Lab Center – Hands-On Lab – Part THREE

Forecasting with SARIMAX

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

20200504

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.

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

1 Objective

In this part of the LAB we will set up a WatsonStudio Instance with a Jupyter Notebook to forecast Rainy and Dry days. We will create some queries which we trigger from Node-RED and display on the Node-RED Dashboard.

***Note**: We will walk through Part Three together first before you start.*

1.1 Pre-Requisite

Please sign up for a free WatsonStudio/IBM Cloud account Register for WatsonStudio
https://dataplatform.cloud.ibm.com/registration/stepone

Download the github to your VM/Desktop
https://github.com/markusvankempen/ThinkLab1239

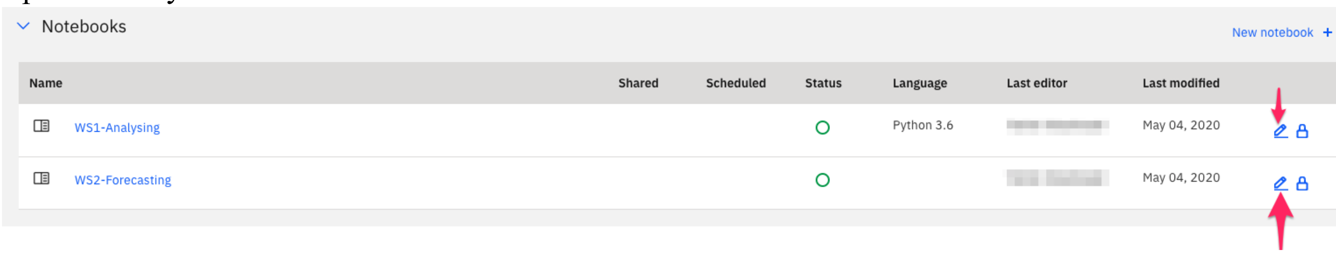
1.2 Setup WatsonStudio

If you worked through the LAB Part One already you should have Watson Studio already set up and can skip this step and just import the Python notebook (LabPart Three / Notebook #2).
If you have not set up your WatsonStudio instance yet follow the instruction for LAB Part two 1st.

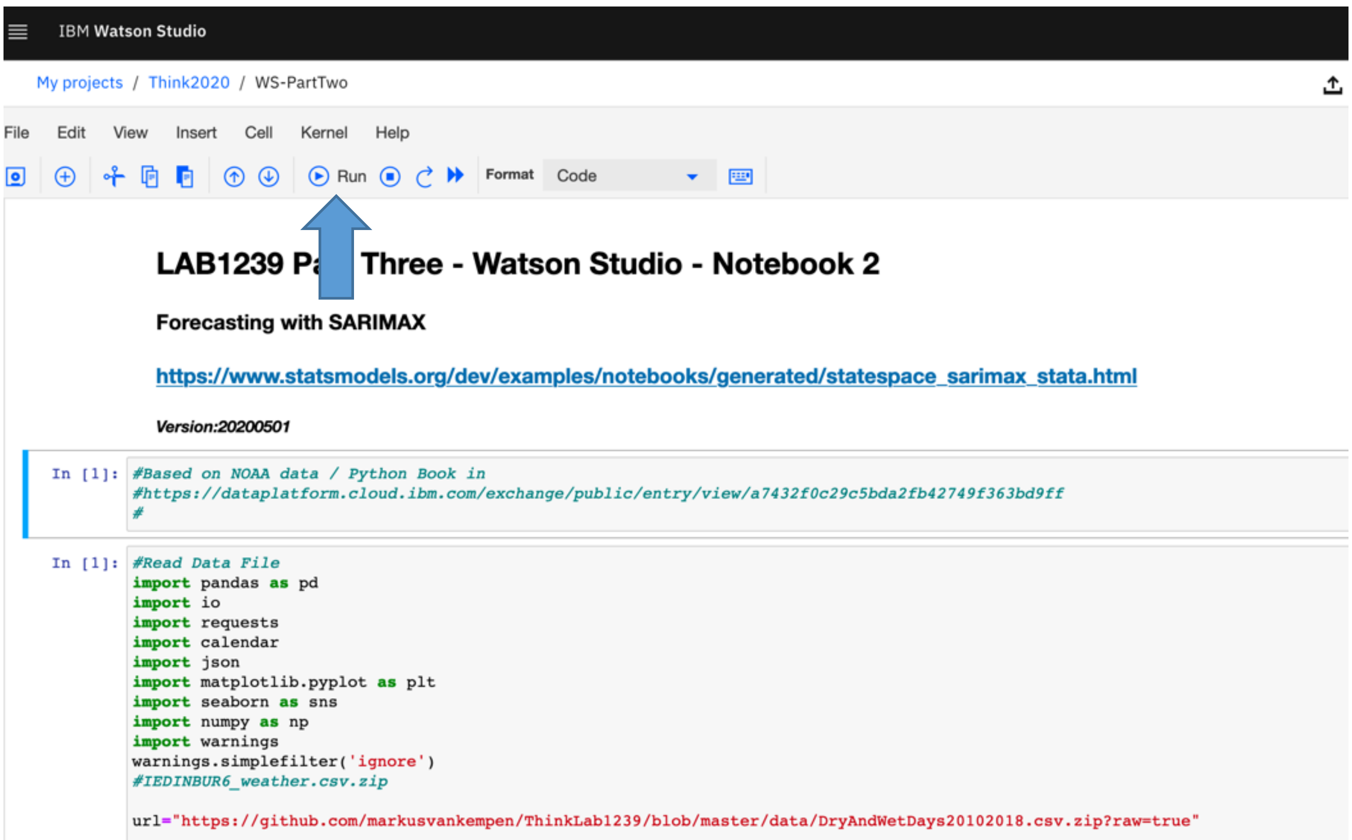
Tip: use WatsonStudio with FireFox and Node-RED / LAB instructions in Chrome so you can switch back and forth between the environments easier.

1.3 Forecasting using a Python Notebook

Upload the Python Notebook and click Edit.

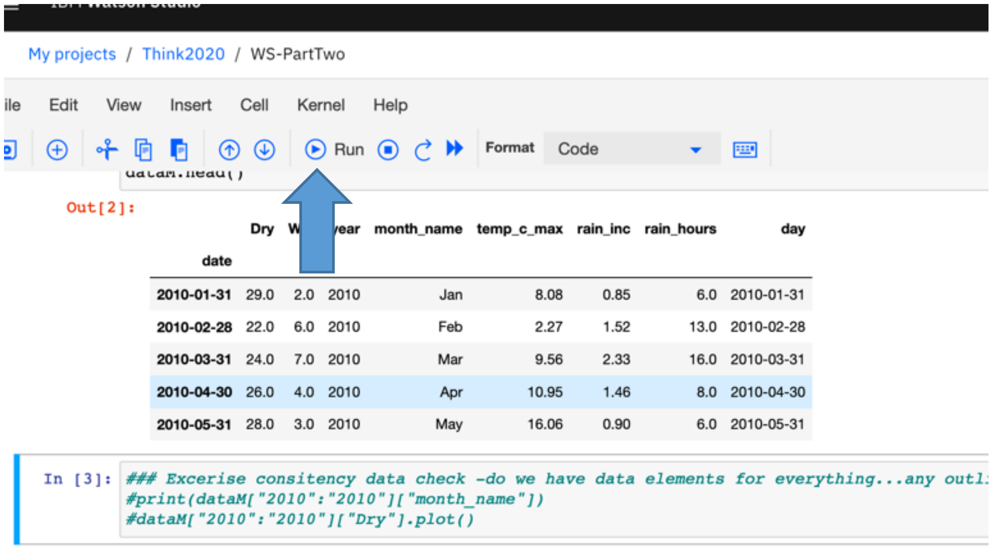


You should see the following screen



The notebook will read the cleaned monthly weather data file form the previous LAB from the github. See https://github.com/markusvankempen/ThinkLab1239/blob/master/data/DryAndWetDays20102018.csv.zip

Put the notebook into Edit mode and step thru it via the Run button.



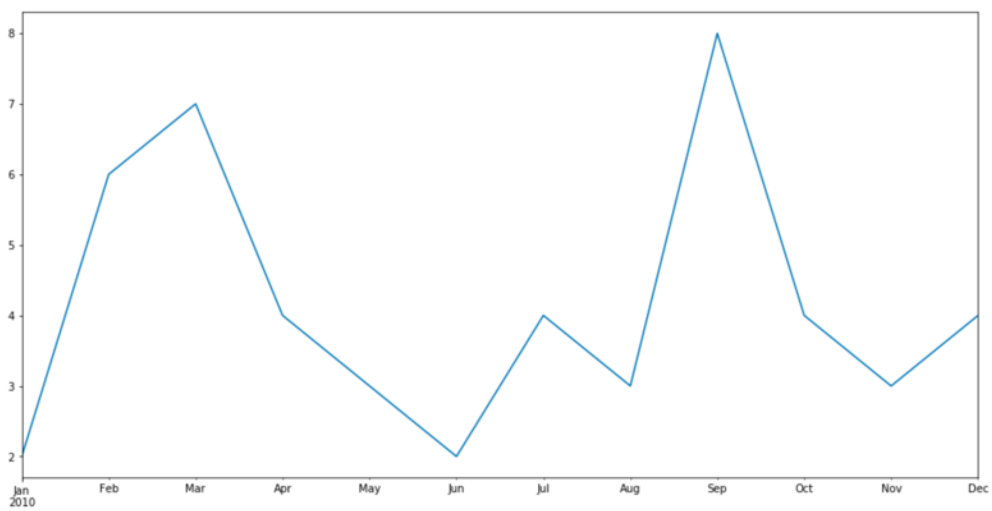
date	Dry	W	year	month_name	temp_c_max	rain_inc	rain_hours	day
2010-01-31	29.0	2.0	2010	Jan	8.08	0.85	6.0	2010-01-31
2010-02-28	22.0	6.0	2010	Feb	2.27	1.52	13.0	2010-02-28
2010-03-31	24.0	7.0	2010	Mar	9.56	2.33	16.0	2010-03-31
2010-04-30	26.0	4.0	2010	Apr	10.95	1.46	8.0	2010-04-30
2010-05-31	28.0	3.0	2010	May	16.06	0.90	6.0	2010-05-31

```
### Excerise consitency data check -do we have data elements for everything...any outl
#print(dataM["2010":"2010"]["month_name"])
#dataM["2010":"2010"]["Dry"].plot()
```

When you have parsed the data, you can display/plot some of the data elements to make sure we have no outliers or odd data spikes.

```
### Excerise consitency data check -do we have data elements for everything...any outliers
#print(dataM["2010":"2010"]["month_name"])
plt.figure(figsize=(16,8))
dataM["2010":"2010"]["Wet"].plot()
```

```
[6]: <matplotlib.axes._subplots.AxesSubplot at 0x7fa823a3ef60>
```



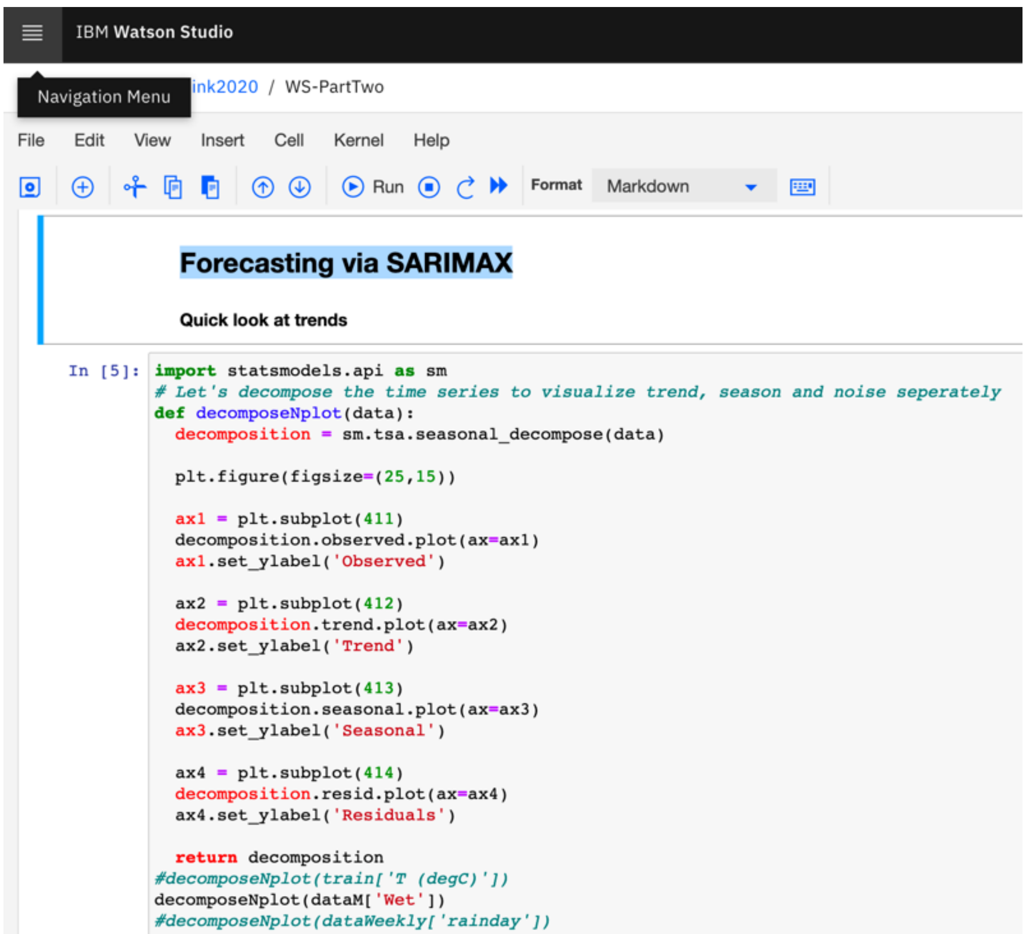
This looks ok. Less rainy days in the summer? Try the “Dry” data element.

1.4 Forecasting via SARIMAX

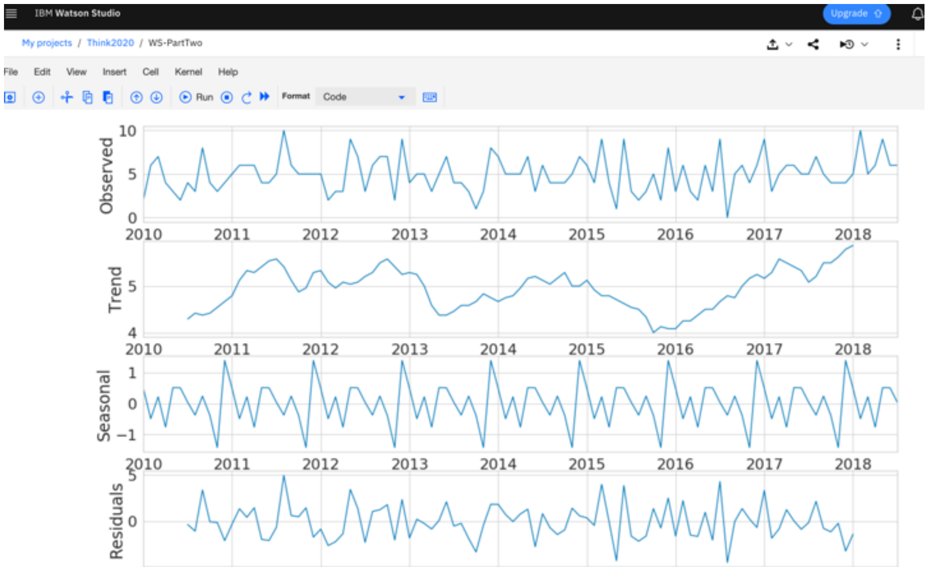
For forecasting we using a SARIMAX as a model for our seasonal data to predict the Dry weather days. You can read up on the ARIMA and SARIMAX here — lots of math ;)
https://www.statsmodels.org/dev/examples/notebooks/generated/statespace_sarimax_stata.html

In the LAB we will not go through the details of modeling. There are lot of other model approaches especially for the time series data set, but for the simplicity we just use ARIMA / SARIMAX in this LAB. You can use Watson AI as well create a model which can deployed and called via APIs. See here
https://dataplatform.cloud.ibm.com/docs/content/wsj/analyze-data/autoai-overview.html
Check also the instruction folder form more excerise around AutoAI for extra credit. In this LAB we kept the approach simple and generic so it can be used in different Python Runtime environments.

The 1st step to create a model is to check if we have a trend and consistency and to see if we have seasonal data.

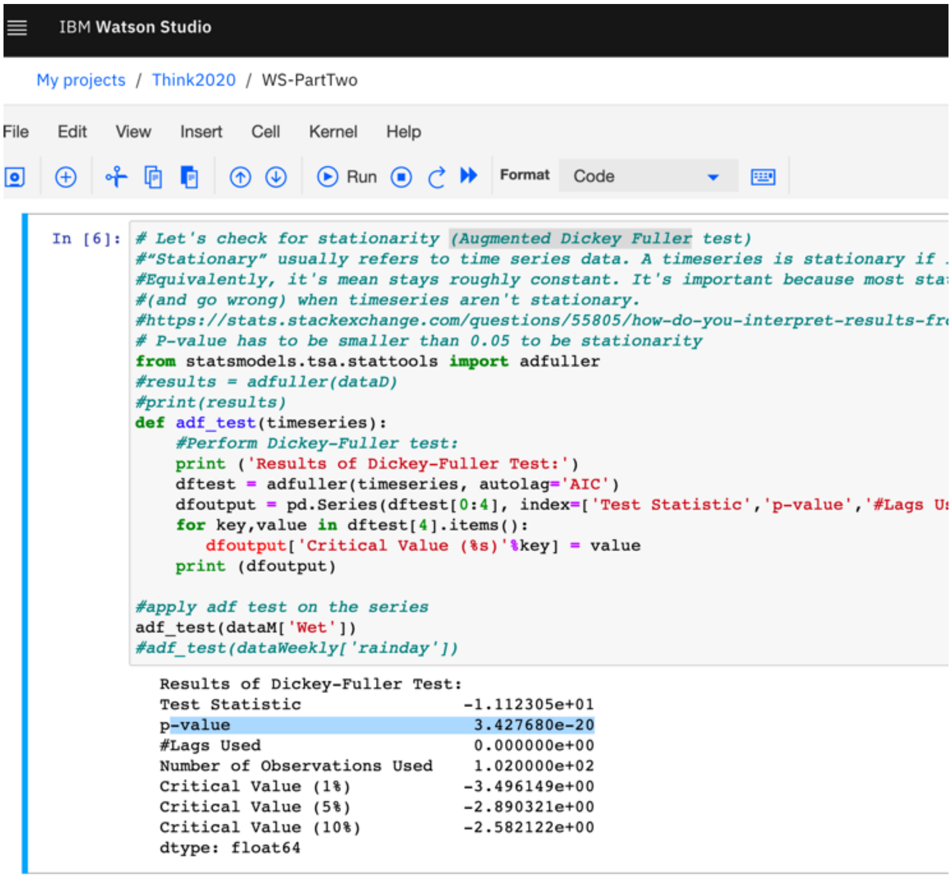


Based on the plot below and of course based on our previous data analyzing exercise we can see that the data is seasonal.



We know can further test the data to see if it is “stationarity”. We using a Augmented Dickey Fuller test. https://en.wikipedia.org/wiki/Augmented_Dickey%E2%80%93Fuller_test

Again, a lot of math there but the key thing for us is that the p-value is smaller than 0.05




```
# Let's check for stationarity (Augmented Dickey Fuller test)
#"Stationary" usually refers to time series data. A timeseries is stationary if .
#Equivalently, it's mean stays roughly constant. It's important because most sta
#(and go wrong) when timeseries aren't stationary.
#https://stats.stackexchange.com/questions/55805/how-do-you-interpret-results-fr
# P-value has to be smaller than 0.05 to be stationarity
from statsmodels.tsa.stattools import adfuller
#results = adfuller(dataD)
#print(results)
def adf_test(timeseries):
    #Perform Dickey-Fuller test:
    print ('Results of Dickey-Fuller Test:')
    dftest = adfuller(timeseries, autolag='AIC')
    dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','#Lags Us
    for key,value in dftest[4].items():
       dfoutput['Critical Value (%s)'%key] = value
    print (dfoutput)

#apply adf test on the series
adf_test(dataM['Wet'])
#adf_test(dataWeekly['rainday'])
```

```
Results of Dickey-Fuller Test:
Test Statistic                -1.112305e+01
p-value                        3.427680e-20
#Lags Used                     0.000000e+00
Number of Observations Used    1.020000e+02
Critical Value (1%)           -3.496149e+00
Critical Value (5%)           -2.890321e+00
Critical Value (10%)          -2.582122e+00
dtype: float64
```

1.5 Creating the "Model"

Next we create the Model to predict the Dry weather days. We will divide the data up in Training and Test data and then execute the modeling function. There is a lot of information as a result.

My projects / Think2020 / WS-PartTwo

File Edit View Insert Cell Kernel Help

Run Format Code

Create SARIMAX Model

In [7]:

```
## Resampling the data to mothly and averaging out the temperature & we will predict the mont
train = dataM[:'2016']
test =  dataM['2016':]
alldata=dataM[:'2018-03']['Dry']
#alldata=dataM[:'2017-07']['Dry']
ftraindata = train['Dry']
ftestdata = test['Dry']

#dataWeekly['rainday']
#train = dataWeekly['2015':'2016']
#test =  dataWeekly['2016':]
#alldata=dataM['dataWeekly']
#alldata=dataWeekly['2015':'2018-04']['rainday']
#ftraindata = train['rainday']
#ftestdata = test['rainday']

from statsmodels.tsa.statespace.sarimax import SARIMAX
#endog.index = endog.datetime
#
#endog = endog.drop(['T (degC)'])

#exog.index = exog.dates
#exog = exog.drop(['dates'], axis = 1)
#Weekly
#model = SARIMAX(alldata,order=(2,0,2),seasonal_order=(1,1,0,52),trend='ct', freq='W')#,endog
#Monthly
model = SARIMAX(alldata,order=(1, 0, 1),seasonal_order=(1, 1, 0, 12),trend='n',  freq='M',enf
#model = SARIMAX(dataM['Dry'],order=(2,0,2),trend='n',freq='M',enforce_stationarity=True)
#(2, 0, 2)xh(1, 1, 0, 52)
results = model.fit()

print(np.mean(np.abs(results.resid)))

results.summary()
```

5.011193571004436

Out[7]:

5.011193571004436

Out[7]:

Statespace Model Results

Dep. Variable:	Dry	**No. Observations:**	99
Model:	SARIMAX(1, 0, 1)x(1, 1, 0, 12)	**Log Likelihood**	-206.418
Date:	Fri, 01 May 2020	**AIC**	420.836
Time:	03:41:29	**BIC**	430.700
Sample:	01-31-2010	**HQIC**	424.808
	- 03-31-2018		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.8886	0.117	7.587	0.000	0.659	1.118
ma.L1	-0.9998	7.527	-0.133	0.894	-15.752	13.752
ar.S.L12	-0.5579	0.121	-4.626	0.000	-0.794	-0.322
sigma2	6.2280	46.795	0.133	0.894	-85.488	97.944

Ljung-Box (Q):	29.65	**Jarque-Bera (JB):**	0.57
Prob(Q):	0.88	**Prob(JB):**	0.75
Heteroskedasticity (H):	1.14	**Skew:**	-0.12
Prob(H) (two-sided):	0.72	**Kurtosis:**	2.69

Again lots of math which you can learn more about here (https://www.machinelearningplus.com/time-series/arima-model-time-series-forecasting-python/) . The interesting value we looking for is MAE
The Mean Absolute Error (MAE) see here for more information
https://medium.com/@ewuramaminka/mean-absolute-error-mae-machine-learning-ml-b9b4afc63077

The MEA basically tells us how good our model is. In our case the prediction could be 5days off ☹

Reason for this is that we used monthly data if we get more data for example using weekly or daily data and work with the SARIMAX / ARIMA parameters more we would get much better results. See the Reference Section for more information

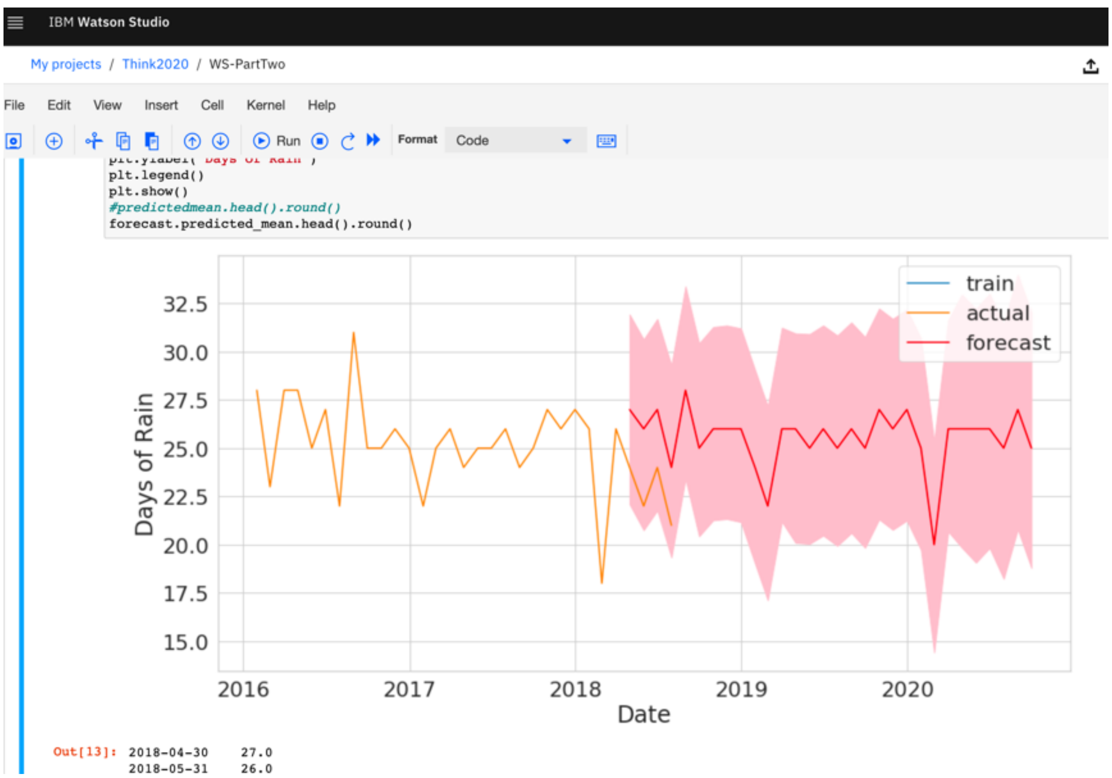
1.6 Visualizing the Model

Once create the model and execute the prediction functions like:
#STEPS = Month   ex., 2 years

```
forecast = results.get_forecast(steps=30)#len(ftestdata))
forecast.predicted_mean.head(10).round()
```

We can use our Test, Training and forecast data to visualize the result. See below.

#Exercise: Try using different date ranges for the plot, such as 2018



.

1.7 Merging Forecast with the History data

Once we have the forecast data, we merge the data with some of the history and calculate the Rainy days.

IBM Watson Studio

My projects / Think2020 / WS-PartTwo

File Edit View Insert Cell Kernel Help

Run Format Code

```
yy="2020"
mm="Jun"
dry=dataF[yy:yy].loc[dataF['month_name'] == mm].values[0][0]
wet=dataF[yy:yy].loc[dataF['month_name'] == mm].values[0][1]
getInfo = {'Wet': wet,'Dry' : dry,"Month" : mm ,"Year":yy}
print(getInfo)
print(dataF['2020':'2020'].loc[dataF['month_name'] == "Jun"].values[0][6
#dataF['2020':'2020'].loc[dataF['month_name'] == "Jan"]
dataF['2020':]
#sel= dataF["2020-01":"2020-03"]
#sel.reset_index().to_json(orient='records')
```

```
{'Wet': 4.0, 'Dry': 26.0, 'Month': 'Jun', 'Year': '2020'}
Jun
```

Out[11]:

	Dry	Wet	date	year	month	days	month_name
2020-01-31	25.0	6.0	2020-01-31	2020	1	31	Jan
2020-02-29	20.0	9.0	2020-02-29	2020	2	29	Feb
2020-03-31	26.0	5.0	2020-03-31	2020	3	31	Mar
2020-04-30	26.0	4.0	2020-04-30	2020	4	30	Apr
2020-05-31	26.0	5.0	2020-05-31	2020	5	31	May
2020-06-30	26.0	4.0	2020-06-30	2020	6	30	Jun
2020-07-31	25.0	6.0	2020-07-31	2020	7	31	Jul
2020-08-31	27.0	4.0	2020-08-31	2020	8	31	Aug
2020-09-30	25.0	5.0	2020-09-30	2020	9	30	Sep

Note: That we have 2 dataset dataM (historic data) and dataF (forecast) .

1.8 Query Playground

In the previous LAB where we analyzed the data and created queries for Node-RED and we will do the same here. Simply go through the different cells and explore the different queries, Feel free to add/change data elements.

IBM Watson Studio

My projects / Think2020 / WS-PartTwo

File Edit View Insert Cell Kernel Help

Run Format Code

Query Playgroud

In [12]:
```
# GetInfo
dataY = dataF.resample('Y').mean(); # yearly averages
dataY.year.count()
getInfo = '{"cmd":"getInfo","tablesize":'+str(len(dataM))+',"years":'+str(dataY.year.count())+'}'; # yearly

print(getInfo)
```

```
{"cmd":"getInfo","tablesize":103,"years":3}
```

In [13]:
```
#getMay2020
datain = {'cmd' : 'getMay2020'}
dry=dataF["2020":].loc[dataF['month_name'] == 'Apr'].values[0][0] #fDry
wet=dataF["2020":].loc[dataF['month_name'] == 'Apr'].values[0][1]
yy= dataF["2020":].loc[dataF['month_name'] == 'Apr'].values[0][3]
mm= dataF["2020":].loc[dataF['month_name'] == 'Apr'].values[0][6]
#temp=dataF["2020":].loc[dataF['month_name'] == 'Apr'].values[0][4]
mvk = {'Wet': wet,'Dry' : dry,"Month" : mm ,"Year":yy}
mvk["cmd"]=datain['cmd']
mvk
```

Out[13]:
```
{'Wet': 4.0, 'Dry': 26.0, 'Month': 'Apr', 'Year': 2020, 'cmd': 'getMay2020'}
```

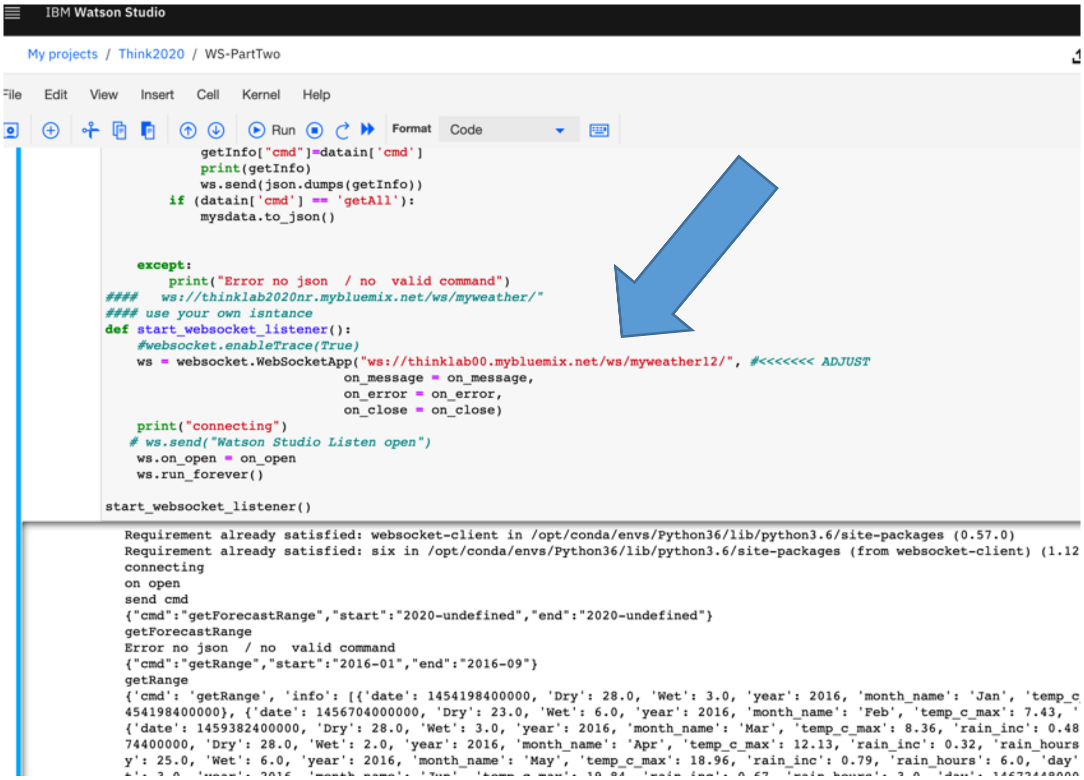
In [14]:
```
#getForecastMonth
yy='2020'
mm='Jun'
dry=dataF[yy:yy].loc[dataF['month_name'] == mm].values[0][0]
wet=dataF[yy:yy].loc[dataF['month_name'] == mm].values[0][1]
getInfo = {'Wet': wet,'Dry' : dry,"Month" : mm ,"Year":yy}

print(getInfo)
```

```
{'Wet': 4.0, 'Dry': 26.0, 'Month': 'Jun', 'Year': '2020'}
```

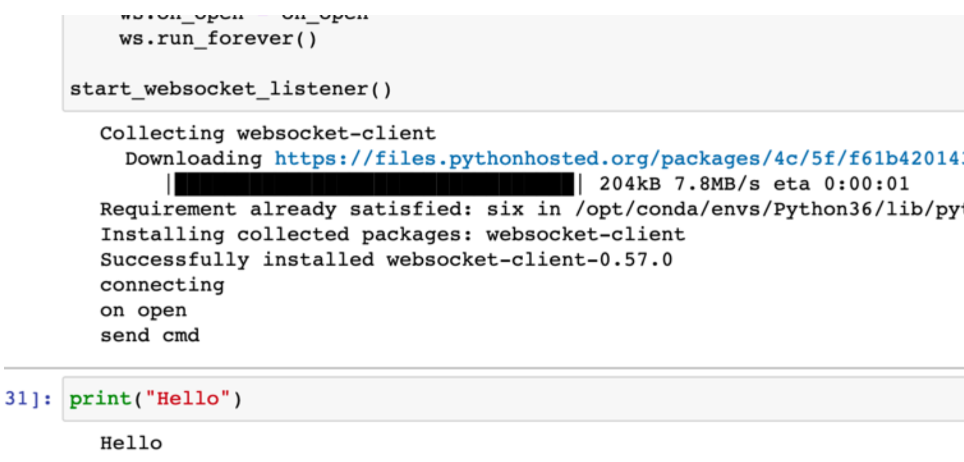
Note: *Make sure to execute each cell in the Notebook.*

1.9 Python <-> Node-Red connection

As before, now we need to connect our python notebook to your Node-RED instance.



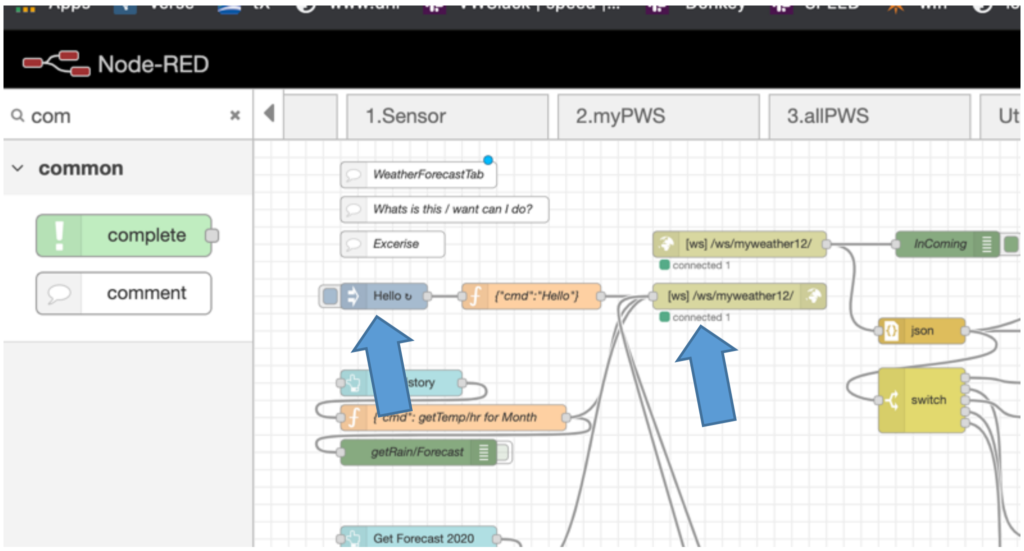
Make sure to adjust the websock URL with your instance/LABID number. You may also need to check the websocket path which is defined in Node-RED. Ex., **/ws/myweather12/**. Execute the cell and you should see something like:



This means the connection is working. If you run into issues make sure you Node-RED host name is correct and the WebSocket url also check the **troubleshoot section.**

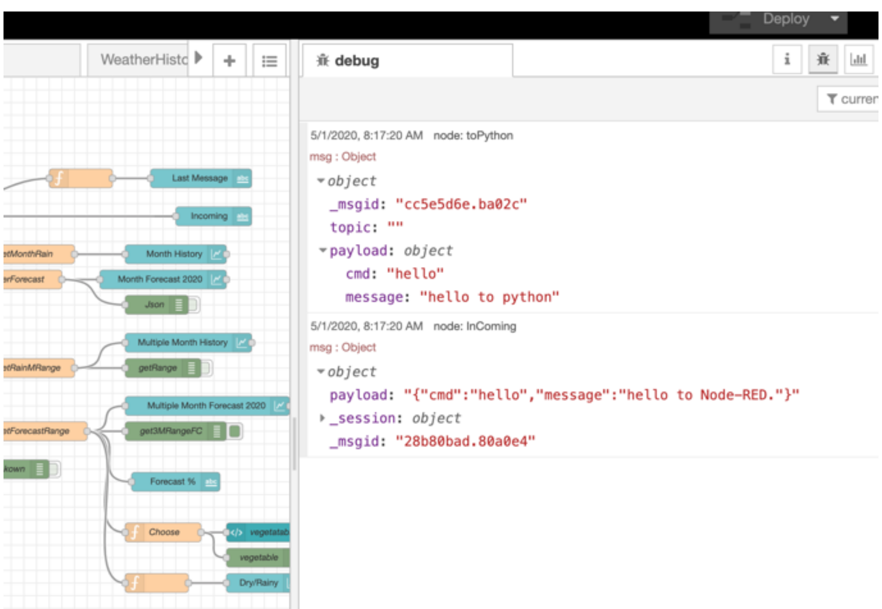
1.10 Querying Python from Node-RED

Switch back to your Node-RED instance and look at the WeatherForecast Tab. You should see 2 green icons under the WebSocket and debug messages, like Heartbeats from your Notebook.



Send a hello message from Node-RED to python and check in the Python Notebook if you see the messages.

Sending a Hello back and forth—see below



```
start_websocket_listener()
```

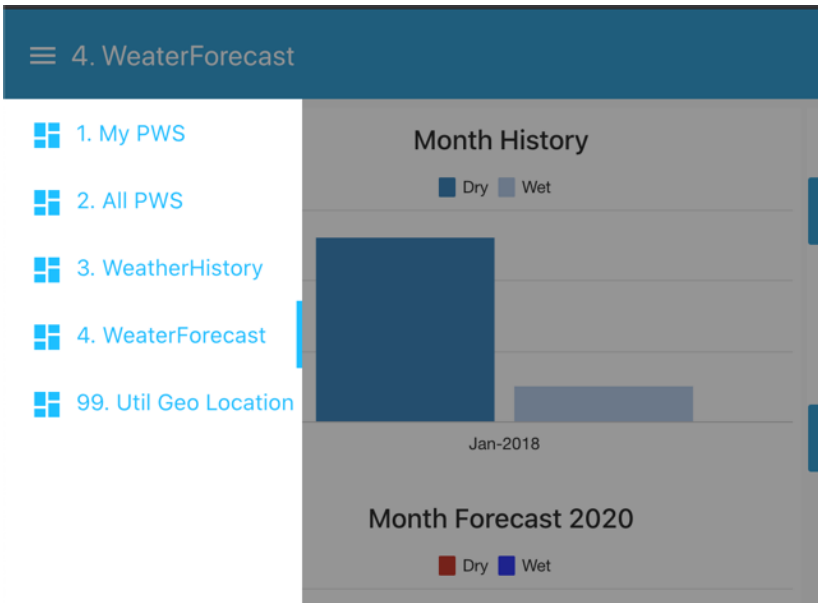
```
Requirement already satisfied: websocket-client in /opt/conda/envs/Py
Requirement already satisfied: six in /opt/conda/envs/Python36/lib/py
connecting
on open
send cmd
{"cmd":"hello","message":"hello to python"}
hello
{"cmd":"hello","message":"hello to Node-RED."}
{"cmd":"hello","message":"hello to python"}
hello
{"cmd":"hello","message":"hello to Node-RED."}
```

```
[*]: print("Hello")
```

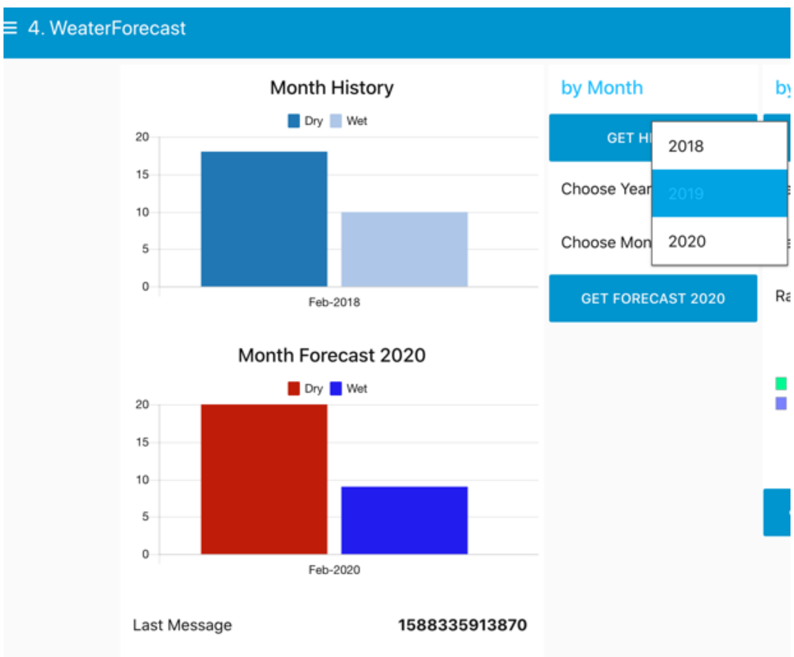
2 Forecasting

2.1 Using the Forecasting Dashboards

Open your Node-RED your instance.mybluemix.net/ui dashboard and select menu # 4. You should see a screen below



Try to get data for Feb 2018 and Feb 2020, in the Month Section:

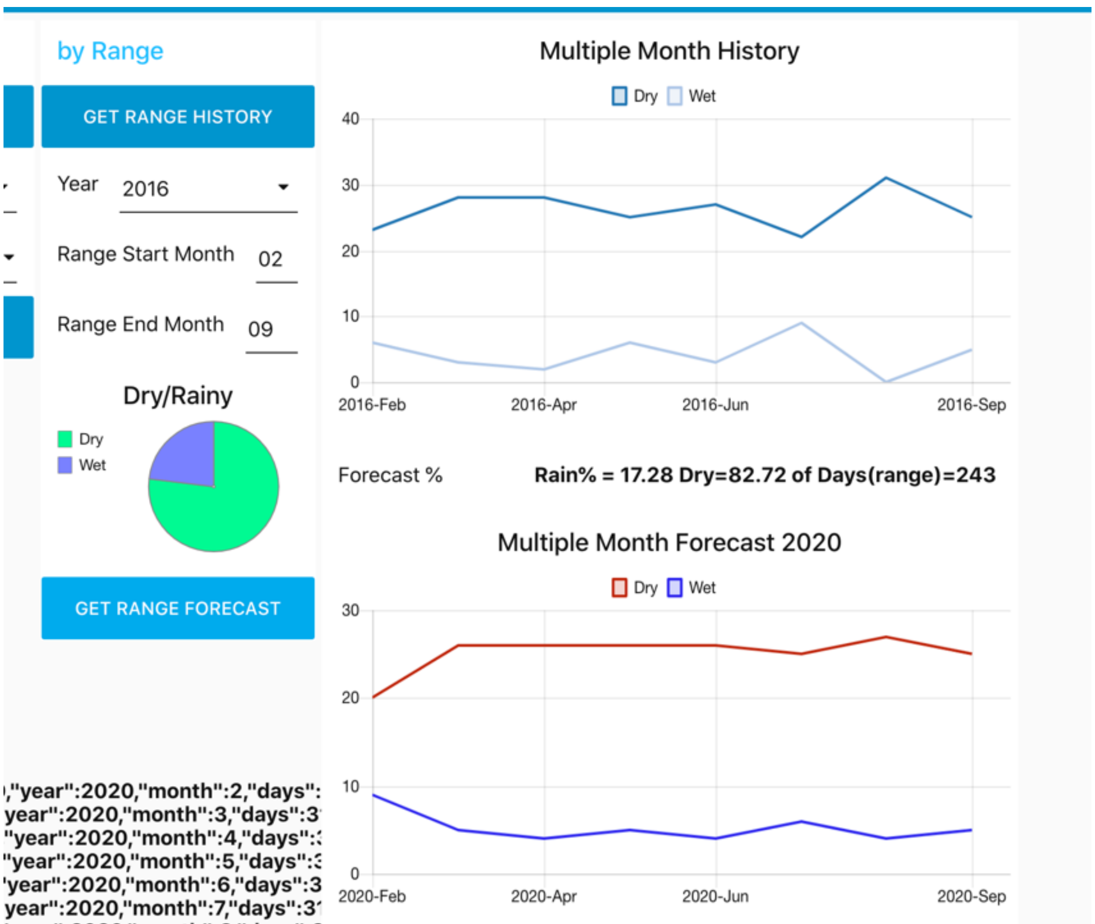


Note: that the forecast Month is hard coded with the year 2020.

#Exercise Add the Missing Month and years (2010-2017)

2.2 The Range Dashboard

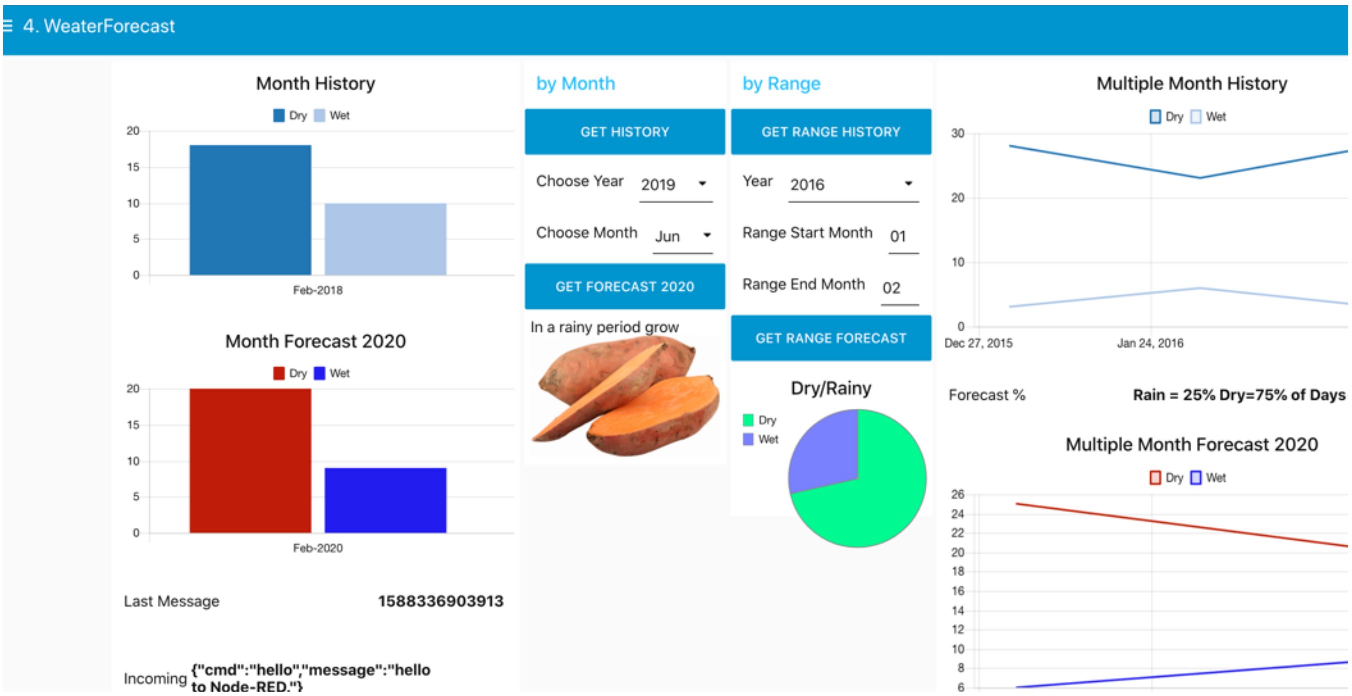
The Range section on the Right will let you query multiple month. Try to query Feb-Sep etc.



Again the forecast year is hard coded to 2020. Add the missing years in the Node-RED

2.3 Vegetable choice

The dashboard as some hidden functionality like displaying the vegetable which one can grow based on Dry/Rainy days. Try to discover and enable the function and display images.



Feel free change the flow/code.

THE END !

3 Trouble shooting

3.1 WebSocket connection

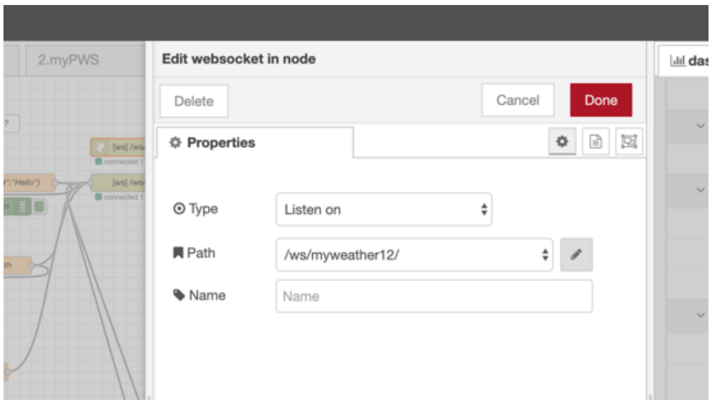
Make sure install the websocket client in the cell via **!pip install websocket-client**

There are 2 parts to connecting

1. Make sure the the url in the the Python Notebook is correct.

Like
ws://thinklab??.mybluemix.net/ws/myweather12
where ??? is you LabID / Node-RED hostname

2. Make sure Node-RED code is deployed and the websocket path is the same e.g **/ws/myweather12**



Sometimes it helps if you change the path on bth ends .. like to **/ws/myweather99** or so

3.2 WebSocket is stuck

Sometimes you cannot tell if the socket works from the Python notebook. Than you the Stop button in the notebook to stop the cell and start a gain … try to execute a different cell 1st before starting the notebook again like the hello work once. If you cannot get any reaction form the Notebook try to reload the whole notebook and execute each cell again.

3.3 Queries not working / No data coming into Node-RED

Sometime the queries are not working. You can see that Node-RED send a command like:

```
    ws.run_forever()

start_websocket_listener()

    Requirement already satisfied: websocket-client in /opt/conda/envs/Python36/lib/p
    Requirement already satisfied: six in /opt/conda/envs/Python36/lib/python3.6/site
    connecting
    on open
    send cmd
    {"cmd":"getForecastMonth","month":"Jun","year":"2020"}
    getForecastMonth
    Error no json  / no  valid command
```

In that case a dataset is most likely not initialize like dataF or so … make sure to stop the notebook than and go thru the query section of the notebook again.

There is a nice developer recipe which show you how to connect the weather information with Alexa and Google Home (https://developer.ibm.com/recipes/tutorials/automatic-broadcasting-weather-information-to-alexa-and-or-chromecast/)

4 References and Information

https://dataplatform.cloud.ibm.com/exchange/public/entry/view/a7432f0c29c5bda2fb42749f363bd9ff
https://machinelearningmastery.com/sarima-for-time-series-forecasting-in-python/
https://www.statsmodels.org/dev/examples/notebooks/generated/statespace_sarimax_stata.html
https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/
https://www.kaggle.com/poiupoiu/how-to-use-sarimax
https://www.kaggle.com/amar09/time-series-delhi-weather-forecasting-arima

Get the instructions for LAB 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

Version:20200501