

## **Lab Center – Hands-On Lab – Part TWO**

Session 1239 - IBM Think2020 IoT Lab

# Hyper-Local Weather and Crop prediction using Watson: Analysing Weather Data using Jupyter Notebook

Markus van Kempen – mvk@ca.ibm.com



## DISCLAIMER

IBM's statements regarding its plans, directions, and intent are subject to change or withdrawal without notice at IBM's sole discretion. Information regarding potential future products is intended to outline potential future products is intended to outline our general product direction and it should not be relied on in making a purchasing decision.

The information mentioned regarding potential future products is not a commitment, promise, or legal obligation to deliver any material, code or functionality. Information about potential future products may not be incorporated into any contract.

The development, release, and timing of any future features or functionality described for our products remains at our sole discretion I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve results like those stated here.

Information in these presentations (including information relating to products that have not yet been announced by IBM) has been reviewed for accuracy as of the date of initial publication and could include unintentional technical or typographical errors. IBM shall have no responsibility to update this information. **This document is distributed "as is" without any warranty, either express or implied. In no event, shall IBM be liable for any damage arising from the use of this information, including but not limited to, loss of data, business interruption, loss of profit or loss of opportunity.** IBM products and services are warranted per the terms and conditions of the agreements under which they are provided.

IBM products are manufactured from new parts or new and used parts. In some cases, a product may not be new and may have been previously installed. Regardless, our warranty terms apply."

**Any statements regarding IBM's future direction, intent or product plans are subject to change or withdrawal without notice.**

Performance data contained herein was generally obtained in controlled, isolated environments. Customer examples are presented as illustrations of how those customers have used IBM products and the results they may have achieved. Actual performance, cost, savings or other results in other operating environments may vary. References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business.

Workshops, sessions and associated materials may have been prepared by independent session speakers, and do not necessarily reflect the views of IBM. All materials and discussions are provided for informational purposes only, and are neither intended to, nor shall constitute legal or other guidance or advice to any individual participant or their specific situation.

It is the customer's responsibility to insure its own compliance with legal requirements and to obtain advice of competent legal counsel as to the identification and interpretation of any relevant laws and regulatory requirements that may affect the customer's business and any actions the customer may need to take to comply with such laws. IBM does not provide legal advice or represent or warrant that its services or products will ensure that the customer follows any law.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products about this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products. IBM does not warrant the quality of any third-party products, or the ability of any such third-party products to interoperate with IBM's products. **IBM expressly disclaims all warranties, expressed or implied, including but not limited to, the implied warranties of merchantability and fitness for a purpose.**

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents, copyrights, trademarks or other intellectual property right.

IBM, the IBM logo, and ibm.com are trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies.



IV

A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at:  
[www.ibm.com/legal/copytrade.shtml](http://www.ibm.com/legal/copytrade.shtml).

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

OpenShift is a trademark of Red Hat, Inc.

UNIX is a registered trademark of The Open Group in the United States and other countries.

© 2020 International Business Machines Corporation. No part of this document may be reproduced or transmitted in any form without written permission from IBM.

**U.S. Government Users Restricted Rights — use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM.**

## We Value Your Feedback

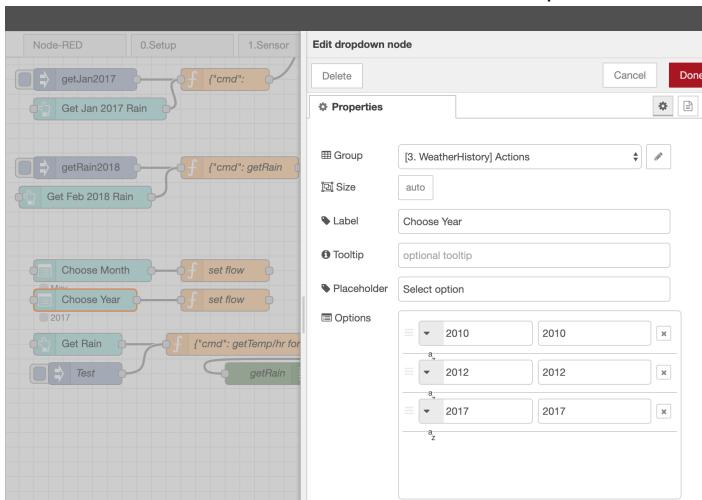
Don't forget to submit your Think 2020 session and speaker feedback! Your feedback is very important to us – we use it to continually improve the conference.

Access the Think 2020 agenda tool to quickly submit surveys from your smartphone or laptop.

## Table of Contents

<b>1 Objective.....</b>	<b>6</b>
1.1 Pre-Requisite .....	6
1.2 Download the github repository.....	7
1.3 Import the Jupyter Notebook .....	7
1.4 How to navigate Jupyter Notebook.....	10
1.5 Fixing Outliers in our Weather data.....	11
1.6 Preprocessing Weather data.....	13
1.6.1 Group the data by DAY .....	14
1.6.2 Group the data and calculating Dry and Wet days.....	15
1.6.3 Sampling Monthly data.....	16
1.6.4 Plot the Monthly data.....	17
1.7 Playground for Query testing.....	18
<b>2 Connecting to Node-RED .....</b>	<b>19</b>
<b>3 Sending a command to Python .....</b>	<b>21</b>
<b>4 Using the Dashboard and data Queries .....</b>	<b>23</b>
4.1 Adding more functionality .....	24
Exercise .....	24

- Add more Month and Years to the Drop down on the Dashboard.



24



2020

Think 2020



# 1 Objective

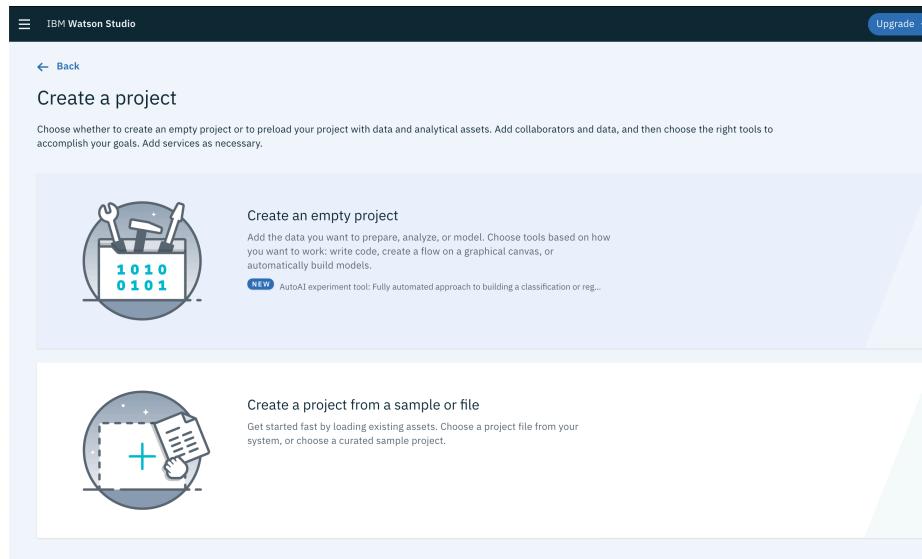
In this part of the LAB we will setup a WatsonStudio Instance with a Jupyter Notebook to analyse some weather data (ideally our own weather data we collected in previous years). Once we analyse the data we will setup a connection to our Node-RED instance and query the data and display it in Node-RED.

**Note:** We will walkthrough Part two together 1<sup>s.t</sup>

## 1.1 Pre-Requisite

Please sign up for a free WatsonStudio/IBM Cloud account [Register for WatsonStudio](https://dataplatform.cloud.ibm.com/registration/stepone)  
<https://dataplatform.cloud.ibm.com/registration/stepone>

Create an empty project with the name Think2020. You will be ask to add an object store just follow the steps.



The screenshot shows the 'Create a project' interface in IBM Watson Studio. At the top, there's a navigation bar with a 'Back' button and an 'Upgrade' dropdown. Below the header, there are two main project creation options:

- Create an empty project:** This option features a circular icon with a wrench and a screwdriver, and binary code '1010 0101'. A sub-section titled 'Create an empty project' explains that users can add data, tools, and collaborators. It highlights the 'AutoAI experiment tool' as a new feature.
- Create a project from a sample or file:** This option features a circular icon with a plus sign and a document. A sub-section explains that users can get started fast by loading existing assets.



The screenshot shows the IBM Watson Studio interface. At the top, there's a navigation bar with 'IBM Watson Studio', 'Upgrade', 'Launch IDE', 'Add to project', and a user icon. Below the bar, the project name 'Think2020' is displayed, along with the last update date: 'Apr 28, 2020'. The interface includes tabs for 'Overview', 'Assets', 'Environments', 'Jobs', 'Deployments', 'Access Control', and 'Settings'. The 'Overview' tab is selected. On the left, there's a sidebar with sections for 'Recent activity', 'Overview' (Date created: Apr 28, 2020; Description: No description available; Storage: 0 Byte used, Cloud Object Storage), 'Collaborators' (mcd crfun, Admin, View all (1)), and 'Readme' (Document your project using standard Markdown syntax). The main area is currently empty under 'Recent activity'.

## 1.2 Download the github repository

### Downloading the github repository

The screenshot shows a GitHub repository page for 'markusvankempen / ThinkLab1239'. The repository title is 'IBM Think2020 IoT Lab - Hyper Localized Weather and Crop prediction using Watson'. It has 56 commits, 1 branch, 0 packages, 0 releases, and 1 contributor. The 'Clone or download' button is highlighted. The repository page lists several files and folders: 'data', 'extra', 'images', 'instructions', and 'node-red'. The 'node-red' folder contains a file named 'final.json'. The 'Clone with HTTPS' and 'Download ZIP' buttons are also visible.

Download the github repository form here  
<https://github.com/markusvankempen/ThinkLab1239>  
and unzip the file onto your VM / Desktop.

## 1.3 Import the Jupyter Notebook

Go back to the WatsonStudio browser window and import the notebook

IBM Watson Studio

My projects / Think2020

Last Updated: Apr 28, 2020

Overview Assets Environments Jobs Deployments Access Control Settings

## Think2020

Readme

**Overview**

Date created Apr 28, 2020

Description No description available

Storage 0 Byte used Cloud Object Storage

Collaborators mcd crfun Admin

[View all \(1\)](#)

**Choose asset type**

**Notebook**  
Run small pieces of code to process your data and immediately view the results.

Available asset types

- Data
- Connection
- Connected data
- AutoAI experiment
- Notebook
- Dashboard
- Visual Recognition ...
- Natural Language Cl...
- Watson Machine Lea...
- Deep learning experi...
- Modeler flow
- Data Refinery flow
- Streams flow
- Decision Optimizatio... **NEW**

Find then notebook [WS-PartOne.ipynb](#) in your downloaded/zip file on you VM/Desktop. It should be in a directory that looks something like : ... /thinklab1239/note-books/

IBM Watson Studio

## New notebook

**Blank** From file From URL

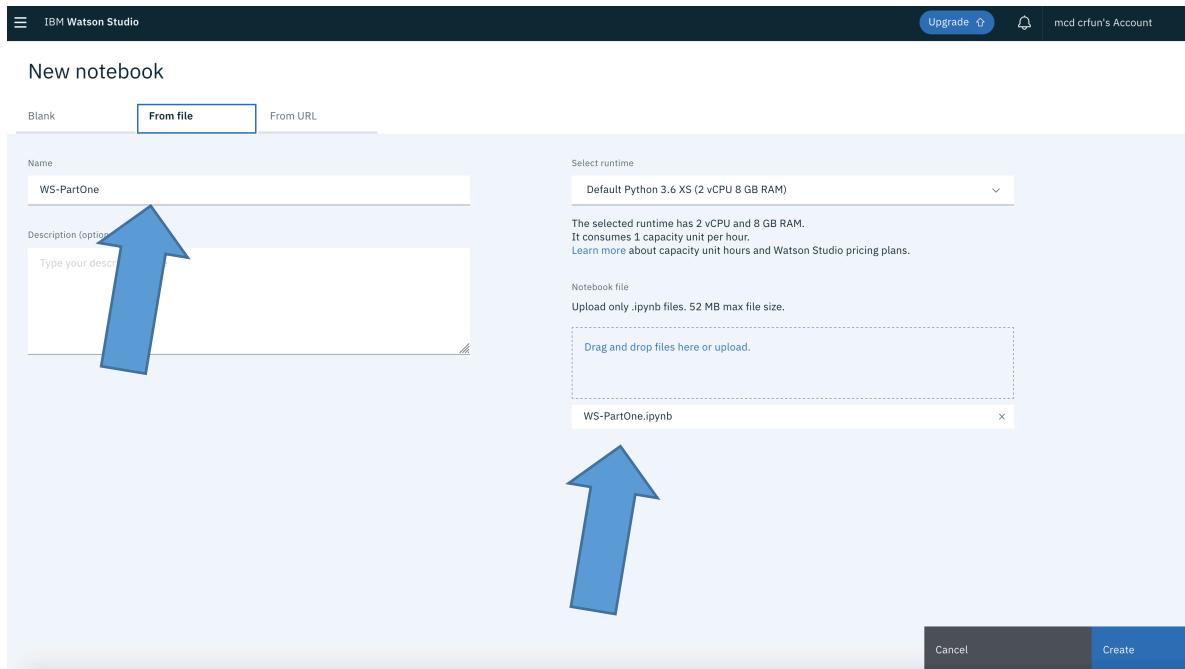
Name

Description (optional)

Select runtime  
Default Python 3.6 XS (2 vCPU 8 GB)

The selected runtime has 2 vCPU and 8 GB memory. It consumes 1 capacity unit per hour. [Learn more](#) about capacity unit hours and usage.

Language  
 Python 3.6



Click create. You should see the following screen.

The screenshot shows the Watson Studio notebook interface with the title 'LAB1239 - Watson Studio - Part1'. The notebook contains three code cells:

- In [1]:**

```
#Based on NOAA data / Python Book in
#https://dataplatform.cloud.ibm.com/exchange/public/entry/view/a7432f0c29c5bda2fb42749f363bd9ff
#
```
- In [2]:**

```
ls
```

Output:

```
FabirxSensor8x8.ipynb           forecast.ipynb
Krakow_2007_2018.csv             input_data.xlsx
LICENSE                           jfk_weather.csv
MonthlyRainPrediction.ipynb       jfk_weather.csv 2.zip
MonthlyRainPrediction20200425.ipynb jfk_weather.csv.zip
MontlyRainPredictions.ipynb       jfk_weather_cleaned.csv
Part 1 - Data Cleaning (1).ipynb jfk_weather_cleaned.csv.zip
README.md                         machine-learning-predict-weather-master
Untitled.ipynb                   model_selection.ipynb
Untitled1.ipynb                  open_csv.py
WU-Ground                      realtimeplottingtest.ipynb
Weather_forecasting-master    requirements.txt
exploratory_analysis.ipynb       results.xlsx
fabfabric-5.csv                   tensorflow-lstm-regression-master
```
- In [2]:**

```
#Read Data File
import pandas as pd
import io
import requests
import calendar
import json
import matplotlib.pyplot as plt
import seaborn as sns
#IEDINBURG_weather.csv.zip
```

## 1.4 How to navigate Jupyter Notebook

Just a quick couple of pointers to how to execute a Notebook. The Notebook has documentation and code cell both can be “execute” with the arrow button on the top. After execution of a cell, you may see result at the bottom on the cell. The cursors will be automatically move to the next cell.

The screenshot shows a Jupyter Notebook interface with the following details:

- File Bar:** File, Edit, View, Insert, Cell, Kernel, Help.
- Toolbar:** Includes icons for file operations like Open, Save, Run, and Cell.
- Title Bar:** My projects / Think2020 / WS-PartOne.
- Code Cells:**
  - In [2]: `#Read Data File`
  - In [3]: `import pandas as pd`
  - In [4]: `import io`
  - In [5]: `!ls /cmd` (Output: bin evm lib lib64 opt root sbin sys usr dev home lib64 mnt proc run srv tmp var)
  - In [6]: `#Read Data File` (Continues with data reading and cleaning code)
- Execution Status:** Shows "In [2] 100% Complete" and "In [6] 100% Complete".
- Annotations:** Blue arrows point to the "Run" button in the toolbar, the "Cell" button in the menu, and the "Edit Mode" button in the menu.

**Note:** Will see execution result once the Notebook is imported but these are just from save from an old session you always need to start from the top to execute the cell once you opened a Notebook /Put in Edit Mode

The screenshot shows a Jupyter Notebook interface with the following details:

- File Bar:** File, Edit, View, Insert, Cell, Kernel, Help.
- Toolbar:** Includes icons for file operations like Open, Save, Run, and Cell.
- Title Bar:** My projects / Think2020 / WS-PartOne.
- Code Cells:**
  - In [6]: `data_raw = pd.read_csv(url, compression='zip')`
  - In [7]: `data_raw['DATE'] = pd.to_datetime(data_raw['DATE'])`
  - In [8]: `data_raw['day'] = data_raw['DATE'].apply(lambda x: x.date())`
  - In [9]: `data_raw['day'] = pd.to_datetime(data_raw['DATE'])`
  - In [10]: `data_raw.head()`
- Result:** An arrow points to the output of In [10], which is a data frame named "data\_raw" with columns: DATE, visibility, dry\_bulb\_temp\_f, wet\_bulb\_temp\_f, dew\_point\_temp\_f, relative\_humidity, wind\_speed, station\_pressure, sea\_level\_pressure, precip, altimeter\_setting, wind\_c.
- Code Cell:** In [11]: `Checking Data`
- Plot:** Out[11]: A line plot showing precipitation over time.
- Annotations:** A large blue arrow points to the "Result" data frame output.

**TIP: Use the Save Button regularly**

## 1.5 Fixing Outliers in our Weather data

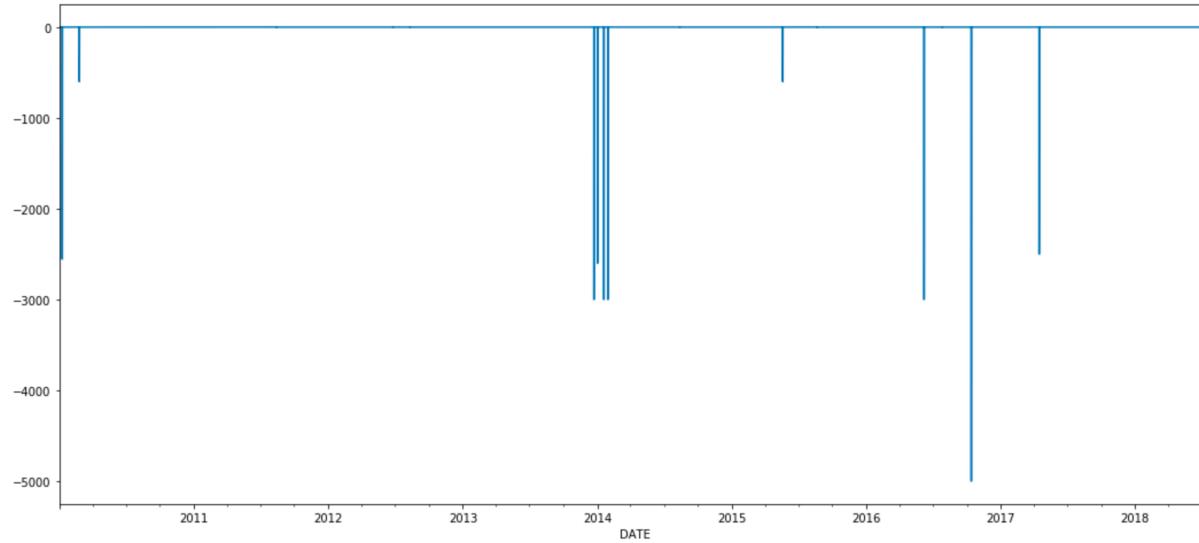
We will read a csv file with weather data in the Notebook/Python and display the columns and rows of data. Data is taken by the hour from 2010 -2018. When you have executed the first code cell you will see the data/structure of the file (*See above picture*). Our first task is to get a picture of the data to check consistency etc.

Below we simply plot the precipitation (rain) data into a chart. We can see there is something wrong with some of the data. This kind of data glitch can happen when your PWS gets cleaned or if there is an power outage/surge. It's always good practice to analyze the consistency to the data/columns and remove the outliers.

### Checking Data

```
#Check data like precip - we will see some error so we need to adjust them
plt.figure(figsize=(18,8))
data_raw[['precip']].plot()
```

: <matplotlib.axes.\_subplots.AxesSubplot at 0x7fba5e45a4e0>



### Exercise:

We see in image above that there is negative rain on some days hours of the year. In order to get consistency we need to add some code in the Code Cell bellow ... Something which simply zeros the entries like:

```
data_raw = data_raw[data_raw['precip'] > -500] #Fixing
```

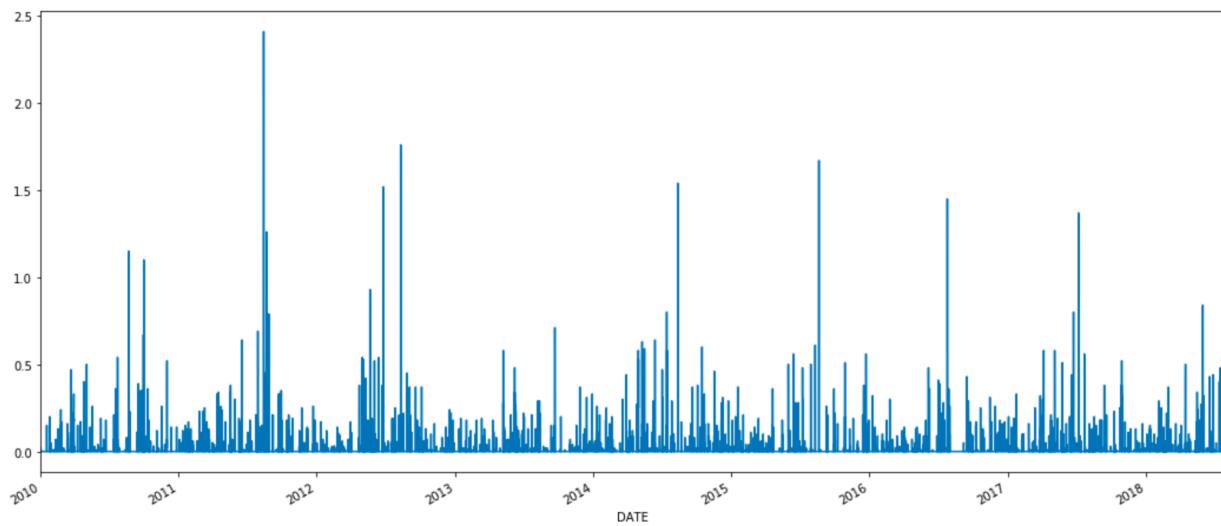
### Feel free to experiment with the code!

<https://stackoverflow.com/questions/23199796/detect-and-exclude-outliers-in-pandas-data-frame>

Once the code is executed with the fix we should see all rain data being positive in the chart.

## Fixing Data

```
[8]: # Can we fix the outliers ????  
data_raw = data_raw[data_raw['precip'] > -500] #Fixing  
plt.figure(figsize=(18,8))  
data_raw["precip"].plot()  
  
ut[8]: <matplotlib.axes._subplots.AxesSubplot at 0x7fba5d2a7be0>
```



## 1.6 Preprocessing Weather data

At the end we want to the data by month with Dry and Rainy days. In the next code sections, we group the data add some additional information and we will drop some columns form out data set.

```
In [5]:
import calendar
#hourly_data_renamed['precip'].sum()
# Assign the "day" to every date entry
#hourly_data_renamed['day'] = hourly_data_renamed.apply(lambda x: x.date())
#len(hourly_data_renamed['relative_humidity']) > 96
#hourly_data_renamed[hourly_data_renamed['rainy'] == True].groupby('day').any()
mydata=data_raw
mydata['day'] = mydata.index
mydata['rain'] = data_raw['relative_humidity'] > 96
mydata['day'] = mydata['day'].apply(lambda x: x.date())
mydata['raintotal'] = mydata.groupby('day')[['precip']].sum()
mydata['month'] = mydata['day'].apply(lambda x: x.month)
mydata['month_name'] = mydata['month'].apply(lambda x: calendar.month_abbr[x])
mydata.head()

Out[5]:
   f  relative_humidity  wind_speed  station_pressure  sea_level_pressure  precip ...  wind_direction_sin  wind_direction_cos  pressure_tendency_incr  pressure_tendency_decr  press
0    92.0          0.0        29.97        29.99  0.01 ...           0.0            1.0             0              1
0    96.0          0.0        29.97        29.99  0.02 ...           0.0            1.0             0              1
0    96.0          0.0        29.97        29.99  0.00 ...           0.0            1.0             0              1
0    96.0          0.0        29.95        29.97  0.00 ...           0.0            1.0             0              1
0    92.0          0.0        29.93        29.96  0.00 ...           0.0            1.0             0              1
```

```
In [12]:
columns_drop = [
    'dew_point_temp_f',
    'sea_level_pressure',
    'altimeter_setting',
    'wind_direction_sin',
    'wind_direction_cos',
    'pressure_tendency_incr',
    'pressure_tendency_decr',
    'pressure_tendency_const'
]
mydata=mydata.drop(columns=columns_drop)
print(len(mydata[mydata.precip >= 0.06]))
#print(len(mydata[mydata.relative_humidity >= 99]))
#mydata['rainh']= if(mydata.relative_humidity > 0.1
#mydata.replace({'rainh': {True: 1, False:0}}, inplace=True)
mydata['rainh']=mydata.precip.apply(lambda x: (x>=0.06),1,0)
mydata["2010-01-17":].head()

Out[12]:
DATE  visibility  dry_bulb_temp_f  wet_bulb_temp_f  relative_humidity  wind_speed  station_pressure  precip
DATE
2010-01-17 00:00:00  2010-01-17 0:00  10.0  33.0  31.0  82.0  7.0  30.12  0.0  2010-C
2010-01-17 01:00:00  2010-01-17 1:00  9.0  34.0  32.0  85.0  7.0  30.11  0.0  2010-C
2010-01-17 02:00:00  2010-01-17 2:00  9.0  31.0  30.0  89.0  0.0  30.09  0.0  2010-C
2010-01-17 03:00:00  2010-01-17 3:00  9.0  32.0  31.0  88.0  0.0  30.10  0.0  2010-C
2010-01-17 04:00:00  2010-01-17 4:00  9.0  33.0  32.0  89.0  0.0  30.11  0.0  2010-C
```

Exercise: Drop the **web\_bulb\_temp\_f** and the **station pressure** columns as well .

## 1.6.1 Group the data by DAY

Here is the data group from hours to days and calculating hours of rain during the day and rain in mm/in.

### Calculating Rain per Day

```
In [26]: #
data=mydata.groupby('day')['relative_humidity'].max().reset_index()
data=data.set_index(data['day'])
data.index.name="index"
data['rain_mm'] = pd.Series(mydata.groupby('day')['precip'].sum(), index=data.index)
data['rain_daily_hours'] = pd.Series(mydata.groupby('day')['rainh'].sum(), index=data.index)
data['temp_f_max']= pd.Series(mydata.groupby('day')['dry_bulb_temp_f'].mean(), index=data.index)
#
#
#
print("Records by day since 2010 - 2018-07 ="+str(len(data)))
print("Total Days with data.relative_humidity >= 96 = "+ str(len(data[data.relative_humidity > 95]))) ## use prec/rain
print("Total Days with data.rain_mm > 0.15 = "+ str(len(data[data.rain_mm >= 0.17])))
#
#
data=data.set_index(data['day'])
data.index = pd.to_datetime(data.index)
data.index.name="index"
data['date'] = pd.to_datetime(data['day'])
data['ym'] = data.date.apply(lambda x: x.strftime('%Y%m'))
#
#
#
data['rainy']= data.rain_mm >= 0.15
data['rainday'] = data.rainy.apply(lambda x: (x>=0.15),1,0)
#
#
#
print("Rainy days in 2017 with data.rain_mm >= 0.15 = "+str(len(data["2017":"2017"][data.rain_mm >= 0.17])))
print("Rainy days in 2016 with data.rain_mm >= 0.15 = "+str(len(data["2016":"2016"][data.rain_mm >= 0.17])))
data["2010-01-17":"2012"].head(10)
```

Records by day since 2010 - 2018-07 =3130  
 Total Days with data.relative\_humidity >= 96 = 488  
 Total Days with data.rain\_mm > 0.15 = 488  
 Rainy days in 2017 with data.rain\_mm >= 0.15 = 58  
 Rainy days in 2016 with data.rain\_mm >= 0.15 = 51

### Exercise:

There are some discrepancies between **rain\_mm** and **relative\_humidity**. Try to get the days of **rain\_mm** and **relative\_humidity** to match up. See above; 488 days in 8 years see to be a good number.

## 1.6.2 Group the data and calculating Dry and Wet days

Creating a couple of dataframes and merging back together to get Wet/Dry days per month:

### Grouping and Calculating Dry and Wet days.

```
[?7]: #Calucate Wet and Dry days
monthly = pd.DataFrame(data.groupby('ym')['rainy'].value_counts())
monthly.columns = ['Days']
monthly.reset_index(inplace = True)
monthly.columns = ['month', 'Rainy', 'Days']
monthly.replace({'Rainy': {True: "Wet", False: "Dry"}}, inplace=True)
monthly.head()

#
#
ndata=monthly.pivot(index ='month', columns ='Rainy')
#
#
ndata=ndata.droplevel(0, axis=1)
ndata.reset_index(inplace = True)
ndata.fillna(0)
dataX=data.merge(ndata, left_on='ym', right_on='month')
dataX.reset_index(inplace = True)

dataX=dataX.set_index(dataX['date'])

dataX.drop(columns='index')
dataX[ "2010-01-17": "2012"].head(10)
```

	index	day	relative_humidity	rain_mm	rain_daily_hours	temp_f_max	date	ym	rainy	rainday	month	Dry	Wet	
t[27]:														
	date													
	2010-01-17	16	2010-01-17	96.0	0.64	5.0	37.250000	2010-01-17	201001	True	True	201001	29.0	2.0
	2010-01-18	17	2010-01-18	89.0	0.02	0.0	42.416667	2010-01-18	201001	False	False	201001	29.0	2.0
	2010-01-19	18	2010-01-19	96.0	0.00	0.0	40.416667	2010-01-19	201001	False	False	201001	29.0	2.0
	2010-01-20	19	2010-01-20	79.0	0.00	0.0	37.791667	2010-01-20	201001	False	False	201001	29.0	2.0
	2010-01-21	20	2010-01-21	75.0	0.00	0.0	31.000000	2010-01-21	201001	False	False	201001	29.0	2.0

Next we check the distribution of Dry /Wet days.

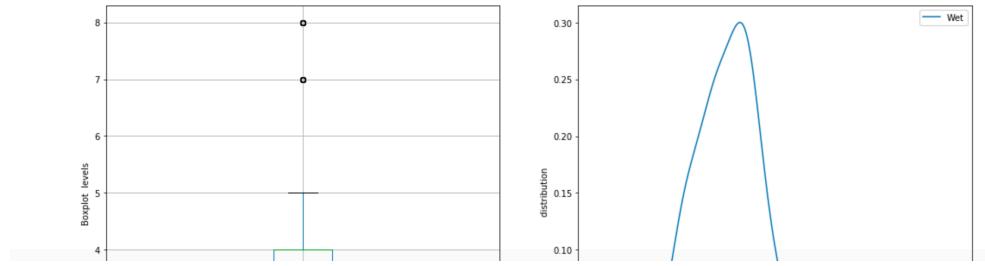
### Check Distribution

```
[30]: import numpy as np
import matplotlib.pyplot as plt
def plotBoxNdendity(data,col=None):
    if col in data.columns:
        plt.figure(figsize=(18,8))

        ax1 = plt.subplot(121)
        data.boxplot(col,ax=ax1)
        ax1.set_ylabel('Boxplot levels', fontsize=10)

        ax2 = plt.subplot(122)
        data[col].plot(ax=ax2,legend=True,kindle='density')
        ax2.set_ylabel('distribution', fontsize=10)

    else:
        print("Column not in the data")
plotBoxNdendity(dataX[ '2010' ],'Wet')
#plotBoxNdendity(dataWeekly,'rainday')
```



## Exercise: Try multiple plots with different years.

Like

```
plotBoxNdendity(dataX[ : '2012' ],'Wet')
plotBoxNdendity(dataX[ : '2014' ],'Wet')
```

### 1.6.3 Sampling Monthly data

Here we resample the daily / hourly data to a month, so we get a nice small dataset.

#### Creating Monthly Data

```
?3]: #data.index = pd.to_datetime(data.index, unit='s')
#data['date']=pd.to_datetime(data['date'], infer_datetime_format=True)
#data.index = pd.to_datetime(data.index)
dataM=dataX.resample('M').max(); # monthly
dataM[ 'year' ] = dataM[ 'day' ].apply(lambda x: x.year)
dataM[ 'month' ] = dataM[ 'day' ].apply(lambda x: x.month)
dataM[ "month_name" ] = dataM[ 'month' ].apply(lambda x: calendar.month_abbr[x])
dataM[ "temp_c_max" ] = round((dataM[ "temp_f_max" ] - 32) * 5/9,2)
dataM=dataM.drop(columns = "relative_humidity")
dataM=dataM.drop(columns = "index")
dataM=dataM.drop(columns = "day")
dataM=dataM.drop(columns = "rain_mm")
dataM=dataM.drop(columns = "rain_daily_hours")
dataM=dataM.drop(columns = "rainday")
dataM=dataM.drop(columns = "ym")
dataM=dataM.drop(columns = "temp_f_max")
dataM=dataM.drop(columns = "rainy")
dataM=dataM.drop(columns = "month")
#dataM=dataM.drop(columns = "temp_c_max")
dataM=dataM.drop(columns = "date")

#dataM[dataM.relative_humidity >= 98].head()
dataM[ '2017-01':].head()
```

```
t[23]:      Dry  Wet  year month_name temp_c_max
          date
2017-01-31  22.0   9.0  2017        Jan     10.51
2017-02-28  25.0   3.0  2017        Feb     14.05
2017-03-31  22.0   5.0  2017        ..     16.10
```

#### #Exercise:

Try adding or leaving out some columns, like average temperature (`mean()`) or temperature in Fahrenheit

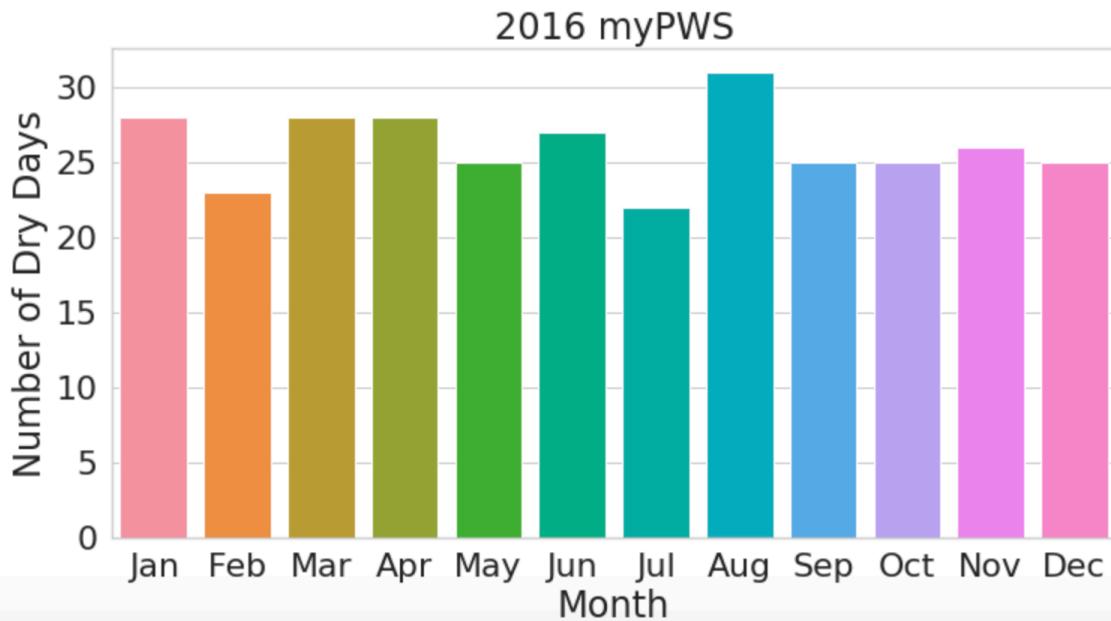
## 1.6.4 Plot the Monthly data.

### Plot the Monthly sample data

```
37]: import matplotlib.pyplot as plt
      import seaborn as sns
      # Monthly plot of rainy days
      plt.figure(figsize=(12,6))
      sns.set_style("whitegrid")
      sns.set_context("notebook", font_scale=2)
      sns.barplot(x="month_name", y="Dry", data=dataM["2016-01":"2016-12"])

      plt.xlabel("Month")
      plt.ylabel("Number of Dry Days")
      plt.title("2016 {}".format(station))
      #Exercise display different years / data Elements

      it[37]: Text(0.5, 1.0, '2016 myPWS')
```



### Exercise

Display different years / data Elements like temperature

## 1.7 Playground for Query testing

Here we will work on the data queries, which we will send via commands from Node-RED, to get back the data. From Node-RED we will send a command call getInfo which should give us information about the data size — maybe the amount of years we have in the dataset. Take a look at the different function examples and execute them.

```
In [49]: # GetInfo
dataY = dataM.resample('Y').mean(); # yearly averages
dataY.year.count()
getInfo = '{"cmd": "getInfo", "tablesize": "' + str(len(dataM)) + '", "years": "' + str(dataY.year.count()) + '"}'; # yearly av
print(getInfo)

{"cmd": "getInfo", "tablesize": 103, "years": 9}

In [48]: #getJan2017
datain = {'cmd' : 'getInfo'}
dry=dataM[ "2017": "2017"].loc[dataM[ 'month_name'] == 'Apr'].values[0][0]
wet=dataM[ "2017": "2017"].loc[dataM[ 'month_name'] == 'Apr'].values[0][1]
yy=dataM[ "2017": "2017"].loc[dataM[ 'month_name'] == 'Apr'].values[0][2]
mm=dataM[ "2017": "2017"].loc[dataM[ 'month_name'] == 'Apr'].values[0][3]
temp=dataM[ "2017": "2017"].loc[dataM[ 'month_name'] == 'Apr'].values[0][4]
mvk = { 'Wet': wet, 'Dry' : dry, "Month" : mm , "Year":yy}
mvk[ "cmd"] = datain[ "cmd"]
mvk

Out[48]: {'Wet': 6.0, 'Dry': 24.0, 'Month': 'Apr', 'Year': 2017, 'cmd': 'getInfo'}

In [50]: #Get A range
sel= dataM["2017-01":"2017-03"]
mvk=sel.reset_index().to_json(orient='records')
getInfo={"cmd": "cmd", "info": []}
info=json.loads(mvk)
getInfo[ 'info']=info
getInfo[ "cmd"] = datain[ "cmd"]
print(getInfo)

{'cmd': 'getInfo', 'info': [{ 'date': 1485820800000, 'Dry': 22.0, 'Wet': 9.0, 'year': 2017, 'month_name': 'Ja
3.0, 'year': 2017, 'month_name': 'Feb', 'temp_c_max': 14.05}, { 'date': 1490918400000, 'Dry': 26.0, 'Wet': 5.

In [54]: #GetRain
-----
```

Above data example are the dataset/json we will send back to Node-RED based on the query/command. We will need to implement these functions in the next sections.

## 2 Connecting to Node-RED

In the next section we will install some websocket packages and adjust the URL to our Node-RED instance.

We need to install the `websocket-client` client via a unix command like:

```
!pip install websocket-client
```

Just uncomment the command and execute it.



### Connecting to Node-RED

```
In [26]: #Create commincation with Node-RED instance
#You need to instal the client 1st
!pip install websocket-client
import websocket
import _thread
import time
import json

def on_open(ws):
    print("on open")
    ws.send(mysdata.loc[mysdata['month_name'] == 'Jan'])
    def run(*args):
        for i in range(10000):
            hbeat = '{"cmd":"Python NB HeartBeat"}'
            print("send cmd")
            ws.send(hbeat)
            time.sleep(1000)

    _thread.start_new_thread(run, ())

# ws.send("Watson Studio Listen open")
ws.on_open = on_open
ws.run_forever()

start_websocket_listener()
```

Once executed you will see some output at the bottom to of the cell —see below

```
# ws.send("Watson Studio Listen open")
ws.on_open = on_open
ws.run_forever()

start_websocket_listener()

Collecting websocket-client
  Downloading https://files.pythonhosted.org/packages/4c/5f/f61b420143ed1c8dc69f9eaec5ff1ac361
  |██████████| 204kB 9.3MB/s eta 0:00:01
Requirement already satisfied: six in /opt/conda/envs/Python36/lib/python3.6/site-packages (fr
Installing collected packages: websocket-client
Successfully installed websocket-client-0.57.0
connecting
on open
```

Once you executed the code - comment the line again . Like `#!pip install websocket-client`

Adjust the line below with your Node-RED instance, such as: thinklab???.mybluemix.net/ws/myweather/

```

69         getInfo={"cmd": "cmd", "info":[]}
70         info=json.loads(mvk)
71         getInfo['info']=info
72         getInfo["cmd"]=datain['cmd']
73         print(getInfo)
74         ws.send(json.dumps(getInfo))
75
76     if (datain['cmd'] == 'getTemp'):
77         yy=datain['year']
78         mm=datain['month']
79         temp=dataM[yy:yy].loc[dataM['month_name'] == mm].values[0][4]
80         getInfo = {'Temp' :temp, "Month" : mm , "Year":yy}
81         getInfo["cmd"]=datain['cmd']
82         print(getInfo)
83         ws.send(json.dumps(getInfo))
84     if (datain['cmd'] == 'getAll'):
85         mysdata.to_json()
86
87
88 except:
89     print("Error no json / no valid command")
90 ##### ws://thinklab2020nr.mybluemix.net/ws/myweather/
91 ##### use your own instance
92 def start_websocket_listener():
93     #websocket.enableTrace(True)
94     ws = websocket.WebSocketApp("ws://thinklab1239.mybluemix.net/ws/myweather/", #<<<<< ADJUST
95                                     on_message = on_message,
96                                     on_error = on_error,
97                                     on_close = on_close)
98     print("connecting")
99     # ws.send("Watson Studio Listen open")
100    ws.on_open = on_open
101    ws.run_forever()
102
103

```

When you have adjusted the cell with your instance name, execute the cell again  
You should see at the bottom of the cell some text like:

```

99     # ws.send("Watson Studio Listen open")
100    ws.on_open = on_open
101    ws.run_forever()
102
103 start_websocket_listener()

connecting
on open

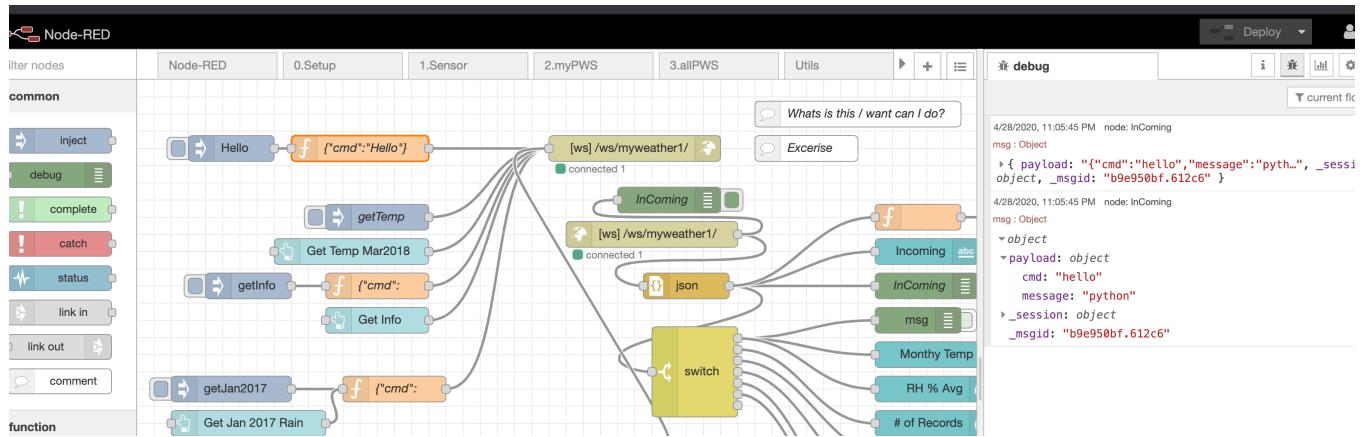
```

In [ ]: 1 **### Start with Part Two**

Once you see this, let's switch back to Node-RED and the Weather History Tab  
*Note: Keep WatsonStudio open. We will switch between Node-RED our Dashboard and the Watson studio.*

## 3 Sending a command to Python

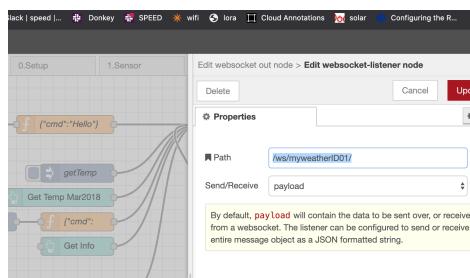
Back in Node-RED, click on the Weather history tab. You should see a green info message on the websocket connection with a message like “connected 1”. If this is the case, send a hello message and see if you receive a hello back, and check the message in Node-RED



*Note: If everything is working you should see as above in the debug console.*

### Troubleshooting:

1. Check the hostname in the Python Notebook should be ws://thinklab???.mybluemix.net/ws/myweather/ With your lab ID instead of ??
2. If you cannot get a connection change the websocket endpoint in the Notebook and Node-RED



Make sure it's the same on both sides. You will need to stop and restart the cell in the Notebook

3. The Notebook websocket might be closed, in which case you must restart the cell.

```
hello
{"cmd": "hello", "message": "python"}

closed
```

[27]: **## Start with Part Two**

In the Notebook cell you should see message at the bottom like.

```
ws.on_open = on_open
ws.run_forever()

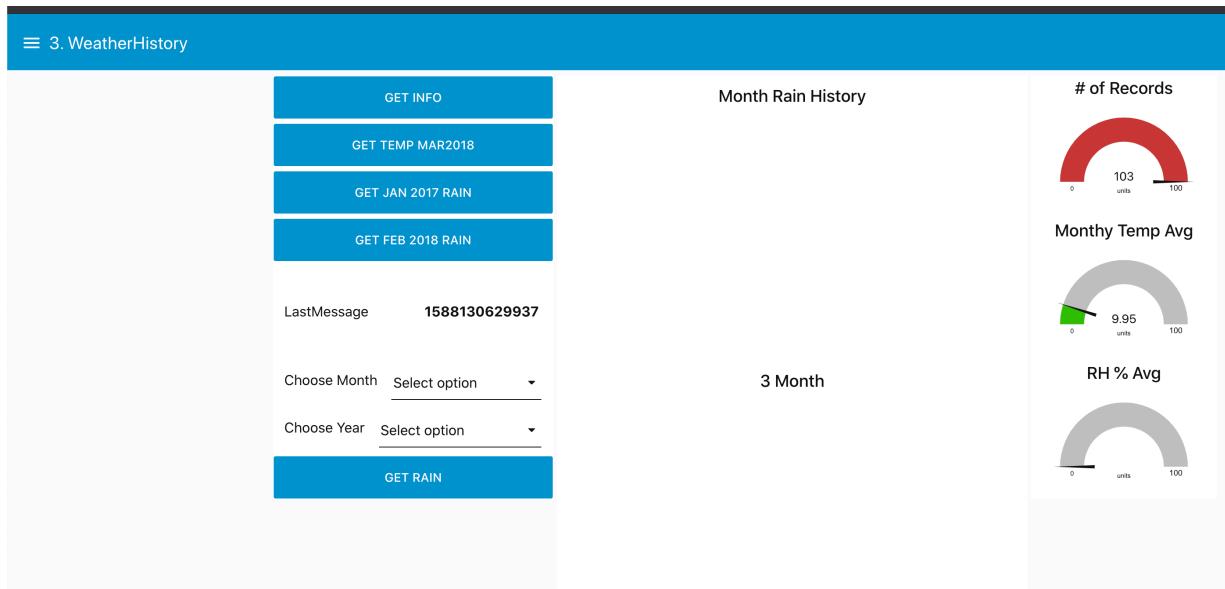
start_websocket_listener()

connecting
on open
{"cmd": "hello"}
hello
{"cmd": "hello", "message": "python"}
{"cmd": "hello"}
hello
{"cmd": "hello", "message": "python"}
{"cmd": "hello"}
hello
{"cmd": "hello", "message": "python"}  
{"cmd": "hello"}  
hello
{"cmd": "hello", "message": "python"}
```

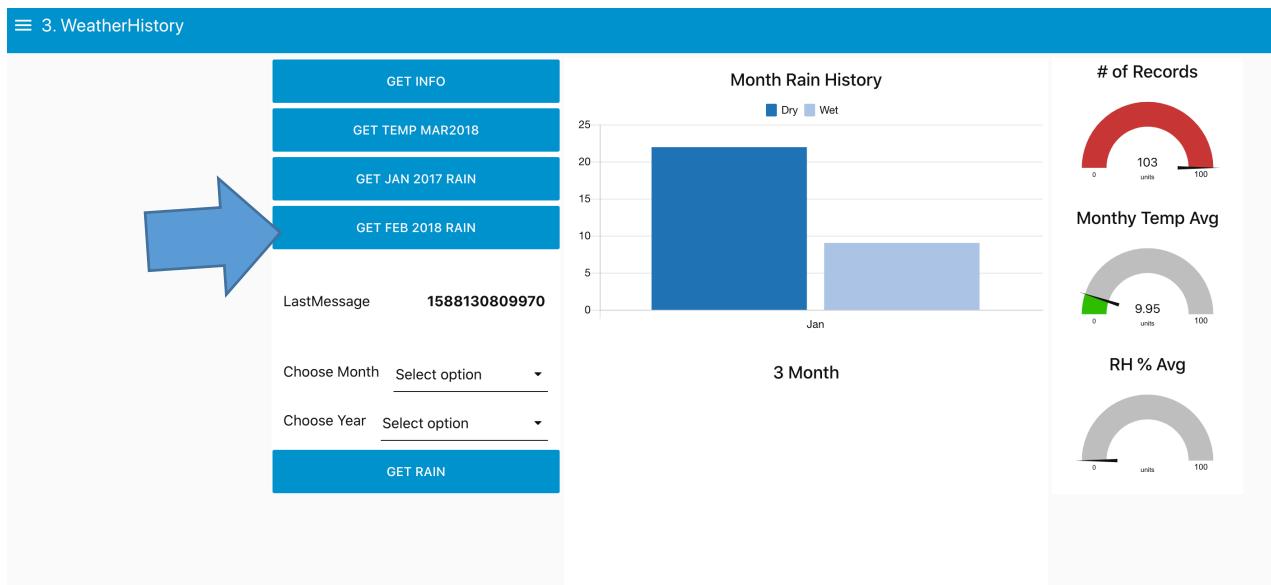
```
[27]: ### Start with Part Two
```

## 4 Using the Dashboard and data Queries

Once we got all the connections working let's look at our Dashboard.  
Select the 3. WeatherHistory Menu



Click on the Jan and Feb Rain buttons to retrieve the data from your Notebook. You should get some data back which is displayed.



Flip back and check the message on Node-RED and NoteBook.

## 4.1 Adding more functionality

We now have functionality to get Temperature in some Months with Dry and Wet Days. We need to add more flexible function to get Rain Data from Notebook. We also need to add some more data such as years and months to the dashboard drop-downs.

### Exercise

- Add more Month and Years to the Drop down on the Dashboard.

The screenshot shows a Node-RED flow and its configuration dialog for a dropdown node. The flow consists of several nodes: a 'getJan2017' node, a 'Get Jan 2017 Rain' node, a 'getRain2018' node, a 'Get Feb 2018 Rain' node, a 'Choose Month' node, a 'Choose Year' node, a 'Get Rain' node, a 'Test' node, and a 'getRain' node. The 'Choose Month' and 'Choose Year' nodes are connected to each other and to the 'Get Rain' node via 'set flow' nodes. A blue arrow points from the 'Edit dropdown node' dialog to the 'Options' section, which displays a dropdown menu with options for 2010, 2012, and 2017.

- Fix the code in the Notebook to return the data base on the drop down selection ... see getRain command.

```

ws.send(getInfo)

if (datain['cmd'] == 'getJan2017'):
    dry=dataM["2017":"2017"].loc[dataM['month_name'] == 'Jan'].values[0][0]
    wet=dataM["2017":"2017"].loc[dataM['month_name'] == 'Jan'].values[0][1]
    yy=dataM["2017":"2017"].loc[dataM['month_name'] == 'Jan'].values[0][2]
    mm=dataM["2017":"2017"].loc[dataM['month_name'] == 'Jan'].values[0][3]
    getInfo = {'Wet': wet,'Dry' : dry,"Month" : mm , "Year":yy}
    getInfo["cmd"]的数据['cmd']
    print(getInfo)
    ws.send(json.dumps(getInfo))

#Get Rain for a month Month in coming {"cmd": "getRain", "month": "Feb", "year": "2018"}
if (datain['cmd'] == 'getRain'):
#
# Excercise Change here see above function
#
    getInfo = {'Wet': wet,'Dry' : dry,"Month" : "FixCode" , "Year": "FixCode"}
    getInfo["cmd"]的数据['cmd']
    print(getInfo)
    ws.send(json.dumps(getInfo))

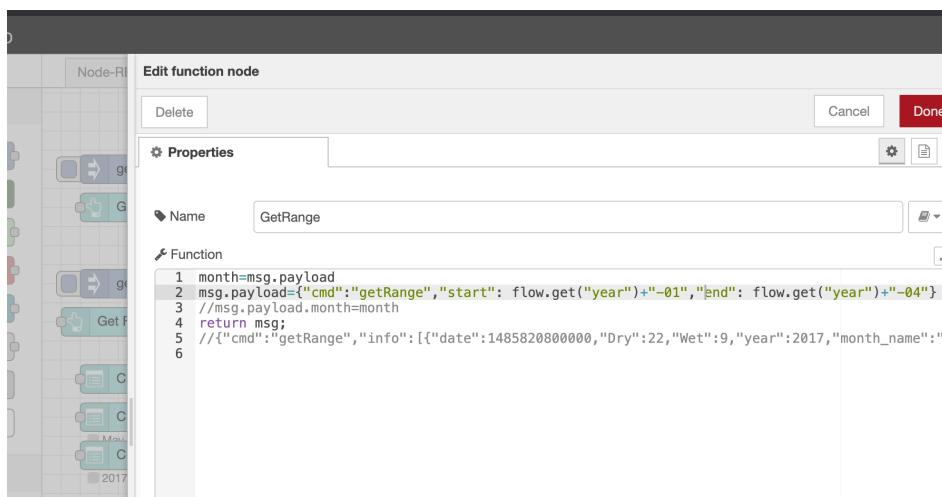
```

A blue arrow points from the code block to the line '#Excercise Change here see above function'.

(see next page)

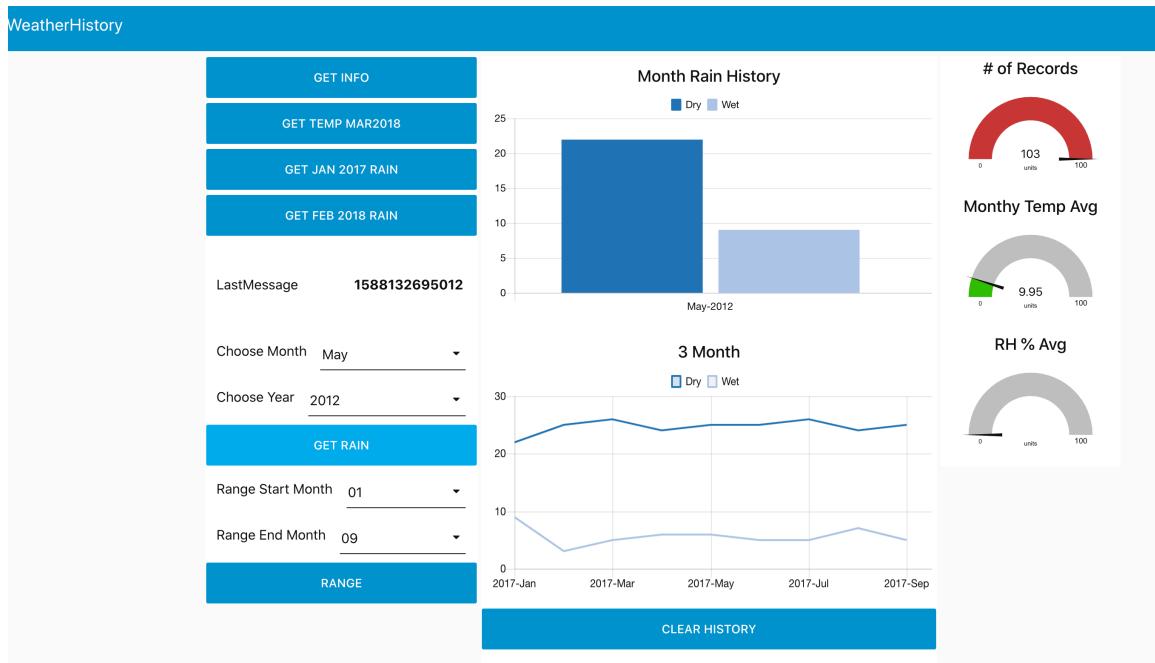
```
#Get Rain for a Month in coming
{"cmd": "getRain", "month": "Feb", "year": "2018"}
if (datain['cmd'] == 'getRain'):
    yy=datain['year']
    mm=datain['month']
    dry=dataM[yy:yy].loc[dataM['month_name'] ==
mm].values[0][0]
    wet=dataM[yy:yy].loc[dataM['month_name'] ==
mm].values[0][1]
    getInfo = {'Wet': wet, 'Dry' : dry, "Month" : mm , "Year":yy}
    getInfo["cmd"]=datain['cmd']
    print(getInfo)
    ws.send(json.dumps(getInfo))
```

- The getRange command is hard coded in Node-RED . I always retrieve Jan to March. Add a other dropdown and add the range flow var to the code in Node-RED



```
msg.payload={"cmd": "getRange", "start": flow.get("year")+-01", "end": flow.get("year")+-04}
//msg.payload.month=month
return msg;
//{"cmd": "getRange", "info": [{"date": 1485820800000, "Dry": 22, "Wet": 9, "year": 2017, "month_name": "Jan"}]}
```

You should see something like this if everything is working



**Please note:** the above exercises are not required for the next part of the lab.  
Feel free to move on without coding.

In Part Three of the LAB we will work with Watson Studio so please make sure to sign up for an free account. [Register for WatsonStudio](#)  
<https://dataplatform.cloud.ibm.com/registration/stepone>

Get the instructions for LAB Part Three here  
<https://github.com/markusvankempen/ThinkLab1239/tree/master/instructions>

For more details got to the github

<https://github.com/markusvankempen/ThinkLab1239>

Cheers  
Markus van Kempen  
[mvk@ca.ibm.com](mailto:mvk@ca.ibm.com)  
Version:20200428