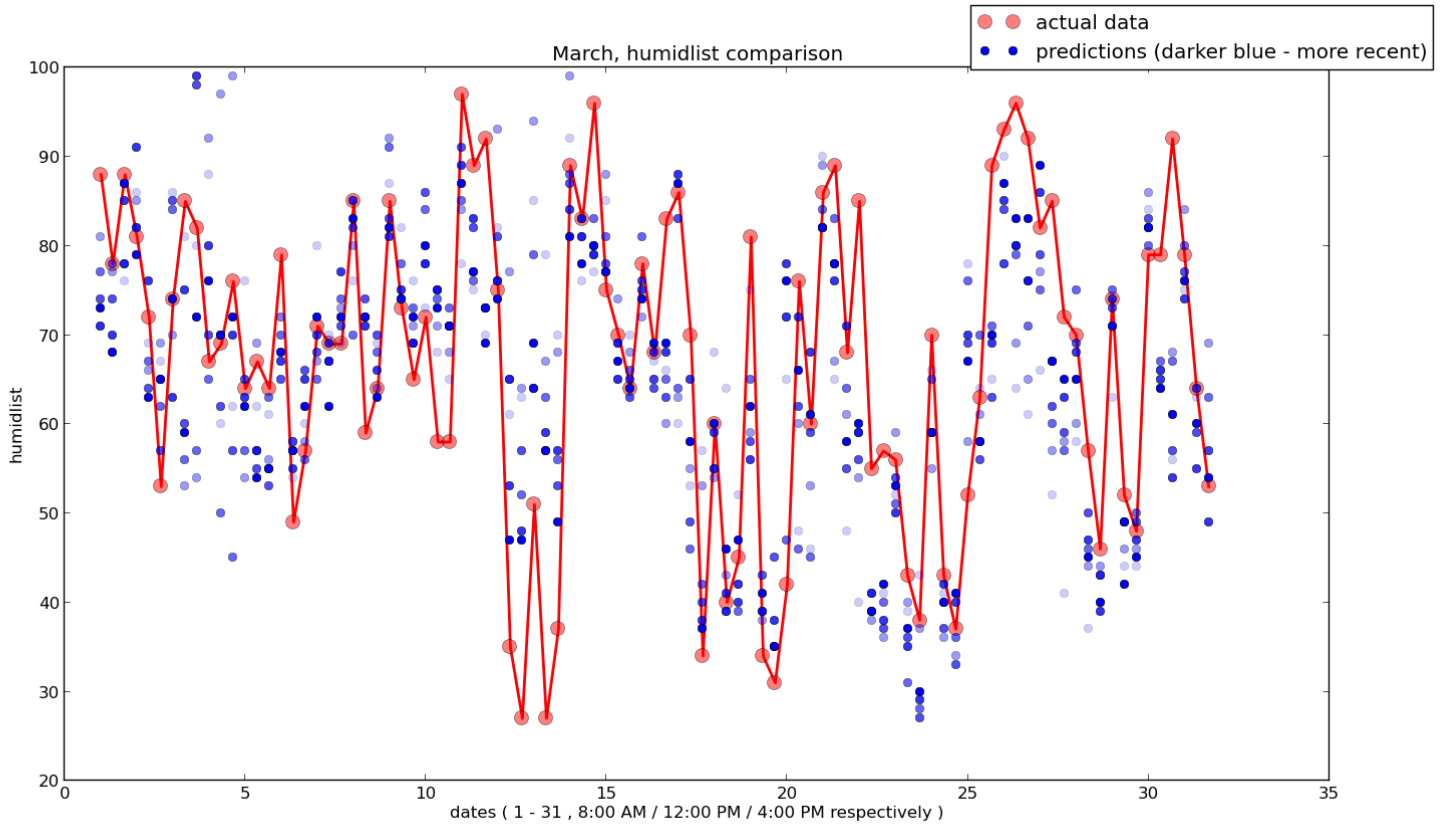
Output:

```
March - warmer? colder?
_____________________________________________
average warmer error of six days ago predictions: 5.11
average colder error of six days ago predictions: -4.75
average warmer error of five days ago predictions: 4.06
average colder error of five days ago predictions: -4.18
average warmer error of four days ago predictions: 4.25
average colder error of four days ago predictions: -2.97
average warmer error of three days ago predictions: 3.47
average colder error of three days ago predictions: -2.24
average warmer error of two days ago predictions: 3.07
average colder error of two days ago predictions: -2.68
average warmer error of one day ago predictions: 3.25
average colder error of one day ago predictions: -1.77

April - warmer? colder?
_____________________________________________
average warmer error of six days ago predictions: 5.57
average colder error of six days ago predictions: -5.82
average warmer error of five days ago predictions: 4.58
average colder error of five days ago predictions: -6.45
average warmer error of four days ago predictions: 4.62
average colder error of four days ago predictions: -5.47
average warmer error of three days ago predictions: 4.38
average colder error of three days ago predictions: -4.41
average warmer error of two days ago predictions: 4.0
average colder error of two days ago predictions: -3.87
average warmer error of one day ago predictions: 3.62
average colder error of one day ago predictions: -3.32
```

As I expected from the plots, in March it was more likely to expect temperature warmer than actual, and in April both warmer and colder expectations' averages are quite similar.

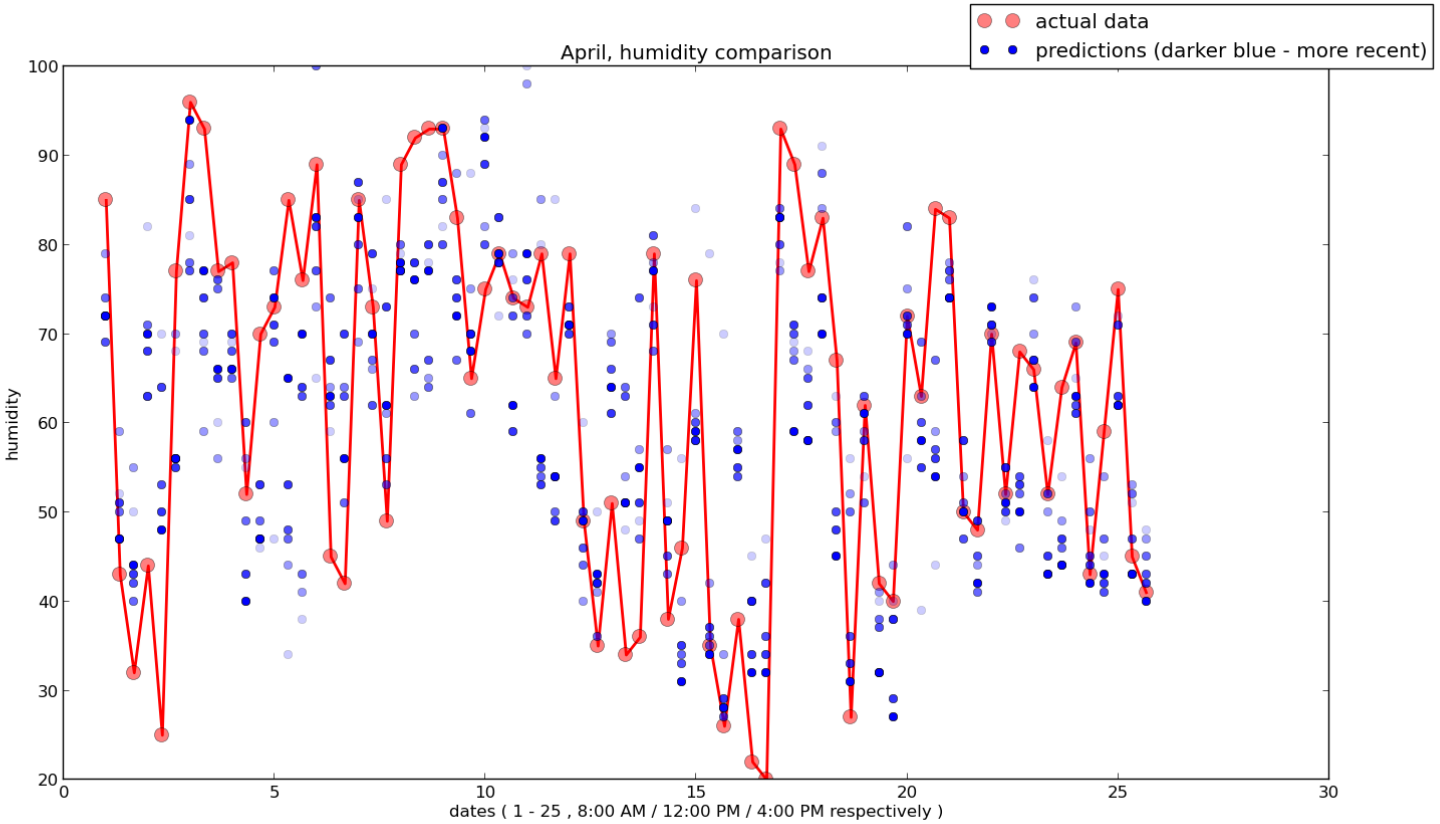
March_humidity comparison:

Humidity predictions are not as good as the temperature predictions. Also, if we look at the blue dots in the same x-axis, they vary too much, which means the predictions are not stable compared to the temperature ones. Fortunately, we human beings do not usually care about humidity, but pay more attention in temperature and conditions (especially if it rains or not), so I think it's okay for the weather forecasts to have bad predictions in humidity.

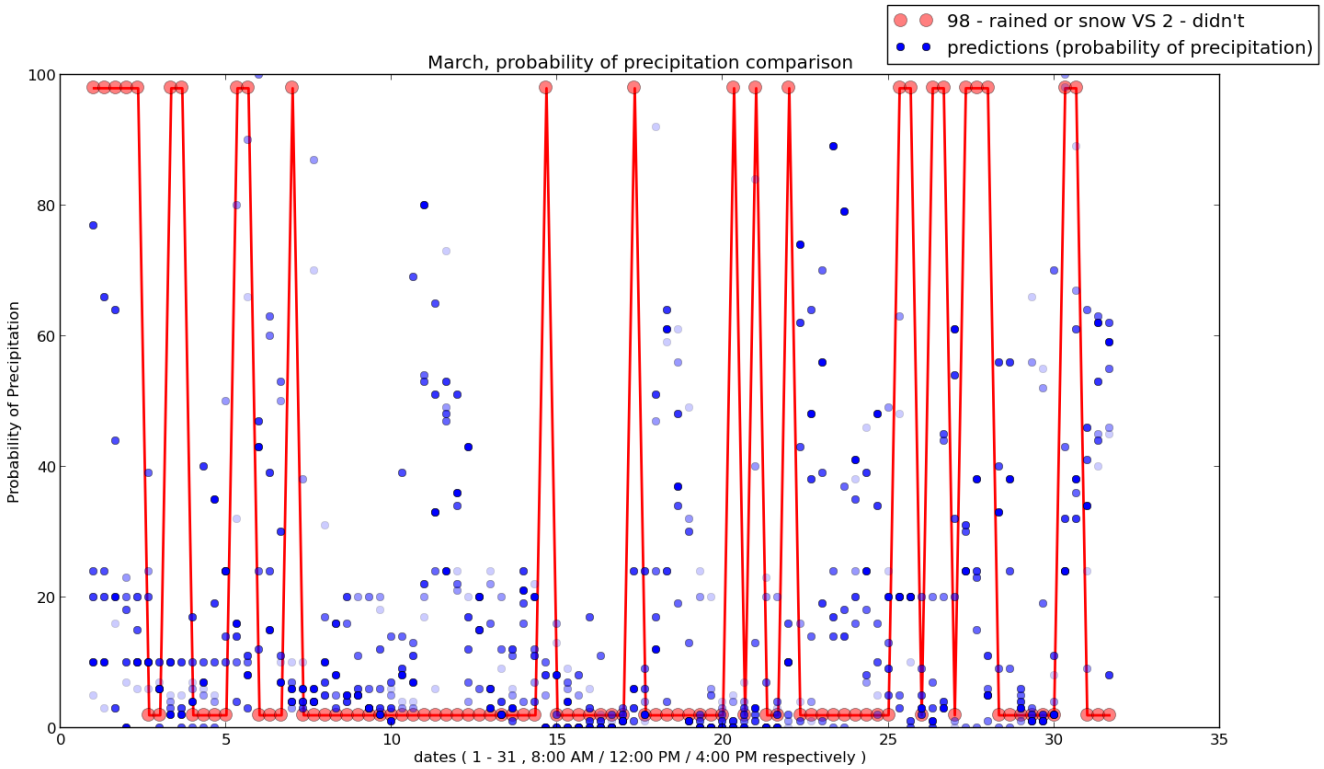


April_humidity comparison:

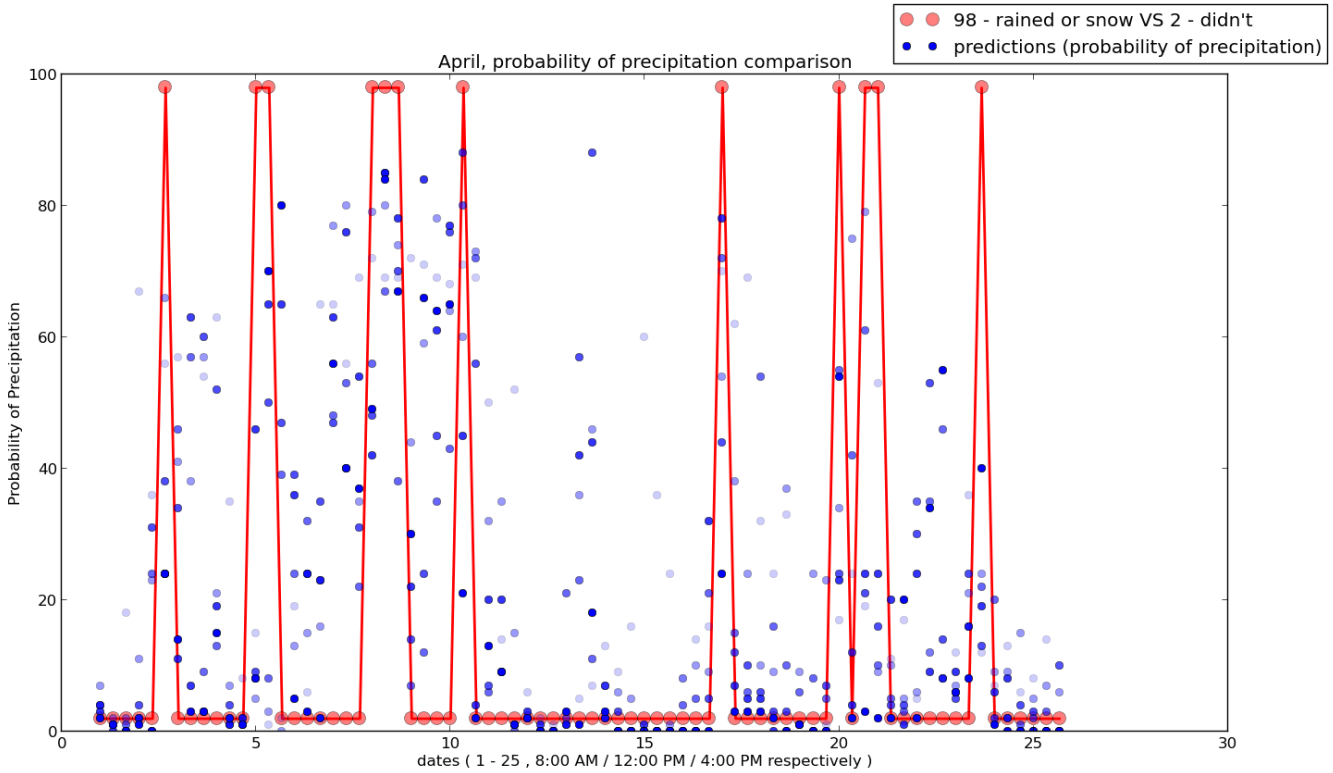
Just like March_humidity, this April_humidity predictions are not that good either.



March_probability of precipitation comparison:
By looking at these plots, the predictions are not very good either as there aren't that many blue dots matching the red dots in the same or close positions.



April_probability of precipitation comparison:



At least I have a faith in the weather forecast website's temperature predictions as the errors were quite small.

Step 7. Conclusion:

1. Compared to other aspects like humidity and probability of precipitation, the temperature predictions were the most accurate with less errors.
2. For humidity and temperature predictions, March had higher accuracy than April.
3. The more recent dates, the more accurate information.