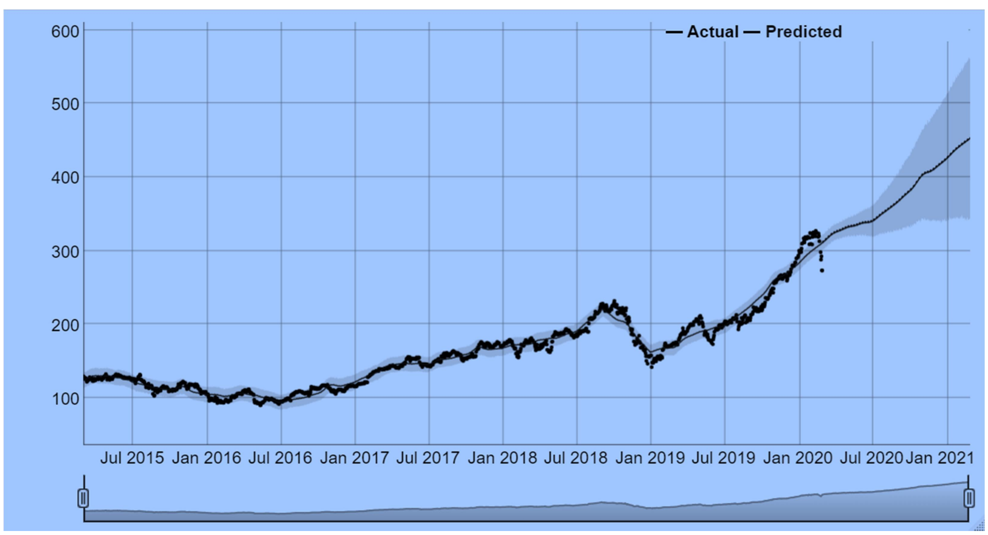
Ing. Dario Di Marzo

IBM Advanced Data Science Capstone Project

Intelligent Irrigation System Predictive Model

with Time Series Forecasting



Capstone Project IBM Advanced Data Science

Use Case: To project an intelligent system for optimizing irrigation based on forecasting of moisture and temperature sensors values.

Literature : An example of an algorithm for an intelligent irrigation system using soil moisture and rainfall prediction* (figura 1).

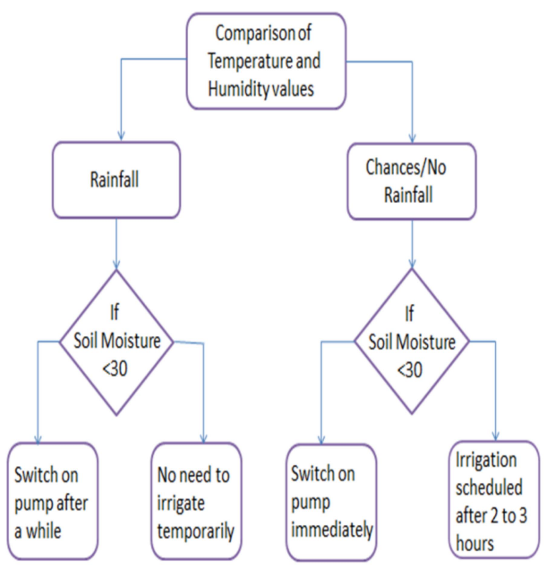


Figura 1

Solution: A similar approach using time series data of soil mosture and temperature forecasting (figura 2).

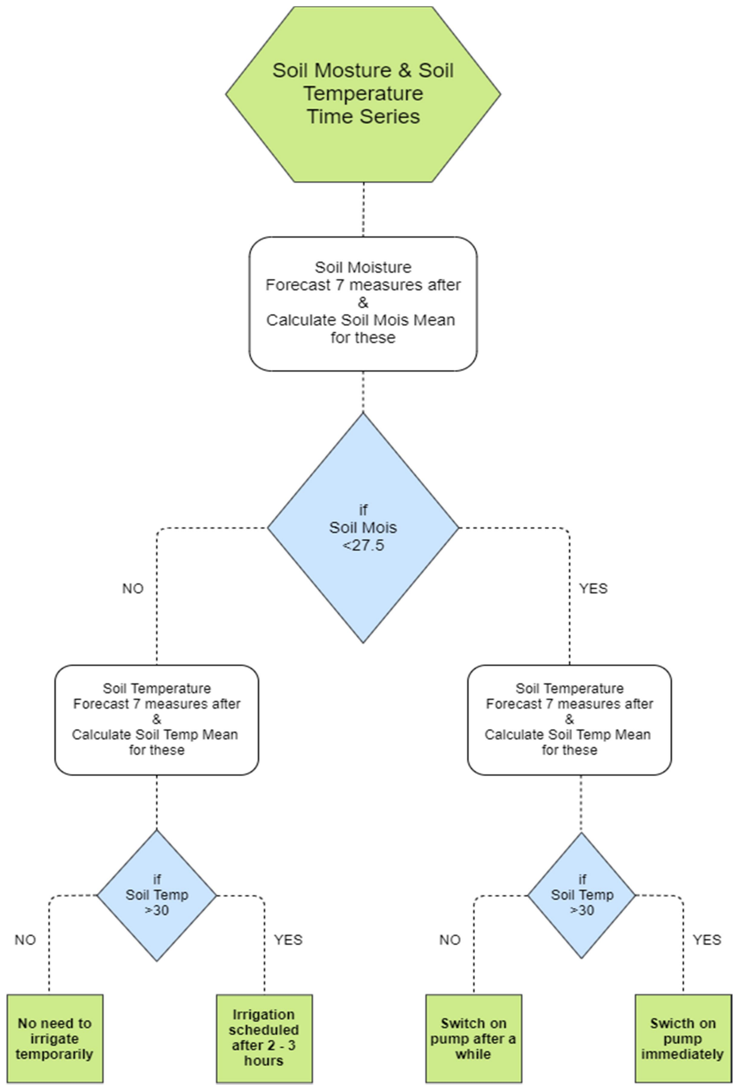


Figura 2

** An IOT based Smart Irrigation System using Soil Moisture and Weather Prediction*

S. Velmurugan, V. Balaji, T. Manoj Bharathi, K. Saravanan

International Journal of Engineering Research & Technology (IJERT) - 2020

Intelligent Irrigation: Architectural Decisions Document

1 Architectural Components Overview

1.1 Data Source

1.1.1 Technology Choice

Dataset "Hyperspectral benchmark dataset on soil moisture" by Felix M. Riese & Sina Keller is imported from https://github.com/awesomedata/awesome-public-datasets.
Format is .csv.

1.1.2 Justification

These data are attractive to develop a machine learning algorithm for a smart system in agriculture.

1.2 Discovery and Exploration

1.2.1 Technology Choice

Data are understood in a Jupyter Notebook, so using python language. Pandas and Matplotlib framework are used to statistical and visual exploration.

1.2.2 Justification

Python and its frameworks like Pandas and Matplotlib are state-of-art technologies for Data Preparation and Understanding.

1.3 ETL

1.3.1 Technology Choice

Data are imported as table in Excel, so final file “soil mosture storic” is a .xlsx cartel.
Columns of spectral measures are deleted graphically in Excel.
File is imported in Watson Studio so technology for repository is Object Store in the cloud.

1.3.2 Justification

Import of a csv file in excel is no code needed.
Spectral measures are out of consideration for project. No code needed using Excel for data preparation.
Watson Studio is a complete cloud environment for Data Science and on this platform data management is easy.

1.4 Feature Creation

1.4.1 Technology Choice

A generic control of dataframe characteristics is done in Pandas.
Time series quantization is done with a IBM Data Refinery Flow.

1st Iteration :

Normalization of series is done with Pandas functions (mean & std).

2nd Iteration :

Normalization is done with MinMaxScaler algorithm of ScikitLearn library.

1.4.2 Justification

The choice of Data Refinery is done to minimize use of code in Watson Studio environment.
Two normalization strategies are necessary to compare effects on model performances as capstone rules.

1.5 Actionable Insights (Model Definition & Evaluation)

1.5.1 Technology Choice

To develop a model of prediction of time series I use a neural network with LSTM stateful neurons.

1.5.2 Justification

So I have tuned model's parameters like Epochs, Number of neurons, Timesteps. So more flexibility is gained respect ARMA traditional non-deeplearning model.

1.6 Applications / Data Products

1.6.1 Technology Choice

The choice for final product is a PDF report, with a brief reference to a case study in Smart IOT System literature.

1.6.2 Justification

The complete notebook as PDF report allows stakeholders (others students in this course) to evaluate this work.

IBM Capstone Project Advanced Data Science

by Dario Di Marzo

Intelligent Irrigation: Time Series Forecasting of soil measures with Deep Learning

Intelligent Irrigation : Data Exploration

Dataset is imported from [https://github.com/awesomedata/awesome-public-datasets (https://github.com/awesomedata/awesome-public-datasets)](https://github.com/awesomedata/awesome-public-datasets)

"Hyperspectral benchmark dataset on soil moisture"

by Felix M. Riese & Sina Keller

Original dataset is a .csv (figure 1)

	A	B	C	D	E	F	G
1	index,datetime,soil_moisture,soil_temperature,454,458,462,466,470,474,478,482,						
2	0,2017-05-23 14:06:17,33.51,34.8,0.0821307072515667,0.0558630119148382,0.0						
3	1,2017-05-23 14:08:17,33.49,35.2,0.07951013936920101,0.05532617054796055,						
4	2,2017-05-23 14:10:17,33.46,35.4,0.08059854381119759,0.0540652664975547,0						
5	3,2017-05-23 14:12:17,33.33,35.0,0.07802406938798218,0.05497247120314093,						
6	4,2017-05-23 14:14:17,33.32,35.3,0.0799729842179957,0.05533482614097687,0						
7	5,2017-05-23 14:16:17,33.23,35.5,0.08147394013067923,0.05482453704217998,						
8	6,2017-05-23 14:18:17,33.16,35.4,0.07903598963201612,0.05527217358203273,						
9	7,2017-05-23 14:20:17,33.05,35.1,0.07909177721686968,0.05530979271602115,						
10	8,2017-05-23 14:22:17,32.96,35.0,0.08028857492013806,0.05401074477394826,						
11	9,2017-05-23 14:24:17,32.96,34.8,0.08070840413017906,0.054309010949206864						
12	10,2017-05-23 14:26:17,32.95,34.5,0.07986969071435833,0.05393984293969476						

Data represent captures values by moisture and temperature sensors in soil for few days in May 2017 (figure 2)

step 1 : Identify quality issues (e.g. missing values…)

In [1]:

```
import os, types
import pandas as pd
from botocore.client import Config
import ibm_boto3

def __iter__(self): return 0

# @hidden_cell
# The following code accesses a file in your IBM Cloud Object Storage. It includes your credentials.
# You might want to remove those credentials before you share the notebook.

if os.environ.get('RUNTIME_ENV_LOCATION_TYPE') == 'external':
    endpoint_1268b64820f3487ab4ae95172d9bd2ee = 'https://s3.us.cloud-object-storage.appdomain.cloud'
else:
    endpoint_1268b64820f3487ab4ae95172d9bd2ee = 'https://s3.private.us.cloud-object-storage.appdomain.cloud'

client_1268b64820f3487ab4ae95172d9bd2ee = ibm_boto3.client(service_name='s3',
    ibm_api_key_id='_On84ldL4wmI0CCMT3T_YvlXDmjXQZuLa2dVhTYLYpQE',
    ibm_auth_endpoint="https://iam.cloud.ibm.com/oidc/token",
    config=Config(signature_version='oauth'),
    endpoint_url=endpoint_1268b64820f3487ab4ae95172d9bd2ee)

body = client_1268b64820f3487ab4ae95172d9bd2ee.get_object(Bucket='capstoneprojectadvanceddatascienc-donotdelete-p
r-xk8sgwtkjowsrk',Key='soil mosture storic.xlsx')['Body']

df_pd = pd.read_excel(body.read())
```

Shift from .csv file to .xlsx file is explaines in ETL deliverable.

In [7]:

```
df_pd.head()
```

Out[7]:

	index	datetime	soil_moisture	soil_temperature
0	1	2017-05-16 11:26:07	35.51	26.4
1	2	2017-05-16 11:28:07	35.40	28.2
2	3	2017-05-16 11:30:07	35.21	29.5
3	4	2017-05-16 11:32:07	35.08	30.4
4	5	2017-05-16 11:34:07	34.88	30.8

In [5]:

```
df_pd.shape
```

Out[5]:

```
(679, 4)
```

In [6]:

```
df_pd.dropna()
```

Out[6]:

	index	datetime	soil_moisture	soil_temperature
0	1	2017-05-16 11:26:07	35.51	26.4
1	2	2017-05-16 11:28:07	35.40	28.2
2	3	2017-05-16 11:30:07	35.21	29.5
3	4	2017-05-16 11:32:07	35.08	30.4
4	5	2017-05-16 11:34:07	34.88	30.8
...	...	...	...	...
674	675	2017-05-26 14:00:10	29.95	40.5
675	676	2017-05-26 14:02:10	29.85	39.5
676	677	2017-05-26 14:04:10	29.78	39.5
677	678	2017-05-26 14:06:10	29.90	39.5
678	679	2017-05-26 14:08:10	29.75	39.7

679 rows × 4 columns

No missing values are present

step 2 : Assess feature quality – how relevant is a certain measurement (e.g. use correlation matrix)

In [7]:

```
df_pd.corr()
```

Out[7]:

	index	soil_moisture	soil_temperature
index	1.000000	0.424569	-0.319772
soil_moisture	0.424569	1.000000	-0.792451
soil_temperature	-0.319772	-0.792451	1.000000

Soil Mosture and Temperature appear poorly (Temperature) or partially (Moisture) correlated with DateTime (Index) probably because measures from sensors are catched only in

May.

Soil Mosture and Temperature are negatively correlated each other.

step 3 : Get an idea on the value distribution of your data using statistical measures and visualizations

In [8]:

```
df_pd.describe()
```

Out[8]:

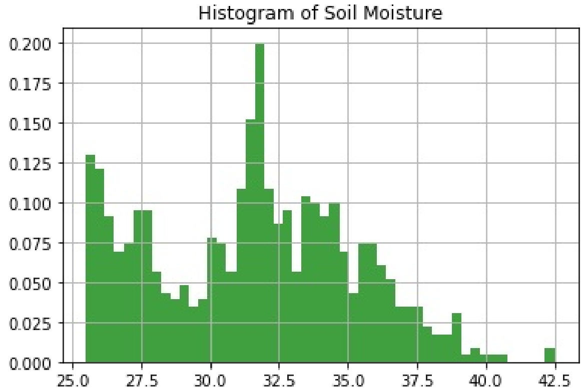
	index	soil_moisture	soil_temperature
count	679.000000	679.000000	679.000000
mean	340.000000	31.568336	37.498380
std	196.154701	3.645354	4.660603
min	1.000000	25.500000	26.400000
25%	170.500000	28.255000	33.600000
50%	340.000000	31.770000	36.700000
75%	509.500000	34.190000	41.150000
max	679.000000	42.500000	47.100000

In [14]:

```
x=df_pd['soil_moisture']
y=df_pd['soil_temperature']
i=df_pd['index']
d=df_pd['datetime']
```

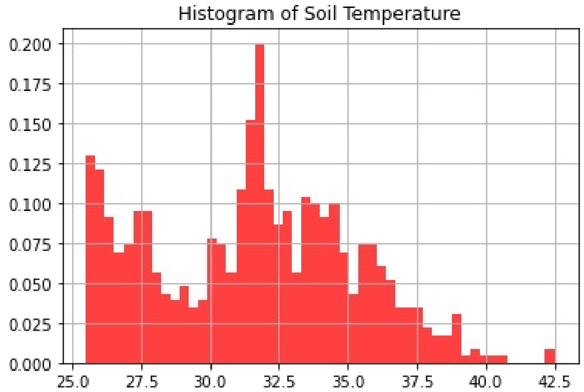
In [4]:

```python
import matplotlib.pyplot as plt


# the histogram of soil moisture

plt.hist(x, 50, density=True, facecolor='g', alpha=0.75)
plt.title('Histogram of Soil Moisture')
plt.grid(True)
plt.show()
```



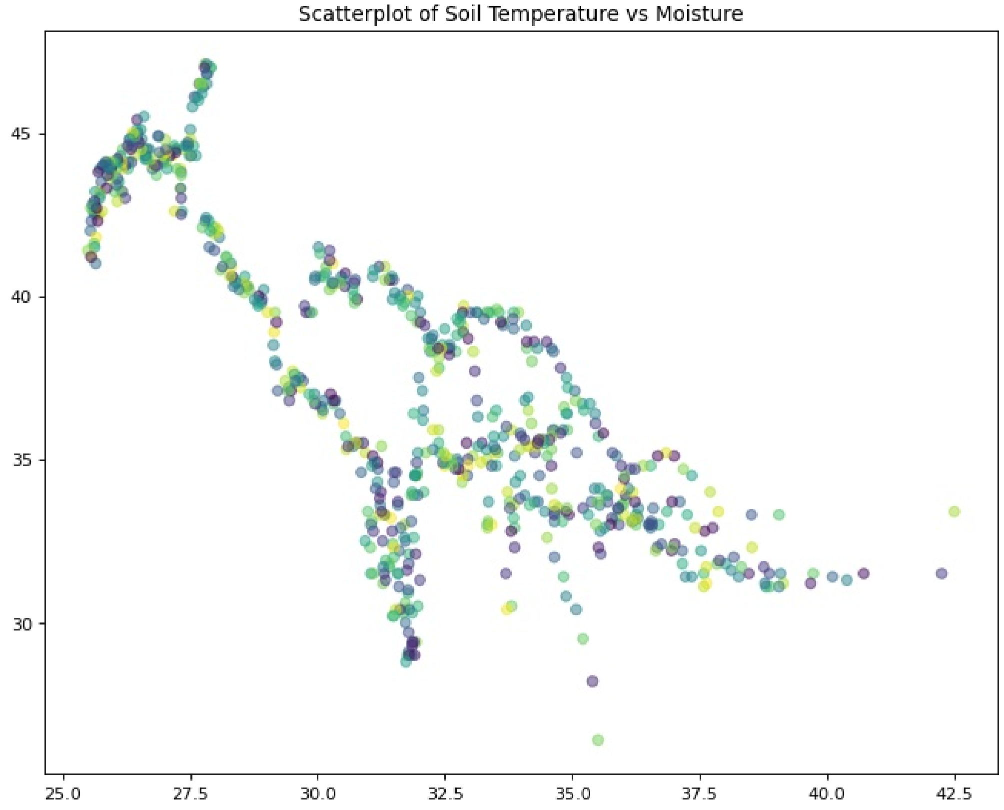
In [11]:

```python
# the histogram of soil temperature

plt.hist(x, 50, density=True, facecolor='r', alpha=0.75)
plt.title('Histogram of Soil Temperature')
plt.grid(True)
plt.show()
```



In [12]:

```
import numpy as np
from matplotlib.pyplot import figure

figure(figsize=(10, 8), dpi=80)
# scatterplot of moisture vs temperature

N = 679
colors = np.random.rand(N)
plt.scatter(x, y, c=colors, alpha=0.5)
plt.title('Scatterplot of Soil Temperature vs Moisture')
plt.show()
```

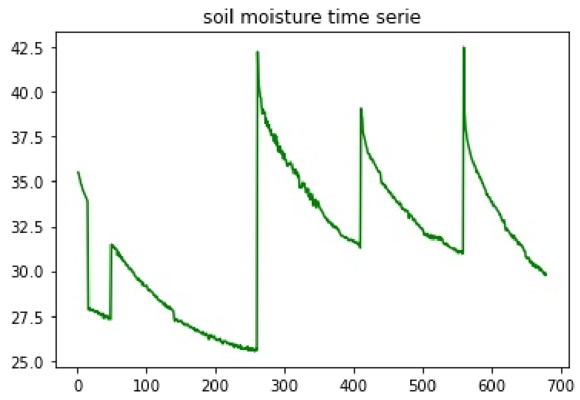


Scatterplot highlights negative correlation between moisture and temperature

Finally a line plot is the best way to visualize a time serie

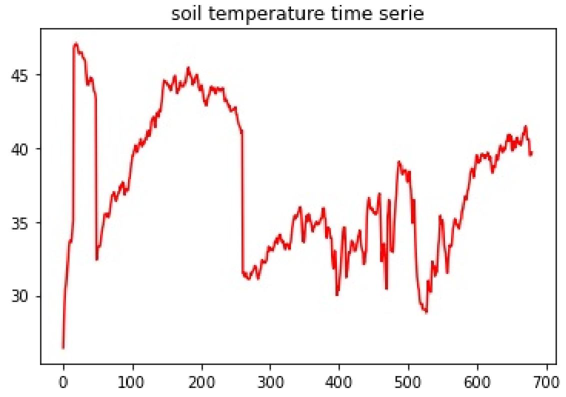
In [17]:

```
plt.plot(i,x, color="green")
plt.title("soil moisture time serie")
plt.show()
```



In [13]:

```
plt.plot(i,y, color="red")
plt.title("soil temperature time serie")
plt.show()
```



Every "hard" jump of value in each line plots is caused by a jump in time of sensors' capture, in same day or in not consecutive days (figure 2-3)

14	16/05/2017 11:52	33.98	34.5
15	16/05/2017 11:54	33.87	35.1
16	16/05/2017 16:19	27.84	46.8
17	16/05/2017 16:21	27.9	47

259	17/05/2017 18:00	25.65	41
260	17/05/2017 18:02	25.56	41.2
261	23/05/2017 10:38	42.25	31.5
262	23/05/2017 10:40	40.72	31.5

We will try to solve this issue later in feature engineering stage.

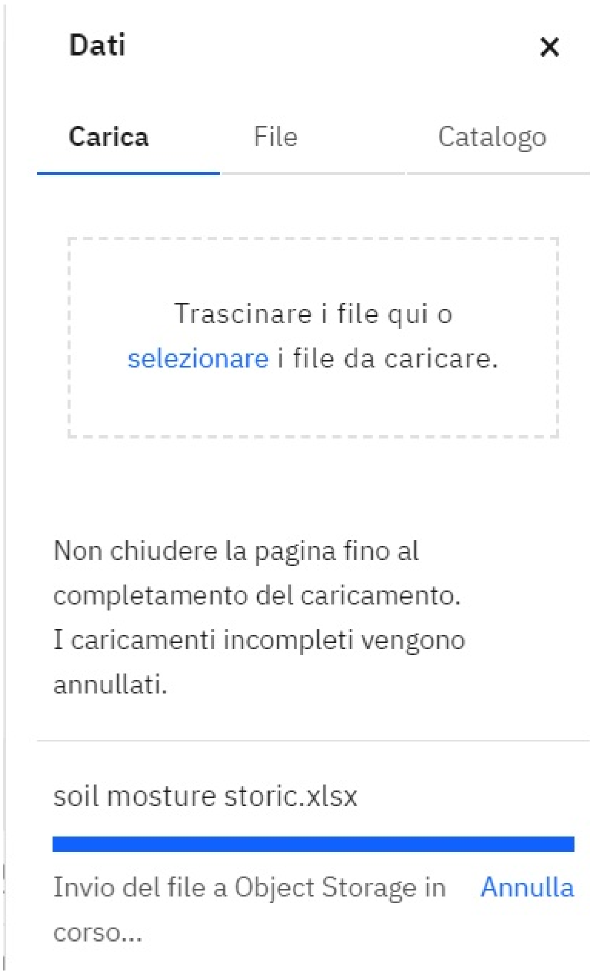
Intelligent_Irrigation: ETL

Original dataset is a .csv (figure 1) imported as table in Excel. Columns of spectral values are deleted. Values are ordered by DateTime (figura 2).

	A	B	C	D	E	F	G
1	index,datetime,soil_moisture,soil_temperature,454,458,462,466,470,474,478,482,						
2	0,2017-05-23 14:06:17,33.51,34.8,0.0821307072515667,0.0558630119148382,0.(						
3	1,2017-05-23 14:08:17,33.49,35.2,0.07951013936920101,0.05532617054796055,						
4	2,2017-05-23 14:10:17,33.46,35.4,0.08059854381119759,0.0540652664975547,0						
5	3,2017-05-23 14:12:17,33.33,35.0,0.07802406938798218,0.05497247120314093,						
6	4,2017-05-23 14:14:17,33.32,35.3,0.0799729842179957,0.05533482614097687,0						
7	5,2017-05-23 14:16:17,33.23,35.5,0.08147394013067923,0.05482453704217998,						
8	6,2017-05-23 14:18:17,33.16,35.4,0.07903598963201612,0.05527217358203273,						
9	7,2017-05-23 14:20:17,33.05,35.1,0.07909177721686968,0.05530979271602115,						
10	8,2017-05-23 14:22:17,32.96,35.0,0.08028857492013806,0.05401074477394826,						
11	9,2017-05-23 14:24:17,32.96,34.8,0.08070840413017906,0.054309010949206864						
12	10.2017-05-23 14:26:17.32.95.34.5.0.07986969071435833.0.05393984293969476						

	A	B	C	D
1	index	datetime	soil_moisture	soil_temperature
2	1	16/05/2017 11:26	35.51	26.4
3	2	16/05/2017 11:28	35.4	28.2
4	3	16/05/2017 11:30	35.21	29.5
5	4	16/05/2017 11:32	35.08	30.4
6	5	16/05/2017 11:34	34.88	30.8
7	6	16/05/2017 11:36	34.83	31.4

The .xlsx file is loaded in Watson Studio as Data Asset (Object Store in IBM Cloud). So no code is needed (figure 3).



Intelligent Irrigation : Features Creation

Data Cleansing

In [1]:

```
import os, types
import pandas as pd
from botocore.client import Config
import ibm_boto3

def __iter__(self): return 0

# @hidden_cell
# The following code accesses a file in your IBM Cloud Object Storage. It includes your credentials.
# You might want to remove those credentials before you share the notebook.

if os.environ.get('RUNTIME_ENV_LOCATION_TYPE') == 'external':
    endpoint_1268b64820f3487ab4ae95172d9bd2ee = 'https://s3.us.cloud-object-storage.appdomain.cloud'
else:
    endpoint_1268b64820f3487ab4ae95172d9bd2ee = 'https://s3.private.us.cloud-object-storage.appdomain.cloud'

client_1268b64820f3487ab4ae95172d9bd2ee = ibm_boto3.client(service_name='s3',
    ibm_api_key_id='_On84ldL4wmI0CCMT3T_YvlXDmjXQZuLa2dVhTYLYpQE',
    ibm_auth_endpoint="https://iam.cloud.ibm.com/oidc/token",
    config=Config(signature_version='oauth'),
    endpoint_url=endpoint_1268b64820f3487ab4ae95172d9bd2ee)

body = client_1268b64820f3487ab4ae95172d9bd2ee.get_object(Bucket='capstoneprojectadvanceddatascienc-donotdelete-p
r-xk8sgwtkjowsrk',Key='soil mosture storic.xlsx')['Body']

df = pd.read_excel(body.read())
df.head()
```

Out[1]:

	index	datetime	soil_moisture	soil_temperature
0	1	2017-05-16 11:26:07	35.51	26.4
1	2	2017-05-16 11:28:07	35.40	28.2
2	3	2017-05-16 11:30:07	35.21	29.5
3	4	2017-05-16 11:32:07	35.08	30.4
4	5	2017-05-16 11:34:07	34.88	30.8

In [2]:

```
df.shape
```

Out[2]:

```
(679, 4)
```

In [3]:

```
df.dtypes
```

Out[3]:

```
index                        int64
datetime            datetime64[ns]
soil_moisture              float64
soil_temperature           float64
dtype: object
```

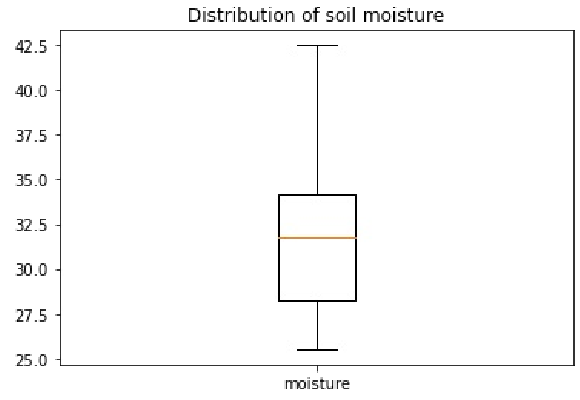
Data types respect columns' values nature

In [3]:

```
x=df['soil_moisture']
y=df['soil_temperature']
```

In [7]:

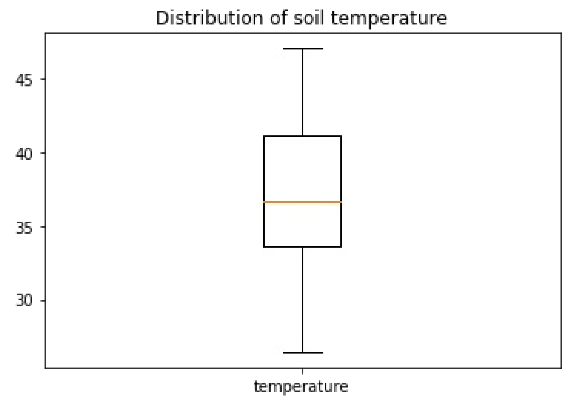
```
import matplotlib.pyplot as plt

plt.boxplot(x, labels=["moisture"])
plt.title("Distribution of soil moisture")
plt.show()
```



Distribution of soil moisture don't reveal anomalies in values range .No outliers are present.

In [8]:

```
plt.boxplot(y, labels=["temperature"])
plt.title("Distribution of soil temperature")
plt.show()
```



Also distribution of soil temperature don't reveal anomalies in values range .No outliers are present.

Features Engineering

Imputed time-series quantization: To regularize time series are extracted only measures for those hours (from 10 to 14) that are present in every day captured. Also measures for days having poor number of captures are deleted. This step is done by a Data Refinery Flow (figure 1). A new data asset is created.

In [4]:

```
body = client_1268b64820f3487ab4ae95172d9bd2ee.get_object(Bucket='capstoneprojectadvanceddatascienc-donotdelete-p
r-xk8sgwtkjowsrk',Key='data_asset/soil_mosture_storic_xlsx_shaped_7kly8rwbhh1mq2w1xspdd817i')['Body']

df_reshaped = pd.read_excel(body.read())
df_reshaped.head()
```

```
/opt/conda/envs/Python-3.8-main/lib/python3.8/site-packages/openpyxl/styles/stylesheet.py:221: UserW
arning: Workbook contains no default style, apply openpyxl's default
  warn("Workbook contains no default style, apply openpyxl's default")
```

Out[4]:

	index	datetime	soil_moisture	soil_temperature
0	49	2017-05-17 10:10:22	31.48	32.4
1	50	2017-05-17 10:12:22	31.47	32.8
2	51	2017-05-17 10:14:22	31.44	33.2
3	52	2017-05-17 10:16:22	31.32	33.3
4	53	2017-05-17 10:18:22	31.34	33.3

In [9]:

```
df_reshaped.shape
```

Out[9]:

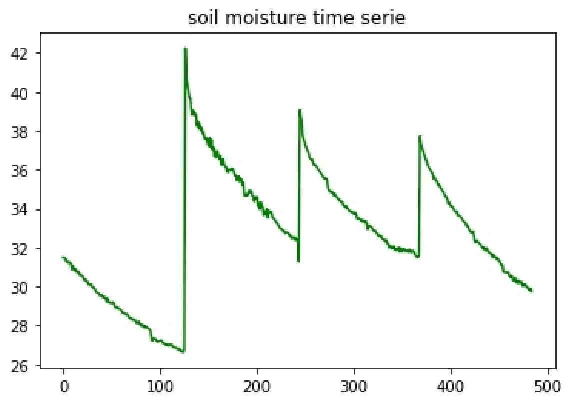
```
(484, 4)
```

In [5]:

```
x=df_reshaped['soil_moisture']
y=df_reshaped['soil_temperature']
```

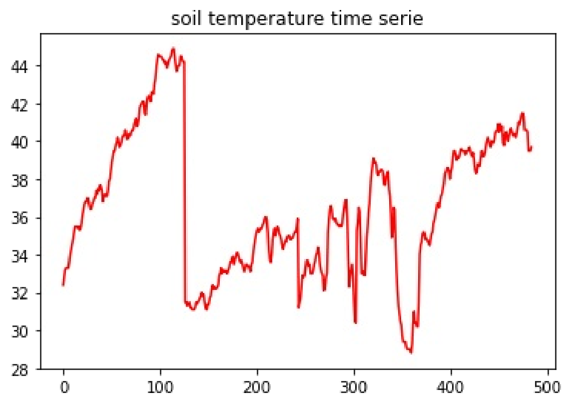
In [6]:

```
import matplotlib.pyplot as plt

plt.plot(x, color="green")
plt.title("soil moisture time serie")
plt.show()
```



In [7]:

```
plt.plot(y, color="red")
plt.title("soil temperature time serie")
plt.show()
```



As we can see filtering series lead to more realistic periodic series

Normalizing : to center data around zero and scale values to a standard deviation of one

In [8]:

```
mean=df_reshaped['soil_moisture'].mean()
mean
```

Out[8]:

32.66878099173554

In [9]:

```
std=df_reshaped['soil_moisture'].std()
std
```

Out[9]:

3.162235751784753

In [10]:

```
df_reshaped['soil_moisture']=(df_reshaped['soil_moisture']-mean)/std
df_reshaped.head()
```

Out[10]:

	index	datetime	soil_moisture	soil_temperature
0	49	2017-05-17 10:10:22	-0.375931	32.4
1	50	2017-05-17 10:12:22	-0.379093	32.8
2	51	2017-05-17 10:14:22	-0.388580	33.2
3	52	2017-05-17 10:16:22	-0.426528	33.3
4	53	2017-05-17 10:18:22	-0.420203	33.3

In [11]:

```
mean=df_reshaped['soil_temperature'].mean()
mean
```

Out[11]:

36.59359504132232

In [12]:

```
std=df_reshaped['soil_temperature'].std()
std
```

Out[12]:

3.797544740372474

In [13]:

```
df_reshaped['soil_temperature']=(df_reshaped['soil_temperature']-mean)/std
df_reshaped.head()
```

Out[13]:

	index	datetime	soil_moisture	soil_temperature
0	49	2017-05-17 10:10:22	-0.375931	-1.104291
1	50	2017-05-17 10:12:22	-0.379093	-0.998960
2	51	2017-05-17 10:14:22	-0.388580	-0.893629
3	52	2017-05-17 10:16:22	-0.426528	-0.867296
4	53	2017-05-17 10:18:22	-0.420203	-0.867296

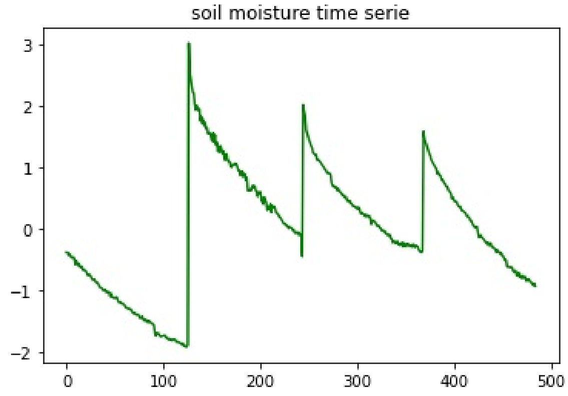
Take a look to our time series normalized

In [14]:

```
x=df_reshaped['soil_moisture']
y=df_reshaped['soil_temperature']
```

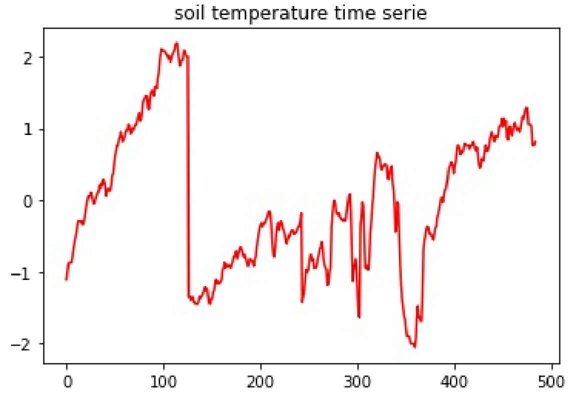
In [15]:

```
plt.plot(x, color="green")
plt.title("soil moisture time serie")
plt.show()
```



In [16]:

```
plt.plot(y, color="red")
plt.title("soil temperature time serie")
plt.show()
```



Now we are going to export reengineered dataset

In [28]:

```
# @hidden_cell
# The project token is an authorization token that is used to access project resources like data sources, connect
ions, and used by platform APIs.
from project_lib import Project
project = Project(project_id='697aab51-ea8a-4a5b-8293-151f3223e235', project_access_token='p-7a8de364fdda9bf6c893
de026095666a3b0af059')
pc = project.project_context
```

In [29]:

```
project.save_data("feature_creation.csv", df_reshaped.to_csv())
```

Out[29]:

```
{'file_name': 'feature_creation.csv',
 'message': 'File saved to project storage.',
 'bucket_name': 'capstoneprojectadvanceddatascienc-donotdelete-pr-xk8sgwtkjowsrk',
 'asset_id': 'a9d39621-ea3e-4481-bb57-65db2a55fcc4'}
```

Intelligent Irrigation: Model Definition

Loading normalized dataframe reengineered in feature creation stage

In [1]:

```
import os, types
import pandas as pd
from botocore.client import Config
import ibm_boto3

def __iter__(self): return 0

# @hidden_cell
# The following code accesses a file in your IBM Cloud Object Storage. It includes your credentials.
# You might want to remove those credentials before you share the notebook.

if os.environ.get('RUNTIME_ENV_LOCATION_TYPE') == 'external':
    endpoint_1268b64820f3487ab4ae95172d9bd2ee = 'https://s3.us.cloud-object-storage.appdomain.cloud'
else:
    endpoint_1268b64820f3487ab4ae95172d9bd2ee = 'https://s3.private.us.cloud-object-storage.appdomain.cloud'

client_1268b64820f3487ab4ae95172d9bd2ee = ibm_boto3.client(service_name='s3',
    ibm_api_key_id='_On84ldL4wmI0CCMT3T_YvlXDmjXQZuLa2dVhTYLYpQE',
    ibm_auth_endpoint="https://iam.cloud.ibm.com/oidc/token",
    config=Config(signature_version='oauth'),
    endpoint_url=endpoint_1268b64820f3487ab4ae95172d9bd2ee)

body = client_1268b64820f3487ab4ae95172d9bd2ee.get_object(Bucket='capstoneprojectadvanceddatascienc-donotdelete-p
r-xk8sgwtkjowsrk',Key='feature_creation.csv')['Body']
# add missing __iter__ method, so pandas accepts body as file-like object
if not hasattr(body, "__iter__"): body.__iter__ = types.MethodType( __iter__, body )

df = pd.read_csv(body)
df.head()
```

Out[1]:

	Unnamed: 0	index	datetime	soil_moisture	soil_temperature
0	0	49	2017-05-17 10:10:22	-0.375931	-1.104291
1	1	50	2017-05-17 10:12:22	-0.379093	-0.998960
2	2	51	2017-05-17 10:14:22	-0.388580	-0.893629
3	3	52	2017-05-17 10:16:22	-0.426528	-0.867296
4	4	53	2017-05-17 10:18:22	-0.420203	-0.867296

In [2]:

```
df.drop("Unnamed: 0", axis=1,inplace=True)
```

In [3]:

```
df.head()
```

Out[3]:

	index	datetime	soil_moisture	soil_temperature
0	49	2017-05-17 10:10:22	-0.375931	-1.104291
1	50	2017-05-17 10:12:22	-0.379093	-0.998960
2	51	2017-05-17 10:14:22	-0.388580	-0.893629
3	52	2017-05-17 10:16:22	-0.426528	-0.867296
4	53	2017-05-17 10:18:22	-0.420203	-0.867296

In [4]:

```
df.shape
```

Out[4]:

```
(484, 4)
```

Now it’s time to start modelling

It's choiced to implement a LSTM Neural Net to predict Time Series

In [5]:

```
!pip3 install --upgrade tensorflow
!pip install Keras
```

```
Requirement already satisfied: tensorflow in /opt/conda/envs/Python-3.8-main/lib/python3.8/site-pack
ages (2.4.3)
Collecting tensorflow
  Downloading tensorflow-2.6.0-cp38-cp38-manylinux2010_x86_64.whl (458.4 MB)
     |████████████████████████████████| 458.4 MB 54 kB/s s eta 0:00:01
Requirement already satisfied: google-pasta~=0.2 in /opt/conda/envs/Python-3.8-main/lib/python3.8/si
te-packages (from tensorflow) (0.2.0)
Collecting keras~=2.6
  Downloading keras-2.6.0-py2.py3-none-any.whl (1.3 MB)
     |████████████████████████████████| 1.3 MB 26.5 MB/s eta 0:00:01
Collecting opt-einsum~=3.3.0
  Downloading opt_einsum-3.3.0-py3-none-any.whl (65 kB)
     |████████████████████████████████| 65 kB 9.9 MB/s  eta 0:00:01
Requirement already satisfied: six~=1.15.0 in /opt/conda/envs/Python-3.8-main/lib/python3.8/site-pac
kages (from tensorflow) (1.15.0)
Requirement already satisfied: protobuf>=3.9.2 in /opt/conda/envs/Python-3.8-main/lib/python3.8/site
-packages (from tensorflow) (3.11.2)
Collecting h5py~=3.1.0
  Downloading h5py-3.1.0-cp38-cp38-manylinux1_x86_64.whl (4.4 MB)
     |████████████████████████████████| 4.4 MB 60.6 MB/s eta 0:00:01
Requirement already satisfied: wheel~=0.35 in /opt/conda/envs/Python-3.8-main/lib/python3.8/site-pac
kages (from tensorflow) (0.35.1)
Collecting tensorboard~=2.6
  Downloading tensorboard-2.7.0-py3-none-any.whl (5.8 MB)
     |████████████████████████████████| 5.8 MB 63.3 MB/s eta 0:00:01
Requirement already satisfied: termcolor~=1.1.0 in /opt/conda/envs/Python-3.8-main/lib/python3.8/sit
e-packages (from tensorflow) (1.1.0)
Collecting clang~=5.0
  Downloading clang-5.0.tar.gz (30 kB)
Requirement already satisfied: typing-extensions~=3.7.4 in /opt/conda/envs/Python-3.8-main/lib/pytho
n3.8/site-packages (from tensorflow) (3.7.4.3)
Collecting grpcio<2.0,>=1.37.0
  Downloading grpcio-1.41.1-cp38-cp38-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (3.9 MB)
     |████████████████████████████████| 3.9 MB 61.2 MB/s eta 0:00:01
Collecting gast==0.4.0
  Downloading gast-0.4.0-py3-none-any.whl (9.8 kB)
Collecting flatbuffers~=1.12.0
  Downloading flatbuffers-1.12-py2.py3-none-any.whl (15 kB)
Requirement already satisfied: astunparse~=1.6.3 in /opt/conda/envs/Python-3.8-main/lib/python3.8/si
te-packages (from tensorflow) (1.6.3)
Requirement already satisfied: numpy~=1.19.2 in /opt/conda/envs/Python-3.8-main/lib/python3.8/site-p
ackages (from tensorflow) (1.19.2)
Requirement already satisfied: wrapt~=1.12.1 in /opt/conda/envs/Python-3.8-main/lib/python3.8/site-p
ackages (from tensorflow) (1.12.1)
Requirement already satisfied: absl-py~=0.10 in /opt/conda/envs/Python-3.8-main/lib/python3.8/site-p
ackages (from tensorflow) (0.10.0)
Requirement already satisfied: keras-preprocessing~=1.1.2 in /opt/conda/envs/Python-3.8-main/lib/pyt
hon3.8/site-packages (from tensorflow) (1.1.2)
Collecting tensorflow-estimator~=2.6
  Downloading tensorflow_estimator-2.7.0-py2.py3-none-any.whl (463 kB)
     |████████████████████████████████| 463 kB 63.6 MB/s eta 0:00:01
Requirement already satisfied: setuptools in /opt/conda/envs/Python-3.8-main/lib/python3.8/site-pack
ages (from protobuf>=3.9.2->tensorflow) (52.0.0.post20211006)
Requirement already satisfied: requests<3,>=2.21.0 in /opt/conda/envs/Python-3.8-main/lib/python3.8/
site-packages (from tensorboard~=2.6->tensorflow) (2.25.1)
Requirement already satisfied: tensorboard-plugin-wit>=1.6.0 in /opt/conda/envs/Python-3.8-main/lib/
python3.8/site-packages (from tensorboard~=2.6->tensorflow) (1.6.0)
Requirement already satisfied: google-auth-oauthlib<0.5,>=0.4.1 in /opt/conda/envs/Python-3.8-main/l
ib/python3.8/site-packages (from tensorboard~=2.6->tensorflow) (0.4.4)
Requirement already satisfied: werkzeug>=0.11.15 in /opt/conda/envs/Python-3.8-main/lib/python3.8/si
te-packages (from tensorboard~=2.6->tensorflow) (1.0.1)
Requirement already satisfied: markdown>=2.6.8 in /opt/conda/envs/Python-3.8-main/lib/python3.8/site
-packages (from tensorboard~=2.6->tensorflow) (3.1.1)
Collecting tensorboard-data-server<0.7.0,>=0.6.0
  Downloading tensorboard_data_server-0.6.1-py3-none-manylinux2010_x86_64.whl (4.9 MB)
     |████████████████████████████████| 4.9 MB 58.7 MB/s eta 0:00:01
Requirement already satisfied: google-auth<3,>=1.6.3 in /opt/conda/envs/Python-3.8-main/lib/python3.
8/site-packages (from tensorboard~=2.6->tensorflow) (1.23.0)
Requirement already satisfied: pyasn1-modules>=0.2.1 in /opt/conda/envs/Python-3.8-main/lib/python3.
8/site-packages (from google-auth<3,>=1.6.3->tensorboard~=2.6->tensorflow) (0.2.8)
Requirement already satisfied: cachetools<5.0,>=2.0.0 in /opt/conda/envs/Python-3.8-main/lib/python3
.8/site-packages (from google-auth<3,>=1.6.3->tensorboard~=2.6->tensorflow) (4.2.2)
Requirement already satisfied: rsa<5,>=3.1.4 in /opt/conda/envs/Python-3.8-main/lib/python3.8/site-p
ackages (from google-auth<3,>=1.6.3->tensorboard~=2.6->tensorflow) (4.7.2)
Requirement already satisfied: requests-oauthlib>=0.7.0 in /opt/conda/envs/Python-3.8-main/lib/pytho
```

```
n3.8/site-packages (from google-auth-oauthlib<0.5,>=0.4.1->tensorboard~=2.6->tensorflow) (1.3.0)
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in /opt/conda/envs/Python-3.8-main/lib/python3.8
/site-packages (from pyasn1-modules>=0.2.1->google-auth<3,>=1.6.3->tensorboard~=2.6->tensorflow) (0.
4.8)
Requirement already satisfied: chardet<5,>=3.0.2 in /opt/conda/envs/Python-3.8-main/lib/python3.8/si
te-packages (from requests<3,>=2.21.0->tensorboard~=2.6->tensorflow) (3.0.4)
Requirement already satisfied: certifi>=2017.4.17 in /opt/conda/envs/Python-3.8-main/lib/python3.8/s
ite-packages (from requests<3,>=2.21.0->tensorboard~=2.6->tensorflow) (2021.10.8)
Requirement already satisfied: idna<3,>=2.5 in /opt/conda/envs/Python-3.8-main/lib/python3.8/site-pa
ckages (from requests<3,>=2.21.0->tensorboard~=2.6->tensorflow) (2.8)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /opt/conda/envs/Python-3.8-main/lib/python3.
8/site-packages (from requests<3,>=2.21.0->tensorboard~=2.6->tensorflow) (1.26.6)
Requirement already satisfied: oauthlib>=3.0.0 in /opt/conda/envs/Python-3.8-main/lib/python3.8/site
-packages (from requests-oauthlib>=0.7.0->google-auth-oauthlib<0.5,>=0.4.1->tensorboard~=2.6->tensor
flow) (3.1.1)
Building wheels for collected packages: clang
  Building wheel for clang (setup.py) ... done
  Created wheel for clang: filename=clang-5.0-py3-none-any.whl size=30705 sha256=8a7b4bb56c86425d2d6
19d8180f27201b69c2e28e91c2f2424d110762bb9cf0c
  Stored in directory: /tmp/wsuser/.cache/pip/wheels/f1/60/77/22b9b5887bd47801796a856f47650d9789c74d
c3161a26d608
Successfully built clang
Installing collected packages: tensorboard-data-server, grpcio, tensorflow-estimator, tensorboard, o
pt-einsum, keras, h5py, gast, flatbuffers, clang, tensorflow
  Attempting uninstall: grpcio
    Found existing installation: grpcio 1.35.0
    Uninstalling grpcio-1.35.0:
      Successfully uninstalled grpcio-1.35.0
  Attempting uninstall: tensorflow-estimator
    Found existing installation: tensorflow-estimator 2.4.0
    Uninstalling tensorflow-estimator-2.4.0:
      Successfully uninstalled tensorflow-estimator-2.4.0
  Attempting uninstall: tensorboard
    Found existing installation: tensorboard 2.4.1
    Uninstalling tensorboard-2.4.1:
      Successfully uninstalled tensorboard-2.4.1
  Attempting uninstall: opt-einsum
    Found existing installation: opt-einsum 3.1.0
    Uninstalling opt-einsum-3.1.0:
      Successfully uninstalled opt-einsum-3.1.0
  Attempting uninstall: h5py
    Found existing installation: h5py 2.10.0
    Uninstalling h5py-2.10.0:
      Successfully uninstalled h5py-2.10.0
  Attempting uninstall: gast
    Found existing installation: gast 0.3.3
    Uninstalling gast-0.3.3:
      Successfully uninstalled gast-0.3.3
  Attempting uninstall: flatbuffers
    Found existing installation: flatbuffers 20210226132247
    Uninstalling flatbuffers-20210226132247:
      Successfully uninstalled flatbuffers-20210226132247
  Attempting uninstall: tensorflow
    Found existing installation: tensorflow 2.4.3
    Uninstalling tensorflow-2.4.3:
      Successfully uninstalled tensorflow-2.4.3
Successfully installed clang-5.0 flatbuffers-1.12 gast-0.4.0 grpcio-1.41.1 h5py-3.1.0 keras-2.6.0 op
t-einsum-3.3.0 tensorboard-2.7.0 tensorboard-data-server-0.6.1 tensorflow-2.6.0 tensorflow-estimator
-2.7.0
Requirement already satisfied: Keras in /opt/conda/envs/Python-3.8-main/lib/python3.8/site-packages
(2.6.0)
```

In [6]:

```
#import packages
import numpy as np
import pandas as pd
from keras.preprocessing import sequence
from keras.models import load_model
```

In [11]:

```
# defining the batch size and number of epochs
# batch size better if a multiple of 8
batch_size = 32
epochs = 60
```

In [12]:

```
timesteps = 15
```

In [13]:

```
def get_train_length(dataset, batch_size, test_percent):
    # substract test_percent to be excluded from training, reserved for testset
    length = len(dataset)
    length *= 1 - test_percent
    train_length_values = []
    for x in range(int(length) - 100,int(length)):
        modulo=x%batch_size
        if (modulo == 0):
            train_length_values.append(x)
    return (max(train_length_values))
```

In [14]:

```
length = get_train_length(df, batch_size, 0.3)
print(length)
```

```
320
```

In [15]:

```
#Adding timesteps * 2
upper_train = length + timesteps*2
df_train = df[0:upper_train]
training_set = df_train.iloc[:,2:3].values
training_set.shape
```

Out[15]:

```
(350, 1)
```

In Feature Creation stage we have already normalized the data.

So later, in an another iteration, we will test different performance of models for data normalized manually, or with MinMaxScaler algorithm of Scikit Learn library

In [16]:

```
"""
# Feature Scaling
#scale between 0 and 1. the weights are esier to find.
from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler(feature_range = (0, 1))
training_set_scaled = sc.fit_transform(np.float64(training_set))
training_set_scaled.shape
"""
```

Out[16]:

```
'\n# Feature Scaling\n#scale between 0 and 1. the weights are esier to find.\nfrom sklearn.preproces
sing import MinMaxScaler\nsc = MinMaxScaler(feature_range = (0, 1))\ntraining_set_scaled = sc.fit_tr
ansform(np.float64(training_set))\ntraining_set_scaled.shape\n'
```

In [17]:

```
training_set_scaled=training_set
```

In [18]:

```
X_train = []
y_train = []

# Creating a data structure with n timesteps

print(length + timesteps)
for i in range(timesteps, length + timesteps):
    X_train.append(training_set_scaled[i-timesteps:i,0])
    y_train.append(training_set_scaled[i:i+timesteps,0])

print(len(X_train))
print(len(y_train))
#create X_train matrix
#30 items per array (timestep)
print(X_train[0:2])
print(np.array(X_train).shape)
#create Y_train matrix
#30 items per array (timestep)
print(y_train[0:2])
print(np.array(y_train).shape)
```

```
335
320
320
[array([-0.37593054, -0.37909286, -0.38857982, -0.42652765, -0.42020301,
       -0.45815085, -0.45498853, -0.44866389, -0.47080013, -0.57199435,
       -0.50558564, -0.5466958 , -0.58780595, -0.55934507, -0.60677987]), array([-0.37909286, -0.388
57982, -0.42652765, -0.42020301, -0.45815085,
       -0.45498853, -0.44866389, -0.47080013, -0.57199435, -0.50558564,
       -0.5466958 , -0.58780595, -0.55934507, -0.60677987, -0.62259147])]
(320, 15)
[array([-0.62259147, -0.60677987, -0.66370162, -0.65421466, -0.66686394,
       -0.67951322, -0.70481177, -0.74275961, -0.73011033, -0.74275961,
       -0.76173353, -0.80284368, -0.81865528, -0.81233064, -0.80916832]), array([-0.60677987, -0.663
70162, -0.65421466, -0.66686394, -0.67951322,
       -0.70481177, -0.74275961, -0.73011033, -0.74275961, -0.76173353,
       -0.80284368, -0.81865528, -0.81233064, -0.80916832, -0.85660311])]
(320, 15)
```

In [19]:

```
# Reshaping
X_train, y_train = np.array(X_train), np.array(y_train)
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
y_train = np.reshape(y_train, (y_train.shape[0], y_train.shape[1], 1))
print(X_train.shape)
print(y_train.shape)
```

```
(320, 15, 1)
(320, 15, 1)
```

In [20]:

```
# Building the LSTM
# Importing the Keras libraries and packages

from keras.layers import Dense
from keras.layers import Input, LSTM
from keras.models import Model
import h5py
```

In [21]:

```
# Initialising the LSTM Model with MAE Loss-Function
# Using Functional API

inputs_1_mae = Input(batch_shape=(batch_size,timesteps,1))
#each layer is the input of the next layer
lstm_1_mae = LSTM(1, stateful=True, return_sequences=True)(inputs_1_mae)
lstm_2_mae = LSTM(1, stateful=True, return_sequences=True)(lstm_1_mae)

output_1_mae = Dense(units = 1)(lstm_2_mae)

regressor_mae = Model(inputs=inputs_1_mae, outputs = output_1_mae)

#adam is fast starting off and then gets slower and more precise
#mae -> mean absolute error loss function
regressor_mae.compile(optimizer='adam', loss = 'mae')
regressor_mae.summary()
```

```
Model: "model"

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(32, 15, 1)]             0
_________________________________________________________________
lstm (LSTM)                  (32, 15, 1)               12
_________________________________________________________________
lstm_1 (LSTM)                (32, 15, 1)               12
_________________________________________________________________
dense (Dense)                (32, 15, 1)               2
=================================================================
Total params: 26
Trainable params: 26
Non-trainable params: 0

_________________________________________________________________
```

In [22]:

```
#Training model
#Statefull
for i in range(epochs):
    print("Epoch: " + str(i))
    #run through all data but the cell, hidden state are used for the next batch.
    regressor_mae.fit(X_train, y_train, shuffle=False, epochs = 1, batch_size = batch_size)
    #resets only the states but the weights, cell and hidden are kept.
    regressor_mae.reset_states()
```

```
Epoch: 0
10/10 [==============================] - 3s 27ms/step - loss: 0.9600
Epoch: 1
10/10 [==============================] - 0s 9ms/step - loss: 0.9585
Epoch: 2
10/10 [==============================] - 0s 10ms/step - loss: 0.9577
Epoch: 3
10/10 [==============================] - 0s 9ms/step - loss: 0.9569
Epoch: 4
10/10 [==============================] - 0s 9ms/step - loss: 0.9561
Epoch: 5
10/10 [==============================] - 0s 9ms/step - loss: 0.9551
Epoch: 6
10/10 [==============================] - 0s 7ms/step - loss: 0.9542
Epoch: 7
10/10 [==============================] - 0s 7ms/step - loss: 0.9531
Epoch: 8
10/10 [==============================] - 0s 7ms/step - loss: 0.9520
Epoch: 9
10/10 [==============================] - 0s 8ms/step - loss: 0.9508
Epoch: 10
10/10 [==============================] - 0s 8ms/step - loss: 0.9495
Epoch: 11
10/10 [==============================] - 0s 7ms/step - loss: 0.9482
Epoch: 12
10/10 [==============================] - 0s 7ms/step - loss: 0.9468
Epoch: 13
10/10 [==============================] - 0s 8ms/step - loss: 0.9452
Epoch: 14
10/10 [==============================] - 0s 7ms/step - loss: 0.9435
Epoch: 15
10/10 [==============================] - 0s 8ms/step - loss: 0.9417
Epoch: 16
10/10 [==============================] - 0s 9ms/step - loss: 0.9397
Epoch: 17
10/10 [==============================] - 0s 9ms/step - loss: 0.9376
```

```
Epoch: 18
10/10 [==============================] - 0s 8ms/step - loss: 0.9352
Epoch: 19
10/10 [==============================] - 0s 7ms/step - loss: 0.9327
Epoch: 20
10/10 [==============================] - 0s 7ms/step - loss: 0.9299
Epoch: 21
10/10 [==============================] - 0s 7ms/step - loss: 0.9270
Epoch: 22
10/10 [==============================] - 0s 7ms/step - loss: 0.9239
Epoch: 23
10/10 [==============================] - 0s 7ms/step - loss: 0.9206
Epoch: 24
10/10 [==============================] - 0s 8ms/step - loss: 0.9172
Epoch: 25
10/10 [==============================] - 0s 8ms/step - loss: 0.9135
Epoch: 26
10/10 [==============================] - 0s 7ms/step - loss: 0.9097
Epoch: 27
10/10 [==============================] - 0s 7ms/step - loss: 0.9058
Epoch: 28
10/10 [==============================] - 0s 7ms/step - loss: 0.9017
Epoch: 29
10/10 [==============================] - 0s 7ms/step - loss: 0.8976
Epoch: 30
10/10 [==============================] - 0s 7ms/step - loss: 0.8934
Epoch: 31
10/10 [==============================] - 0s 8ms/step - loss: 0.8890
Epoch: 32
10/10 [==============================] - 0s 7ms/step - loss: 0.8846
Epoch: 33
10/10 [==============================] - 0s 7ms/step - loss: 0.8801
Epoch: 34
10/10 [==============================] - 0s 7ms/step - loss: 0.8755
Epoch: 35
10/10 [==============================] - 0s 7ms/step - loss: 0.8710
Epoch: 36
10/10 [==============================] - 0s 9ms/step - loss: 0.8664
Epoch: 37
10/10 [==============================] - 0s 8ms/step - loss: 0.8618
Epoch: 38
10/10 [==============================] - 0s 8ms/step - loss: 0.8573
Epoch: 39
10/10 [==============================] - 0s 7ms/step - loss: 0.8529
Epoch: 40
10/10 [==============================] - 0s 10ms/step - loss: 0.8485
Epoch: 41
10/10 [==============================] - 0s 9ms/step - loss: 0.8442
Epoch: 42
10/10 [==============================] - 0s 9ms/step - loss: 0.8401
Epoch: 43
10/10 [==============================] - 0s 9ms/step - loss: 0.8361
Epoch: 44
10/10 [==============================] - 0s 7ms/step - loss: 0.8323
Epoch: 45
10/10 [==============================] - 0s 7ms/step - loss: 0.8287
Epoch: 46
10/10 [==============================] - 0s 8ms/step - loss: 0.8252
Epoch: 47
10/10 [==============================] - 0s 8ms/step - loss: 0.8219
Epoch: 48
10/10 [==============================] - 0s 8ms/step - loss: 0.8186
Epoch: 49
10/10 [==============================] - 0s 8ms/step - loss: 0.8154
Epoch: 50
10/10 [==============================] - 0s 8ms/step - loss: 0.8122
Epoch: 51
10/10 [==============================] - 0s 7ms/step - loss: 0.8089
Epoch: 52
10/10 [==============================] - 0s 7ms/step - loss: 0.8057
Epoch: 53
10/10 [==============================] - 0s 7ms/step - loss: 0.8024
Epoch: 54
10/10 [==============================] - 0s 7ms/step - loss: 0.7991
Epoch: 55
10/10 [==============================] - 0s 7ms/step - loss: 0.7957
Epoch: 56
10/10 [==============================] - 0s 7ms/step - loss: 0.7923
Epoch: 57
10/10 [==============================] - 0s 7ms/step - loss: 0.7889
Epoch: 58
10/10 [==============================] - 0s 7ms/step - loss: 0.7854
Epoch: 59
```

```
10/10 [==============================] - 0s 7ms/step - loss: 0.7819
```

In [23]:

```
def get_test_length(dataset, batch_size):

    test_length_values = []
    for x in range(len(dataset) - 200, len(dataset) - timesteps*2):
        modulo=(x-upper_train)%batch_size
        if (modulo == 0):
            test_length_values.append(x)
            print(x)
    return (max(test_length_values))
```

In [24]:

```
test_length = get_test_length(df, batch_size)
print(test_length)
upper_test = test_length + timesteps*2
testset_length = test_length - upper_train
print(testset_length)
```

```
286
318
350
382
414
446
446
96
```

In [25]:

```
print(upper_train, upper_test, len(df))
```

```
350 476 484
```

In [27]:

```
# construct test set

#subsetting
df_test = df[upper_train:upper_test]
test_set = df_test.iloc[:,2:3].values

#scaling
# scaled_real_bcg_values_test = sc.fit_transform(np.float64(test_set))

scaled_real_bcg_values_test=np.float64(test_set)

#creating input data
X_test = []
for i in range(timesteps, testset_length + timesteps):
    X_test.append(scaled_real_bcg_values_test[i-timesteps:i, 0])
X_test = np.array(X_test)


#reshaping
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
```

In [28]:

```
X_test.shape
```

Out[28]:

```
(96, 15, 1)
```

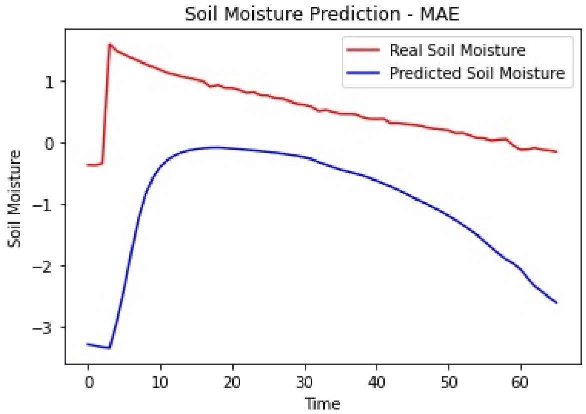
In [31]:

```
from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler(feature_range = (0, 1))
training_set_scaled = sc.fit_transform(np.float64(training_set))

#prediction
predicted_bcg_values_test_mae = regressor_mae.predict(X_test, batch_size=batch_size)
regressor_mae.reset_states()

print(predicted_bcg_values_test_mae.shape)

#reshaping
predicted_bcg_values_test_mae = np.reshape(predicted_bcg_values_test_mae,
                                        (predicted_bcg_values_test_mae.shape[0],
                                         predicted_bcg_values_test_mae.shape[1]))

print(predicted_bcg_values_test_mae.shape)
#inverse transform
predicted_bcg_values_test_mae = sc.inverse_transform(predicted_bcg_values_test_mae)


#creating y_test data
y_test = []
for j in range(0, testset_length - timesteps):
    y_test = np.append(y_test, predicted_bcg_values_test_mae[j, timesteps-1])

# reshaping
y_test = np.reshape(y_test, (y_test.shape[0], 1))

print(y_test.shape)
```

```
(96, 15, 1)
(96, 15)
(81, 1)
```

In [32]:

```
# Visualising the results
import matplotlib.pyplot as plt

plt.plot(test_set[timesteps:len(y_test)].astype(float), color = 'red', label = 'Real Soil Moisture')
plt.plot(y_test[0:len(y_test) - timesteps].astype(float), color = 'blue', label = 'Predicted Soil Moisture')
plt.title('Soil Moisture Prediction - MAE')
plt.xlabel('Time')
plt.ylabel('Soil Moisture')
plt.legend()
plt.show()
```



We can see prediction for 66 points in time

We can apply as model performance indicator MSE - mean squared error, to change respect MAE - mean absolute error that we use in the model of neural net

In [33]:

```
#MSE (mean squared error)
import math
from sklearn.metrics import mean_squared_error
rmse = math.sqrt(mean_squared_error(test_set[timesteps:len(y_test)], y_test[0:len(y_test) - timesteps]))
print(rmse)
```

```
1.7927620588947637
```

Model Performance express with MSE in 1st Iteration with manually normalized data is very low

epochs = 60 -> MSE = 1.7927620588947637

In second Iteration with MinMaxScaler normalization of data, performance is improved infact

epochs = 60 -> MSE = 0.5485747466827837

Now we try to improve model performance before increasing number of epochs, later changing model structure with an hidden layer of 10 neurons

epochs = 60 -> MSE = 0.5485747466827837

epochs = 120 -> MSE = 0.5479207378756074

epochs = 30 -> MSE = 0.875947556148119

epochs = 600 -> MSE = 0.41438538874002684

epochs = 6000 -> MSE = 0.3055890836518183

As we can see in the figure above an epochs = 6000 produce better results and minimize MSE value

(in cwd is stored model trained after MinMaxScaler normalization and an epochs value of 6000.

In notebook is visible model trained with manually normalization and an epochs value of 60)

In [199]:

```
#save model
import h5py
regressor_mae.save(filepath="my_model_with_mae.h5")
```

In [ ]:

```
#load model
import h5py
regressor_mae = load_model(filepath="my_model_with_mae.h5")
```