hatari labs |Sustainable water management

Python for Hydrology

Session 8 – Filling missing precipitation data

Objective:

Implementation of simple AA, normal ratio, multi-linear regression, and neural network methods for filling gaps in precipitation.

Introduction

Create the folder and notebook for this session.

Start importing all the necessary libraries.

```
import pandas as pd
import numpy as np
```

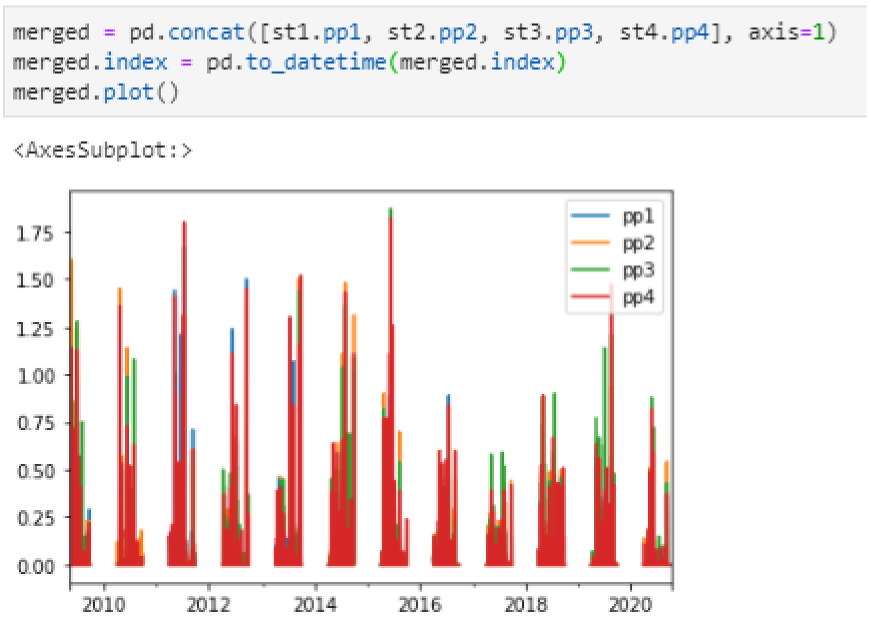
As we have imported lots of files with strange formats, let’s open directly four precipitation data files, change its column names and convert its values to numeric values. All the files are located in the “Data” directory of this session.

```
st1 =
pd.read_csv('st1.txt',skiprows=28,delimiter='\t',index_col='datetime',
parse_dates=True)[1:]
st2 =
pd.read_csv('st2.txt',skiprows=28,delimiter='\t',index_col='datetime',
parse_dates=True)[1:]
st3 =
pd.read_csv('st3.txt',skiprows=28,delimiter='\t',index_col='datetime',
parse_dates=True)[1:]
st4 =
pd.read_csv('st4.txt',skiprows=28,delimiter='\t',index_col='datetime',
parse_dates=True)[1:]
st1.columns = ['agency','site','pp1','code']
st2.columns = ['agency','site','pp2','code']
st3.columns = ['agency','site','pp3','code']
st4.columns = ['agency','site','pp4','code']
st1.pp1 = pd.to_numeric(st1.pp1, errors='coerce')
st2.pp2 = pd.to_numeric(st2.pp2, errors='coerce')
st3.pp3 = pd.to_numeric(st3.pp3, errors='coerce')
```

```
st4.pp4 = pd.to_numeric(st4.pp4, errors='coerce')
```

Concatenate all the stations in one single DataFrame, fix the index as the dates, and plot to see its distribution.

```
merged = pd.concat([st1.pp1, st2.pp2, st3.pp3, st4.pp4], axis=1)
merged.index = pd.to_datetime(merged.index)
merged.plot()
```

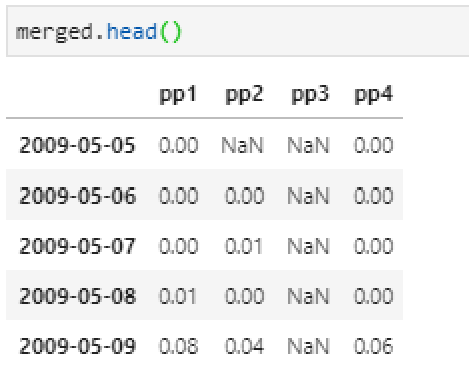


```
merged = pd.concat([st1.pp1, st2.pp2, st3.pp3, st4.pp4], axis=1)
merged.index = pd.to_datetime(merged.index)
merged.plot()
```

```
<AxesSubplot:>
```

If you want to confirm that the concatenated DataFrame is well formatted, check its first values.

```
merged.head()
```



```
merged.head()
```

	pp1	pp2	pp3	pp4
2009-05-05	0.00	NaN	NaN	0.00
2009-05-06	0.00	0.00	NaN	0.00
2009-05-07	0.00	0.01	NaN	0.00
2009-05-08	0.01	0.00	NaN	0.00
2009-05-09	0.08	0.04	NaN	0.06

You can export the DataFrame to an Excel file if you want to keep the records. However, before continuing, you would need the "openpyxl" library. In case you don't have this library, you can install it directly inside Jupyter Lab.

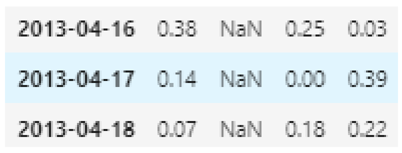
```
!pip install openpyxl
```

To save your DataFrame use:

```
merged.to_excel('merged.xlsx')
```

For the first two filling precipitation data methods, we use the data regarding the year 2013 and the month of April. If you see its records, you could find 3 values with "Nan" for the column "pp2"

```
merged['2013-04']
```



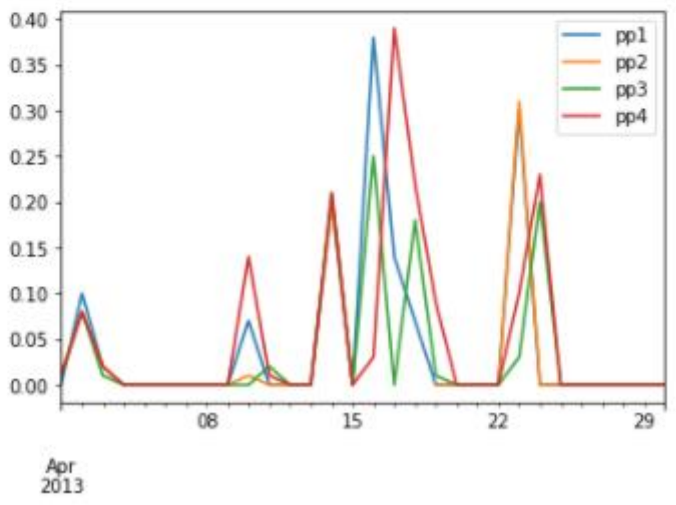
2013-04-16	0.38	NaN	0.25	0.03
2013-04-17	0.14	NaN	0.00	0.39
2013-04-18	0.07	NaN	0.18	0.22

You can see its distribution by making a line plot of all of them

```
merged['2013-04'].plot(kind='line')
```

```
merged['2013-04'].plot(kind='line')
```

```
<AxesSubplot:>
```



We choose this month because there are only 3 missing values, so it is useful for better visualization of our calculations. Store this month in the variable “ppf”

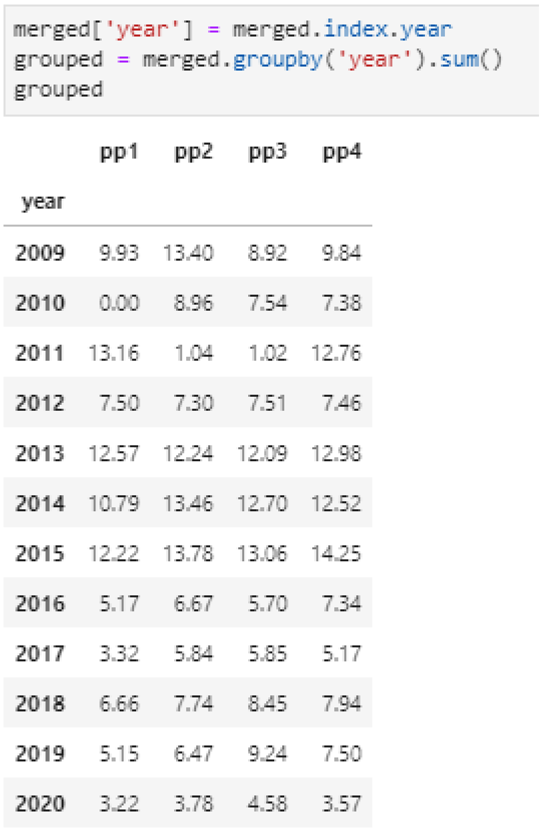
```
ppf = merged['2013-04']
```

Simple AA

Simple Arithmetic Mean Method/ Local Mean Method / AA: This is the simplest method commonly used to fill in missing meteorological data in meteorology and climatology. If the normal annual precipitations at surrounding gauges are within the range of 10% of the normal annual precipitation at station X, then the Arithmetic procedure could be adopted to estimate the missing observation of station X. This assumes equal weights from all nearby rain gauge stations and uses the arithmetic mean of precipitation records of them as an estimate

To start, let's check the assumption. Let's compare the yearly values of precipitation in the four weather stations. Some may vary a lot because of the information gaps. If you see the years 2012 till 2018, you could see a strong relationship in the precipitation.

```
merged['year'] = merged.index.year
grouped = merged.groupby('year').sum()
grouped
```

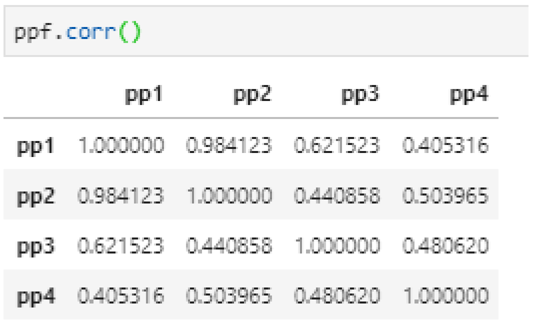


```
merged['year'] = merged.index.year
grouped = merged.groupby('year').sum()
grouped
```

year	pp1	pp2	pp3	pp4
2009	9.93	13.40	8.92	9.84
2010	0.00	8.96	7.54	7.38
2011	13.16	1.04	1.02	12.76
2012	7.50	7.30	7.51	7.46
2013	12.57	12.24	12.09	12.98
2014	10.79	13.46	12.70	12.52
2015	12.22	13.78	13.06	14.25
2016	5.17	6.67	5.70	7.34
2017	3.32	5.84	5.85	5.17
2018	6.66	7.74	8.45	7.94
2019	5.15	6.47	9.24	7.50
2020	3.22	3.78	4.58	3.57

Another way to see the relationship between them is to calculate the correlation coefficients.

```
ppf.corr()
```

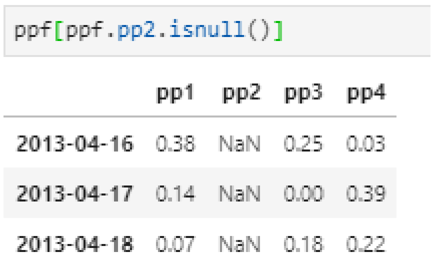


```
ppf.corr()
```

	pp1	pp2	pp3	pp4
pp1	1.000000	0.984123	0.621523	0.405316
pp2	0.984123	1.000000	0.440858	0.503965
pp3	0.621523	0.440858	1.000000	0.480620
pp4	0.405316	0.503965	0.480620	1.000000

We want to separate the rows that have no data in the "pp2" column, do it as follows:

```
ppf[ppf.pp2.isnull()]
```



```
ppf[ppf.pp2.isnull()]
```

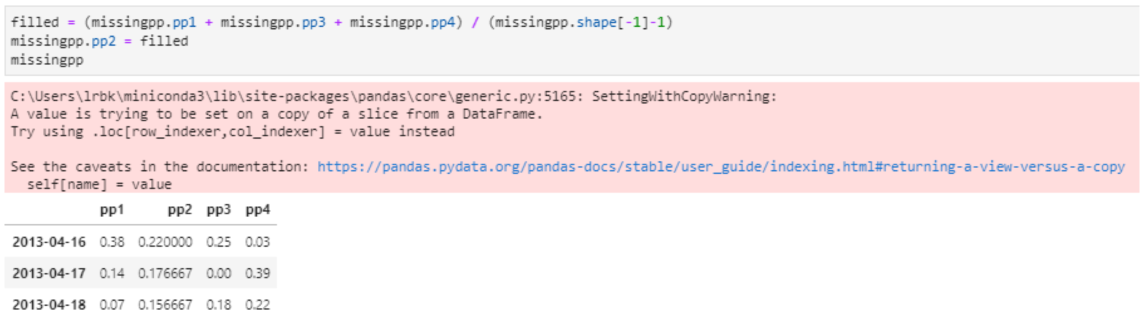
	pp1	pp2	pp3	pp4
2013-04-16	0.38	NaN	0.25	0.03
2013-04-17	0.14	NaN	0.00	0.39
2013-04-18	0.07	NaN	0.18	0.22

Store these values in a new variable called "missingpp"

```
missingpp = ppf[ppf.pp2.isnull()]
missingpp
```

Finally, fill the values with the "simple AA" method, which makes an average of the other stations with data.

```
filled = (missingpp.pp1 + missingpp.pp3 + missingpp.pp4) /
(missingpp.shape[-1]-1)
missingpp.pp2 = filled
missingpp
```



```
filled = (missingpp.pp1 + missingpp.pp3 + missingpp.pp4) / (missingpp.shape[-1]-1)
missingpp.pp2 = filled
missingpp
```

```
C:\Users\lrbk\miniconda3\lib\site-packages\pandas\core\generic.py:5165: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self[name] = value
```

	pp1	pp2	pp3	pp4
2013-04-16	0.38	0.220000	0.25	0.03
2013-04-17	0.14	0.176667	0.00	0.39
2013-04-18	0.07	0.156667	0.18	0.22

Normal ratio method

This method is used if any surrounding gauges have the normal annual precipitation exceeding 10% of the considered gauge. This weighs the effect of each surrounding station. If the normal precipitations vary considerably then, Px is estimated by weighting the precipitation at various stations by the ratios of normal annual precipitation.

To calculate this method, you need to weigh the variables.

We can lump all the parameters needed as follows:

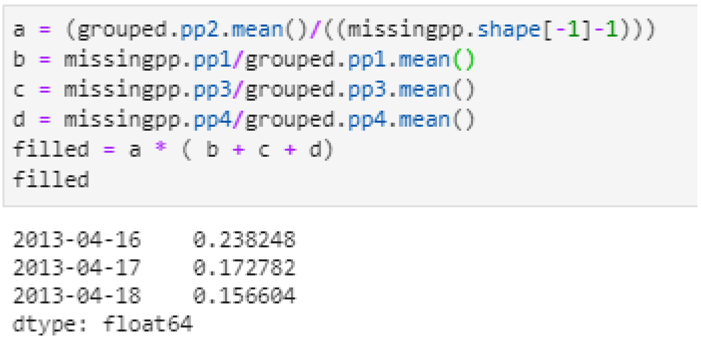
```
filled = (grouped.pp2.mean()/((missingpp.shape[-1]-
1)))*(missingpp.pp1/grouped.pp1.mean() +
missingpp.pp3/grouped.pp3.mean() + missingpp.pp4/grouped.pp4.mean())
filled
```

But the previous equation doesn't allow us to have adequate control of the equation. It could not be very clear if you have additional variables.

In the following script, we can see how to build this method.

- You need to calculate the yearly means based on the data you have. This value is stored in the variable "a" in this variable. Divide by the amount of the other stations we are using.\
- Variables "b," "c," and "d" store the values divided by the yearly mean of the data available
- Variable "filled" stores our result

```
a = (grouped.pp2.mean()/((missingpp.shape[-1]-1)))
b = missingpp.pp1/grouped.pp1.mean()
c = missingpp.pp3/grouped.pp3.mean()
d = missingpp.pp4/grouped.pp4.mean()
filled = a * ( b + c + d)
filled
```



```
a = (grouped.pp2.mean()/((missingpp.shape[-1]-1)))
b = missingpp.pp1/grouped.pp1.mean()
c = missingpp.pp3/grouped.pp3.mean()
d = missingpp.pp4/grouped.pp4.mean()
filled = a * ( b + c + d)
filled
```

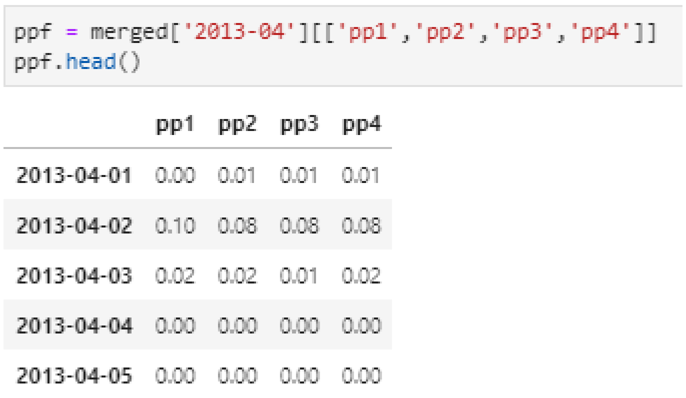
```
2013-04-16    0.238248
2013-04-17    0.172782
2013-04-18    0.156604
dtype: float64
```

Multiple linear regression

Multiple Linear Regressions (MLR) is a statistical method for estimating the relationship between a dependent variable and two or more independent, or predictor, variables. MLR identifies the best-weighted combination of independent variables to predict the dependent, or criterion, variable.

To start, let's select only the data with information regarding precipitation

```
ppf = merged['2013-04'][['pp1','pp2','pp3','pp4']]
ppf.head()
```

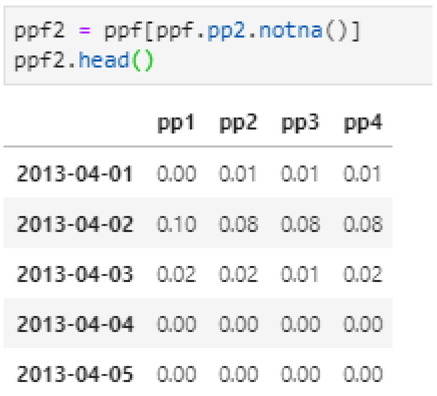


```
ppf = merged['2013-04'][['pp1','pp2','pp3','pp4']]
ppf.head()
```

	pp1	pp2	pp3	pp4
2013-04-01	0.00	0.01	0.01	0.01
2013-04-02	0.10	0.08	0.08	0.08
2013-04-03	0.02	0.02	0.01	0.02
2013-04-04	0.00	0.00	0.00	0.00
2013-04-05	0.00	0.00	0.00	0.00

For this method, we need the values with data

```
ppf2 = ppf[ppf.pp2.notna()]
ppf2.head()
```



```
ppf2 = ppf[ppf.pp2.notna()]
ppf2.head()
```

	pp1	pp2	pp3	pp4
2013-04-01	0.00	0.01	0.01	0.01
2013-04-02	0.10	0.08	0.08	0.08
2013-04-03	0.02	0.02	0.01	0.02
2013-04-04	0.00	0.00	0.00	0.00
2013-04-05	0.00	0.00	0.00	0.00

We need to import the "sklearn" library

```
from sklearn.linear_model import LinearRegression
```

If you don't have it, install it.

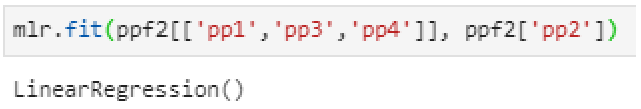
```
!pip install sklearn
```

Create an empty Linear Regression object

```
mlr = LinearRegression()
```

Add the values of precipitation. They need to have the shape (X, Y), "X" can be a matrix.

```
mlr.fit(ppf2[['pp1','pp3','pp4']], ppf2['pp2'])
```

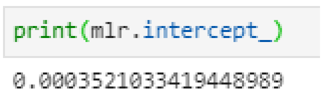


```
mlr.fit(ppf2[['pp1','pp3','pp4']], ppf2['pp2'])
```

```
LinearRegression()
```

This method returns us the intercept of the regression.

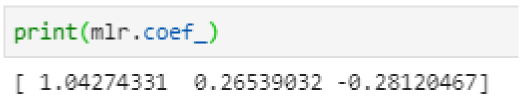
```
print(mlr.intercept_)
```



```
print(mlr.intercept_)
0.0003521033419448989
```

And the coefficients of the regression.

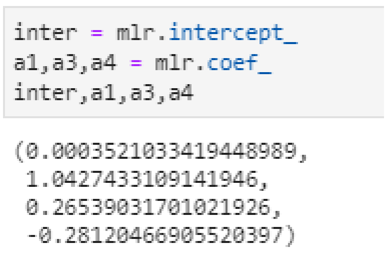
```
print(mlr.coef_)
```



```
print(mlr.coef_)
[ 1.04274331  0.26539032 -0.28120467]
```

Store these coefficients in variables

```
inter = mlr.intercept_
a1,a3,a4  = mlr.coef_
inter,a1,a3,a4
```

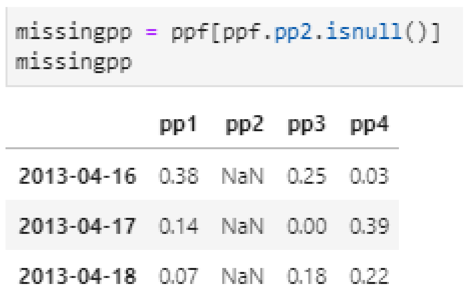


```
inter = mlr.intercept_
a1,a3,a4 = mlr.coef_
inter,a1,a3,a4

(0.0003521033419448989,
 1.0427433109141946,
 0.26539031701021926,
 -0.28120466905520397)
```

Filter the DataFrame with the rows with missing values.

```
missingpp = ppf[ppf.pp2.isnull()]
missingpp
```

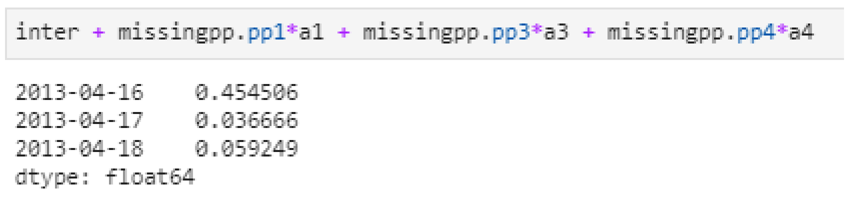


```
missingpp = ppf[ppf.pp2.isnull()]
missingpp
```

	pp1	pp2	pp3	pp4
2013-04-16	0.38	NaN	0.25	0.03
2013-04-17	0.14	NaN	0.00	0.39
2013-04-18	0.07	NaN	0.18	0.22

Based on this new DataFrame, we can calculate the new parameters with the interception value and correlations.

```
inter + missingpp.pp1*a1 + missingpp.pp3*a3 + missingpp.pp4*a4
```



```
inter + missingpp.pp1*a1 + missingpp.pp3*a3 + missingpp.pp4*a4
```

```
2013-04-16    0.454506
2013-04-17    0.036666
2013-04-18    0.059249
dtype: float64
```

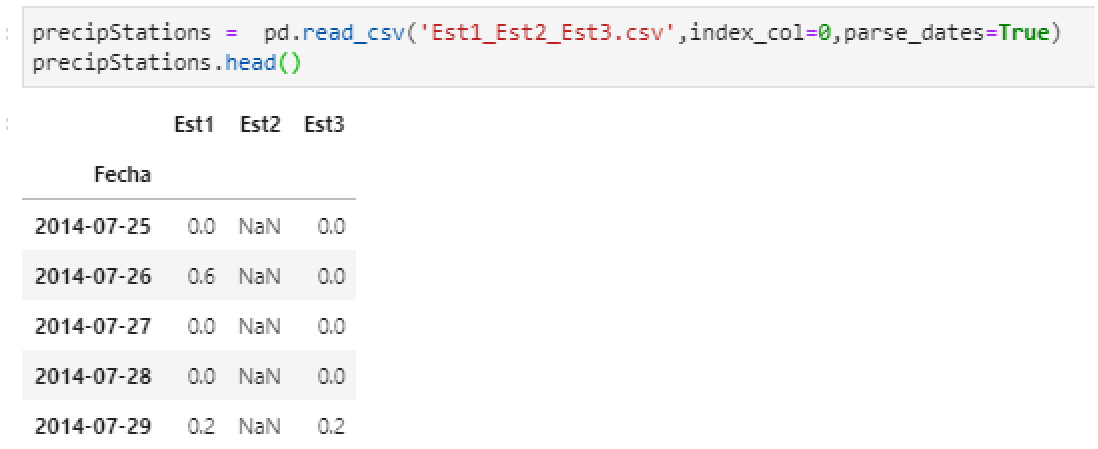
Neural Networks

Start importing all the libraries required

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPRegressor
```

Read the DataFrame with the precipitation stations.

```
precipStations =
pd.read_csv('Est1_Est2_Est3.csv',index_col=0,parse_dates=True)
precipStations.head()
```

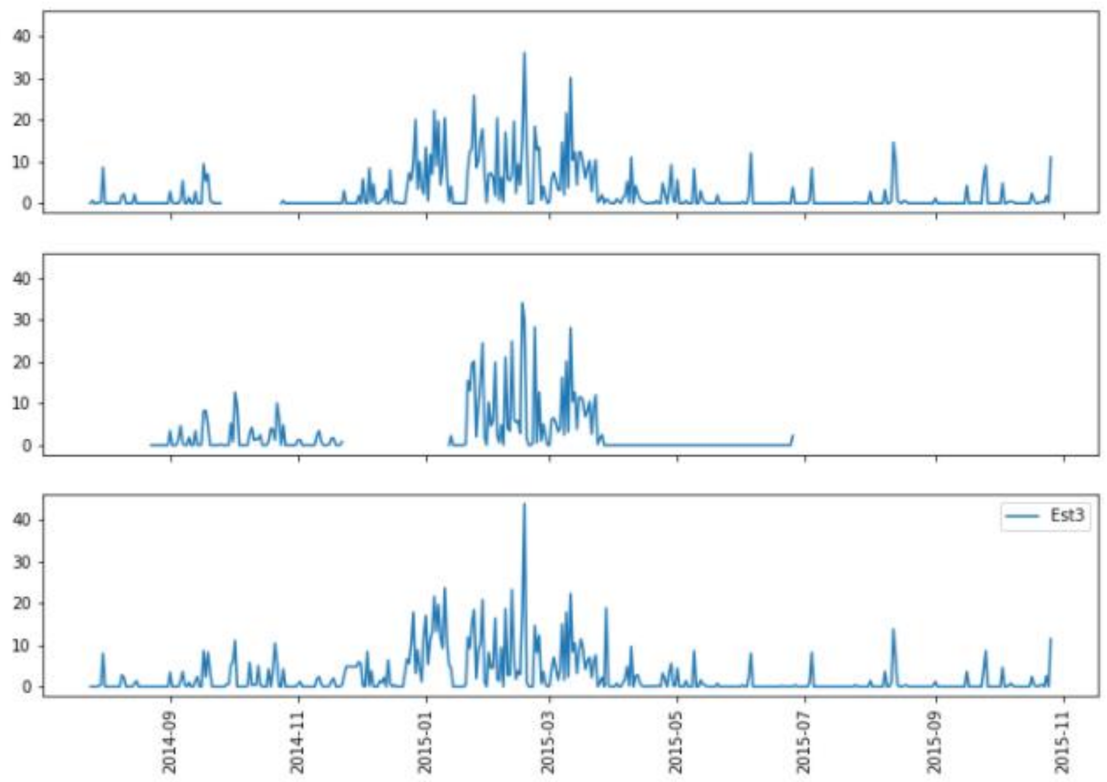


```
precipStations =  pd.read_csv('Est1_Est2_Est3.csv',index_col=0,parse_dates=True)
precipStations.head()
```

	Est1	Est2	Est3
Fecha			
2014-07-25	0.0	NaN	0.0
2014-07-26	0.6	NaN	0.0
2014-07-27	0.0	NaN	0.0
2014-07-28	0.0	NaN	0.0
2014-07-29	0.2	NaN	0.2

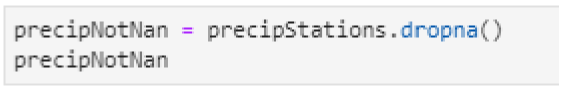
Plot each station in a different axis

```
#precipitation hidrogram
fig, axs=plt.subplots(3,1,figsize=(12,8),sharex=True,sharey=True)
axs[0].plot(precipStations.index,precipStations['Est1'],label='Est1')
axs[1].plot(precipStations.index,precipStations['Est2'],label='Est2')
axs[2].plot(precipStations.index,precipStations['Est3'],label='Est3')
plt.legend()
plt.xticks(rotation='vertical')
plt.show()
```

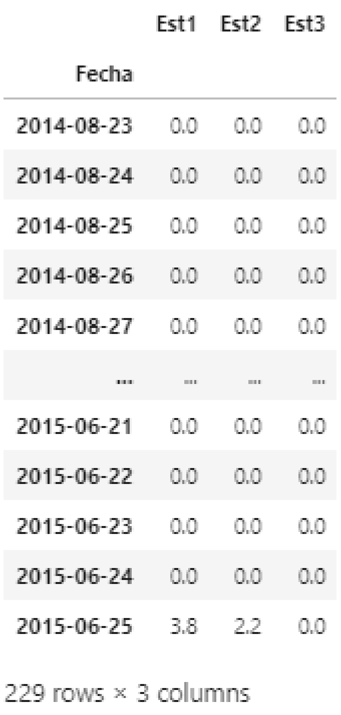


Erase all rows with “Nan” values:

```
precipNotNan = precipStations.dropna()
precipNotNan
```



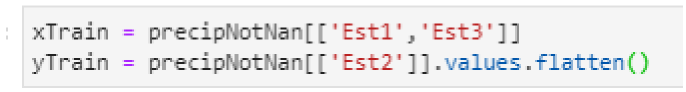
```
precipNotNan = precipStations.dropna()
precipNotNan
```



	Est1	Est2	Est3
Fecha			
2014-08-23	0.0	0.0	0.0
2014-08-24	0.0	0.0	0.0
2014-08-25	0.0	0.0	0.0
2014-08-26	0.0	0.0	0.0
2014-08-27	0.0	0.0	0.0
...	...	...	...
2015-06-21	0.0	0.0	0.0
2015-06-22	0.0	0.0	0.0
2015-06-23	0.0	0.0	0.0
2015-06-24	0.0	0.0	0.0
2015-06-25	3.8	2.2	0.0

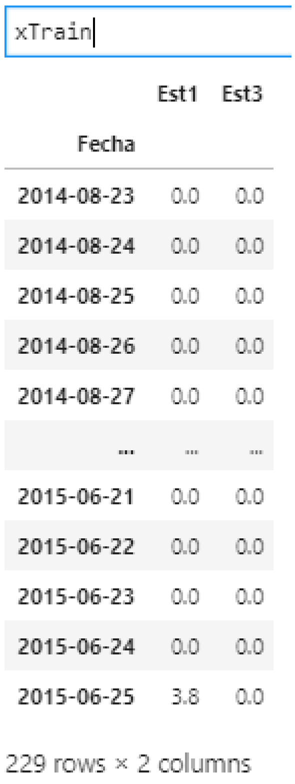
229 rows × 3 columns

Create your train variables. Our target station is “Est2”



```
xTrain = precipNotNan[['Est1','Est3']]
yTrain = precipNotNan[['Est2']].values.flatten()
```

Our “xTrain” is as follows:

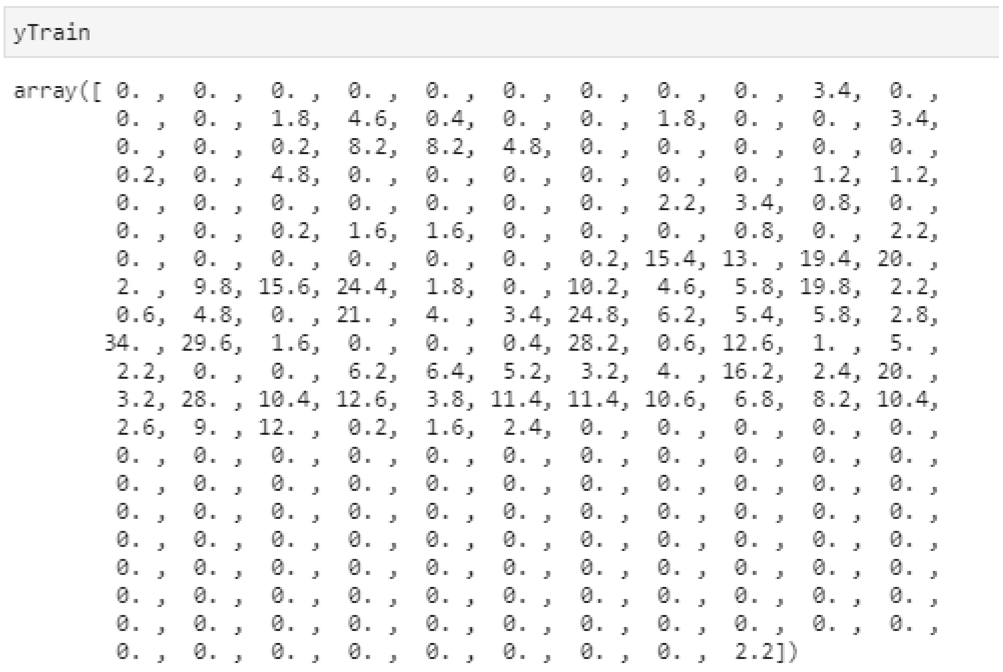


```
xTrain
```

	Est1	Est3
Fecha		
2014-08-23	0.0	0.0
2014-08-24	0.0	0.0
2014-08-25	0.0	0.0
2014-08-26	0.0	0.0
2014-08-27	0.0	0.0
...	...	...
2015-06-21	0.0	0.0
2015-06-22	0.0	0.0
2015-06-23	0.0	0.0
2015-06-24	0.0	0.0
2015-06-25	3.8	0.0

229 rows × 2 columns

Our “yTrain” values:



```
yTrain
```

```
array([ 0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  3.4,  0. ,
        0. ,  0. ,  1.8,  4.6,  0.4,  0. ,  0. ,  1.8,  0. ,  0. ,  3.4,
        0. ,  0. ,  0.2,  8.2,  8.2,  4.8,  0. ,  0. ,  0. ,  0. ,  0. ,
        0.2,  0. ,  4.8,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  1.2,  1.2,
        0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  2.2,  3.4,  0.8,  0. ,
        0. ,  0. ,  0.2,  1.6,  1.6,  0. ,  0. ,  0. ,  0.8,  0. ,  2.2,
        0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0.2, 15.4, 13. , 19.4, 20. ,
        2. ,  9.8, 15.6, 24.4,  1.8,  0. , 10.2,  4.6,  5.8, 19.8,  2.2,
        0.6,  4.8,  0. , 21. ,  4. ,  3.4, 24.8,  6.2,  5.4,  5.8,  2.8,
       34. , 29.6,  1.6,  0. ,  0. ,  0.4, 28.2,  0.6, 12.6,  1. ,  5. ,
        2.2,  0. ,  0. ,  6.2,  6.4,  5.2,  3.2,  4. , 16.2,  2.4, 20. ,
        3.2, 28. , 10.4, 12.6,  3.8, 11.4, 11.4, 10.6,  6.8,  8.2, 10.4,
        2.6,  9. , 12. ,  0.2,  1.6,  2.4,  0. ,  0. ,  0. ,  0. ,  0. ,
        0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,
        0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,
        0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,
        0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,
        0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,
        0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,
        0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,
        0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  2.2])
```

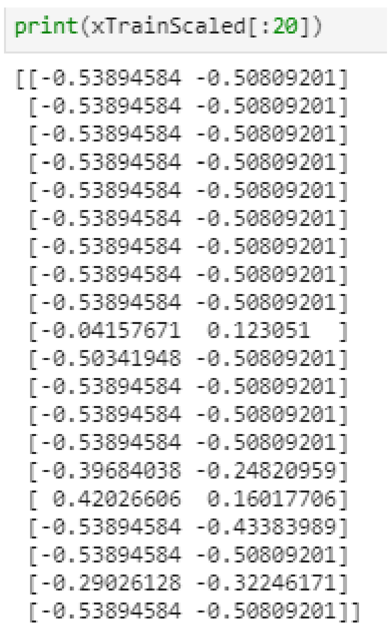
Create the "Standard score" based on our "xTrain" values. Data scaling or normalization is a process of making model data in a standard format so that the training is improved, accurate, and faster. The method of scaling data in neural networks is similar to data normalization in any machine learning problem.

```
scaler = StandardScaler().fit(xTrain)
```

Perform standardization by centering and scaling, to do this use the "scaler" variable to create an "xTrainScaled" variable

```
xTrainScaled = scaler.transform(xTrain)
```

You would get an array as follows:

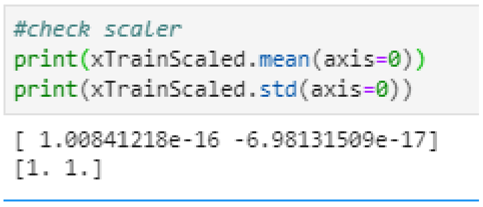


```
print(xTrainScaled[:20])
```

```
[[-0.53894584 -0.50809201]
 [-0.53894584 -0.50809201]
 [-0.53894584 -0.50809201]
 [-0.53894584 -0.50809201]
 [-0.53894584 -0.50809201]
 [-0.53894584 -0.50809201]
 [-0.53894584 -0.50809201]
 [-0.53894584 -0.50809201]
 [-0.53894584 -0.50809201]
 [-0.04157671  0.123051  ]
 [-0.50341948 -0.50809201]
 [-0.53894584 -0.50809201]
 [-0.53894584 -0.50809201]
 [-0.53894584 -0.50809201]
 [-0.39684038 -0.24820959]
 [ 0.42026606  0.16017706]
 [-0.53894584 -0.43383989]
 [-0.53894584 -0.50809201]
 [-0.29026128 -0.32246171]
 [-0.53894584 -0.50809201]]
```

You can check the "scaler" by seeing the mean and standard deviation of the "xTrainedScalar"

```
#check scaler
print(xTrainScaled.mean(axis=0))
print(xTrainScaled.std(axis=0))
```



```
#check scaler
print(xTrainScaled.mean(axis=0))
print(xTrainScaled.std(axis=0))
```

```
[ 1.00841218e-16 -6.98131509e-17]
[1. 1.]
```

Calculate the regressor, and fit the "xTrainScaled" and "yTrain" variables

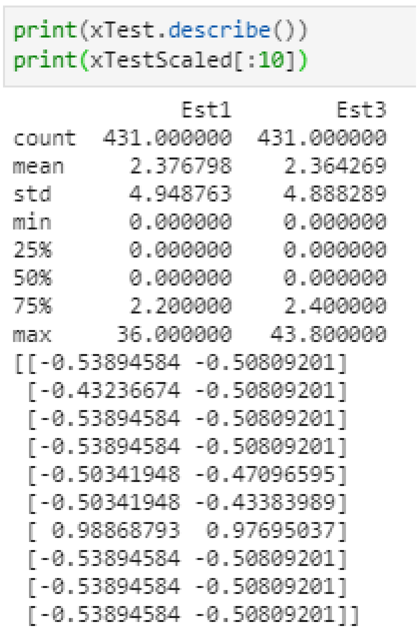
```
#regressor
regr = MLPRegressor(random_state=1, max_iter=5000).fit(xTrainScaled,
yTrain)
```

Test the scalar; we delete the "nan" values inside "Est1" and "Est2" because if they have missing values won't be useful for our neural network

```
#test
xTest = precipStations[['Est1','Est3']].dropna()
xTestScaled = scaler.transform(xTest)
```

Print its results of this new Scaler.

```
print(xTest.describe())
print(xTestScaled[:10])
```

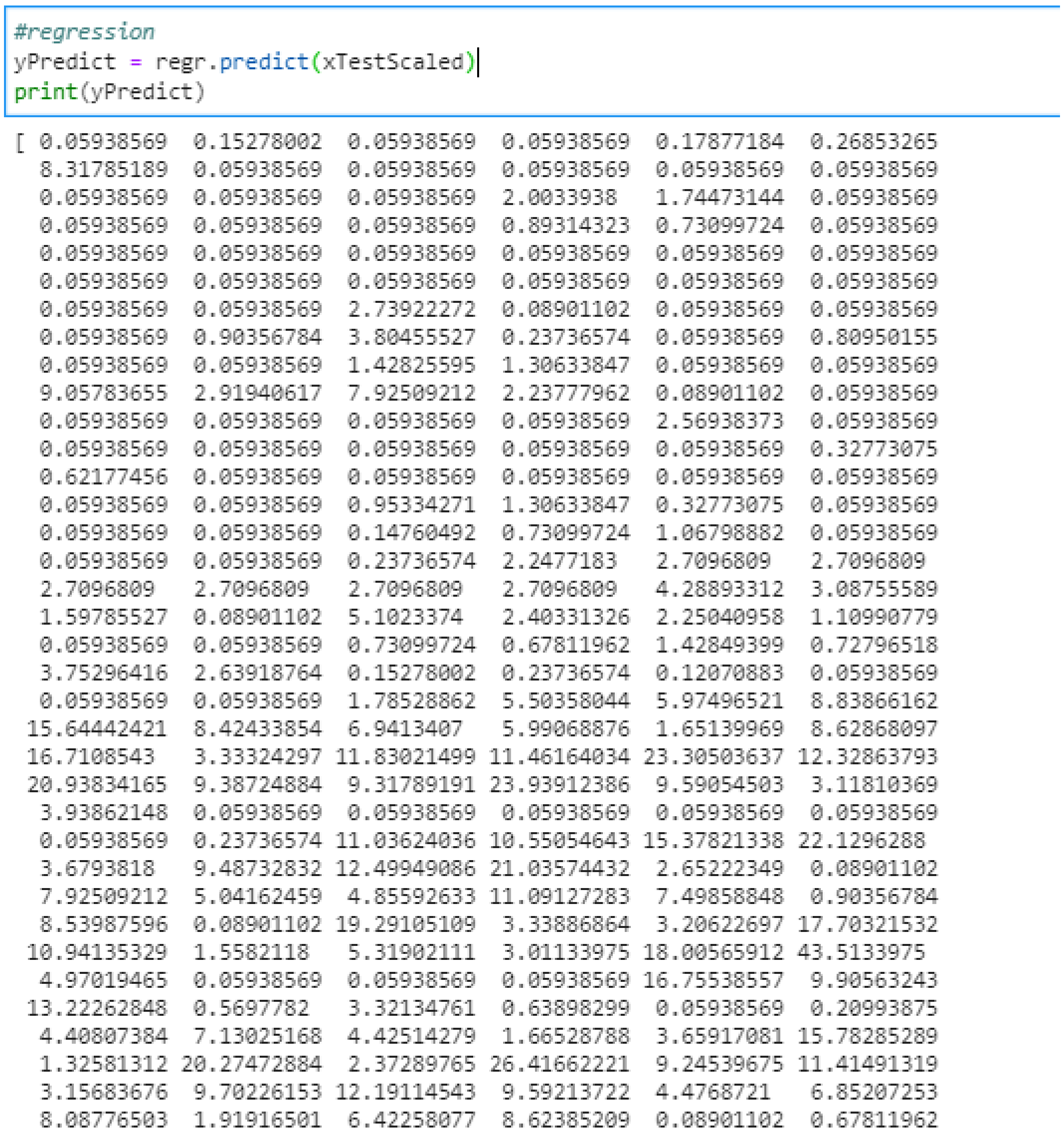


```
print(xTest.describe())
print(xTestScaled[:10])
```

```
             Est1        Est3
count  431.000000  431.000000
mean     2.376798    2.364269
std      4.948763    4.888289
min      0.000000    0.000000
25%      0.000000    0.000000
50%      0.000000    0.000000
75%      2.200000    2.400000
max     36.000000   43.800000
[[-0.53894584 -0.50809201]
 [-0.43236674 -0.50809201]
 [-0.53894584 -0.50809201]
 [-0.53894584 -0.50809201]
 [-0.50341948 -0.47096595]
 [-0.50341948 -0.43383989]
 [ 0.98868793  0.97695037]
 [-0.53894584 -0.50809201]
 [-0.53894584 -0.50809201]
 [-0.53894584 -0.50809201]]
```

Based on the previous training, we can predict the “Est2” values based on the “xTestScaled” array.

```
#regression
yPredict = regr.predict(xTestScaled)
print(yPredict)
```

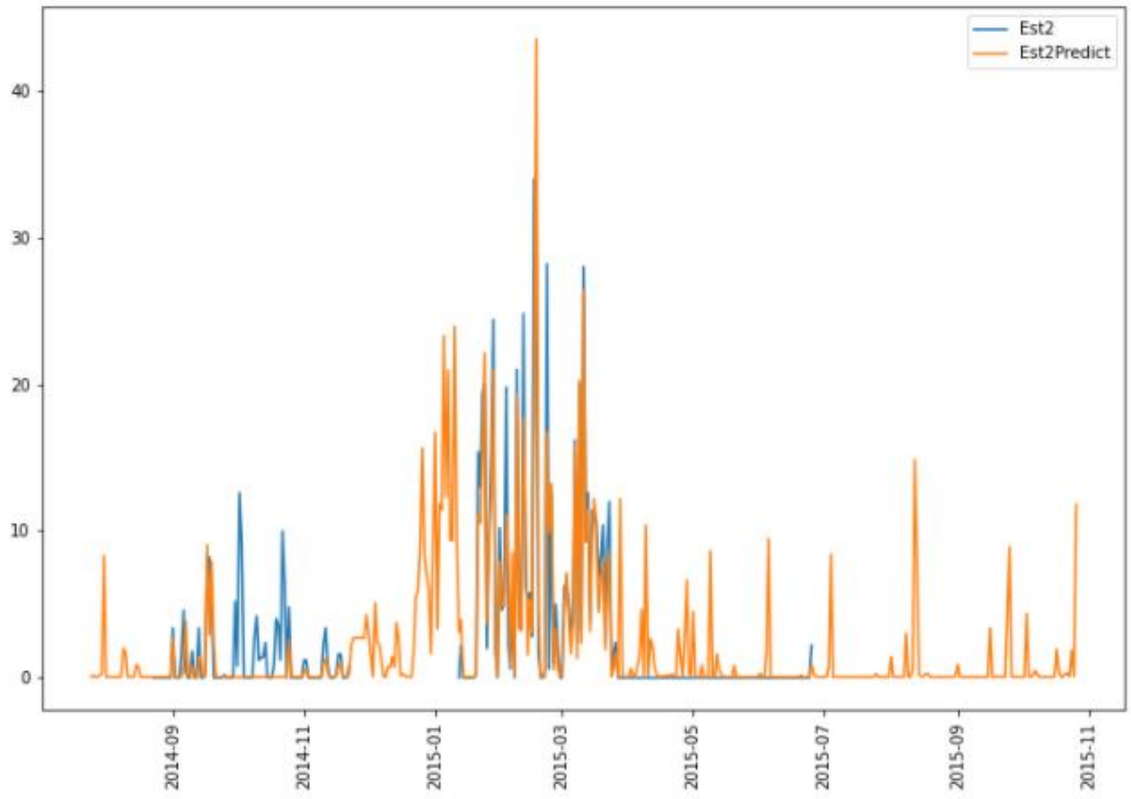


```
#regression
yPredict = regr.predict(xTestScaled)
print(yPredict)
```

```
[ 0.05938569  0.15278002  0.05938569  0.05938569  0.17877184  0.26853265
  8.31785189  0.05938569  0.05938569  0.05938569  0.05938569  0.05938569
  0.05938569  0.05938569  0.05938569  2.0033938   1.74473144  0.05938569
  0.05938569  0.05938569  0.05938569  0.89314323  0.73099724  0.05938569
  0.05938569  0.05938569  0.05938569  0.05938569  0.05938569  0.05938569
  0.05938569  0.05938569  0.05938569  0.05938569  0.05938569  0.05938569
  0.05938569  0.05938569  2.73922272  0.08901102  0.05938569  0.05938569
  0.05938569  0.90356784  3.80455527  0.23736574  0.05938569  0.80950155
  0.05938569  0.05938569  1.42825595  1.30633847  0.05938569  0.05938569
  9.05783655  2.91940617  7.92509212  2.23777962  0.08901102  0.05938569
  0.05938569  0.05938569  0.05938569  0.05938569  2.56938373  0.05938569
  0.05938569  0.05938569  0.05938569  0.05938569  0.05938569  0.32773075
  0.62177456  0.05938569  0.05938569  0.05938569  0.05938569  0.05938569
  0.05938569  0.05938569  0.95334271  1.30633847  0.32773075  0.05938569
  0.05938569  0.05938569  0.14760492  0.73099724  1.06798882  0.05938569
  0.05938569  0.05938569  0.23736574  2.2477183   2.7096809   2.7096809
  2.7096809   2.7096809   2.7096809   2.7096809   4.28893312  3.08755589
  1.59785527  0.08901102  5.1023374   2.40331326  2.25040958  1.10990779
  0.05938569  0.05938569  0.73099724  0.67811962  1.42849399  0.72796518
  3.75296416  2.63918764  0.15278002  0.23736574  0.12070883  0.05938569
  0.05938569  0.05938569  1.78528862  5.50358044  5.97496521  8.83866162
 15.64442421  8.42433854  6.9413407   5.99068876  1.65139969  8.62868097
 16.7108543   3.33324297 11.83021499 11.46164034 23.30503637 12.32863793
 20.93834165  9.38724884  9.31789191 23.93912386  9.59054503  3.11810369
  3.93862148  0.05938569  0.05938569  0.05938569  0.05938569  0.05938569
  0.05938569  0.23736574 11.03624036 10.55054643 15.37821338 22.1296288
  3.6793818   9.48732832 12.49949086 21.03574432  2.65222349  0.08901102
  7.92509212  5.04162459  4.85592633 11.09127283  7.49858848  0.90356784
  8.53987596  0.08901102 19.29105109  3.33886864  3.20622697 17.70321532
 10.94135329  1.5582118   5.31902111  3.01133975 18.00565912 43.5133975
  4.97019465  0.05938569  0.05938569  0.05938569 16.75538557  9.90563243
 13.22262848  0.5697782   3.32134761  0.63898299  0.05938569  0.20993875
  4.40807384  7.13025168  4.42514279  1.66528788  3.65917081 15.78285289
  1.32581312 20.27472884  2.37289765 26.41662221  9.24539675 11.41491319
  3.15683676  9.70226153 12.19114543  9.59213722  4.4768721   6.85207253
  8.08776503  1.91916501  6.42258077  8.62385209  0.08901102  0.67811962
```

You can compare the predicted “Est2” with the initial conditions.

```
#comparison of station 2
fig, ax=plt.subplots(figsize=(12,8),sharex=True,sharey=True)
ax.plot(precipStations.index,precipStations['Est2'],label='Est2')
ax.plot(xTest.index,yPredict,label='Est2Predict')
plt.legend()
plt.xticks(rotation='vertical')
plt.show()
```



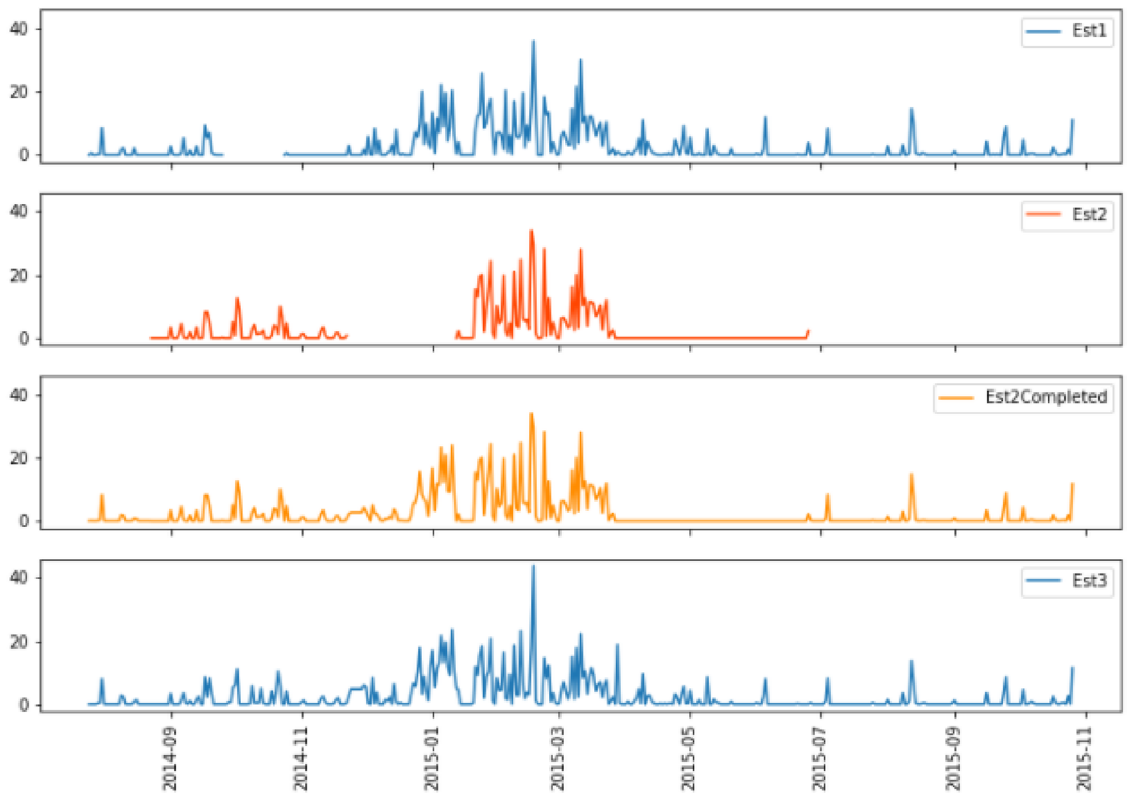
Iterate over the new values created and check if the values are "nan" to replace them with new calculations.

```
precipStations['Est2Completed'] = 0
for index, row in precipStations.iterrows():
    if np.isnan(row['Est2']) and ~np.isnan(row['Est1']) and
~np.isnan(row['Est3']):
        rowScaled = scaler.transform([[row['Est1'],row['Est3']]])
        precipStations.loc[index,['Est2Completed']] =
regr.predict(rowScaled)
    elif ~np.isnan(row['Est2']):
        precipStations.loc[index,['Est2Completed']] = row['Est2']
    else:
        row['Est2Completed'] = np.nan
precipStations['Est2Completed'] = 0
```

Finally, to plot the results and compare:

```
fig, axs=plt.subplots(4,1,figsize=(12,8),sharex=True,sharey=True)
axs[0].plot(precipStations.index,precipStations['Est1'],label='Est1')
```

```
axs[0].legend()
axs[1].plot(precipStations.index,precipStations['Est2'],label='Est2',c
olor='orangered')
axs[1].legend()
axs[2].plot(precipStations.index,precipStations['Est2Completed'],label
='Est2Completed',color='darkorange')
axs[2].legend()
axs[3].plot(precipStations.index,precipStations['Est3'],label='Est3')
axs[3].legend()
plt.xticks(rotation='vertical')
plt.show()
```



Reference:

- https://scienceforecastoa.com/Articles/SJEES-V3-E1-1036.pdf