

Predicting soil moisture values 30 days ahead

Nasim Salehi

Dissertation submitted to International Business School for the partial fulfilment
of the requirement for the degree of MASTER OF SCIENCE IN IT FOR BUSINESS
DATA ANALYTICS

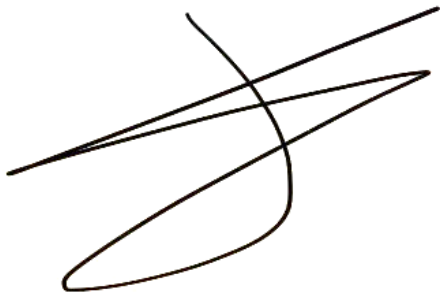
December 2022

DECLARATION

This dissertation is a product of my own work and is not the result of anything done in collaboration.

I consent to the University's free use including online reproduction, including electronically, and including adaptation for teaching and education activities of any whole or part item of this dissertation.

Signature

A handwritten signature in dark ink, consisting of several overlapping loops and a long horizontal stroke extending to the right.

Student name: Nasim Salehi

Word length: 13485 words

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my professors specially Zsófia Gyarmathy, for their great guidance, advice and feedback throughout this project. Without their guidance, it would be impossible for me to complete my project.

I am deeply grateful to my parents and my brother for their continuous support and patience during my studies.

From the bottom of my heart, a special thanks to my love, Hamid Reza. Without his support and encouragement during my study, it would be impossible for me to complete it.

I also would like to thank my friends and classmates who were always available whenever I need a helping hand.

ABSTRACT

Today, machine and deep learning algorithms have been successful in various fields. Soil moisture is one of the important factors in agriculture and water management. Forecasting soil moisture is usually complicated due to its different characteristics and time, and it is difficult to predict it by basic statistical methods. Predicting the amount of soil moisture can be considered as a regression and time problem in machine learning. In this work, the aim is to implement machine learning techniques such as K-Nearest-Neighbour algorithm, Decision Tree, Random forest and deep Long Short-Term Memory (LSTM) approach to predict the amount of soil moisture. Of course, in order to achieve high accuracy in predicting the amount of soil moisture, a series of methods have been used to prepare the data set. In general, the working method here consists of several important parts including: (1) reading the dataset, (2) Exploratory data analysis, (3) pre-processing stage, (4) feature selection stage, (5) Preparation data (for machine learning and deep learning), (6) resampling, (7) implementation stage of machine learning and deep learning models, (8) Comparison and evaluation stage. The results of various analyses and experiments show that among the machine learning methods, ExtraTreeRegressor and K-Nearest-Neighbour algorithm have obtained better results in terms of MSE, R2 criteria compared to other models. This LSTM method has also shown better results.

TABLE OF CONTENTS

ABSTRACT.....	IV
TABLE OF CONTENTS.....	V
LIST OF FIGURES.....	VII
LIST OF TABLES	VIII
1. INTRODUCTION	1
2. READ DATA	2
2.1. Download dataset	2
2.2. Read file csv	2
2.3. Fix and check dataset	3
3. EXPLORATY DATA ANALYSIS	6
3.1. Relationship between variables	7
3.2. Pair plot	10
4. PREPROCESSING	17
4.1. MinMaxScaler.....	18
4.2. StandardScaler.....	19
4.3. Outlier.....	20
4.3.1 Boxplot.....	20
4.3.2 Local Outlier Factor	27
5. TIME SERIES	28
6. FEATURE SELECTION	31
6.1. Random Forest (tree base).....	32
6.2. Linear regression (linear base).....	33
6.3. Final feature selection.....	35
7. POTENTIAL RESAMPLING	36
8. PREPARATION DATA (FOR MACHINE LEARNING AND DEEP LEARNING)	39
8.1. Candle model (past_history).....	40
8.2. Next model	41
9. TESTING ALL MODEL ON NEXT DATA	42
9.1. Train and test split.....	42
9.2. Machine learning model	43
9.2.1. KNeighborsRegressor	43
9.2.2. Linear Regression	44
9.2.3. DecisionTreeRegressor	44
9.2.4. RandomForestRegressor	45

9.2.5.	ExtraTreeRegressor	45
9.3.	Deep learning	46
9.3.1.	LSTM	46
9.4.	Results.....	49
10.	TESTING ALL MODEL ON TIME WINDOW DATA.....	51
10.1.	Train and test split.....	51
10.2.	Machine learning model	52
10.2.1.	KNeighborsRegressor	53
10.2.2.	LinearRegression	53
10.2.3.	DecisionTreeRegressor	53
10.2.4.	RandomForestRegressor	54
10.2.5.	ExtraTreeRegressor	54
10.3.	Deep learning	55
10.3.1.	LSTM	55
10.4.	Results.....	56
11.	EVALUATION AND REFLECTION	58
12.	CONCLUSIONS	59
13.	REFERENCES	60

LIST OF FIGURES

Figure 1 Correlation of all features	7
Figure 2 Measurement points back in time –relatively1	9
Figure 3 Measurement points back in time –relatively2	10
Figure 4 analyse the different features of the soil moisture dataset	11
Figure 5 analyse the target variable BF10-1	12
Figure 6 analyse the target variable BF10-2	13
Figure 7 exploratory analysis of VGSL, Month, TT_TER and TS05	15
Figure 8 the monthly exploratory analysis of VGSL, RF_TER, TT_TER and TS05.....	16
Figure 9 Min-Max-scaled plots.....	19
Figure 10 box plot-the features of VGSL, TT_TER, RF_TER outliers.....	21
Figure 11 box plot- removed outlier data from the soil moisture data set.....	24
Figure 12 Detection outlier just for VGSL with boxplot method	26
Figure 13 Detection outlier just for TT_TER with boxplot method.....	26
Figure 14 Detection outlier just for RF_TER with boxplot method.....	27
Figure 15 Local Outlier Factor with boxplot method.....	28
Figure 16 values of years about the situation of soil moisture	29
Figure 17 values of months about the situation of soil moisture	30
Figure 18 temporality and seasonality of the BF10 feature	31
Figure 19 Soil moisture feature importance's	33
Figure 20 Soil moisture feature importance's	34
Figure 21 resample_BF10 feature and the predictor variable1.....	36
Figure 22 resample_BF10 feature and the predictor variable2.....	37
Figure 23 feature TT_TER before and after resampling.....	38
Figure 24 feature RF_TER before and after resampling	39
Figure 25 time window with a time step	40
Figure 26 Train and test plot-next data	43
Figure 27 Algoithm of MSE-RMSE-MAE-R2.....	51
Figure 28 Train and test plot_time window data.....	52
Figure 29 Algoithm of MSE-RMSE-MAE-R2(time window data).....	58

LIST OF TABLES

Table 1 , The munster_hourly data set	3
Table 2 fix and check dataset for pre-processing and normalization 1	4
Table 3 fix and check dataset for pre-processing and normalization 2	5
Table 4 exploratory data on soil moisture data	6
Table 5 normalization of the dataset	18
Table 6 Min-Max-scaled descriptive1	19
Table 7 Min-Max-scaled descriptive2	19
Table 8 filter of VGSL feature	22
Table 9 filter of TT_TER feature	22
Table 10 filter of RF_TER feature	23
Table 11 data frame of final outliers	31
Table 12 dataframe of final feature without QN_4 and day	35
Table 13 data frame of final feature_resampling	37
Table 14 Algorithm_MSE_RMSE_MAE_R2_1(next data)	50
Table 15 Algorithm_MSE_RMSE_MAE_R2_2(next data)	50
Table 16 Algorithm_MSE_RMSE_MAE_R2_1(time window data)	56
Table 17 Algorithm_MSE_RMSE_MAE_R2_2(time window data)	57
Table 18 result based on reg1	58
Table 19 result based on reg2	59

1. INTRODUCTION

The topic of this research is related to predicting the amount of soil moisture using machine learning methods and deep learning methods. As a result, it can be said that the main goal of this research is to implement an important machine learning algorithm including the K-Nearest-Neighbour algorithm, Decision Tree, Random forest, Linear Regression, ExtraTreeRegressor and the deep approach of Long Short-Term Memory (LSTM) to predict the amount of soil moisture. For the next 30 days, it is in the Google colab environment and in the Python programming language. We carried out the detailed implementation of this method in several detailed steps of this implementation, which in summary includes the following.

The first step in the project involves downloading and reading the authoritative collection of soil moisture. In the second step, a series of exploratory analyses will be performed on the soil moisture data set, where the goal is to investigate the correlation and auto-correlation of soil moisture features, and then hexbin and scatter charts will be used for more detailed analysis. In the third step, the initial stage of pre-processing (two types of normalization, StandardScaler and MinMaxScaler) and data preparation for the time mode will be done on the dataset, and outlier data will be removed with boxplot and Local Outlier Factor methods. Of course, the final method of removing outlier data will be the Local Outlier Factor method. In the fourth step, the soil moisture data set was analysed in terms of time series and time analysis, where the p-value for the BF10 feature was investigated. In the fifth step, we performed two feature selection methods including the feature selection method based on linear regression and Random Forest (tree base) to identify some important features in the data set. In the sixth step, we used the resampling method to convert the data into daily time. In the seventh step, we proposed and implemented two time data preparation methods, and finally, in the eighth and ninth steps, all machine learning and deep learning algorithms have been evaluated in terms of different criteria.

According to the mentioned cases, it is possible to imagine different hypotheses for research (such as machine learning methods can well predict the amount of soil moisture, data scaling methods have a positive effect on the output of some machine learning methods). But in general, it is expected that deep and machine learning methods can better predict soil moisture.

2. READ DATA

In this section, our main goal is:

First download the soil moisture dataset, which is in the link below (this is done directly in section 2.1.: [Soil and weather variables in the German city of Münster](#) (Data source: [DWD](#))

After downloading the dataset, in the next step, dataset reading has been done using the pandas library (section 2.2).

Finally, some changes have been made on the dataset for better dataset preparation (section 2.3.).

2.1. Download dataset

In the section below, the soil moisture dataset has been downloaded directly from the Google Drive link with the `gdown` command.

After downloading the data, it can be accessed as `munster_hourly.csv`

Of course, sometimes the following command can also be used:

```
# !wget https://drive.google.com/uc?export=download&id=173tvWt-qPyAqZJclKoBiUHgoSg6nLUtT -O data.csv
```

```
!gdown 173tvWt-qPyAqZJclKoBiUHgoSg6nLUtT
```

```
Downloading...
```

```
From: https://drive.google.com/uc?id=173tvWt-qPyAqZJclKoBiUHgoSg6nLUtT
```

```
To: /content/munster_hourly.csv
```

```
100% 1.76M/1.76M [00:00<00:00, 184MB/s]
```

2.2. Read file csv

In sub-section, the `munster_hourly` data set is read by the `pandas` library and converted into a data frame, and various operations on the data frame are easily possible.

Columns of the dataset are below:

column name	description
DATUM	The datetime column
STATIONS_ID	DWD weather station ID
QN_4	quality level of TT_TER and RF_TER columns [1,2,..10]
TT_TER	air temperature
RF_TER	relative humidity
VGSL	real evapotranspiration over gras and sandy loam (mm)
TS05	mean daily soil temperature in 5 cm depth for uncovered typical soil (°C)
RF_TER	relative humidity
BF10	soil moisture under grass and sandy loam between 0 and 10 cm depth in % plant useable water (%nFK)

```
import pandas as pd
# read data from path download
path_file='/content/munster_hourly.csv'
df_main=pd.read_csv(path_file)
df_main.head(10)
```

Table 1, The munster_hourly data set

	DATUM	STATIONS_ID	QN_4	TT_TER	RF_TER	VGSL	TS05	BF10
0	1991-01-01 07:00:00	1766	10	3.0	91.0	0.3	2.9	102
1	1991-01-01 14:00:00	1766	10	4.8	85.0	0.3	2.9	102
2	1991-01-01 21:00:00	1766	10	3.9	82.0	0.3	2.9	102
3	1991-01-02 07:00:00	1766	10	5.6	94.0	1.4	6.3	110
4	1991-01-02 14:00:00	1766	10	11.0	87.0	1.4	6.3	110
5	1991-01-02 21:00:00	1766	10	11.0	91.0	1.4	6.3	110
6	1991-01-03 07:00:00	1766	10	7.2	87.0	1.0	6.6	110
7	1991-01-03 14:00:00	1766	10	9.4	69.0	1.0	6.6	110
8	1991-01-03 21:00:00	1766	10	6.6	91.0	1.0	6.6	110
9	1991-01-04 07:00:00	1766	10	6.7	79.0	1.2	5.5	106

2.3. Fix and check dataset

In this sub-section, we are looking for some basic changes and corrections on the data so that the dataset is ready for pre-processing and normalization.

This includes the following:

- convert DATUM to datetime types
- rename DATUM to Date names
- extract new feature from Date as year,month,day names
- check NA values
- set Date column to Dataframe index

```
# convert DATUM to datetime and rename Date
df_main=df_main.astype({'DATUM':'datetime64[ns]'})
df_main = df_main.rename(columns={'DATUM':'Date'})
df_main.dtypes

Date datetime64[ns]
STATIONS_ID int64
QN_4 int64
TT_TER float64
RF_TER float64
VGSL float64
TS05 float64
BF10 int64
```

```
dtype: object
```

```
df_main.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 35672 entries, 0 to 35671
Data columns (total 8 columns):
 #   Column          Non-Null Count  Dtype
---  -
 0   Date             35672 non-null  datetime64[ns]
 1   STATIONS_ID      35672 non-null  int64
 2   QN_4             35672 non-null  int64
 3   TT_TER           35672 non-null  float64
 4   RF_TER           35672 non-null  float64
 5   VGSL             35672 non-null  float64
 6   TS05             35672 non-null  float64
 7   BF10             35672 non-null  int64
dtypes: datetime64[ns](1), float64(4), int64(3)
memory usage: 2.2 MB
```

```
df_main.describe(include='all')
```

Table 2 fix and check dataset for pre-processing and normalization 1

	DATE	STATIONS_ID	QN_4	TT_TER	RF_TER	VGSL	TS05	BF10
COUNT		35672	35672	35672.0000	35672.0000	35672.0000	35672.0000	35672.0000
T			.0	00	00	00	00	00
UNIQUE		35672	NaN	NaN	NaN	NaN	NaN	NaN
TOP	1991-01-01 07:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN
FREQ	1	NaN	NaN	NaN	NaN	NaN	NaN	NaN
FIRST	1991-01-01 07:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN
LAST	2021-12-31 00:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN
MEAN		NaN	1766. 0	8.641540	10.755136	76.796675	1.236676	11.096577
STD		NaN	0.0	2.772131	7.591943	18.688003	0.978120	7.684733
MIN		NaN	1766. 0	1.000000	-21.300000	- 999.000000	0.000000	-8.000000
25%		NaN	1766. 0	10.000000	5.200000	65.000000	0.500000	4.600000
50%		NaN	1766. 0	10.000000	10.600000	81.000000	1.000000	10.900000
75%		NaN	1766. 0	10.000000	16.100000	91.000000	1.800000	17.500000

```
## extract feature from Data
```

```
df_main["year"] = df_main["Date"].astype("str").map(lambda x: x[:4]).values.astype(int)
df_main["month"] = df_main["Date"].astype("str").map(lambda x: x[5:7]).values.astype(int)
df_main["day"] = df_main["Date"].astype("str").map(lambda x: x[8:10]).values.astype(int)
```

```
#check the NA values
```

```
nulls = df_main.isnull().sum()
for key,value in nulls.items():
    print(key, "counts NA: ",value)
```

```

Date counts NA: 0
STATIONS_ID counts NA: 0
QN_4 counts NA: 0
TT_TER counts NA: 0
RF_TER counts NA: 0
VGSL counts NA: 0
TS05 counts NA: 0
BF10 counts NA: 0
year counts NA: 0
month counts NA: 0
day counts NA: 0

```

```

#Ok, let's correct this!
print("After dropping all NA values...")
df_main.dropna(inplace=True)

assert not df_main.isnull().sum().any()
print("The run continued without problems.")

```

```

After dropping all NA values...
The run continued without problems.

```

```
df_main.set_index('Date', inplace=True)
```

```
df_main
```

Table 3 fix and check dataset for pre-processing and normalization 2

	QN_4	TT_TER	RF_TER	VGSL	TS05	BF10	YEAR	MONTH	DAY	
STATIONS_ID										
DATE										
1991-01-01 07:00:00	1766	10	3.0	91.0	0.3	2.9	102	1991	1	1
1991-01-01 14:00:00	1766	10	4.8	85.0	0.3	2.9	102	1991	1	1
1991-01-01 21:00:00	1766	10	3.9	82.0	0.3	2.9	102	1991	1	1
1991-01-02 07:00:00	1766	10	5.6	94.0	1.4	6.3	110	1991	1	2
1991-01-02 14:00:00	1766	10	11.0	87.0	1.4	6.3	110	1991	1	2
...
2021-12-30 00:00:00	1766	1	11.6	90.0	0.8	9.4	104	2021	12	30
2021-12-30 06:00:00	1766	1	11.1	98.0	0.8	9.4	104	2021	12	30
2021-12-30 12:00:00	1766	1	14.3	83.0	0.8	9.4	104	2021	12	30
2021-12-30 18:00:00	1766	1	13.5	90.0	0.8	9.4	104	2021	12	30
2021-12-31 00:00:00	1766	1	13.8	83.0	0.8	10.5	101	2021	12	31

35672 rows x 10 columns

```

df_main.dtypes
STATIONS_ID int64
QN_4 int64
TT_TER float64
RF_TER float64
VGSL float64

```

```

TS05 float64
BF10 int64
year int64
month int64
day int64
dtype: object

```

3. EXPLORATY DATA ANALYSIS

In this section, the goal is to analyse exploratory data on soil moisture data. For better understanding, we have divided this section into several sub-sections. In the first section, our goal is to examine the relationship and correlation between the features, which is done by correlation in the pandas library. The second part is related to the analysis of different features of the dataset, and in this part, it is tried to analyse the exploratory data of different features and their relationship with the output feature (BF10).

```

# sort columns name
col_names=['STATIONS_ID', 'QN_4', 'TT_TER', 'RF_TER', 'VGSL', 'TS05',
           'year', 'month', 'day','BF10']
df_main['STATIONS_ID'] =df_main['STATIONS_ID'].astype(float)
df_plot=df_main[col_names]
df_main=df_main[col_names]
df_plot

```

Table 4 exploratory data on soil moisture data

	STATIONS_ID	QN_4	TT_TER	RF_TER	VGSL	TS05	YEAR	MONTH	DAY	BF10
DATE										
1991-01-01 07:00:00	1766.0	10	3.0	91.0	0.3	2.9	1991	1	1	102
1991-01-01 14:00:00	1766.0	10	4.8	85.0	0.3	2.9	1991	1	1	102
1991-01-01 21:00:00	1766.0	10	3.9	82.0	0.3	2.9	1991	1	1	102
1991-01-02 07:00:00	1766.0	10	5.6	94.0	1.4	6.3	1991	1	2	110
1991-01-02 14:00:00	1766.0	10	11.0	87.0	1.4	6.3	1991	1	2	110
...
2021-12-30 00:00:00	1766.0	1	11.6	90.0	0.8	9.4	2021	12	30	104
2021-12-30 06:00:00	1766.0	1	11.1	98.0	0.8	9.4	2021	12	30	104
2021-12-30 12:00:00	1766.0	1	14.3	83.0	0.8	9.4	2021	12	30	104
2021-12-30 18:00:00	1766.0	1	13.5	90.0	0.8	9.4	2021	12	30	104
2021-12-31 00:00:00	1766.0	1	13.8	83.0	0.8	10.5	2021	12	31	101

35672 rows × 10 columns

3.1. Relationship between variables

In this subsection, at first, we aim to analyse the correlation analysis of different features of the soil moisture data set. Therefore, at first, the degree of correlation of all features such as `QN_4`, `TT_TER`, etc. is checked with respect to the output variable or prediction variable BF10, and then all the features will be compared to each other. Here, for the convenience of implementation, this work is done by correlation in the pandas library.

```
import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=(10,10))
cor = df_plot.corr()
sns.heatmap(cor, annot=True, cmap=plt.cm.Blues)
plt.title("correlation all feature")
plt.show()
```

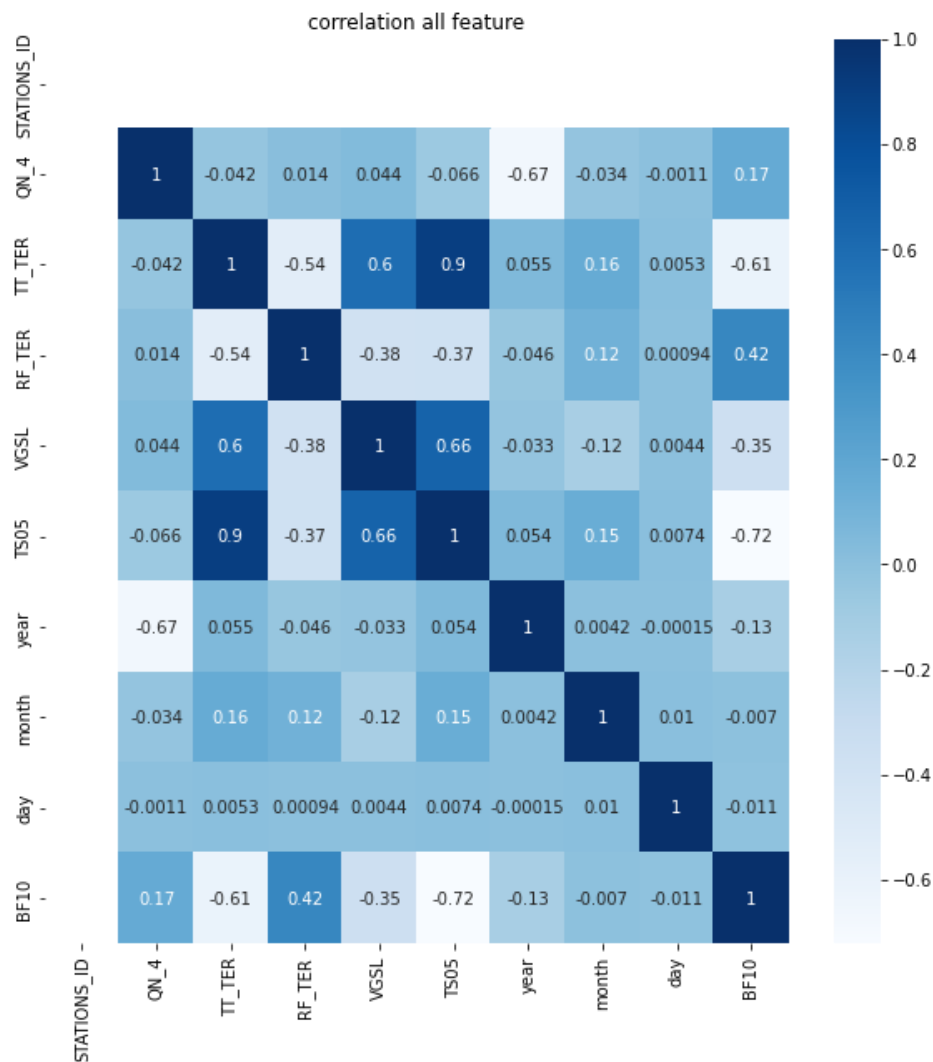


Figure 1 Correlation of all features

```

import numpy as np
cor_abs=np.abs(cor.iloc[1:,-1])
idx_sort=np.argsort(cor_abs)
sort_cor=np.round(cor_abs[idx_sort],4)
col_names_new=np.array(col_names[1:])
col_names_sort=col_names_new[idx_sort]
for i, item in enumerate(col_names_sort):
    print('number: {0} , columns: {1} cor: {2}'.format(i,item,sort_cor[i]))
number: 0 , columns: month cor: 0.007
number: 1 , columns: day cor: 0.0111
number: 2 , columns: year cor: 0.129
number: 3 , columns: QN_4 cor: 0.1719
number: 4 , columns: VGSL cor: 0.3512
number: 5 , columns: RF_TER cor: 0.4236
number: 6 , columns: TT_TER cor: 0.6124
number: 7 , columns: TS05 cor: 0.7213
number: 8 , columns: BF10 cor: 1.0

```

The results of the correlation diagram in the above diagram show the relationship between the features themselves and their relationship with the target variable BF10 in the form of a heatmap diagram along with its values, which shows that in relation to the target variable BF10, the `TS05` feature with a value of 0.72 and the TT_TER feature with a value of 0.61, the `RF_TER` feature with a value of 0.42, `VGSL` feature with a value of 0.35, `QN_4` feature with a value of 0.17 and finally year feature with a value of 0.12 showed correlation. Of course, a variable like `STATIONS_ID` has no correlation due to its fixed value, and features like `day` and `month` have very little correlation compared to the target variable BF10.

In addition, there is also a relationship between the features in the soil moisture data set. So that there is a 66% correlation between the two features of `VGSL` and `TS05`, and there is also a correlation of 66% between the two features of `VGSL` and `TT_TER`. This shows that feature reduction methods such as `PCA` and `feature selection` methods can be effective in reducing data dimensions.

```

# drop STATIONS_ID from dataset
df_plot=df_plot.drop('STATIONS_ID',axis=1)
df_main=df_main.drop('STATIONS_ID',axis=1)

```

In the continuation of this section, our goal is to further examine the target variable BF10 to show how this variable is in `Autocorrelation` and in different lags. `Autocorrelation` A function that calculates `autocorrelation` in terms of a time interval between observations. In fact, `Autocorrelation` is used to detect or identify the structure of time data, whose function is known as `AFC`, and the function of this function is to calculate the correlation between the target variable data of `BF10` according to the desired delay.

```

a=[]
for lag in [0,1,100,200,500,600,10000,20000]:
    print("Autocorrelation with lag", lag, " is\t",
          df_plot["BF10"].autocorr(lag) )

```


a.append(df_plot["BF10"].autocorr(lag))		
Autocorrelation with lag 0	is	1.0
Autocorrelation with lag 1	is	0.9918921949488575
Autocorrelation with lag 100	is	0.5830174393312971
Autocorrelation with lag 200	is	0.2974059055896101
Autocorrelation with lag 500	is	-0.405293966525784
Autocorrelation with lag 600	is	-0.45888075038459936
Autocorrelation with lag 10000	is	0.22737968229196637
Autocorrelation with lag 20000	is	-0.04878072722199644

```

from statsmodels.graphics.tsaplots import plot_acf
plt.close()
plot_acf(df_plot["BF10"].values, lags=300, zero=False)
plt.xlabel("Measurement points back in time - relatively")
plt.show()

```

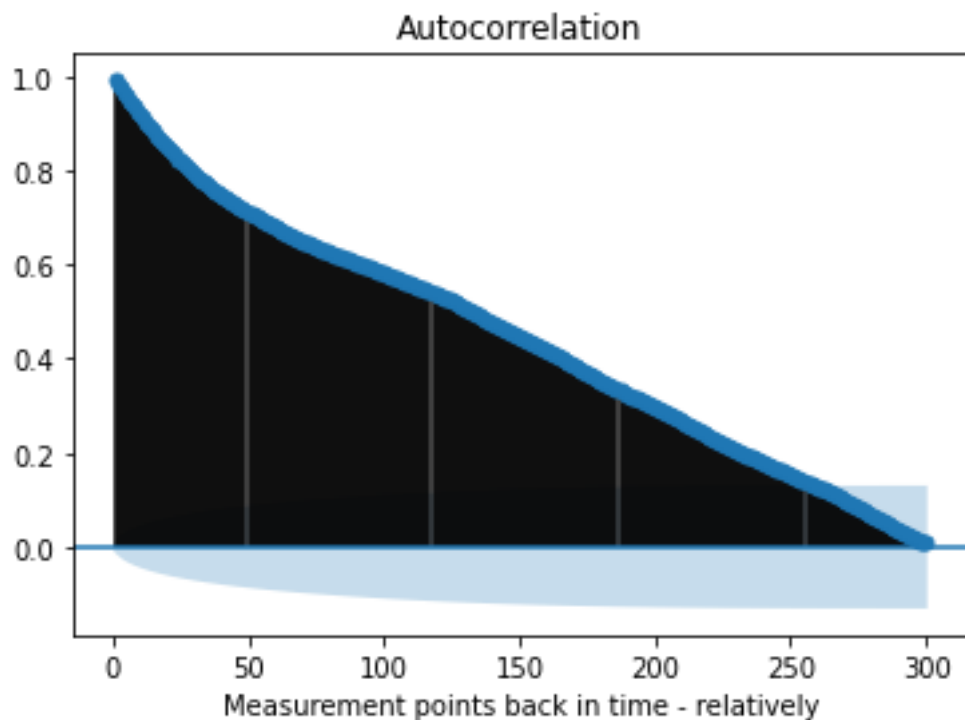


Figure 2 Measurement points back in time –relatively1

```

from statsmodels.graphics.tsaplots import plot_acf
plt.close()
plot_acf(df_plot["BF10"].values, lags=20000, zero=False)
plt.xlabel("Measurement points back in time - relatively")
plt.show()

```

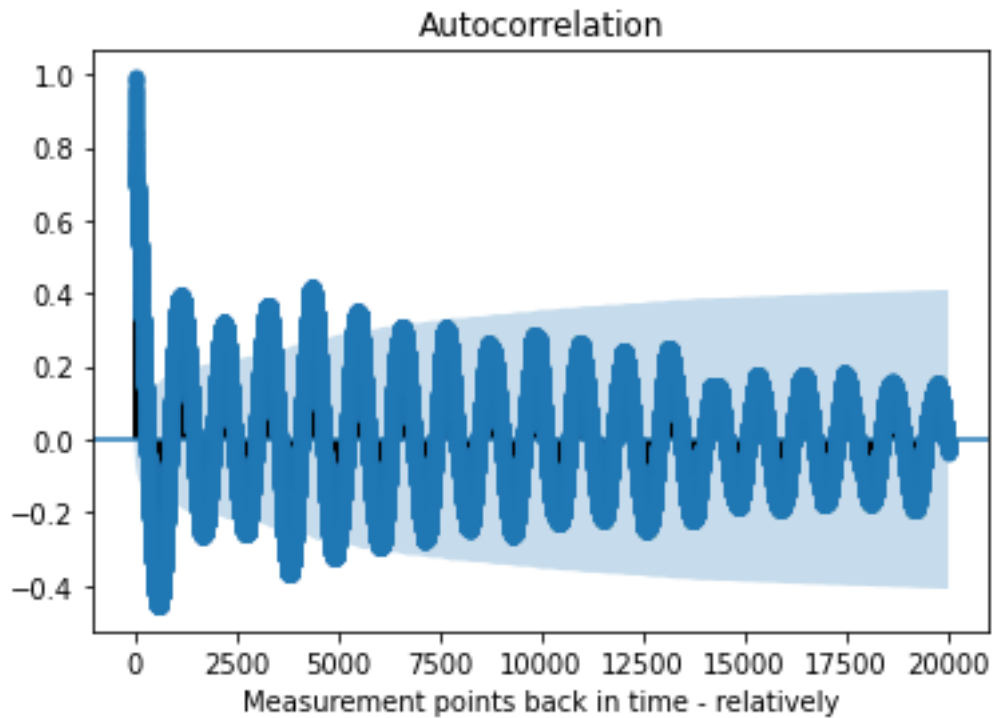


Figure 3 Measurement points back in time –relatively2

Autocorrelation results show that the target variable structure of BF10 has more fluctuations based on different lags, so that 20000 also shows itself. Therefore, it can be concluded that the target variable BF10 has a time structure with different delays. As can be seen, the autocorrelation coefficient for lag 0 (which means series correlation with itself) is equal to 1. Accordingly, with the increase of delays, the correlation value decreases and increases again, but in the end it approaches zero.

3.2. Pair plot

In this section, our goal is to analyse the different features of the dataset. At first, the pair plot diagram is drawn for all the characteristics of the soil moisture dataset, and then the diagrams were drawn and analysed separately with regard to the target variable BF10.

```
# importing packages
import seaborn
# pairplot with hue sex
seaborn.pairplot(df_plot, hue = 'BF10')
# to show
plt.show()
```

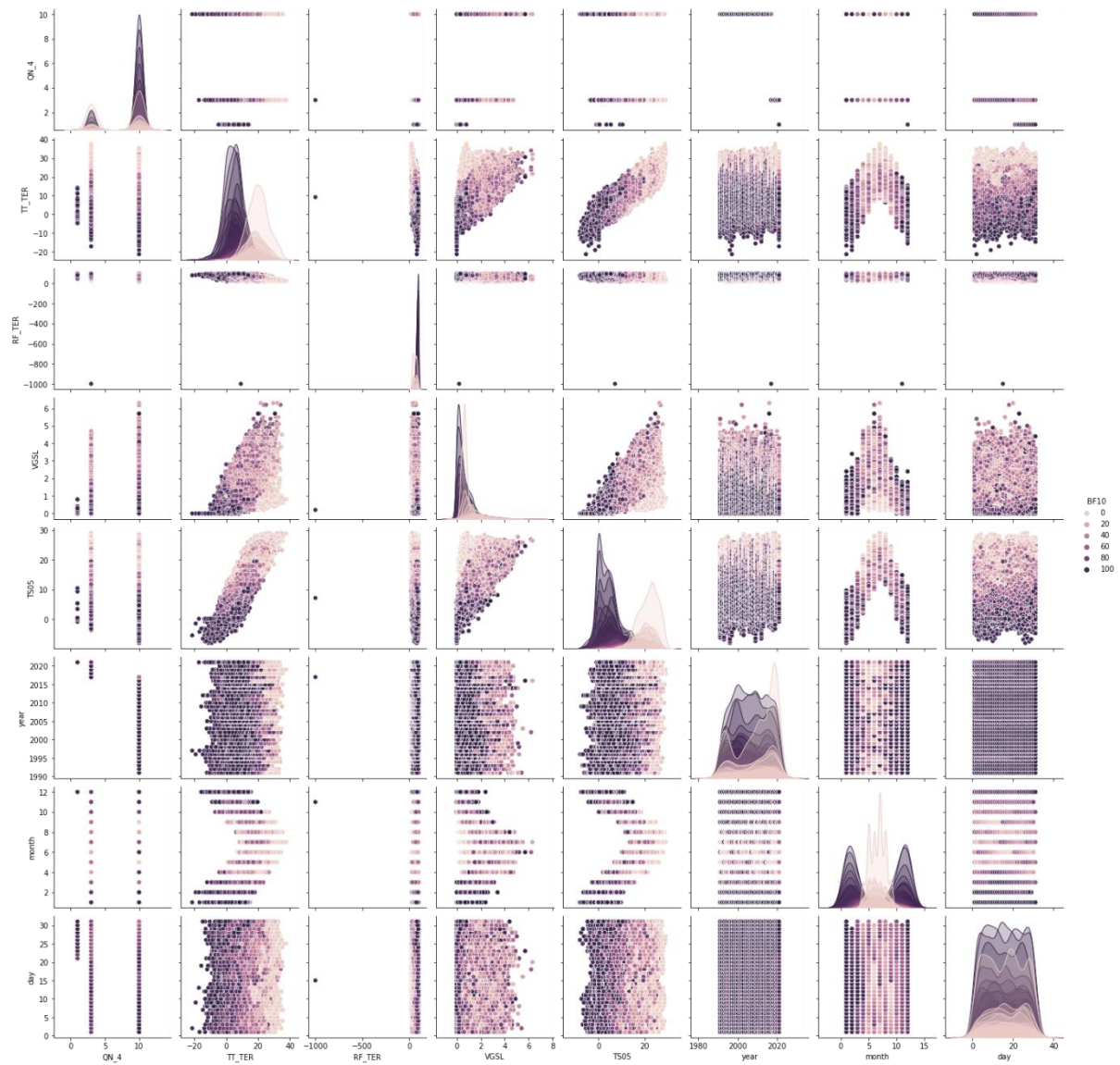


Figure 4 analyse the different features of the soil moisture dataset

```
df_plot.plot(subplots=True,figsize=(10,20));
plt.tight_layout(); # <- this is often necessary when using subplots
to avoid
```

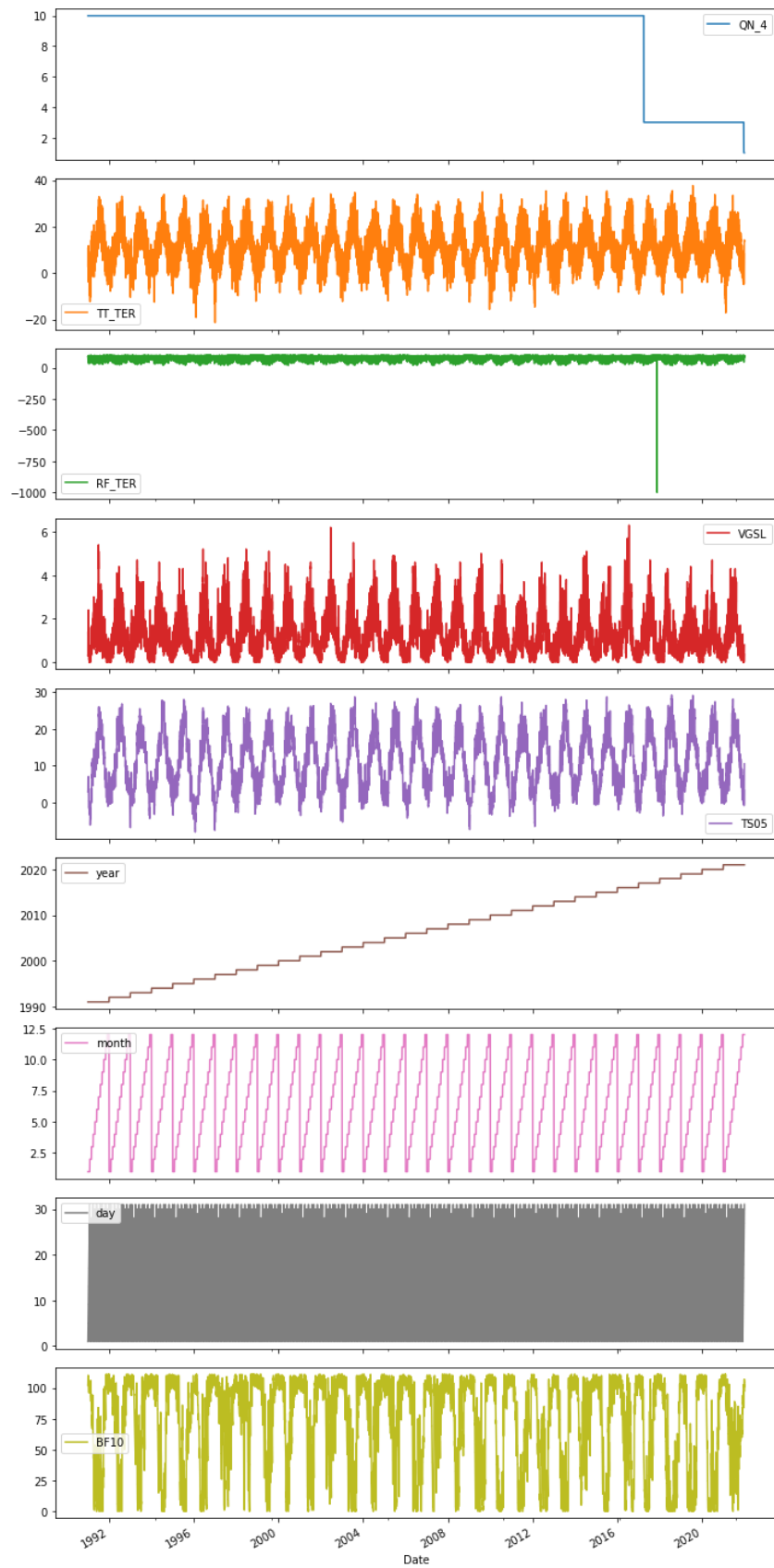


Figure 5 analyse the target variable BF10-1

```
# replace RF_TER -999 with mean of RF_TER
df_plot['RF_TER'][df_plot['RF_TER']<0]=df_plot['RF_TER'].mean()
```

```
df_main['RF_TER'][df_main['RF_TER']<0]=df_main['RF_TER'].mean()
```

```
# create new plots after drop noise from RF_TER
df_plot.plot(subplots=True,figsize=(10,20));
plt.tight_layout(); # <- this is often necessary when using subplots
to avoid
```

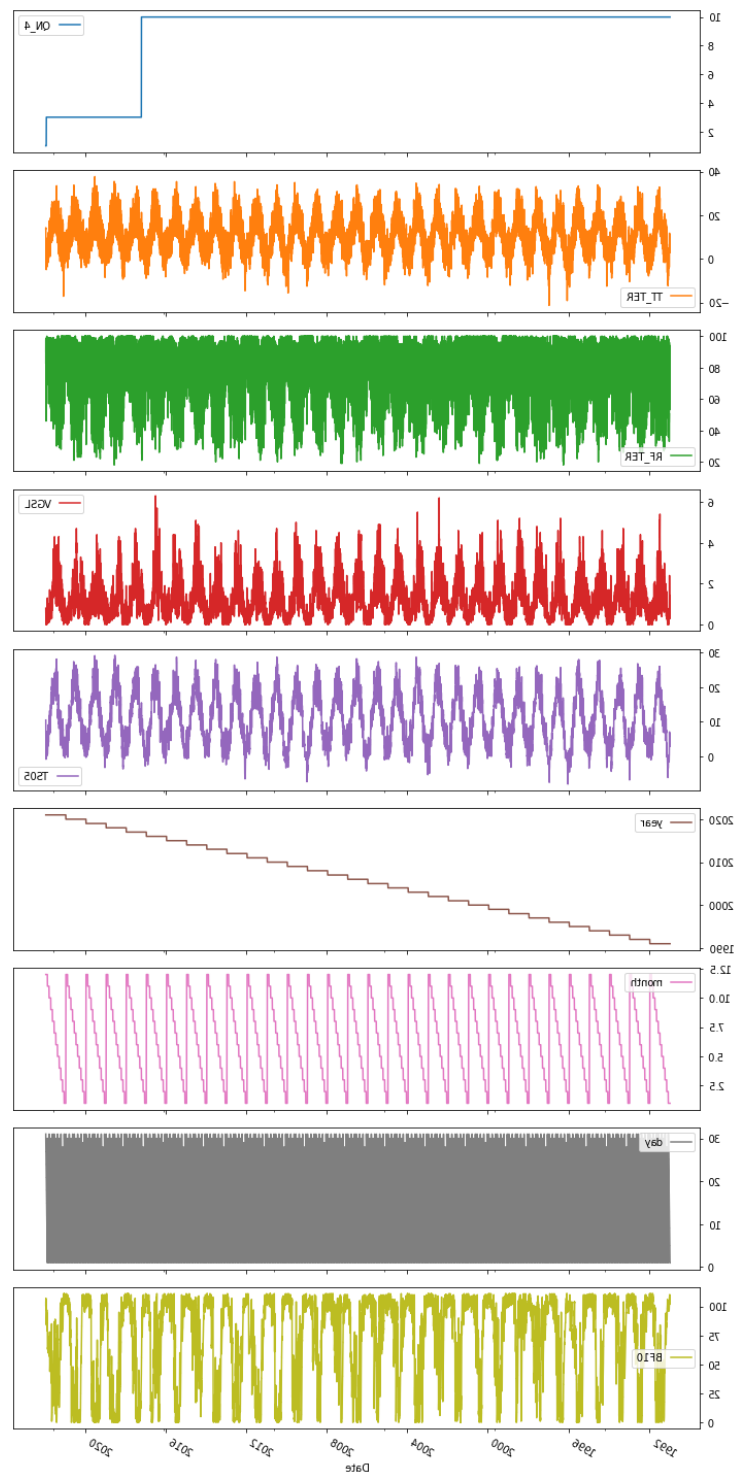
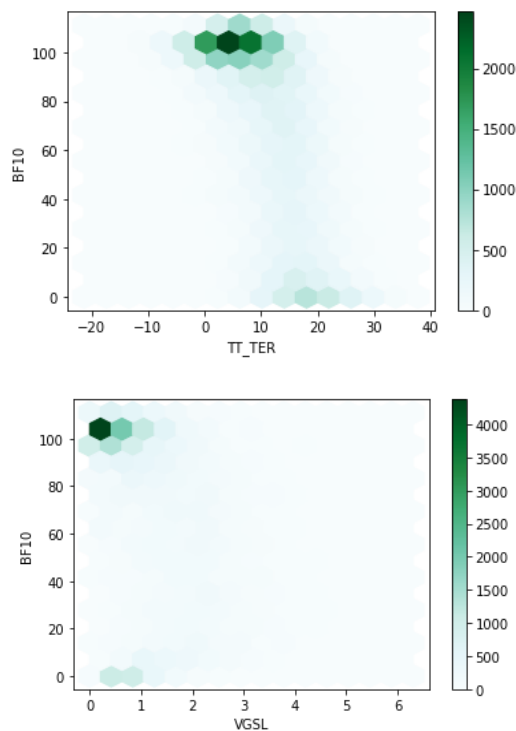


Figure 6 analyse the target variable BF10-2

The above plots show that some features of the soil moisture dataset have a regular and repeated structure. From the method of the month chart, it can be seen that for each year its value starts from 1 and then ends at 12, and on the other hand, for the year chart, it has gradually increased. As for the VGSL charts, TT_TER and TS05 have a repeating annual chart and can somehow contribute a lot to the BF10 target variable pre-value. Among the plots, the RF_TER plot has a significant noise, and here we used the Mean of this feature to remove this noise. The reason for this is to have a better view of the RF_TER plot and also to be able to remove the noise data manually.

```
plt.figure()
df_plot.plot(kind="hexbin", x=df_plot.columns[1], y=df_plot.columns[-1], gridsize=15, sharex=False);
df_plot.plot(kind="hexbin", x=df_plot.columns[3], y=df_plot.columns[-1], gridsize=15, sharex=False);
df_plot.plot(kind="hexbin", x=df_plot.columns[4], y=df_plot.columns[-1], gridsize=15, sharex=False);
df_plot.plot(kind="hexbin", x=df_plot.columns[6], y=df_plot.columns[-1], gridsize=15, sharex=False);
```



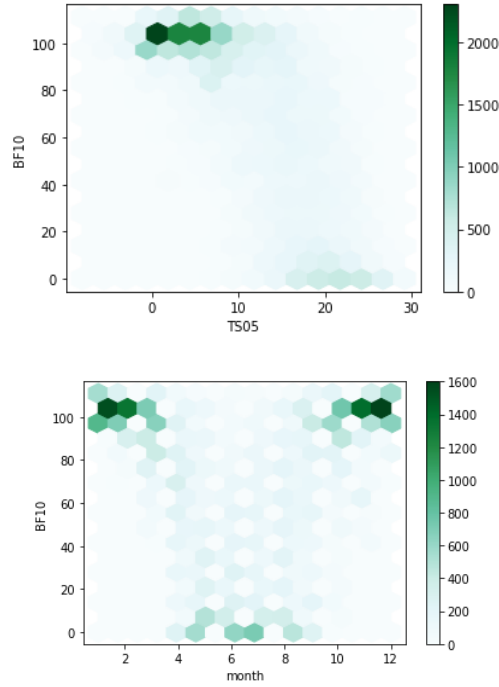


Figure 7 exploratory analysis of VGSL, Month, TT_TER and TS05

In the continuation of this section, we did an exploratory analysis of some main variables including VGSL, Month, TT_TER and TS05. The purpose of this study is to show these variables with the help of hexbin diagram and then analyse it.

The hexbin chart related to the Month variable shows that the first and last months of the year are very similar, and on the other hand, the middle months of the year (months 4 to 9) are very similar in terms of BF10 values. So that soil moisture near 100 is observed in more samples in the first, second, eleventh and twelfth months, and also in the sixth and seventh months, the amount of moisture is less.

The hexbin diagram related to the TS05 variable shows that the difference between the values less than 10 and greater than 10 shows itself in terms of the BF10 values, and also the TS05 feature with values of 7, 8, 9, 10 has the highest value close to 100 for the feature. They have BF10 and it is clearly visible in the diagram. Of course, all these things can help in predicting the exact value of BF10.

The hexbin diagram related to the TT_TER variable shows that the values 0 to 20 for the TT_TER variable have different values in terms of the BF10 value, and also the TT_TER attribute with values from 0 to 20 have the highest value close to 100 for the BF10 attribute and in the diagram it is completely clear. Of course, all these things can help in predicting the exact value of BF10

The hexbin diagram related to the VGSL variable shows that the difference between values less than 3 and greater than 3 shows itself in terms of BF10 values, and also the VGSL feature with values from 0 to 1 has the highest value close to 100 for the BF10 feature and The diagram is very clear. Of course, all these things can help in predicting the exact value of BF10

```
for i in range(1,5):
    df_plot.plot(df_plot.columns[6], df_plot.columns[i], kind="scatter")
```

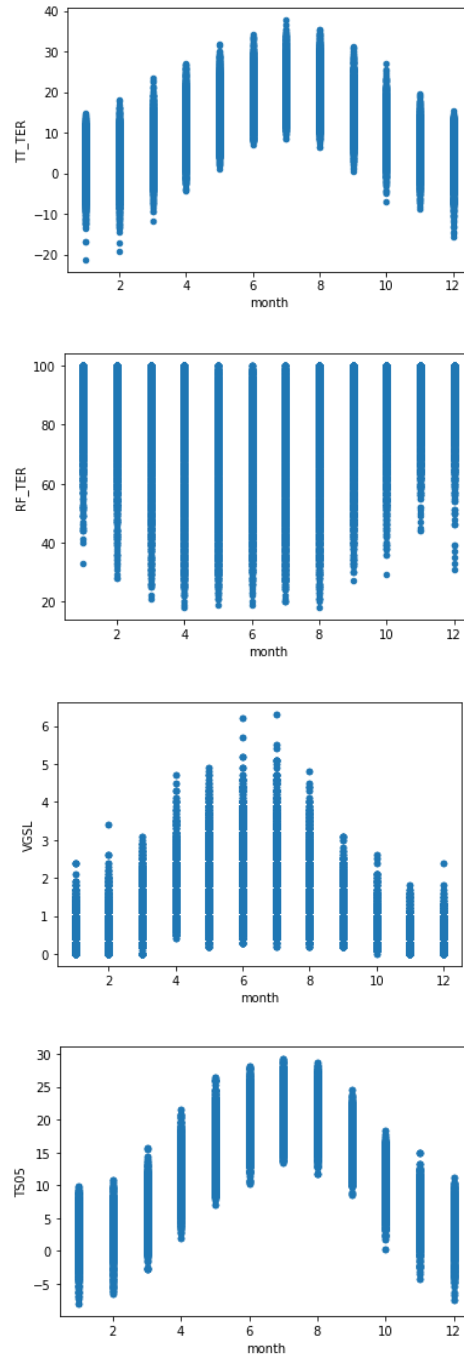


Figure 8 the monthly exploratory analysis of VGSL, RF_TER, TT_TER and TS05

In the continuation of this section, our goal is to examine the monthly exploratory analysis of some main variables including VGSL, RF_TER, TT_TER and TS05. Therefore, according to the above graphs, the following results can be obtained:

The graph related to the variable TT_TER shows that the values of this feature gradually increase

until it reaches the highest value in months 7 and 8, and then this variable decreases and even in the months The end of the year can take negative values. All these things can help in predicting the exact value of BF10.

The graph related to the RF_TER variable, unlike the TT_TER feature, shows that the values of this feature gradually decrease until it reaches the lowest value in months 7 and 8, and then this variable increases and even In the last months of the year, it can reach more than 20. All these things can help in predicting the exact value of BF10

The graph related to the VGSL variable, like the TT_TER feature, shows that the values of this feature gradually increase until it reaches the highest value in the 6th and 7th months, and then this variable takes a decreasing value. Even in the last months of the year, it can take zero values. All these things can help in predicting the exact value of BF10.

The graph related to the variable TS05 shows that the values of this feature gradually increase until it reaches the highest value in months 6 and 7, and then this variable decreases and even in the months the end of the year can take negative values. This feature shows that many of its values depend on different months. All these things can help in predicting the exact value of BF10.

4. PREPROCESSING

Scaling or normalization of the dataset is one of the important steps that should be done before running machine learning methods on the dataset so that the data are in a similar range. For example, if we use a linear regression to predict soil moisture. If we do not scale the features, it will cause some features to have a greater impact or weight (of course, without any special rule, but according to their values) in predicting soil moisture and cause problems for the performance of the linear regression algorithm. . Therefore, scaling or normalization is an important step in data mining. Of course, there are many methods for data normalization, such as min-max normalization, z-score normalization, etc.

In this implementation, we used min-max and standard-scaled methods for data normalization. Since in this implementation we used linear regression and deep method to predict soil moisture, the MinMaxScaler method is more compatible due to linear transformation on the original data. However, in a different test with two min-max and standard-scaled methods in this implementation, the results show that the MinMaxScaler method is better.

```
use_normalization=0# 0 : MinMaxScaler 1:StandardScaler
df_numeric=df_main.copy()
df_numeric
```

Table 5 normalization of the dataset

QN_4	TT_TER	RF_TER	VGSL	TS05	YEAR	MONTH	DAY	BF10
DATE								
1991-01-01 07:00:00	10	3.0	91.0	0.3	2.9	1991	1	1 102
1991-01-01 14:00:00	10	4.8	85.0	0.3	2.9	1991	1	1 102
1991-01-01 21:00:00	10	3.9	82.0	0.3	2.9	1991	1	1 102
1991-01-02 07:00:00	10	5.6	94.0	1.4	6.3	1991	1	2 110
1991-01-02 14:00:00	10	11.0	87.0	1.4	6.3	1991	1	2 110
...
2021-12-30 00:00:00	1	11.6	90.0	0.8	9.4	2021	12	30 104
2021-12-30 06:00:00	1	11.1	98.0	0.8	9.4	2021	12	30 104
2021-12-30 12:00:00	1	14.3	83.0	0.8	9.4	2021	12	30 104
2021-12-30 18:00:00	1	13.5	90.0	0.8	9.4	2021	12	30 104
2021-12-31 00:00:00	1	13.8	83.0	0.8	10.5	2021	12	31 101

35672 rows x 9 columns

4.1. MinMaxScaler

MinMaxScaler performs a linear transformation on the original data. Assume that Min_a and Max_a are the minimum and maximum values for attribute A. Min-max normalization of a value v from A to v' in the range ($NewMin_a$, $NewMax_a$) can be done according to the relation (1). To customize the normalization output to a desired scale, the range transformation method can be applied. In this thesis, the new range is set for all features between the new limit of 0 and 1.(Akanbi et al., 2015)

$v' = \frac{v - Min_a}{Max_a - Min_a} \times (NewMax_a - NewMin_a) + NewMin_a$	(1)
<pre> import matplotlib.pyplot as plt from sklearn.preprocessing import MinMaxScaler df_numeric=df_main.copy() if(use_normalization==0): print("Original descriptives:") display(df_numeric.describe().round(2)) print("\nMin-Max-scaled descriptives:") scaler = MinMaxScaler() scaler.fit(df_numeric) data_scaled = scaler.transform(df_numeric) df_scaled = pd.DataFrame(data_scaled, index=df_numeric.index, columns=df_numeric.columns) display(df_scaled.describe().round(2)) data_normal=df_scaled fig, axs = plt.subplots(ncols=2, figsize=(15,6)) df_numeric.plot.box(ax=axs[0], title="original") df_scaled.plot.box(ax=axs[1], title="scaled") plt.tight_layout() plt.show() else: </pre>	

```
print('Got to next cell for StandardScaler...')
```

Table 6 Min-Max-scaled descriptive1

	QN_4	TT_TER	RF_TER	VGSL	TS05	YEAR	MONTH	DAY	BF10
COUNT	35672.00	35672.00	35672.00	35672.00	35672.00	35672.00	35672.00	35672.00	35672.00
MEAN	8.64	10.76	76.83	1.24	11.10	2006.63	6.54	15.73	70.37
STD	2.77	7.59	17.80	0.98	7.68	9.17	3.45	8.80	37.33
MIN	1.00	-21.30	18.00	0.00	-8.00	1991.00	1.00	1.00	0.00
25%	10.00	5.20	65.00	0.50	4.60	1999.00	4.00	8.00	40.00
50%	10.00	10.60	81.00	1.00	10.90	2007.00	7.00	16.00	87.00
75%	10.00	16.10	91.00	1.80	17.50	2015.00	10.00	23.00	102.00
MAX	10.00	37.70	100.00	6.30	29.20	2021.00	12.00	31.00	111.00

Table 7 Min-Max-scaled descriptive2

	QN_4	TT_TER	RF_TER	VGSL	TS05	YEAR	MONTH	DAY	BF10
COUNT	35672.00	35672.00	35672.00	35672.00	35672.00	35672.00	35672.00	35672.00	35672.00
MEAN	0.85	0.54	0.72	0.20	0.51	0.52	0.50	0.49	0.63
STD	0.31	0.13	0.22	0.16	0.21	0.31	0.31	0.29	0.34
MIN	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
25%	1.00	0.45	0.57	0.08	0.34	0.27	0.27	0.23	0.36
50%	1.00	0.54	0.77	0.16	0.51	0.53	0.55	0.50	0.78
75%	1.00	0.63	0.89	0.29	0.69	0.80	0.82	0.73	0.92
MAX	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00

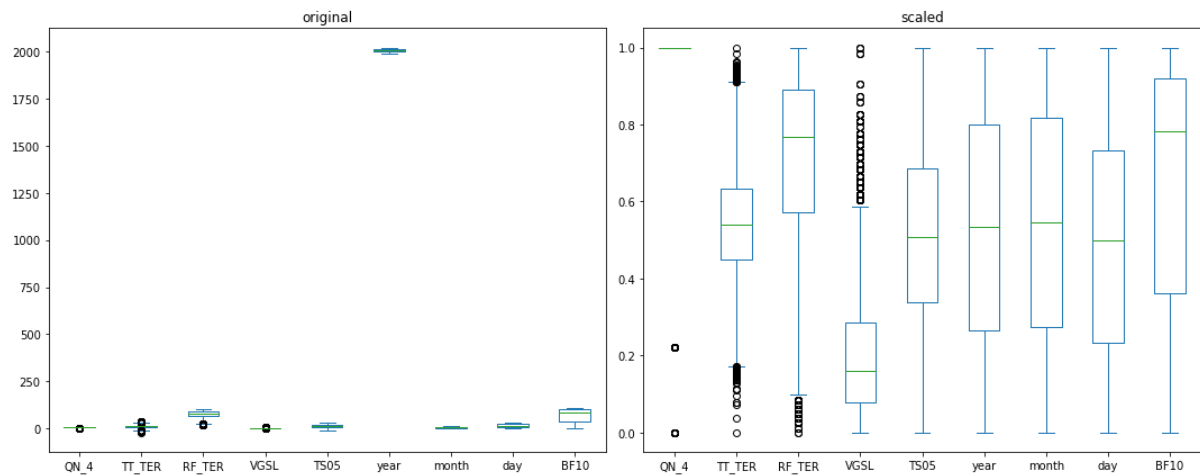


Figure 9 Min-Max-scaled plots

4.2. StandardScaler

StandardScaler normalization or scaling method, which is usually called Z-Score. In this method, the distance between the data is calculated by the average and divided by the standard deviation. In fact, during StandardScaler normalization, the values of a feature f are normalized using the mean and standard deviation of (Shiue, Lee, & Su, 2018).

$$v' = \frac{v - f_{mean}}{f_{standarddeviation}} \quad (2)$$

```

if(use_normalization==1):
    print("Standard-scaled descriptives:")
    from sklearn.preprocessing import StandardScaler
    scaler = StandardScaler()
    scaler.fit(df_numeric)
    data_scaled = scaler.transform(df_numeric)
    df_scaled = pd.DataFrame(data_scaled, index=df_numeric.index, columns=df_numeric.columns)
    display(df_scaled.describe().round(2))
    data_normal=df_scaled
    fig, axs = plt.subplots(ncols=2, figsize=(15,6))
    df_numeric.plot.box(ax=axs[0], title="original")
    df_scaled.plot.box(ax=axs[1], title="scaled")
    plt.tight_layout()
    plt.show()
else:
    print('Got to Before cell for StandardScaler...')

```

4.3. Outlier

Detection of outliers can be considered as a pre-processing step in data mining, and sometimes as an independent operation in data mining. So far, there are many methods to detect outlier data, including the robust covariance estimate method, Isolation Forest, Local Outlier Factor, boxplot, etc. Each of these methods has its advantages and disadvantages. But since we are dealing with temporal data it is a bit hard to detect outliers. Therefore, here we use the Local Outlier Factor and boxplot method to be more careful, which tries to identify points based on the neighbourhood.

4.3.1 Boxplot

The boxplot method is one of the most important and simple methods for detecting outliers. So that this method can be used better when the data is non-normal and random distribution, and as a result, in this method, instead of mean and standard deviation, two other criteria including median and inter quartile range are used respectively. The title of estimation of concentration and dispersion point is used for better detection of outlier data. Based on this model, almost a general rule (if a sample of soil moisture data is three times the range of the first or third quartile, consider it as outlier data) is used to detect and identify outlier data (Gijbels & Hubert, 2009).

We calculate the quantiles based on the following relationship.

$q(p)=x(k)+\alpha(x(k+1)-x(k))$	(3)
---------------------------------	-----

Therefore, to calculate the first and third quartiles, it is enough to set the value of p equal to 0.25 and 0.75, then obtain the values of q(p). In this way, the range of values that are considered as permissible and meaningful is within three IQRs of the first and third quartiles.

$$x \in (q(0.25) - 3 \times IQR, q(0.75) + 3 \times IQR) \quad (4)$$

```
boxplot = data_normal.boxplot(figsize=(8,6))
```

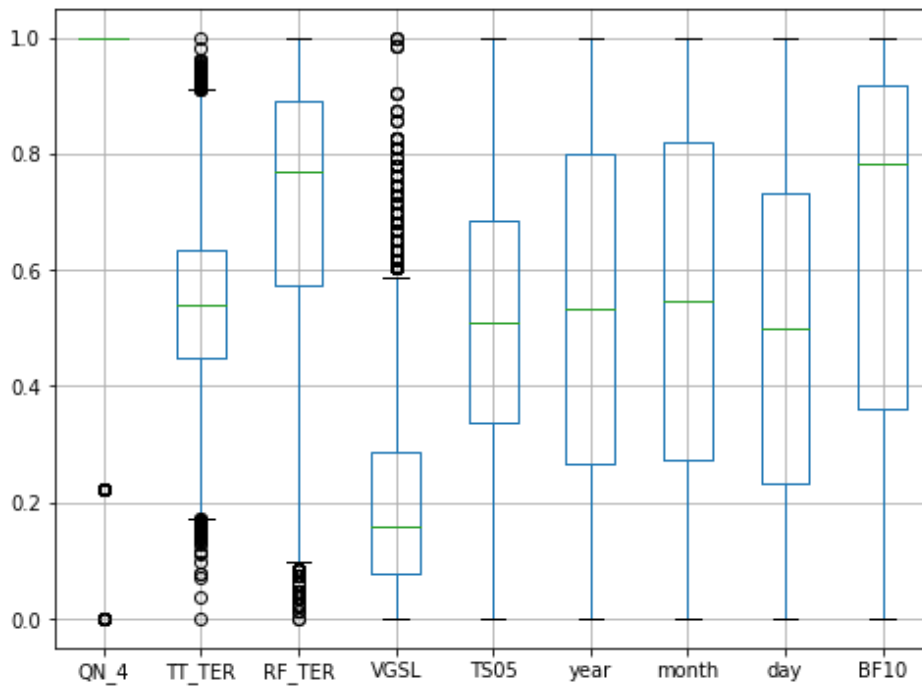


Figure 10 box plot-the features of VGSL, TT_TER, RF_TER outliers

Here, we used the boxplot method to identify outliers on the soil moisture data set, the results of this method show that the features of VGSL, TT_TER, RF_TER have outliers and their outliers should be identified by formulas 3 and 4 and be remove boxplot method to identify outliers on the soil moisture data set removed or replaced from the dataset. In the code below, the first filter related to the VGSL feature is obtained based on the first and third quadrants. As is clear from the results. Based on the feature x, 636 samples have been identified as outliers.

```
quarter1 = data_normal['VGSL'].quantile(0.25)
quarter3 = data_normal['VGSL'].quantile(0.75)
IQR = quarter3 - quarter1    #IQR is interquartile range.

filter1 = (data_normal['VGSL'] >= quarter1 - 1.5 * IQR) & (data_normal['VGSL'] <= quarter3 + 1.5 * IQR)
data_normal.loc[~filter1]
```

Table 8 filter of VGSL feature

QN_4	TT_TER	RF_TER	VGSL	TS05	YEAR	MONTH	DAY	BF10	
DATE									
1991-07-02 07:00:00	1.000000	0.652542	0.951220	0.857143	0.825269	0.0	0.545455	0.033333	0.612613
1991-07-02 14:00:00	1.000000	0.832203	0.317073	0.857143	0.825269	0.0	0.545455	0.033333	0.612613
1991-07-02 21:00:00	1.000000	0.742373	0.597561	0.857143	0.825269	0.0	0.545455	0.033333	0.612613
1991-07-03 07:00:00	1.000000	0.732203	0.487805	0.809524	0.857527	0.0	0.545455	0.066667	0.486486
1991-07-03 14:00:00	1.000000	0.827119	0.292683	0.809524	0.857527	0.0	0.545455	0.066667	0.486486
...
2021-07-26 18:00:00	0.222222	0.766102	0.451220	0.682540	0.852151	1.0	0.545455	0.833333	0.387387
2021-07-30 00:00:00	0.222222	0.577966	0.829268	0.619048	0.758065	1.0	0.545455	0.966667	0.522523
2021-07-30 06:00:00	0.222222	0.615254	0.731707	0.619048	0.758065	1.0	0.545455	0.966667	0.522523
2021-07-30 12:00:00	0.222222	0.754237	0.353659	0.619048	0.758065	1.0	0.545455	0.966667	0.522523
2021-07-30 18:00:00	0.222222	0.733898	0.524390	0.619048	0.758065	1.0	0.545455	0.966667	0.522523

636 rows × 9 columns

In the code below, the first filter related to the TT_TER feature is obtained based on the first and third quadrants. As is clear from the results. Based on the feature x, 108 samples have been identified as outliers.

```

quarter1 = data_normal['TT_TER'].quantile(0.25)
quarter3 = data_normal['TT_TER'].quantile(0.75)
IQR = quarter3 - quarter1      #IQR is interquartile range.

filter2 = (data_normal['TT_TER'] >= quarter1 - 1.5 * IQR) & (data_normal['TT_TER'] <= quarter3 + 1.5 * IQR)
data_normal.loc[~filter2]

```

Table 9 filter of TT_TER feature

QN_4	TT_TER	RF_TER	VGSL	TS05	YEAR	MONTH	DAY	BF10	
DATE									
1991-02-06 07:00:00	1.000000	0.152542	0.658537	0.000000	0.051075	0.000000	0.090909	0.166667	0.864865
1991-02-06 21:00:00	1.000000	0.159322	0.707317	0.000000	0.051075	0.000000	0.090909	0.166667	0.864865
1991-07-11 14:00:00	1.000000	0.918644	0.195122	0.238095	0.911290	0.000000	0.545455	0.333333	0.009009
1992-08-09 14:00:00	1.000000	0.922034	0.268293	0.126984	0.935484	0.033333	0.636364	0.266667	0.000000
1994-02-21 07:00:00	1.000000	0.149153	0.902439	0.000000	0.129032	0.100000	0.090909	0.666667	0.819820
...
2021-02-12 06:00:00	0.222222	0.144068	0.926829	0.000000	0.220430	1.000000	0.090909	0.366667	0.927928
2021-02-13 00:00:00	0.222222	0.171186	0.914634	0.000000	0.220430	1.000000	0.090909	0.400000	0.927928
2021-02-13 06:00:00	0.222222	0.071186	0.865854	0.000000	0.220430	1.000000	0.090909	0.400000	0.927928
2021-06-17 12:00:00	0.222222	0.928814	0.146341	0.269841	0.959677	1.000000	0.454545	0.533333	0.000000
2021-06-17 18:00:00	0.222222	0.913559	0.256098	0.269841	0.959677	1.000000	0.454545	0.533333	0.000000

108 rows × 9 columns

In the code below, the first filter related to the RF_TER feature is obtained based on the first and third quadrants. As is clear from the results. Based on the feature x, 92 samples have been identified as outliers.

```
quarter1 = data_normal['RF_TER'].quantile(0.25)
quarter3 = data_normal['RF_TER'].quantile(0.75)
IQR = quarter3 - quarter1      #IQR is interquartile range.

filter3 = (data_normal['RF_TER'] >= quarter1 - 1.5 * IQR) & (data_normal['RF_TER'] <= quarter3 + 1.5 * IQR)
data_normal.loc[~filter3]
```

Table 10 filter of RF_TER feature

QN_4	TT_TER	RF_TER	VGSL	TS05	YEAR	MONTH	DAY	BF10	
DATE									
1994-07-24 14:00:00	1.000000	0.923729	0.024390	0.126984	0.946237	0.100000	0.545455	0.766667	0.0
1995-08-09 14:00:00	1.000000	0.798305	0.073171	0.095238	0.825269	0.133333	0.636364	0.266667	0.0
1995-08-10 14:00:00	1.000000	0.838983	0.060976	0.095238	0.844086	0.133333	0.636364	0.300000	0.0
1995-08-11 14:00:00	1.000000	0.876271	0.048780	0.111111	0.862903	0.133333	0.636364	0.333333	0.0
1995-08-12 14:00:00	1.000000	0.910169	0.060976	0.095238	0.887097	0.133333	0.636364	0.366667	0.0
...
2020-06-02 12:00:00	0.222222	0.805085	0.085366	0.111111	0.846774	0.966667	0.454545	0.033333	0.0
2020-06-02 18:00:00	0.222222	0.818644	0.060976	0.111111	0.846774	0.966667	0.454545	0.033333	0.0
2020-08-07 12:00:00	0.222222	0.906780	0.036585	0.095238	0.900538	0.966667	0.636364	0.200000	0.0
2020-08-07 18:00:00	0.222222	0.894915	0.073171	0.095238	0.900538	0.966667	0.636364	0.200000	0.0
2020-08-08 12:00:00	0.222222	0.933898	0.085366	0.095238	0.930108	0.966667	0.636364	0.233333	0.0

92 rows × 9 columns

In the continuation of this section, after applying the three desired filters and removing the outlier data, according to the independent characteristics of VGSL, TT_TER, RF_TER, a number of outlier samples are removed from the soil moisture data set, and finally, the new data can be obtained as follows drawing.

```
data_outlier=data_normal.loc[filter3 & filter1 & filter2]
boxplot = data_outlier.boxplot(figsize=(8,6))
```

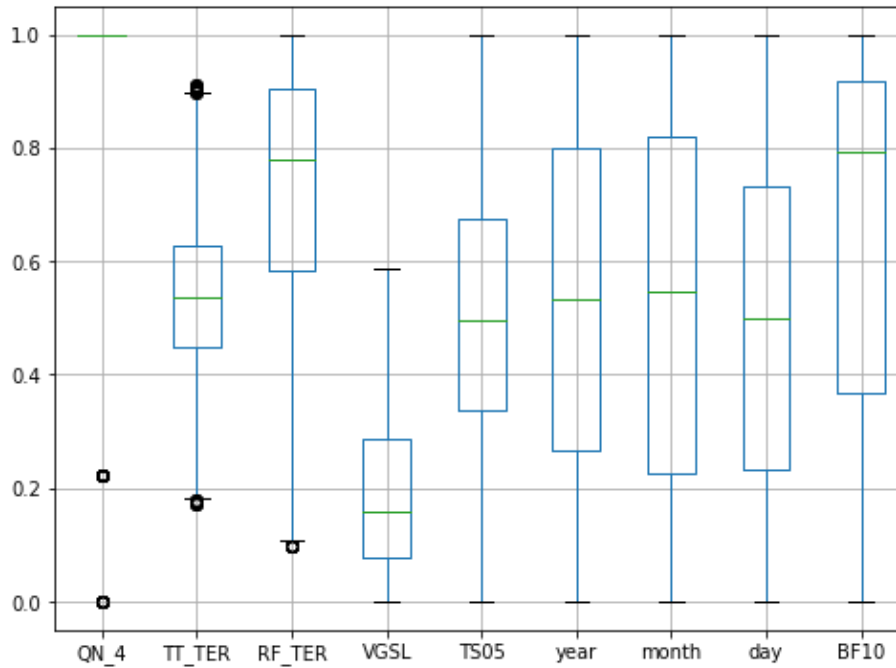


Figure 11 box plot- removed outlier data from the soil moisture data set

According to the features of VGSL, TT_TER, RF_TER, we will examine the outlier data chart of each feature. To reach an important conclusion about the boxplot. In fact, as can be seen, this method is used derivatively for each of the features.

```
idx1=np.argwhere(~filter1.values)
idx1=[ item[0] for item in idx1]
idx2=np.argwhere(~filter2.values)
idx2=[ item[0] for item in idx2]
idx3=np.argwhere(~filter3.values)
idx3=[ item[0] for item in idx3]
idx_outlier=np.concatenate((idx1,idx2,idx3),axis=0)
# idx_outlier=[ item[0] for item in idx_outlier]
idx_outlier

array([ 546,  547,  548,  549,  550,  551,  552,  553,  554,  555,  556,  557,  558,  559,  560,  561,
        562,  563, 1497, 1498, 1499, 1500, 1501, 1502, 1512, 1513, 1514, 1515, 1516, 1517, 1578,
        1579, 1580, 1608, 1609, 1610, 2523, 2524, 2525, 2541, 2542, 2543, 2544, 2545, 2546, 2547,
        2548, 2549, 2550, 2551, 2552, 2577, 2578, 2579, 3687, 3688, 3689, 4722, 4723, 4724, 4908,
        4909, 4910, 4911, 4912, 4913, 4914, 4915, 4916, 5949, 5950, 5951, 5955, 5956, 5957, 6087,
        6088, 6089, 6090, 6091, 6092, 6174, 6175, 6176, 6990, 6991, 6992, 7032, 7033, 7034, 7059,
        7060, 7061, 7200, 7201, 7202, 7206, 7207, 7208, 7224, 7225, 7226, 7239, 7240, 7241, 7242,
        7243, 7244, 7245, 7246, 7247, 8007, 8008, 8009, 8055, 8056, 8057, 8061, 8062, 8063, 8064,
        8065, 8066, 8070, 8071, 8072, 8130, 8131, 8132, 8184, 8185, 8186, 8187, 8188, 8189, 8271,
        8272, 8273, 8274, 8275, 8276, 8328, 8329, 8330, 9177, 9178, 9179, 9204, 9205, 9206, 9210,
        9211, 9212, 9222, 9223, 9224, 9225, 9226, 9227, 9363, 9364, 9365, 9393, 9394, 9395, 9396,
        9397, 9398, 10209, 10210, 10211, 10236, 10237, 10238, 10239, 10240, 10241, 10248, 10249,
        10250, 10323, 10324, 10325, 10341, 10342, 10343, 11346, 11347, 11348, 11349, 11350, 11351,
        11352, 11353, 11354, 11511, 11512, 11513, 11514, 11515, 11516, 11517, 11518, 11519, 11520,
        11521, 11522, 12474, 12475, 12476, 12549, 12550, 12551, 12555, 12556, 12557, 12558, 12559,
```


12560, 12573, 12574, 12575, 12681, 12682, 12683, 13518, 13519, 13520, 13605, 13606, 13607,
 13623, 13624, 13625, 13737, 13738, 13739, 13749, 13750, 13751, 14769, 14770, 14771, 14808,
 14809, 14810, 14838, 14839, 14840, 14877, 14878, 14879, 14934, 14935, 14936, 15777, 15778,
 15779, 15783, 15784, 15785, 15801, 15802, 15803, 15840, 15841, 15842, 15849, 15850, 15851,
 15852, 15853, 15854, 15855, 15856, 15857, 15894, 15895, 15896, 15912, 15913, 15914, 15915,
 15916, 15917, 15918, 15919, 15920, 16806, 16807, 16808, 16809, 16810, 16811, 16812, 16813,
 16814, 16815, 16816, 16817, 16818, 16819, 16820, 16917, 16918, 16919, 16920, 16921, 16922,
 16923, 16924, 16925, 16926, 16927, 16928, 16947, 16948, 16949, 16962, 16963, 16964, 18006,
 18007, 18008, 18009, 18010, 18011, 18039, 18040, 18041, 18096, 18097, 18098, 18114, 18115,
 18116, 18117, 18118, 18119, 18120, 18121, 18122, 18150, 18151, 18152, 18180, 18181, 18182,
 18183, 18184, 18185, 19014, 19015, 19016, 19146, 19147, 19148, 19251, 19252, 19253, 19254,
 19255, 19256, 19257, 19258, 19259, 19275, 19276, 19277, 19284, 19285, 19286, 20154, 20155,
 20156, 20232, 20233, 20234, 20247, 20248, 20249, 20328, 20329, 20330, 20331, 20332, 20333,
 20343, 20344, 20345, 20346, 20347, 20348, 20355, 20356, 20357, 20418, 20419, 20420, 21402,
 21403, 21404, 21423, 21424, 21425, 23364, 23365, 23366, 23487, 23488, 23489, 23625, 23626,
 23627, 23634, 23635, 23636, 23649, 23650, 23651, 23652, 23653, 23654, 24429, 24430, 24431,
 24747, 24748, 24749, 25626, 25627, 25628, 25671, 25672, 25673, 25674, 25675, 25676, 25680,
 25681, 25682, 25683, 25684, 25685, 25752, 25753, 25754, 25761, 25762, 25763, 25797, 25798,
 25799, 25800, 25801, 25802, 25809, 25810, 25811, 25812, 25813, 25814, 25815, 25816, 25817,
 25842, 25843, 25844, 25863, 25864, 25865, 26940, 26941, 26942, 27024, 27025, 27026, 27771,
 27772, 27773, 27774, 27775, 27776, 27777, 27778, 27779, 27780, 27781, 27782, 27858, 27859,
 27860, 27861, 27862, 27863, 27867, 27868, 27869, 27915, 27916, 27917, 27930, 27931, 27932,
 27966, 27967, 27968, 27969, 27970, 27971, 27996, 27997, 27998, 27999, 28000, 28001, 29059,
 29060, 29061, 29062, 29159, 29160, 29161, 29162, 29167, 29168, 29169, 29170, 29171, 29172,
 29173, 29174, 29179, 29180, 29181, 29182, 30263, 30264, 30265, 30266, 30275, 30276, 30277,
 30278, 30363, 30364, 30365, 30366, 30367, 30368, 30369, 30370, 30391, 30392, 30393, 30394,
 31943, 31944, 31945, 31946, 31947, 31948, 31949, 31950, 31959, 31960, 31961, 31962, 31967,
 31968, 31969, 31970, 31983, 31984, 31985, 31986, 33431, 33432, 33433, 33434, 33443, 33444,
 33445, 33446, 33447, 33448, 33449, 33450, 33451, 33452, 33453, 33454, 33455, 33456, 33457,
 33458, 33459, 33460, 33461, 33462, 34727, 34728, 34729, 34730, 34819, 34820, 34821, 34822,
 34855, 34856, 34857, 34858, 34895, 34896, 34897, 34898, 34923, 34924, 34925, 34926, 35003,
 35004, 35005, 35006, 35031, 35032, 35033, 35034, 35039, 35040, 35041, 35042, 35055, 35056,
 35057, 35058, 108, 110, 574, 1759, 3441, 3838, 3901, 3910, 3913, 3922, 3934, 4987, 5469,
 5550, 5559, 5592, 5594, 5595, 6566, 6573, 6574, 6575, 6576, 6577, 6578, 6579, 6581, 6582,
 6597, 6599, 7285, 8694, 10372, 10375, 12559, 13173, 13176, 13738, 13750, 13804, 13805,
 13807, 13808, 13816, 13819, 13820, 15528, 17035, 17036, 17038, 17044, 17056, 19740, 19749,
 19752, 20419, 20781, 20783, 20895, 21367, 21368, 21370, 21388, 21391, 21392, 21873, 23109,
 23112, 23115, 23118, 23121, 23704, 23705, 24718, 24748, 24749, 25801, 26845, 26846, 26851,
 26852, 29061, 30649, 30653, 30657, 30658, 30661, 30662, 30689, 30705, 30706, 31993, 31994,
 32109, 32110, 32113, 32114, 32117, 32245, 33633, 33637, 34375, 34383, 34384, 34387, 34388,
 34885, 34886, 3901, 5044, 5047, 5050, 5053, 5761, 5797, 5800, 5812, 6868, 6871, 7042, 7243,
 7246, 7249, 9205, 9274, 9400, 9403, 9406, 9409, 9412, 10267, 10372, 13393, 13394, 13807,
 13816, 15616, 16813, 16816, 16819, 16828, 16831, 17032, 17033, 17035, 17036, 17806, 17845,
 17846, 17849, 17890, 17893, 17896, 19019, 19021, 19027, 20419, 21160, 21163, 21358, 21367,
 21368, 21388, 21389, 21419, 22246, 22297, 22298, 22300, 22301, 23450, 24355, 24442, 24484,
 24718, 24748, 24749, 28741, 28747, 30558, 30562, 30565, 30658, 30705, 30706, 31725, 31729,
 32010, 33137, 33138, 33153, 33157, 33161, 33261, 33262, 33365, 33366, 33629, 33630, 33633]

```
plt.plot(data_normal.iloc[:,3].values)
plt.plot(idx1,data_normal.iloc[idx1,3].values,'*')
plt.title('detection outlier just for VGSL with boxplot method')
```

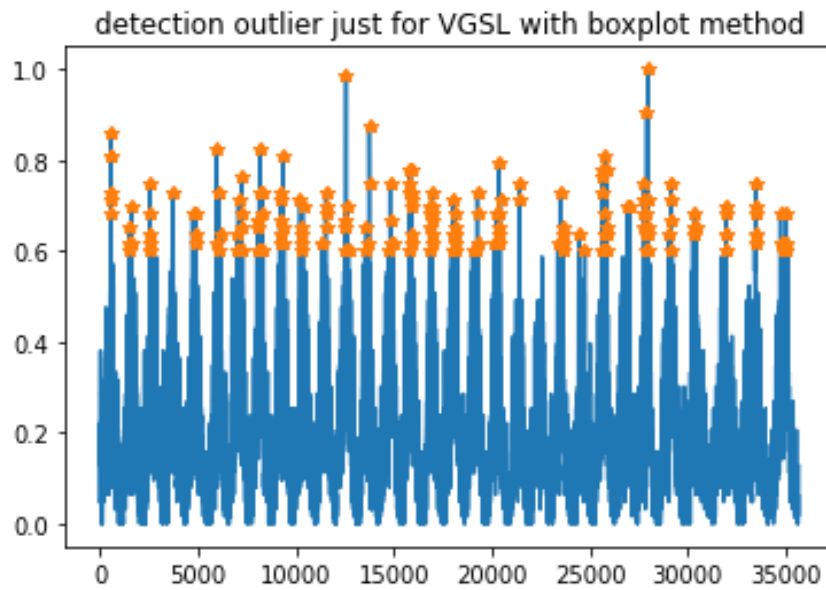


Figure 12 Detection outlier just for VGSL with boxplot method

The results of the boxplot method on the VGSL feature show that this method has identified points higher than 0.60 as outlier data.

```
plt.plot(data_normal.iloc[:,1].values)
plt.plot(idx2,data_normal.iloc[idx2,1].values,'*')
plt.title('detection outlier just for TT_TER with boxplot method')
```

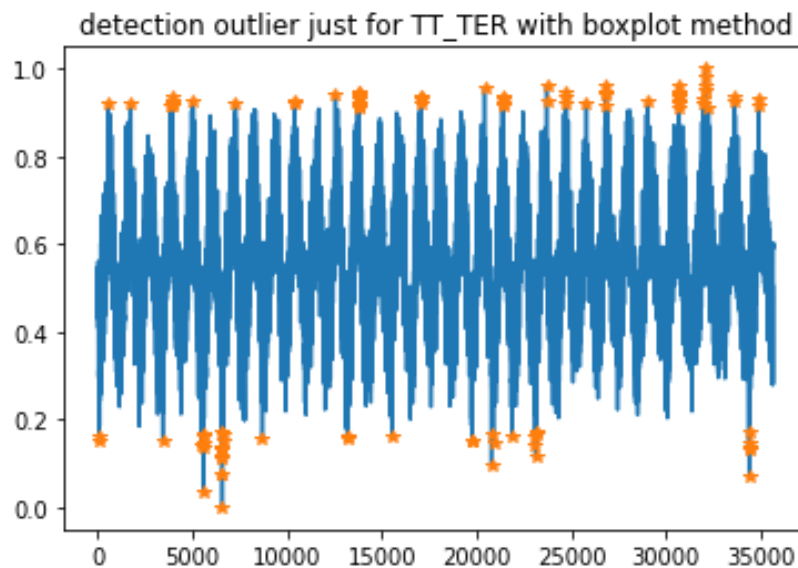


Figure 13 Detection outlier just for TT_TER with boxplot method

The results of the boxplot method on the TT_TER feature show that this method has identified points higher than 0.90 and lower than 0.2 as outlier data.

```
plt.plot(data_normal.iloc[:,2].values)
plt.plot(idx3,data_normal.iloc[idx3,2].values,'*')
plt.title('detection outlier just for RF_TER with boxplot method')
```

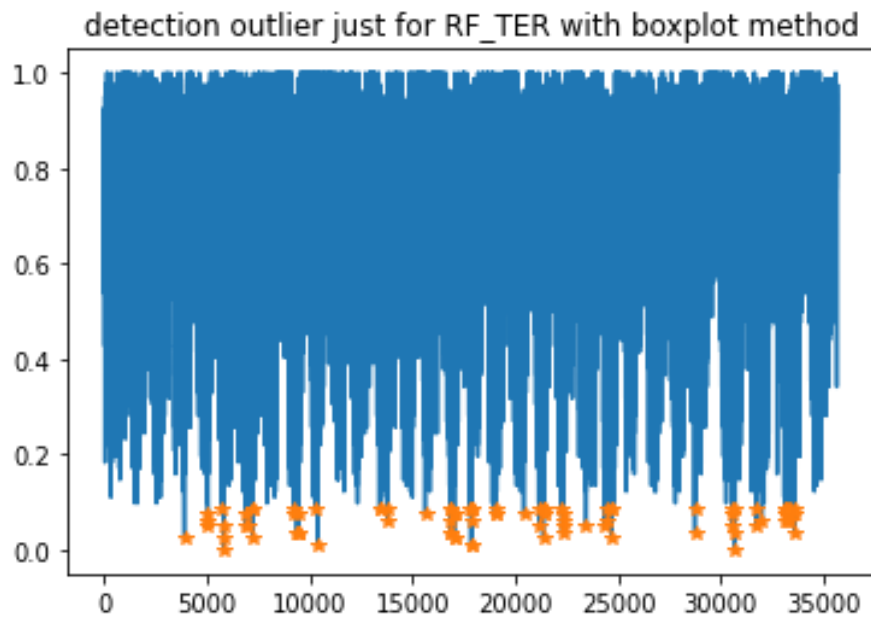


Figure 14 Detection outlier just for RF_TER with boxplot method

The results of the boxplot method on the RF_TER feature show that this method has identified points lower than 0.1 as outlier data.

Important point result boxplot

Drawing a box plot can be used in the best way to detect outliers in the data set in the form of a feature and one dimension. But when our data set has several features like the soil moisture data set here, using a box plot and detecting outliers based on the separate examination of different features in the soil moisture set seems somewhat incorrect. Because it is possible that these outliers are effective samples in the BF10 feature and in some way better predictions can be made from these outlier samples. Therefore, since we are dealing with both time data and multi-feature data, using this method is inefficient. And our goal here was the general examination of the soil moisture dataset with this method, and in the following we will use the neighbourhood point's method to better detect outliers.

4.3.2 Local Outlier Factor

In the boxplot method, we came to the conclusion that this method is used for single feature data sets and we need other methods for multiple features. In fact, here we need methods that can identify outliers in multivariate data sets, and the most important of these methods is the Local Outlier Factor technique, which is based on clustering. Acting on the basis of clustering helps us to examine a sample in terms of several features with its neighbour and identify it as an outlier if it is outside the neighbourhood. In the Local Outlier Factor technique, an outlier score is calculated for each of the samples in the soil moisture data set, and the outliers are determined by calculating the distance of each sample from its cluster centre.

Here we use the sklearn version, which is an unsupervised outlier detection method using the local outlier coefficient. In fact, the abnormality or outlier score of each sample is called the local outlier factor. In this method, the deviation of a sample is also measured based on the local density and the neighbours of that sample. In fact, it can be said that outliers can be identified by comparing the local density of a sample of the data set with the local density of its neighbours, which is a number that can be adjusted. As a result, the sample is among the outliers that have a lower density than their neighbours.

```
from sklearn.neighbors import LocalOutlierFactor
X=data_normal.iloc[:, :-1].values
y=data_normal.iloc[:, -1].values
pred = LocalOutlierFactor(n_neighbors=200).fit_predict(X,y)
# pred=clf.predict(X)
idx=np.argwhere(pred==-1)
idx=[item[0] for item in idx]
plt.plot(y)
plt.plot(idx,y[idx], '*')
```

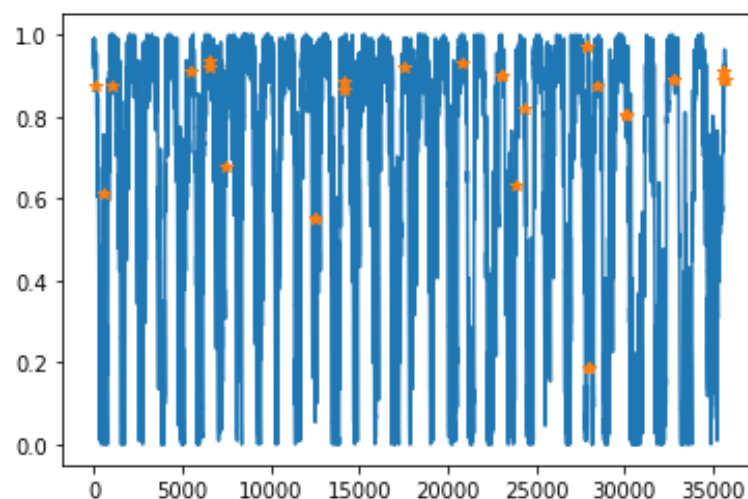


Figure 15 Local Outlier Factor with boxplot method

```
idx=np.argwhere(pred==1)
idx=[item[0] for item in idx]
df_final_outlier=data_normal.iloc[idx,:]
```

5. TIME SERIES

In this section, our goal is to examine the characteristics of BF10 in terms of time, its recurrence period, and its seasonality. Therefore, we are content with showing several graphs and checking the p-value here, and refrain from providing additional explanations. In the following section, the results of p-value show that the value of BF10 is less than 0.05. Its value is very small, which shows that this characteristic is a kind of time and trend.

```

from statsmodels.tsa.stattools import adfuller

result = adfuller(data_normal["BF10"].values)

## the p-value is the second one of the returned objects
print(f"p-
value (~'how sure it is that the series has a trend'): {result[1]:.15
f}")

print("Should be under 0.05 to assume stationarity")
p-value (~'how sure it is that the series has a trend'):
0.000000000000002
Should be under 0.05 to assume stationarity

```

```

import matplotlib.pyplot as plt
import seaborn as sns
reindexed_df = data_normal.copy()
fig, ax = plt.subplots(figsize=(12,6))
sns.boxplot(x=reindexed_df.index.year, y=reindexed_df["BF10"], ax=ax)
#, width=20)
plt.title("values for years")
plt.xlabel("Years")
plt.xticks(rotation=45)
plt.show()

```

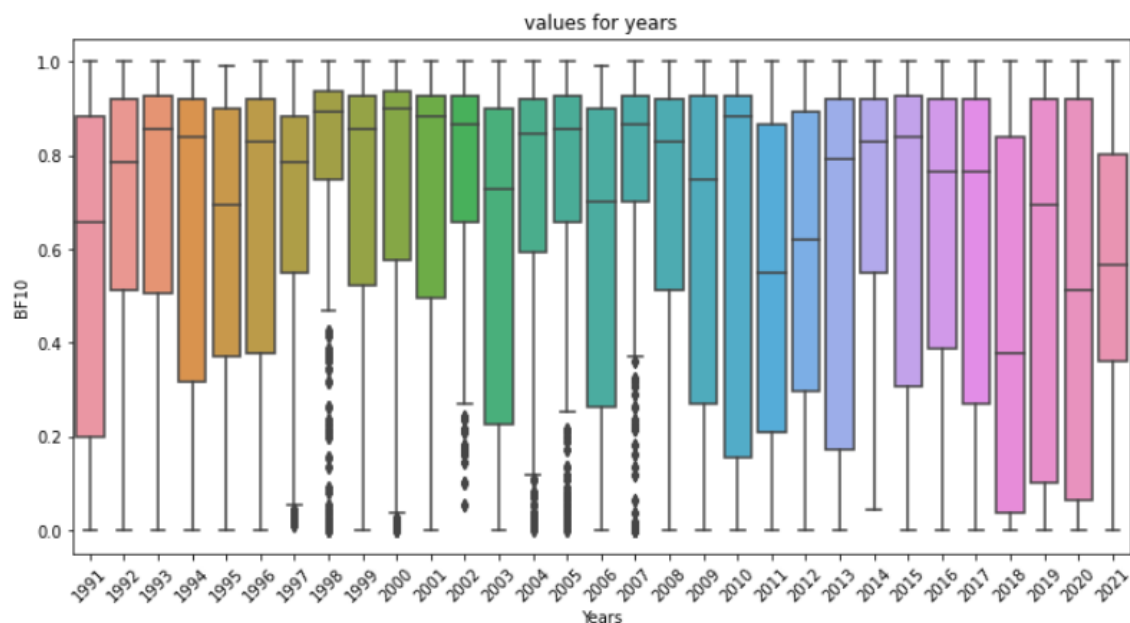


Figure 16 values of years about the situation of soil moisture

In the following, we examined the trend of BF10 features in terms of years, the results of which in the above figure show that some years such as 1998, 2002 and 2007 have recorded higher soil moisture than other years (almost above 0.6). In some years, such as 2018 to 2020, different soil moisture values close to zero to 0.9 have been recorded. With this analysis, it can be concluded that

the years have almost the same situation in terms of soil moisture, but in some years different soil moisture has been recorded. In the following, the monthly chart is also reviewed for further review.

```
fig, ax = plt.subplots(figsize=(12,6))
sns.boxplot(x=reindexed_df.index.strftime("%b"), y=reindexed_df["BF10"], ax=ax)#, width=20)
plt.title("Values for months")
plt.xlabel("Months")
plt.show()
```

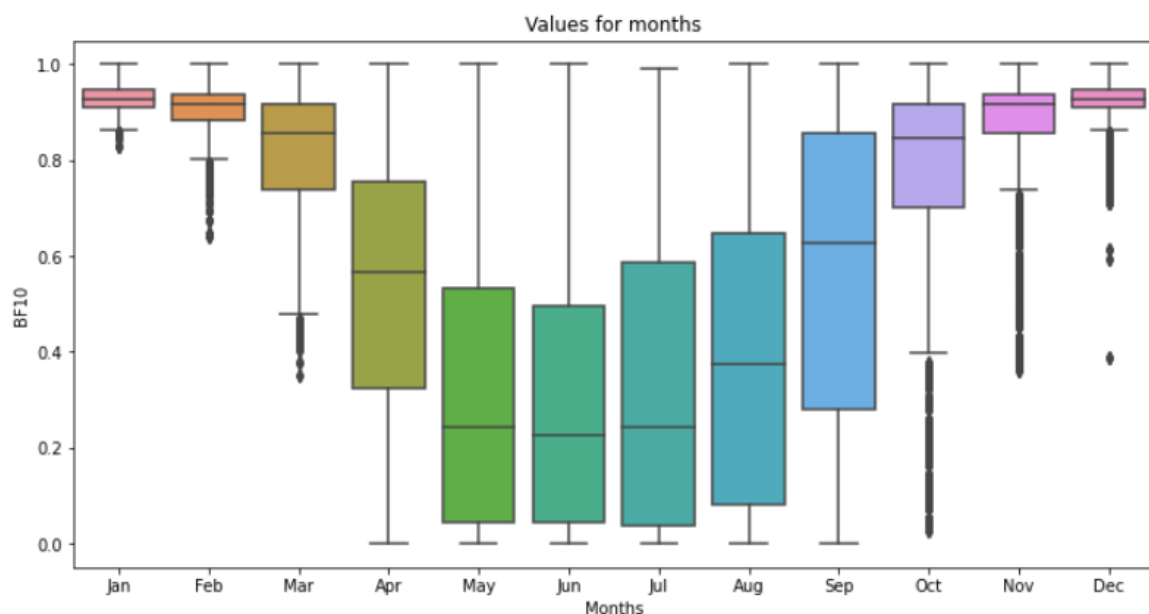


Figure 17 values of months about the situation of soil moisture

In the following, we examined the trend of BF10 features in terms of year, the results of which in the figure above show that the soil moisture has taken different values in different months and this is contrary to the year chart. In fact, the time cycle in the BF10 feature can be different based on different months. This plot shows that soil moisture has higher values in the months of Jan, Feb, Mar, Nov, Dec, and so that it has average values (between 0.2 and 0.8) in other months like May and Sep, and also in months like May, June, July, August assigned the lowest value (0 to 0.6). Next, to further prove the temporality and seasonality of the BF10 feature, we coded its corresponding graph using the statsmodels library and the seasonal_decompose method as below.

```
## for automatic decomposition of a time series:
from statsmodels.tsa.seasonal import seasonal_decompose
analysis = seasonal_decompose(df_final_outlier["BF10"].values, model=
"additive", period=500)
analysis.plot()
plt.show()
```

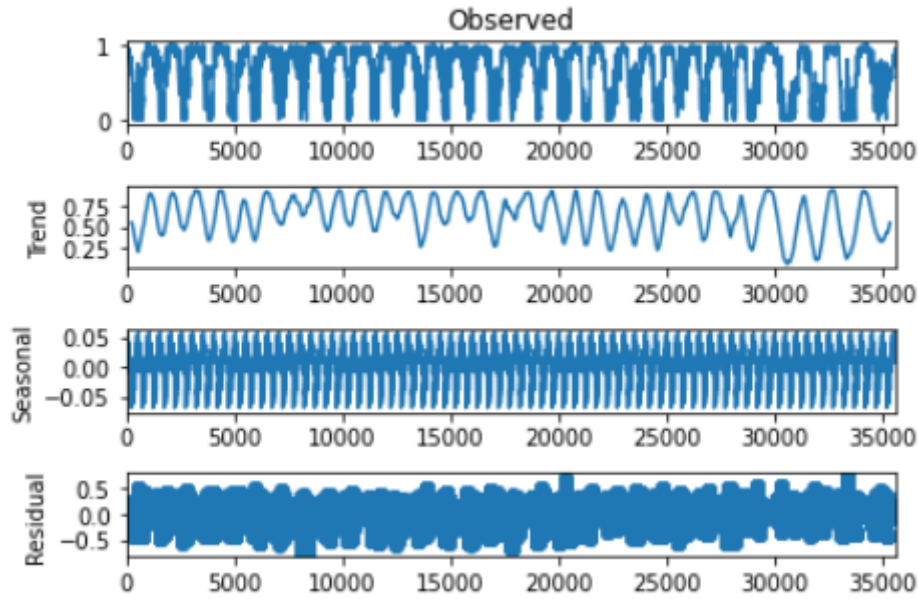


Figure 18 temporality and seasonality of the BF10 feature

6. FEATURE SELECTION

Feature selection is the process of finding the smallest possible number of features of a data set that are able to describe this data set as well as the original features. The purpose of feature selection is to remove unnecessary features and select important features according to the dataset and its class. In our implementation, we will use tree methods and linear methods to accurately predict the amount of soil moisture. Therefore, in this section, we will examine two different methods for feature selection. The first method of feature selection is using RandomForestRegressor, which provides us with important features, and the second method is to use simple linear regression to predict and select important features.

```
df_final_outlier
```

Table 11 data frame of final outliers

	QN_4	TT_TER	RF_TER	VGSL	TS05	YEAR	MONTH	DAY	BF10
DATE									
1991-01-01 07:00:00	1.0	0.411864	0.890244	0.047619	0.293011	0.0	0.0	0.000000	0.918919
1991-01-01 14:00:00	1.0	0.442373	0.817073	0.047619	0.293011	0.0	0.0	0.000000	0.918919
1991-01-01 21:00:00	1.0	0.427119	0.780488	0.047619	0.293011	0.0	0.0	0.000000	0.918919
1991-01-02 07:00:00	1.0	0.455932	0.926829	0.222222	0.384409	0.0	0.0	0.033333	0.990991
1991-01-02 14:00:00	1.0	0.547458	0.841463	0.222222	0.384409	0.0	0.0	0.033333	0.990991
...
2021-12-30 00:00:00	0.0	0.557627	0.878049	0.126984	0.467742	1.0	1.0	0.966667	0.936937
2021-12-30 06:00:00	0.0	0.549153	0.975610	0.126984	0.467742	1.0	1.0	0.966667	0.936937
2021-12-30 12:00:00	0.0	0.603390	0.792683	0.126984	0.467742	1.0	1.0	0.966667	0.936937
2021-12-30 18:00:00	0.0	0.589831	0.878049	0.126984	0.467742	1.0	1.0	0.966667	0.936937
2021-12-31 00:00:00	0.0	0.594915	0.792683	0.126984	0.497312	1.0	1.0	1.000000	0.909910

35634 rows x 9 columns

```
X_scaled=df_final_outlier.loc[:,df_final_outlier.columns!='BF10'].values
y=df_final_outlier['BF10'].values
# 30 step next
X_scaled=X_scaled[:-30]
y=y[30:]
```

6.1. Random Forest (tree base)

In this section, we will use RandomForestRegressor to select features and then plot the scores of all features, and in the next step, we will use the linear regression method to select features, and finally, we will select features that are suitable for the entire model. Tree and regression methods are useful.

```
from sklearn.ensemble import RandomForestRegressor
reg_model =RandomForestRegressor( n_estimators=100,random_state=125)
reg_model.fit(X_scaled, y)
RandomForestRegressor(random state=125)
```

```
importances_score = reg_model.feature_importances_
for i,v in enumerate(importances_score):
    print('Feature: %1d, Score: %.3f' % (i,v))
Feature: 0, Score: 0.008
Feature: 1, Score: 0.011
Feature: 2, Score: 0.013
Feature: 3, Score: 0.060
Feature: 4, Score: 0.437
Feature: 5, Score: 0.201
Feature: 6, Score: 0.193
Feature: 7, Score: 0.076
```

```
import matplotlib.pyplot as plt
plt.figure(figsize=(8,7))
importances_score = reg_model.feature_importances_
col_names=df_final_outlier.columns[df_final_outlier.columns!='BF10']
forest_importances = pd.Series(importances_score, index=col_names)
fig, ax = plt.subplots()
std = np.std([item_tree.feature_importances_ for item_tree in reg_model.estimators_], axis=0)
forest_importances.plot.bar()
ax.set_title("soil moisture Feature importances")
ax.set_ylabel("score")
fig.tight_layout()
```

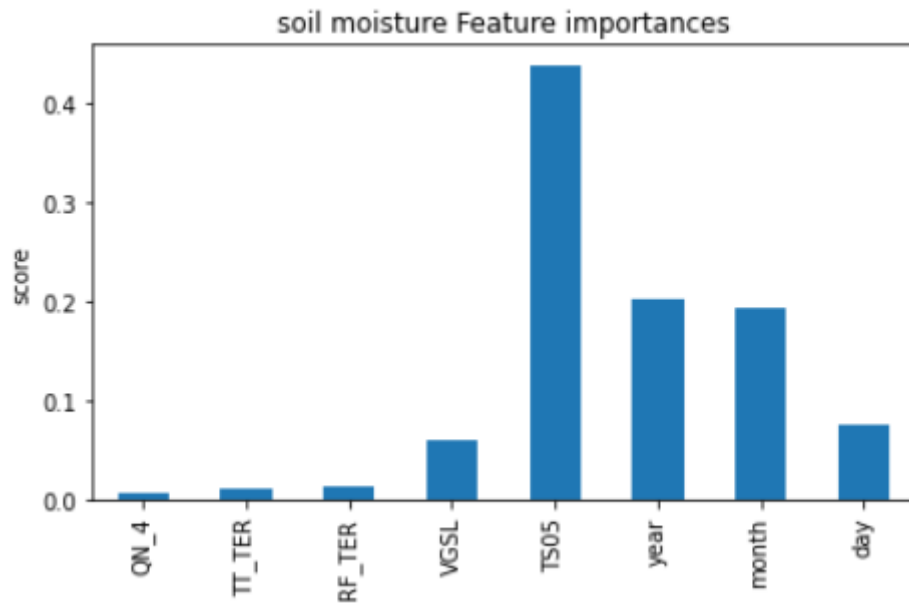



Figure 19 Soil moisture feature importance's

```
df_final_outlier.columns
Index(['QN_4', 'TT_TER', 'RF_TER', 'VGSL', 'TS05', 'year', 'month',
      'day', 'BF10'], dtype='object')
```

The results of using RandomForestRegressor to select the feature in the figure above show that the most important feature is TS05 with a value of 0.43 and then the year feature with a value of 0.20, the month feature with a value of 0.19 and the VGSL feature with a value of 0.07 are the most important features and also the worst. The corresponding feature of QN_4, which is close to zero. The rest of the characteristics also obtained much lower values or attention to the plot.

6.2. Linear regression (linear base)

In this section, we will use Linear Regression to select features and then plot the scores of all features, and in the next step, we will use both RandomForestRegressor and linear regression methods to select features, and finally, we will select features that Useful for alltree and regression models.

```
from sklearn.linear_model import ARDRegression, LinearRegression
reg_model =ARDRegression(compute_score=True)
reg_model.fit(X_scaled, y)
```

```
LinearRegression()
```

```
importances_score = importance = np.abs(reg_model.coef_)
for i,v in enumerate(importances_score):
    print('Feature: %s, Score: %.2f' % (df_final_outlier.columns[i],v))
Feature: QN_4, Score: 0.12
```

```
Feature: TT_TER, Score: 1.34
Feature: RF_TER, Score: 0.43
Feature: VGSL, Score: 0.18
Feature: TS05, Score: 1.75
Feature: year, Score: 0.01
Feature: month, Score: 0.14
Feature: day, Score: 0.00
```

```
df_final_outlier.columns
```

```
Index(['QN_4', 'TT_TER', 'RF_TER', 'VGSL', 'TS05', 'year', 'month',  
      'day', 'BF10'], dtype='object')
```

```
import matplotlib.pyplot as plt
plt.figure(figsize=(8,7))
col_names=df_final_outlier.columns[df_final_outlier.columns!='BF10']
linear_importances = pd.Series(importances_score, index=col_names)
fig, ax = plt.subplots()
linear_importances.plot.bar()
ax.set_title("soil moisture Feature importances")
ax.set_ylabel("score")
fig.tight_layout()
```

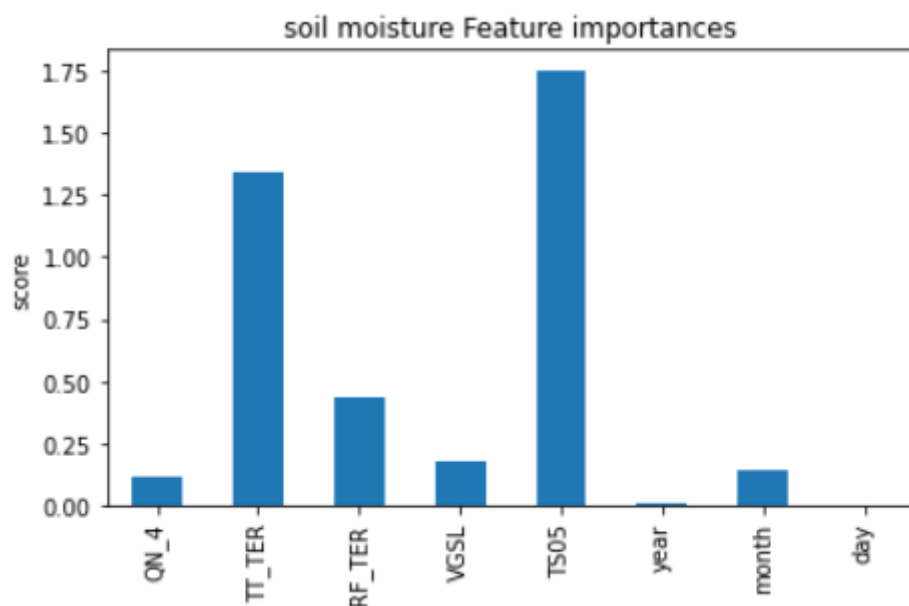


Figure 20 Soil moisture feature importance's

The results of using Linear Regression to select the feature in the figure above show that the most important feature is TS05 with a value of 1.75, and then the TT_TER feature with a value of 1.34, the VGSL feature with a value of 0.18 and the month feature with a value of 0.14 are the most

important features and also the worst. The attribute related to year and day, which is close to zero. The rest of the characteristics also obtained much lower values or attention to the graph.

6.3. Final feature selection

forest importances	
QN_4	0.007597
TT_TER	0.010651
RF_TER	0.013020
VGSL	0.060230
TS05	0.437213
year	0.201471
month	0.193419
day	0.076398
dtype: float64	

linear importances	
QN_4	0.120147
TT_TER	1.342580
RF_TER	0.434354
VGSL	0.180354
TS05	1.747092
year	0.010920
month	0.138561
day	0.000000
dtype: float64	

The results of feature selection of LinearRegression and RandomForestRegressor methods were investigated. The results of these two models show that, according to both models, TS05 feature is one of the most important features, and on the other hand, QN_4 has a low score in both methods of the day feature. Of course, some features have different points in terms of two models. Like the TT_TER feature, in the RandomForestRegressor method, it has a lower score, and vice versa, in the LinearRegression method, it has a higher score. With these results, if we want to choose unimportant and common features between the two models, we have to limit ourselves to removing the two features of the day, QN_4. Of course, if we want to be strict, we can remove some other features.

# drop QN_4 and day features	
df_final_feature	=df_final_outlier.loc[:,df_final_outlier.columns!='QN_4']
df_final_feature	=df_final_feature.loc[:,df_final_feature.columns!='day']
df_final_feature	

Table 12 dataframe of final feature without QN_4 and day

	TT_TER	RF_TER	VGSL	TS05	YEAR	MONTH	BF10
DATE							
1991-01-01 07:00:00	0.411864	0.890244	0.047619	0.293011	0.0	0.0	0.918919
1991-01-01 14:00:00	0.442373	0.817073	0.047619	0.293011	0.0	0.0	0.918919
1991-01-01 21:00:00	0.427119	0.780488	0.047619	0.293011	0.0	0.0	0.918919
1991-01-02 07:00:00	0.455932	0.926829	0.222222	0.384409	0.0	0.0	0.990991
1991-01-02 14:00:00	0.547458	0.841463	0.222222	0.384409	0.0	0.0	0.990991

...
2021-12-30 00:00:00	0.557627	0.878049	0.126984	0.467742	1.0	1.0	0.936937
2021-12-30 06:00:00	0.549153	0.975610	0.126984	0.467742	1.0	1.0	0.936937
2021-12-30 12:00:00	0.603390	0.792683	0.126984	0.467742	1.0	1.0	0.936937
2021-12-30 18:00:00	0.589831	0.878049	0.126984	0.467742	1.0	1.0	0.936937
2021-12-31 00:00:00	0.594915	0.792683	0.126984	0.497312	1.0	1.0	0.909910

35634 rows x 7 columns

7. POTENTIAL RESAMPLING

The purpose of this section is our goal of resampling. But why do we need resampling on the soil moisture dataset? The first point about the soil moisture dataset is related to the BF10 feature and the predictor variable, in which the values are repeated for 6 hours every day, and in fact, we will not lose any information from the BF10 feature by resampling. On the other hand, we are looking for a forecast of 30 days in the future, so converting to daily can have a better forecast than the clock mode with more time steps.

```
plt.figure(figsize=(10,4))
plt.plot(df_final_feature['BF10'])
plt.show()
```

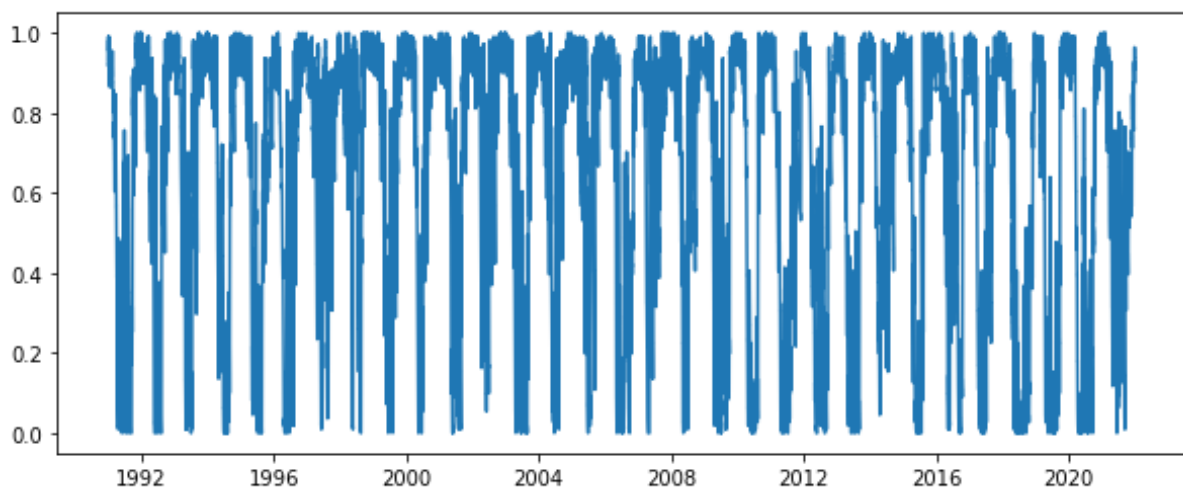


Figure 21 resample_BF10 feature and the predictor variable1

```
df_final_resample=df_final_feature.resample('1d').mean()
plt.figure(figsize=(10,4))
df_final_resample['BF10'].plot()
plt.show()
```

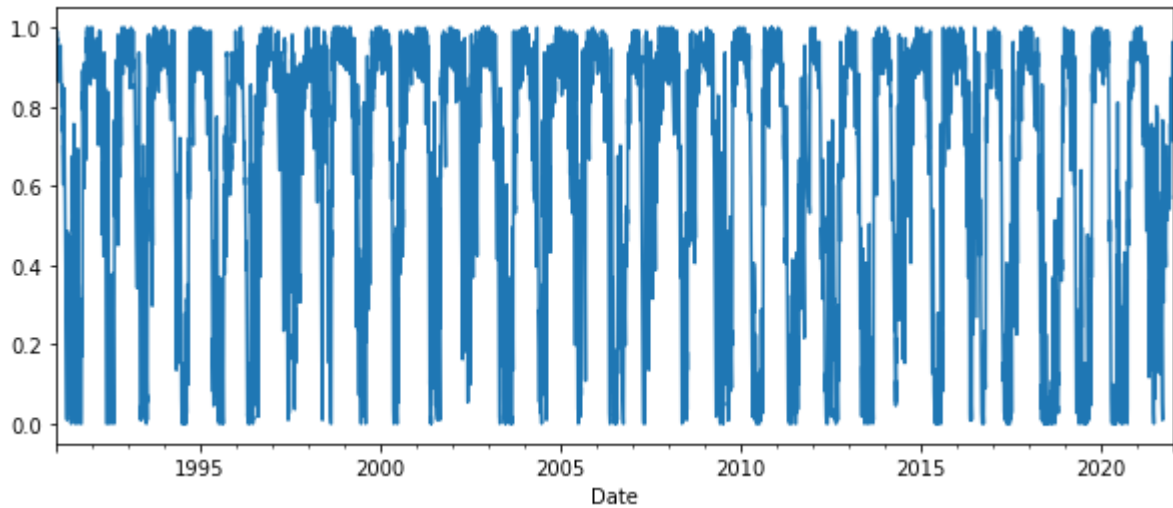


Figure 22 resample_BF10 feature and the predictor variable2

The results of resampling show that we did not give any information about the BF10 feature, but some other features such as TT_TER and RF_TER were calculated from the hourly mode as an average.

```
df_final_feature
```

Table 13 data frame of final feature_resampling

	TT_TER	RF_TER	VGSL	TS05	YEAR	MONTH	BF10
DATE							
1991-01-01 07:00:00	0.411864	0.890244	0.047619	0.293011	0.0	0.0	0.918919
1991-01-01 14:00:00	0.442373	0.817073	0.047619	0.293011	0.0	0.0	0.918919
1991-01-01 21:00:00	0.427119	0.780488	0.047619	0.293011	0.0	0.0	0.918919
1991-01-02 07:00:00	0.455932	0.926829	0.222222	0.384409	0.0	0.0	0.990991
1991-01-02 14:00:00	0.547458	0.841463	0.222222	0.384409	0.0	0.0	0.990991
...
2021-12-30 00:00:00	0.557627	0.878049	0.126984	0.467742	1.0	1.0	0.936937
2021-12-30 06:00:00	0.549153	0.975610	0.126984	0.467742	1.0	1.0	0.936937
2021-12-30 12:00:00	0.603390	0.792683	0.126984	0.467742	1.0	1.0	0.936937
2021-12-30 18:00:00	0.589831	0.878049	0.126984	0.467742	1.0	1.0	0.936937
2021-12-31 00:00:00	0.594915	0.792683	0.126984	0.497312	1.0	1.0	0.909910

35634 rows x 7 columns

```
plt.figure(figsize=(10,4))
df_final_feature['TT_TER'].plot()
```

```
plt.title('feature TT_TER befor resampling')
plt.show()
plt.figure(figsize=(10,4))
df_final_resample['TT_TER'].plot()
plt.title('feature TT_TER after resampling')
plt.show()
```

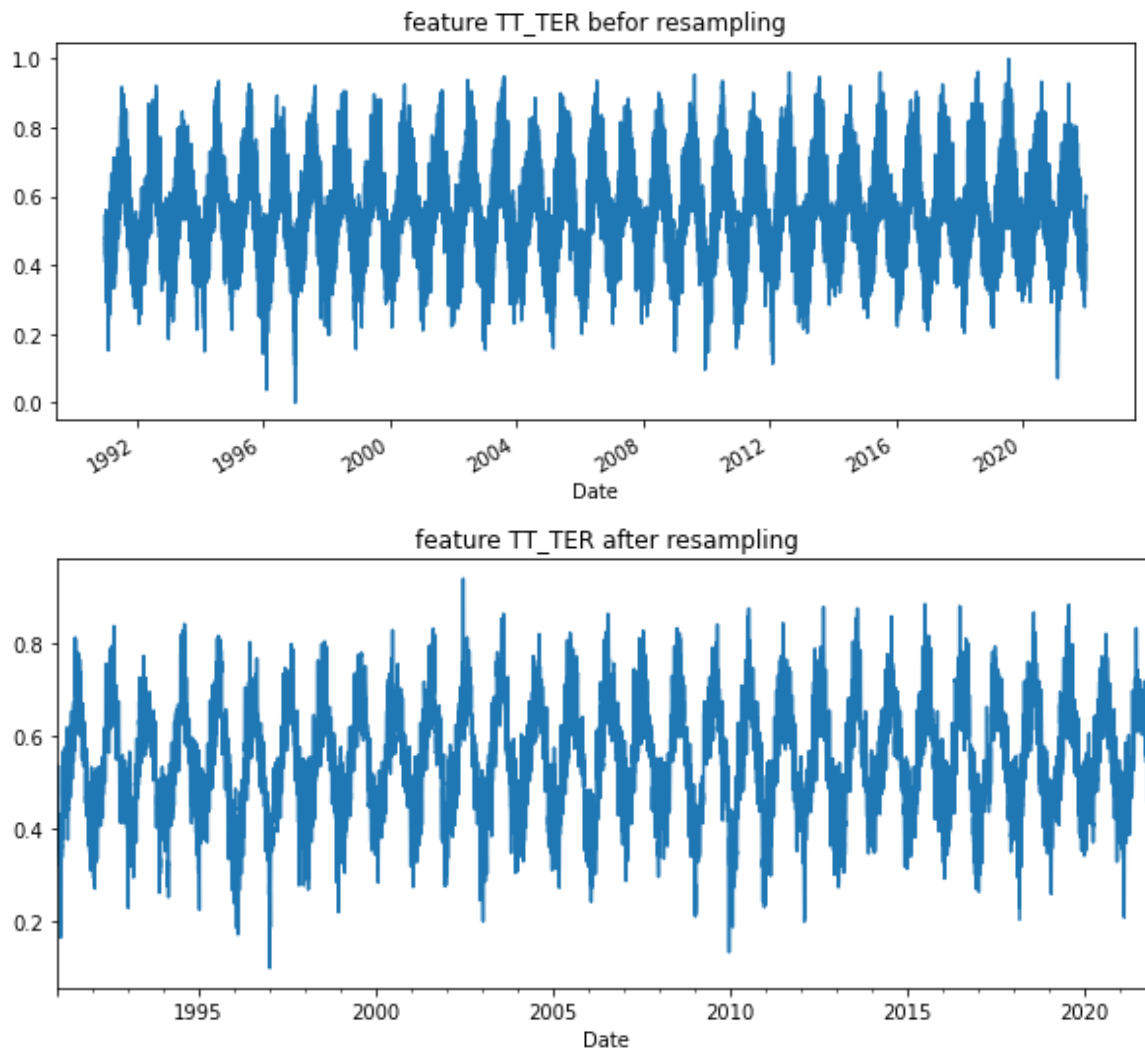


Figure 23 feature TT_TER before and after resampling

```
plt.figure(figsize=(10,4))
df_final_feature['RF_TER'].plot()
plt.title('feature RF_TER befor resampling')
plt.show()
plt.figure(figsize=(10,4))
df_final_resample['RF_TER'].plot()
plt.title('feature RF_TER after resampling')
plt.show()
```

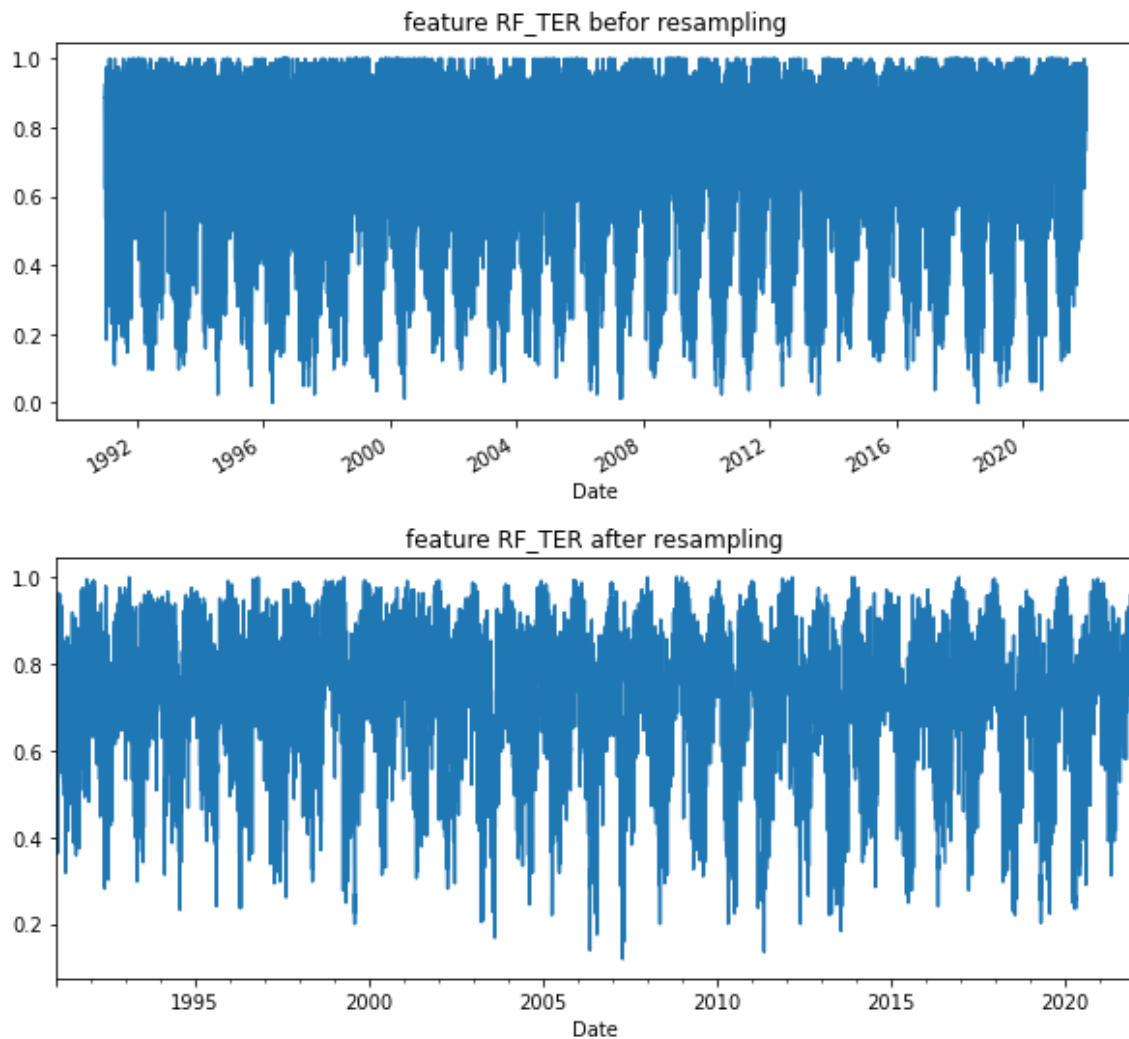


Figure 24 feature RF_TER before and after resampling

8. PREPARATION DATA (FOR MACHINE LEARNING AND DEEP LEARNING)

In this section, our goal is to prepare a dataset for machine learning methods to predict time 30 days ahead. Definitely, the point that we should pay attention to is the type of models and how to look at past data and predict the future. In this work, we rely on two important methods for data preparation time series.

Time window model: In the first model, it is called a candle-based time window mode. In this case, we predict the future by looking at the past data. The data is converted into three dimensions in this method and this model is suitable for deep learning methods.

Current mode model: In the second mode, 30-day forecast is made from current data without using past data. In this method, the data have smaller dimensions and are light, and this model is suitable for machine learning methods.

```
df_final_resample=df_final_resample[~df_final_resample.isnull().any(
axis=1)]
```

```
X_new= df_final_resample.values
y_new=df_final_resample['BF10'].values
print('_'*10)
print('%s shape : %s' % ('X_new', X_new.shape))
print('%s shape : %s' % ('y_new', y_new.shape))
print(X_new.shape)
print('_'*10)
```

```
X_new shape : (11321, 7)
y_new shape : (11321,)
(11321, 7)
```

8.1. Candle model (past_history)

A time window or sliding window is a method that can be defined on time data. In this work, the data is timed and daily. Actually, here the soil moisture data set is defined as a continuous multivariate time series data set D of dimension d with n time steps. so that it can be said that $D=X_1, X_2, X_3, \dots, X_n$ and provided that each $X_i=x^1_i, x^2_i, \dots, x^d_i$ is a vector of the characteristics of the data obtained in The time step is i . Of course, the time window can be defined by two important terms, which are named here as the window width w and step s (an example of a time window with a time step is given in the following figure) (Zhang, Liu, Yu, Feng, & Cheng , 2019)). So the output of the time window is a matrix where each row is a vector (window) w width of the window consisting of n dimensions or features.

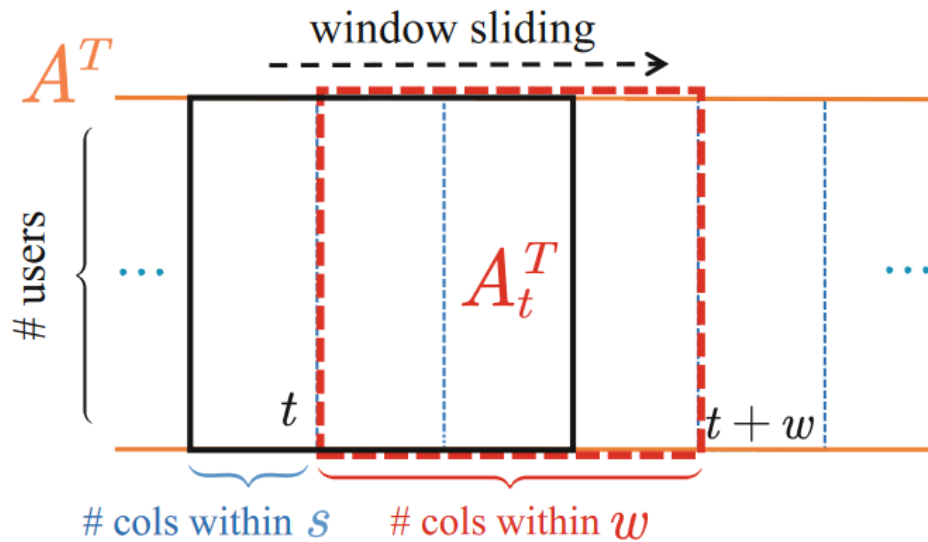


Figure 25 time window with a time step

```
import numpy as np

def multivariate win(data_ordinal, y_out, history_size,
```



```

                                out_size, step_row, one_step=False):
data_array = []
lbl_array = []
start_idx=0
end_idx = None
start_idx = start_idx + history_size
if end_idx is None:
    end_idx = len(data_original) - out_size

for i in range(start_idx, end_idx):
    all_idx = range(i-history_size, i, step_row)
    data_array.append(data_original[all_idx])

    if one_step:
        lbl_array.append(y_out[i+out_size])
    else:
        lbl_array.append(y_out[i:i+out_size])

return np.array(data_array), np.array(lbl_array)

step_row =1
n_step_ahead=30
past_history = 20
index_precentage = -1

data_can, lbl_candel=multivariate_win(X_new,
                                      y_new,
                                      past_history,
                                      n_step_ahead, step_row,
                                      one_step=True)

print("_"*50)
print('-> data_can shape = {0}'.format(data_can.shape))
print('-> lbl shape = {0}'.format(lbl_candel.shape))
print("_"*50)

-> data_can shape = (11271, 20, 7)
-> lbl shape = (11271,)

```

8.2. Next model

```

X_ma=[]
y_ma=[]
n_step_ahead=30
for i in range(X_new.shape[0]-n_step_ahead):
    X_ma.append(X_new[i,:])
    y_ma.append(y_new[i+n_step_ahead])

```

```

X_ma=np.array(X_ma)
y_ma=np.array(y_ma)

print("_"*50)
print('-> X_ma  shape = {0}'.format(X_ma.shape))
print('-> y_ma shape = {0}'.format(y_ma.shape))
print("_"*50)

```

```

-> X_ma  shape = (11291, 7)
-> y_ma shape = (11291,)

```

9. TESTING ALL MODEL ON NEXT DATA

9.1. Train and test split

In this section, we divide the data into two parts, training and testing. In this section, since the data is time, shuffle should be equal to false.(For next data)

```

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_ma , y_ma, test
_size=0.20,shuffle=False, random_state=42)
print("", X_train.shape, "\n", y_train.shape,
      "\n", X test.shape, "\n", y test.shape)

```

```

(9032, 7)
(9032,)
(2259, 7)
(2259,)

```

```

plt.figure(figsize=(10,4))
plt.plot(range(len(y_train)),y_train)
plt.plot(range(len(y_train),len(y_train)+len(y_test)),y_test)
plt.legend(['train', 'test'])
plt.show()

```

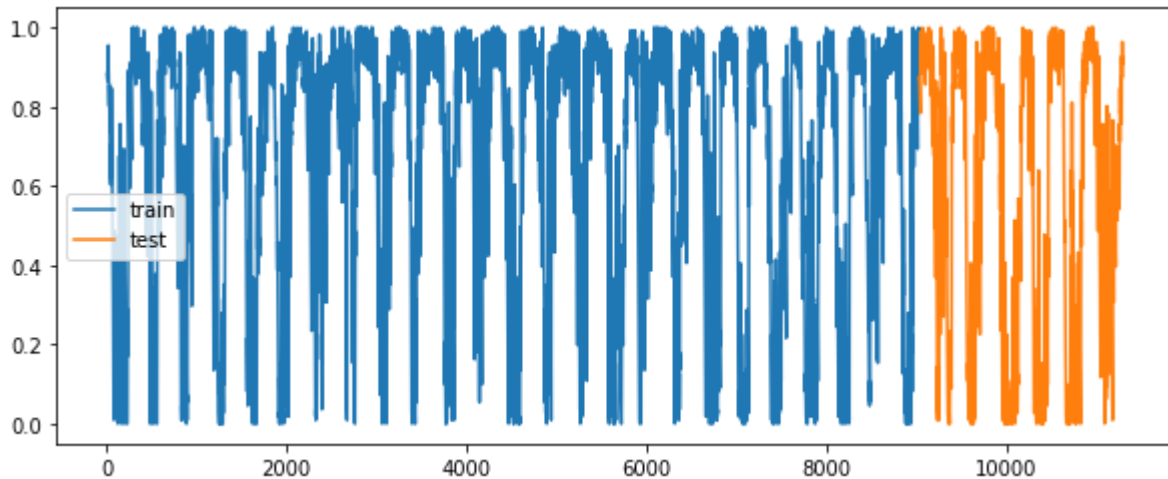


Figure 26 Train and test plot-next data

9.2. Machine learning model

In this section, the goal is to implement the important machine learning algorithm, including K-Nearest-Neighbour, Decision Tree, Random forest, Linear Regression, ExtraTreeRegressor and deep Long Short-Term Memory (LSTM) approach to predict soil moisture for the next 30 days. Also, in this section, various criteria such as mean square error (MSE), root mean square error (RMSE), mean absolute error (MAE) and detection coefficient (R2) are calculated and stored separately for each model.

```
X_train_ma=X_train
X_test_ma=X_test

import numpy as np
algorithms_name=[ 'KNN',
                  'DT',
                  'RF',
                  'XDT',
                  "LSTM"]

alg_num=len(algorithms_name)
mse_array=np.zeros(alg_num)
rmse_array=np.zeros(alg_num)
mae_array=np.zeros(alg_num)
r2_array=np.zeros(alg_num)
K=-1;
```

9.2.1. KNeighborsRegressor

Neighbours-based regression can be used in cases where the data labels are continuous rather than discrete variables. The label assigned to a query point is computed based on the mean of the labels of its nearest neighbours.

```

import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsRegressor
from sklearn import metrics
import math

K=0;
print('-----KNN-----')
reg_model = KNeighborsRegressor(n_neighbors=100)
reg_model.fit(X_train_ma, y_train)
y_pred = reg_model.predict(X_test_ma)
# save to array
mse_array[K]=metrics.mean_squared_error(y_test, y_pred)
rmse_array[K]= math.sqrt(mse_array[K])
mae_array[K]=metrics.mean_absolute_error(y_test,y_pred)
r2_array[K]= metrics.r2_score(y_test, y_pred)

```

9.2.2. Linear Regression

Linear Regression fits a linear model with coefficients $w = (w_1, \dots, w_p)$ to minimize the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation.

```

import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn import metrics
import math

K=1;
print('-----LR-----')
reg_model = LinearRegression()
reg_model.fit(X_train_ma, y_train)
y_pred = reg_model.predict(X_test_ma)
# save to array
mse_array[K]=metrics.mean_squared_error(y_test, y_pred)
rmse_array[K]= math.sqrt(mse_array[K])
mae_array[K]=metrics.mean_absolute_error(y_test,y_pred)
r2_array[K]= metrics.r2_score(y_test, y_pred)

```

9.2.3. DecisionTreeRegressor

Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. A tree can be seen as a piecewise constant approximation.

```

from sklearn.tree import DecisionTreeRegressor

K=1;
print('-----DT-----')

reg_model = DecisionTreeRegressor(min_samples_split=100)
reg_model.fit(X_train_ma, y_train)
y_pred = reg_model.predict(X_test_ma)

# save to array
mse_array[K]=metrics.mean_squared_error(y_test, y_pred)
rmse_array[K]= math.sqrt(mse_array[K])
mae_array[K]=metrics.mean_absolute_error(y_test,y_pred)
r2_array[K]= metrics.r2_score(y_test, y_pred)

```

9.2.4. RandomForestRegressor

A random forest is a Meta estimator that fits a number of classifying decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is controlled with the `max_samples` parameter if `bootstrap=True` (default), otherwise the whole dataset is used to build each tree

```

from sklearn.ensemble import RandomForestRegressor

K=2;
print('-----RandomForestRegressor-----')
reg_model =RandomForestRegressor(max_depth=100, n_estimators=10, random state=125)
reg_model.fit(X_train_ma, y_train)
y_pred = reg_model.predict(X_test_ma)
# save to array
mse_array[K]=metrics.mean_squared_error(y_test, y_pred)
rmse_array[K]= math.sqrt(mse_array[K])
mae_array[K]=metrics.mean_absolute_error(y_test,y_pred)
r2_array[K]= metrics.r2_score(y_test, y_pred)

```

9.2.5. ExtraTreeRegressor

Extra-trees differ from classic decision trees in the way they are built. When looking for the best split to separate the samples of a node into two groups, random splits are drawn for each of the `max_features` randomly selected features and the best split among those is chosen. When `max_features` is set 1, this amounts to building a totally random decision tree.

```

from sklearn.tree import ExtraTreeRegressor
K=3;
print('-----ExtraTreeRegressor-----')
reg_model =ExtraTreeRegressor(random_state=0,min_samples_split=100)
reg_model.fit(X_train_ma, y_train)
y_pred = reg_model.predict(X_test_ma)
# save to array
mse_array[K]=metrics.mean_squared_error(y_test, y_pred)
rmse_array[K]= math.sqrt(mse_array[K])
mae_array[K]=metrics.mean_absolute_error(y_test,y_pred)
r2_array[K]= metrics.r2_score(y_test, y_pred)

```

9.3. Deep learning

Deep learning is an advanced sub-branch of machine learning containing several layers of neurons to realize the learning process. Unlike machine learning methods, deep learning can adapt its performance to macro-scale data. Deep learning techniques have proven their effectiveness in dimensionality reduction, feature extraction, and data classification.

```

X_train_deep=X_train.reshape(X_train.shape[0],
                             X_train.shape[1],1)
X_test_deep=X_test.reshape(X_test.shape[0],
                           X_test.shape[1],1)

```

9.3.1. LSTM

Compared to a standard recurrent neural network, a long-short-term memory network overcomes the problem of gradient (slope) fading to model long-term dependencies and wins. Compared to other versions of recurrent neural networks, the long-short-term memory network, which only computes memory units using different activation functions, has a simple structure. Therefore, short-term long memory network is chosen to predict soil moisture.

```

from tensorflow import keras

layer_input = keras.Input(shape=(X_train_deep.shape[1], X_train_deep.
shape[2]))

x=keras.layers.LSTM(units=64,return_sequences=True,name="lstm1")(layer_input)
x=keras.layers.Activation("relu",name="act_1")(x)
x=keras.layers.Dropout(rate=0.3)(x)
x=keras.layers.LSTM(units=32,return_sequences=True,name="lstm2")(x)

```

```

x=keras.layers.Activation("relu",name="act_2")(x)
x=keras.layers.Flatten()(x)
# finall full layer
out=keras.layers.Dense(1)(x)
model_lstm=keras.models.Model(inputs=layer_input,
                               outputs=out)
opti=keras.optimizers.Adam(learning_rate=0.0001)
model_lstm.compile(loss="mse",
                   optimizer=opti)

```

```
model_lstm.summary()
```

Model: "model"

Layer (type)	Output Shape	Param #
=====		
input_1 (InputLayer)	[(None, 7, 1)]	0
lstm1 (LSTM)	(None, 7, 64)	16896
act_1 (Activation)	(None, 7, 64)	0
dropout (Dropout)	(None, 7, 64)	0
lstm2 (LSTM)	(None, 7, 32)	12416
act_2 (Activation)	(None, 7, 32)	0
flatten (Flatten)	(None, 224)	0
dense (Dense)	(None, 1)	225

```

=====
Total params: 29,537
Trainable params: 29,537
Non-trainable params: 0

```

```

# Train_lable_bin, Test_lable_bin
hist_model_lstm=model_lstm.fit(x=X_train_deep, y=y_train,
                               validation_data=(X_test_deep,y_test),
                               epochs=300,shuffle=False,
                               batch_size=256)

```

```

Epoch 1/400
36/36 [=====] - 10s 32ms/step - loss: 0.5176 - val_loss: 0.4042
Epoch 2/400
36/36 [=====] - 0s 8ms/step - loss: 0.4709 - val_loss: 0.3546
Epoch 3/400
36/36 [=====] - 0s 9ms/step - loss: 0.3922 - val_loss: 0.2675
Epoch 4/400
36/36 [=====] - 0s 8ms/step - loss: 0.2696 - val_loss: 0.1671
Epoch 5/400
36/36 [=====] - 0s 9ms/step - loss: 0.1530 - val_loss: 0.1479

```

```

Epoch 6/400
36/36 [=====] - 0s 8ms/step - loss: 0.1209 - val_loss: 0.1597
Epoch 7/400
36/36 [=====] - 0s 8ms/step - loss: 0.1199 - val_loss: 0.1567
Epoch 8/400
36/36 [=====] - 0s 8ms/step - loss: 0.1192 - val_loss: 0.1565
Epoch 9/400
36/36 [=====] - 0s 9ms/step - loss: 0.1193 - val_loss: 0.1561
Epoch 10/400
36/36 [=====] - 0s 9ms/step - loss: 0.1186 -
.....
.....

36/36 [=====] - 0s 9ms/step - loss: 0.0631 - val_loss: 0.0755
Epoch 391/400
36/36 [=====] - 0s 9ms/step - loss: 0.0626 - val_loss: 0.0760
Epoch 392/400
36/36 [=====] - 0s 8ms/step - loss: 0.0628 - val_loss: 0.0757
Epoch 393/400
36/36 [=====] - 0s 8ms/step - loss: 0.0633 - val_loss: 0.0758
Epoch 394/400
36/36 [=====] - 0s 10ms/step - loss: 0.0628 - val_loss: 0.0754
Epoch 395/400
36/36 [=====] - 0s 9ms/step - loss: 0.0627 - val_loss: 0.0757
Epoch 396/400
36/36 [=====] - 0s 9ms/step - loss: 0.0631 - val_loss: 0.0756
Epoch 397/400
36/36 [=====] - 0s 9ms/step - loss: 0.0631 - val_loss: 0.0764
Epoch 398/400
36/36 [=====] - 0s 9ms/step - loss: 0.0624 - val_loss: 0.0755
Epoch 399/400
36/36 [=====] - 0s 9ms/step - loss: 0.0625 - val_loss: 0.0753
Epoch 400/400
36/36 [=====] - 0s 9ms/step - loss: 0.0631 - val_loss: 0.0757

```

```

K=5;
y_pred = model_lstm.predict(X_test)
# save to array
mse_array[K]=metrics.mean_squared_error(y_test, y_pred)
rmse_array[K]= math.sqrt(mse_array[K])
mae_array[K]=metrics.mean_absolute_error(y_test,y_pred)
r2_array[K]= metrics.r2_score(y_test, y_pred)

```


9.4. Results

In this section, various criteria for evaluating deep and machine learning methods are introduced. Of course, in most cases of prediction and regression, the MSE error measure is used to compare different models. Here, we will use several main criteria for predicting soil moisture such as mean square error (MSE), root mean square error (RMSE), mean absolute error (MAE) and detection coefficient (R^2), which The continuation of the formulas of each of these criteria for evaluating all deep and machine learning models is described along with their explanations.

- **Root Mean Square Error (MSE):** This metric calculates the mean squared distance between the predicted soil moisture values of the deep or machine learning method and the actual soil moisture value. The smaller the MSE values, the better the soil moisture prediction result of the machine or deep learning method. This criterion is stated in equation (5). In equation (5), n represents the number of samples, y_i the actual values of soil moisture and y'_i the predicted values of soil moisture of the machine or deep learning method.

$MSE = \frac{\sum_{i=1}^n (y_i - y'_i)^2}{n}$	(5)
---	-----

- **Root Mean Square Error (RMSE):** This measure is called RMSE if it is taken from the root mean square measure of MSE. In fact, the comparison of MSE and MAE is not correct due to the change of scale of the error value in MSE. Therefore, it is necessary to define the RMSE criterion. This criterion is stated in equation (6). In equation (6), n is the number of samples, y_i is the experimental or actual soil moisture values and y'_i is the predicted soil moisture values of the machine learning or deep learning method. The point is that if the variance in individual errors is greater, the difference between the MAE and RMSE measures will be larger.

$RMSE = \sqrt{\frac{\sum_{i=1}^n (y_i - y'_i)^2}{n}}$	(6)
---	-----

- **Mean absolute error (MAE):** This measure calculates the average absolute difference between the predicted values of the machine learning or deep method and the actual soil moisture values. The smaller the MAE values, the more accurate the prediction result of the machine learning or deep learning method. This criterion is stated in equation (7). In equation (7), n represents the number of samples, y_i the actual values of soil moisture and y'_i the predicted values of soil moisture of the machine or deep learning method.

$MAE = \frac{1}{n} \sum_{i=1}^n y_i - y'_i $	(7)
---	-----

- **Coefficient of determination or detection coefficient (R^2):** This measure calculates the degree to which the predicted values of the machine learning or deep learning method

match the actual values of soil moisture. Unlike other criteria, the closer it is to one, the better. This criterion is stated in equation (8). In equation (8), y_i represents the average of the variables, n is the number of samples, y_i is the actual soil moisture values and y'_i is the predicted soil moisture values of the machine or deep learning method.

$$R2 = 1 - \frac{\sum_{i=1}^n (y_i - y'_i)^2}{\sum_{i=1}^n (y_i - y''_i)^2} \quad (8)$$

```
import pandas as pd
data_zeros=np.zeros((5,5))
data_res_clf=pd.DataFrame(data_zeros,columns=["algorithm_name", "MSE",
"RMSE", "MAE", "R2"])
data_res_clf
```

Table 14 Algorithm_MSE_RMSE_MAE_R2_1(next data)

	ALGORITHM_NAME	MSE	RMSE	MAE	R2
0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0

```
data_res_reg1["algorithm_name"]=algorithms_name

data_res_reg1["MSE"]=np.round(mse_array,4)
data_res_reg1["RMSE"]=np.round(rmse_array,4)
data_res_reg1["MAE"]=np.round(mae_array,4)
data_res_reg1["R2"]=np.round(r2_array,4)
data_res_reg1
```

Table 15 Algorithm_MSE_RMSE_MAE_R2_2(next data)

	ALGORITHM_NAME	MSE	RMSE	MAE	R2
0	KNN	0.0526	0.2294	0.1714	0.5937
1	LR	0.0613	0.2476	0.1959	0.5269
2	DT	0.0756	0.2749	0.1905	0.4165
3	RF	0.0820	0.2864	0.1964	0.3671
4	XDT	0.0567	0.2380	0.1727	0.5627

```
import seaborn as sns
import matplotlib.pyplot as plt
fig=plt.figure(figsize=(10,10))
data_res_reg1.plot(x="algorithm_name",
```

```

y=data_res_reg1.columns[1:5],
kind="bar",figsize=(10,10)).legend(loc='upper le
ft')
plt.ylim([0.0,1])
plt.xlim([-0.5,4.5])
plt.savefig("out.png",dpi=300)
plt.show()

```

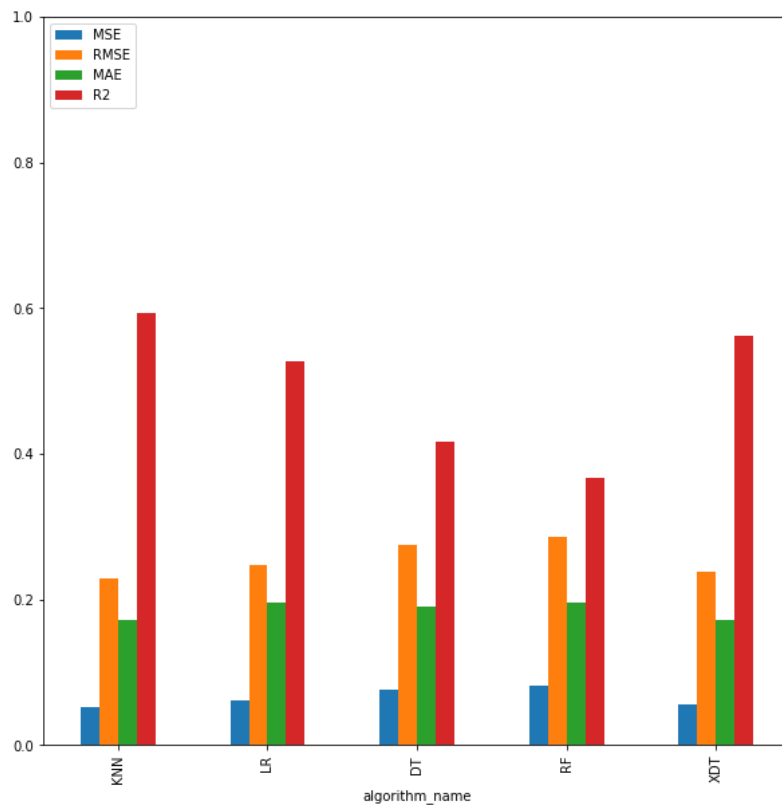


Figure 27 Algorithm of MSE-RMSE-MAE-R2

10. TESTING ALL MODEL ON TIME WINDOW DATA

10.1. Train and test split

In this section, we divide the data into two parts, training and testing. In this section, since the data is time, shuffle should be equal to false. (For time window data)

```

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(data_can , lbl_ca
ndel, test_size=0.20,shuffle=False, random_state=42)
print("", X_train.shape, "\n", y_train.shape,

```

```
"\n", X_test.shape, "\n", y_test.shape)
```

```
(9016, 20, 7)
(9016,)
(2255, 20, 7)
(2255,)
```

```
plt.figure(figsize=(10,4))
plt.plot(range(len(y_train)),y_train)
plt.plot(range(len(y_train),len(y_train)+len(y_test)),y_test)
plt.legend(['train','test'])
plt.show()
```

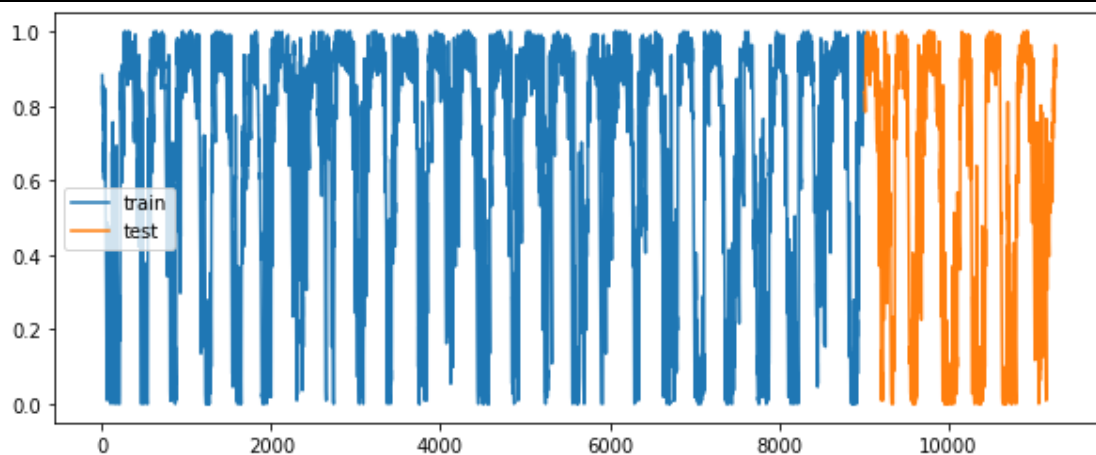


Figure 28 Train and test plot _time window data

10.2. Machine learning model

In this section, the goal is to implement the important machine learning algorithm, including K-Nearest-Neighbor, Decision Tree, Random forest, Linear Regression, ExtraTreeRegressor and deep Long Short-Term Memory (LSTM) approach to predict soil moisture for the next 30 days. Also, in this section, various criteria such as mean square error (MSE), root mean square error (RMSE), mean absolute error (MAE) and detection coefficient (R2) are calculated and stored separately for each model.

```
X_train_ma=X_train.reshape(X_train.shape[0],
                           X_train.shape[1]*X_train.shape[2])
X_test_ma=X_test.reshape(X_test.shape[0],
                         X_test.shape[1]*X_test.shape[2])
```

```
import numpy as np
algorithms_name=['KNN',
                "LR",
                'DT',
                'RF',
                'XDT',
                "LSTM"]
alg_num=len(algorithms_name)
mse_array=np.zeros(alg_num)
```

```
rmse_array=np.zeros(alg_num)
mae_array=np.zeros(alg_num)
r2_array=np.zeros(alg_num)
K=-1;
```

10.2.1. KNeighborsRegressor

```
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsRegressor
from sklearn import metrics
import math

K=0;
print('-----KNN-----')
reg_model = KNeighborsRegressor(n_neighbors=100)
reg_model.fit(X_train_ma, y_train)
y_pred = reg_model.predict(X_test_ma)
# save to array
mse_array[K]=metrics.mean_squared_error(y_test, y_pred)
rmse_array[K]= math.sqrt(mse_array[K])
mae_array[K]=metrics.mean_absolute_error(y_test,y_pred)
r2_array[K]= metrics.r2_score(y_test, y_pred)
```

10.2.2. LinearRegression

```
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn import metrics
import math

K=1;
print('-----LR-----')
reg_model = LinearRegression()
reg_model.fit(X_train_ma, y_train)
y_pred = reg_model.predict(X_test_ma)
# save to array
mse_array[K]=metrics.mean_squared_error(y_test, y_pred)
rmse_array[K]= math.sqrt(mse_array[K])
mae_array[K]=metrics.mean_absolute_error(y_test,y_pred)
r2_array[K]= metrics.r2_score(y_test, y_pred)
```

10.2.3. DecisionTreeRegressor

```
from sklearn.tree import DecisionTreeRegressor
```

```

K=2;
print('-----DT-----')

reg_model = DecisionTreeRegressor(min_samples_split=100)
reg_model.fit(X_train_ma, y_train)
y_pred = reg_model.predict(X_test_ma)

# save to array
mse_array[K]=metrics.mean_squared_error(y_test, y_pred)
rmse_array[K]= math.sqrt(mse_array[K])
mae_array[K]=metrics.mean_absolute_error(y_test,y_pred)
r2_array[K]= metrics.r2_score(y_test, y_pred)

```

10.2.4. RandomForestRegressor

```

from sklearn.ensemble import RandomForestRegressor
K=3;
print('-----RandomForestRegressor-----')
reg_model =RandomForestRegressor(max_depth=100, n_estimators=10,random_state=125)
reg_model.fit(X_train_ma, y_train)
y_pred = reg_model.predict(X_test_ma)
# save to array
mse_array[K]=metrics.mean_squared_error(y_test, y_pred)
rmse_array[K]= math.sqrt(mse_array[K])
mae_array[K]=metrics.mean_absolute_error(y_test,y_pred)
r2_array[K]= metrics.r2_score(y_test, y_pred)

```

10.2.5. ExtraTreeRegressor

```

from sklearn.tree import ExtraTreeRegressor
K=4;
print('-----ExtraTreeRegressor-----')
reg_model =ExtraTreeRegressor(random_state=0,min_samples_split=100)
reg_model.fit(X_train_ma, y_train)
y_pred = reg_model.predict(X_test_ma)
# save to array
mse_array[K]=metrics.mean_squared_error(y_test, y_pred)
rmse_array[K]= math.sqrt(mse_array[K])
mae_array[K]=metrics.mean_absolute_error(y_test,y_pred)
r2_array[K]= metrics.r2_score(y_test, y_pred)

```

10.3. Deep learning

```
X_train_deep=X_train
X_test_deep=X_test
```

10.3.1. LSTM

```
from tensorflow import keras

layer_input = keras.Input(shape=(X_train_deep.shape[1], X_train_deep.
shape[2]))

x=keras.layers.LSTM(units=64,return_sequences=True,name="lstm1")(layer_input)
x=keras.layers.Activation("relu",name="act_1")(x)
x=keras.layers.Dropout(rate=0.3)(x)
x=keras.layers.LSTM(units=32,return_sequences=True,name="lstm2")(x)
x=keras.layers.Activation("relu",name="act_2")(x)
x=keras.layers.Flatten()(x)
# finall full layer
out=keras.layers.Dense(1)(x)
model_lstm=keras.models.Model(inputs=layer_input,
                              outputs=out)
opti=keras.optimizers.Adam(learning_rate=0.0001)
model_lstm.compile(loss="mse",
                  optimizer=opti)
```

```
model_lstm.summary()
```

```
Model: "model_1"
```

Layer (type)	Output Shape	Param #
=====		
input_2 (InputLayer)	[(None, 20, 7)]	0
lstm1 (LSTM)	(None, 20, 64)	18432
act_1 (Activation)	(None, 20, 64)	0
dropout_1 (Dropout)	(None, 20, 64)	0
lstm2 (LSTM)	(None, 20, 32)	12416
act_2 (Activation)	(None, 20, 32)	0
flatten_1 (Flatten)	(None, 640)	0
dense_1 (Dense)	(None, 1)	641

```
=====
Total params: 31,489
Trainable params: 31,489
Non-trainable params: 0
```

```
# Train_lable_bin, Test_lable_bin
hist_model_lstm=model_lstm.fit(x=X_train_deep, y=y_train,
                                validation_data=(X_test_deep,y_test),
                                epochs=100,shuffle=False,
                                batch_size=256)

Epoch 1/100
36/36 [=====] - 4s 28ms/step - loss: 0.3928 - val_loss: 0.2168
Epoch 2/100
36/36 [=====] - 0s 9ms/step - loss: 0.1827 - val_loss: 0.1182
Epoch 3/100
36/36 [=====] - 0s 9ms/step - loss: 0.0967 - val_loss: 0.1185
Epoch 4/100
36/36 [=====] - 0s 8ms/step - loss: 0.0836 - val_loss: 0.0995
Epoch 5/100
36/36 [=====] - 0s 9ms/step - loss: 0.0753 - val loss: 0.085
.....
Epoch 95/100
36/36 [=====] - 0s 10ms/step - loss: 0.0459 - val_loss: 0.0582
Epoch 96/100
36/36 [=====] - 0s 9ms/step - loss: 0.0460 - val_loss: 0.0581
Epoch 97/100
36/36 [=====] - 0s 8ms/step - loss: 0.0461 - val_loss: 0.0584
Epoch 98/100
36/36 [=====] - 0s 8ms/step - loss: 0.0460 - val_loss: 0.0582
Epoch 99/100
36/36 [=====] - 0s 10ms/step - loss: 0.0460 - val_loss: 0.0581
Epoch 100/100
36/36 [=====] - 0s 9ms/step - loss: 0.0460 - val loss: 0.0584
```

10.4. Results

```
import pandas as pd
data_zeros=np.zeros((6,5))
data_res_reg2=pd.DataFrame(data_zeros,columns=["algorithm_name","MSE"
,"RMSE", "MAE", "R2"])
data_res_reg2
```

Table 16 Algorithm_MSE_RMSE_MAE_R2_1(time window data)

	ALGORITHM_NAME	MSE	RMSE	MAE	R2
0		0.0	0.0	0.0	0.0
1		0.0	0.0	0.0	0.0
2		0.0	0.0	0.0	0.0
3		0.0	0.0	0.0	0.0
4		0.0	0.0	0.0	0.0
5		0.0	0.0	0.0	0.0


```

data_res_reg2["algorithm_name"]=algorithms_name

data_res_reg2["MSE"]=np.round(mse_array,4)
data_res_reg2["RMSE"]=np.round(rmse_array,4)
data_res_reg2["MAE"]=np.round(mae_array,4)
data_res_reg2["R2"]=np.round(r2_array,4)
data_res_reg2

```

Table 17 Algorithm_MSE_RMSE_MAE_R2_2(time window data)

	ALGORITHM_NAME	MSE	RMSE	MAE	R2
0	KNN	0.0532	0.2306	0.1682	0.5897
1	LR	0.0602	0.2453	0.1874	0.5356
2	DT	0.0833	0.2886	0.1995	0.3575
3	RF	0.0757	0.2752	0.1943	0.4158
4	XDT	0.0825	0.2872	0.1985	0.3637
5	LSTM	0.0584	0.2416	0.1875	0.5498

```

import seaborn as sns
import matplotlib.pyplot as plt
fig=plt.figure(figsize=(10,10))
data_res_reg2.plot(x="algorithm_name",
                    y=data_res_reg2.columns[1:5],
                    kind="bar",figsize=(10,10)).legend(loc='upper le
ft')
plt.ylim([0.0,1])
plt.xlim([-0.5,4.5])
plt.savefig("out.png",dpi=300)
plt.show()

```

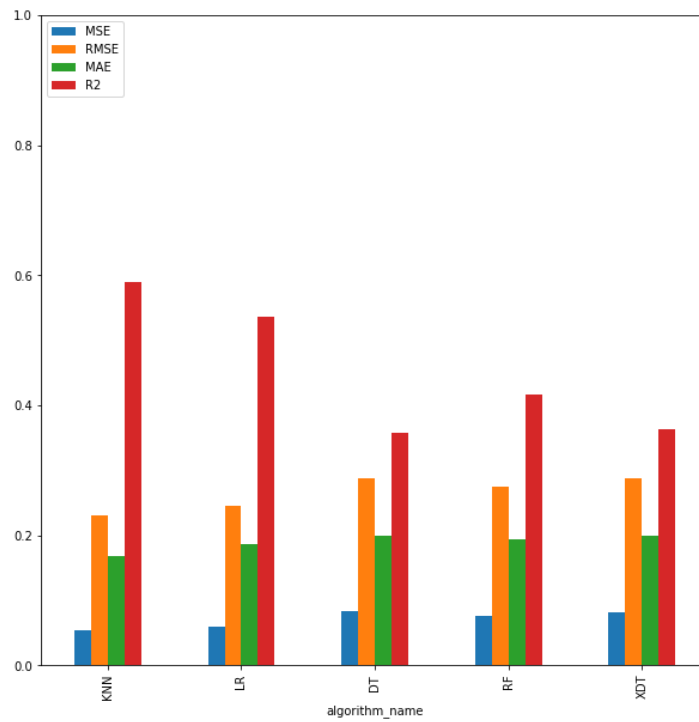


Figure 29 Algorithm of MSE-RMSE-MAE-R2(time window data)

11. EVALUATION AND REFLECTION

data_res_reg1

Table 18 result based on reg1

	ALGORITHM_NAME	MSE	RMSE	MAE	R2
0	KNN	0.0526	0.2294	0.1714	0.5937
1	LR	0.0613	0.2476	0.1959	0.5269
2	DT	0.0756	0.2749	0.1905	0.4165
3	RF	0.0820	0.2864	0.1964	0.3671
4	XDT	0.0567	0.2380	0.1727	0.5627
5	LSTM	0.0757	0.2751	0.2212	0.4161

data_res_reg2

Table 19 result based on reg2

	ALGORITHM_NAME	MSE	RMSE	MAE	R2
0	KNN	0.0532	0.2306	0.1682	0.5897
1	LR	0.0602	0.2453	0.1874	0.5356
2	DT	0.0833	0.2886	0.1995	0.3575
3	RF	0.0757	0.2752	0.1943	0.4158
4	XDT	0.0825	0.2872	0.1985	0.3637
5	LSTM	0.0584	0.2416	0.1875	0.5498

In this section, our goal is to review and evaluate the results of different machine learning algorithms, including the K-Nearest-Neighbor algorithm, Decision Tree, Random forest, Linear Regression, ExtraTreeRegressor, and the Long Short-Term Memory (LSTM) deep approach for predicting the amount of soil moisture. Our results are shown in the two tables above for two different temporal data preparation methods. In the following, we will examine the results of each of the different models in the time data preparation method.

The evaluation results on the first preparation model show that the K-Nearest-Neighbor method performed better in the MSE criterion with a value of 0.052 and ExtraTreeRegressor in the MSE criterion with a value of 0.056 percent compared to other models. Of course, the results of other important criteria such as R2 prove the superiority of these two models in the first case, so that the K-Nearest-Neighbor method has achieved a value of 0.59 and the ExtraTreeRegressor method has achieved a value of 0.56. Of course, among the methods of machine learning and deep decision tree, random forest had relatively poor performance compared to other models. Also, the results prove that the deep approach of Long Short-Term Memory (LSTM) has worked relatively moderately to predict the amount of soil moisture.

The evaluation results on the second preparation model show that the K-Nearest-Neighbor method performed better in the MSE criterion with a value of 0.053 and LSTM in the MSE criterion with a value of 0.056% compared to other models. Of course, the results of other important criteria such as R2 prove the superiority of these two models in the first case, so that the K-Nearest-Neighbor method has achieved a value of 0.58 and the LSTM method has achieved a value of 0.56. Of course, among the methods of machine learning and deep Decision Tree, ExtraTreeRegressor t had relatively poor performance compared to other models. Therefore, the results of this data preparation model prove that the deep Long Short-Term Memory (LSTM) approach has performed relatively better for predicting the amount of soil moisture.

12. CONCLUSIONS

Today, machine and deep learning algorithms have been successful in various fields. Prediction of soil moisture can be considered as a regression and time problem in this research. And then machine learning methods including K-Nearest-Neighbor algorithm, Decision Tree, Random forest, LinearRegression, ExtraTreeRegressor and deep Long Short-Term Memory (LSTM) approach to predict the amount of soil moisture for the next 30 days in the Google colab environment and It was designed and implemented in Python programming language. Of course, we went step by step to predict the price of predicting the amount of soil moisture.

The first step here involved reading a valid soil moisture collection. In the second step, a series of exploratory analysis should be performed on the soil moisture data set. In the third step, we performed pre-processing including normalization and removal of outlier data. In the fourth step, the soil moisture dataset was analysed in terms of time series and time analysis. In the fifth step, we performed two feature selection methods, including the feature selection method based on linear regression and Random Forest (tree base) to identify some important features in the data set. In the sixth step, we used the resampling method to convert the data into daily time. In the seventh step, we proposed and implemented two time data preparation methods, and finally, in the eighth and ninth steps, we tested all the machine learning and deep learning algorithms in terms of different criteria such as mean square error (MSE), root mean square error (RMSE), mean absolute error (MAE) and detection coefficient (R2) were used. In the last part of our work, we evaluated the results of two data preparation models, the results of which include the following.

The evaluation results on the first preparation model show that the K-Nearest-Neighbor method performed better in the MSE criterion with a value of 0.052 and ExtraTreeRegressor in the MSE criterion with a value of 0.056 percent compared to other models. Of course, the results of other important criteria such as R2 prove the superiority of these two models in the first case, so that the K-Nearest-Neighbor method has achieved a value of 0.59 and the ExtraTreeRegressor method has achieved a value of 0.56. Of course, among the methods of machine learning and deep decision tree, random forest had relatively poor performance compared to other models. Also, the results prove that the deep approach of Long Short-Term Memory (LSTM) has worked relatively moderately to predict the amount of soil moisture.

The evaluation results on the second preparation model show that the K-Nearest-Neighbor method performed better in the MSE criterion with a value of 0.053 and LSTM in the MSE criterion with a value of 0.056% compared to other models. Of course, the results of other important criteria such as R2 prove the superiority of these two models in the first case, so that the K-Nearest-Neighbor method has achieved a value of 0.58 and the LSTM method has achieved a value of 0.56. Of course, among the methods of machine learning and deep Decision Tree, ExtraTreeRegressor t had relatively poor performance compared to other models. Therefore, the results of this data preparation model prove that the deep Long Short-Term Memory (LSTM) approach has performed relatively better for predicting the amount of soil moisture.

13. REFERENCES

Akanbi, O. A. A. I. S. F. E. A. O. A. I. & F. E. (., 2015. *Chapter 4—Feature Extraction. A Machine-Learning Approach to Phishing Detection and Defense; Syngress*. Rockland, MA, USA, 45-54.: s.n.

Anon., n.d. <https://machinelearningmastery.com/calculate-feature-importance-with-python/>. [Online].

Anon., n.d. <https://scikit-learn.org>. [Online].

Anon., n.d. https://scikit-learn.org/stable/auto_examples/ensemble/plot_forest_importances.html. [Online].

Durmuş Özkan Şahin, O. E. K. S. A. E. K., 2021. A novel permission-based Android malware detection system using feature selection based on linear regression.

Gijbels, I. & H. M. 1. & Brown, S. T. R. ..., 2009. *Robust and Nonparametric Statistical Methods. Comprehensive Chemometrics*. Walczak, B., Eds, 189-211: s.n.

Shiue, Y.-R. L. K.-C. & S. C.-T. .. 1. 6.-6., 2018. *Real-time scheduling for a smart factory using a reinforcement learning approach. Computers & Industrial Engineering*. s.l.:s.n.

Zhang, J. L. S. Y. W. F. W. & C. X., (2019).. Eigenpulse: Detecting surges in large streaming graphs with row augmentation. Paper presented at the Pacific-Asia Conference on Knowledge Discovery and Data Mining..