Attack detection in

Water Distribution System

Part of

Safeguarding Supervisory Control and Data Acquisition(SCADA)

By

Mr.Sukkarin Ruensukont

G6538138

Introduction

SCADA, which stands for Supervisory Control and Data Acquisition, is a system used in various industries to monitor and control processes. It typically involves a combination of hardware and software components that collect and analyze real-time data. SCADA systems are crucial in managing and controlling complex industrial processes such as manufacturing, energy distribution, and water treatment.

In the context of a water distribution system, SCADA plays a pivotal role in ensuring the efficient and reliable operation of the system. It allows operators to remotely monitor and control various components such as pumps, valves, tanks, and sensors. By providing real-time insights and control capabilities, SCADA enables water utilities to optimize their operations, respond to emergencies, and ensure the delivery of safe and reliable water to consumers.

The war between Ukraine and Israel that occurred in 2023 witnessed SCADA attacks with military objectives. Therefore, knowledge and understanding of the system, attack methods, and defense mechanisms are crucial. The Singaporean government, in collaboration with iTrust - the Center for Research in Cyber Security at Singapore University, has been organizing the Continuous Learning Program on Critical Information Infrastructure Security (CISS) since 2016. The program aims to study, comprehend attacks, and devise protection measures. The systems under consideration include water systems, electrical systems, and gas systems. More information on CISS at https://itrust.sutd.edu.sg/ciss-2023/#ed

Awareness of the importance of understanding SCADA attacks and defenses has continuously increased. In 2024, the first International Conference on the Design of Cyber-Secure Water Plants, DCS-Water'24, will be held on April 23-24, 2024, at The Water Tower, Buford, Georgia, USA. For more details, more information at https://itrust.sutd.edu.sg/first-international-conference-on-the-design-of-cyber-secure-water-plants-dcs-water24/programme-dcs-water24/.

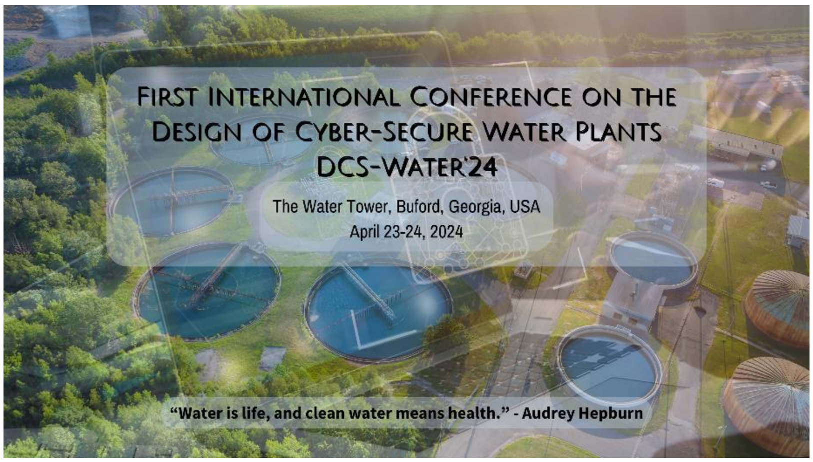


Fig1. First International Conference on the Design of Cyber-Secure Water Plants

"Water is life, and clean water means health." – Audrey Hepburn

Problem and Objective

Objective:

Detect 2 classes, attack and no attack in Water distribution system

Dataset:

Individual need to request for dataset. iTRUST will provide the link to shared google drive, include labeled dataset. iTRUST hold Central Infrastructure Security Showdown (CISS) which is sponsored by the Cyber Security Agency of Singapore and co-organised with the Ministry of Defence, Singapore. CISS competition began in 2016. They cover Secure Water Distribution, Electric power, gas distribution. We focus on Secure water distribution.

Main Dataset:

Water Distribution 2019(WADI2019)

Use for machine learning and attack detection purpose.

https://drive.google.com/drive/folders/12mqpuejSSjq2Wa_0muVjcoQuLN0H6vZt

Additional Dataset:

Battle of the Attack Detection Algorithms(BATADAL2017)

Use to add more understanding of important between features

https://drive.google.com/drive/folders/12-nEv4WaPlgj6SYb-vc7tBFrwxNSWjaV

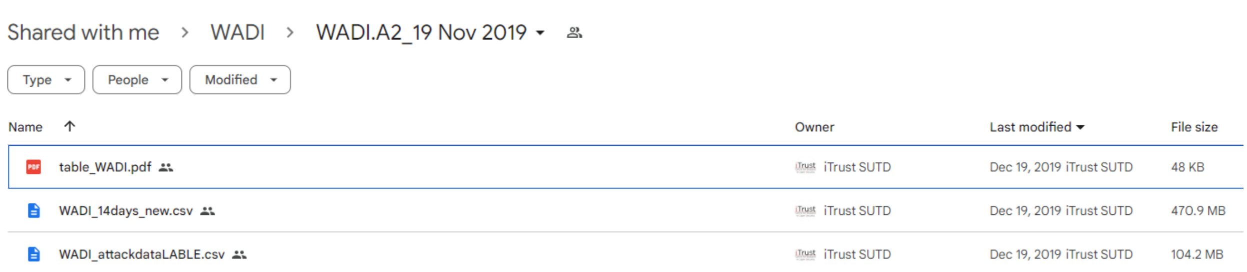


Fig2. WADI2019

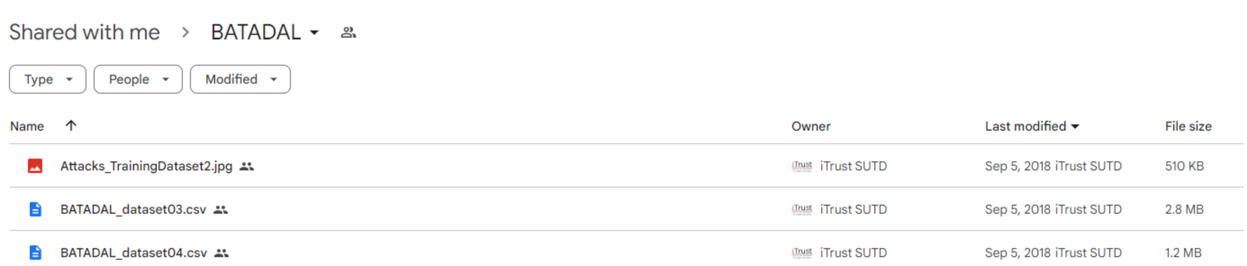


Fig3. BATADAL2017

Dataset details

1. BATtle of Attack Detection Algorithms (BATADAL2016)

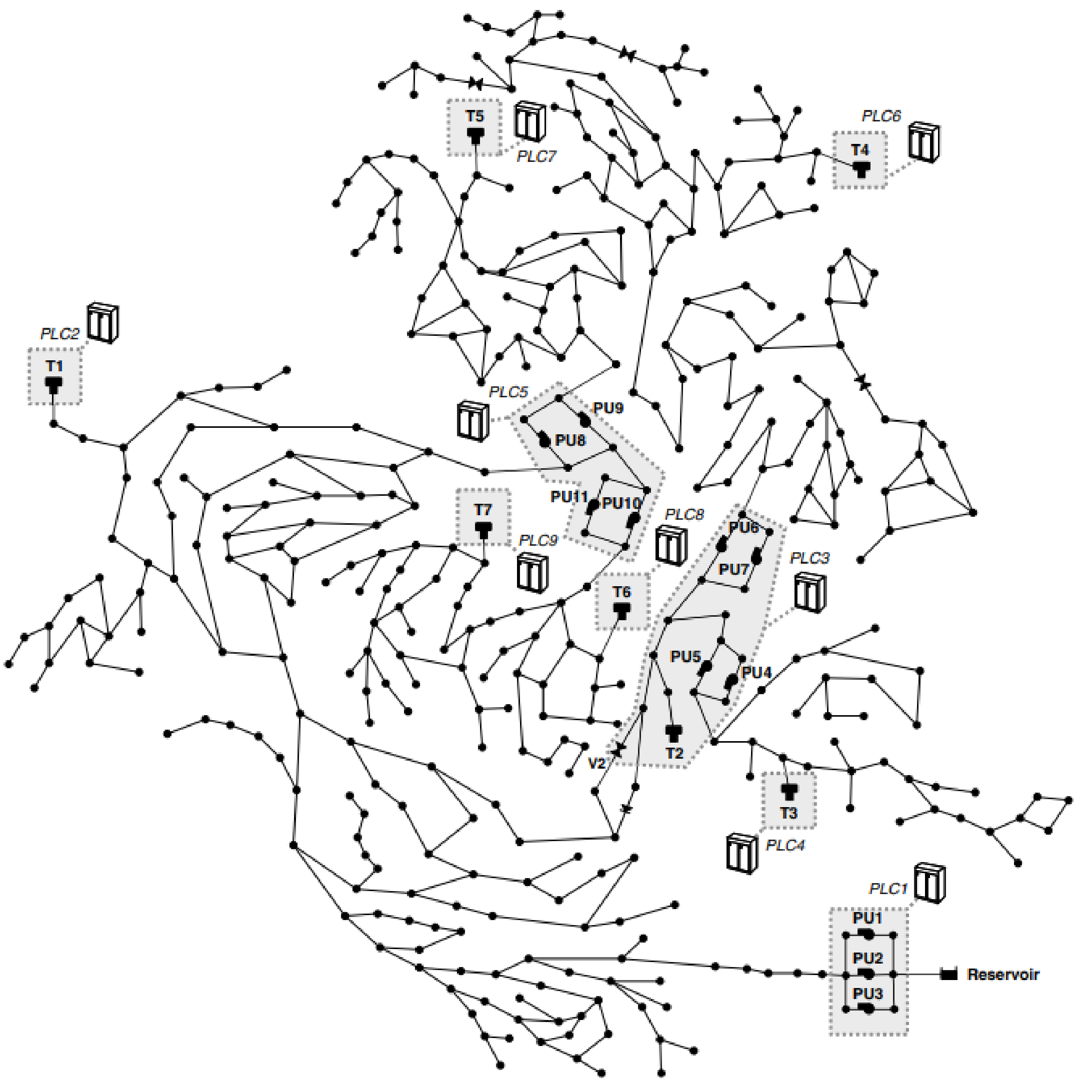


Fig. 1. C-Town water distribution system. (Adapted from Taormina et al. 2017.)

Fig4. Image of water distribution system

Water distribution system consists of

- Pipes to deliver water from pump to destinations
- Pumps to pump water
- Tank to store water, working with PLC in order to know when to turn on the pump
- PLC to control pump, to measure water level in the tank.

Value of components can be transformed to features. There’re 2 files, BATADAL_dataset03.csv all benign, BATADAL_dataset04.csv mix benign and attacks. We will use BATADAL_dataset04.csv in this paper because it contains both benign and malicious.

Features of BATADAL

- 7 Tank water levels, denoted L_<tank_id>
- 12 Pressure for actuated valve, denoted P_<junction id>
- 12 flows for actuated valve, denoted F_<actuator id>
- 12 statuses for actuated valve, denoted S_<actuator id>
- Total 43 features

	A	B	C	D	E	F	G	H	I	J	K	L
1	DATETIME	L_T1	L_T2	L_T3	L_T4	L_T5	L_T6	L_T7	F_PU1	S_PU1	F_PU2	S_PU2
2	04/07/16 00	2.44	5.24	3.19	4.1	2.86	5.5	4.39	93.63	1	93.65	1
3	04/07/16 01	2.66	4.53	3.2	4.18	3.29	5.44	4.53	89.41	1	89.43	1
4	04/07/16 02	3.11	3.66	3.66	4.21	3.87	5.15	3.22	89.88	1	89.89	1
5	04/07/16 03	3.62	3.04	4.17	4.04	3.56	4.98	2.4	88.1	1	88.12	1
6	04/07/16 04	4.08	2.68	4.73	3.2	3.11	5.39	3.46	87.01	1	87.03	1
7	04/07/16 05	4.53	2.1	5.26	3.29	2.76	5.5	4.67	110.09	1	0	0
8	04/07/16 06	4.84	1.57	5.12	3.9	2.29	5.3	3.9	107.71	1	0	0
9	04/07/16 07	5.19	1.07	4.85	4.45	1.87	5	2.66	110.23	1	0	0

Fig5. Head of dataset BATADAL03.csv

	0	1	2	3	4	5	6	7	8	9	129	130
0	Row	Date	Time	1_AIT_001_PV	1_AIT_002_PV	1_AIT_003_PV	1_AIT_004_PV	1_AIT_005_PV	1_FIT_001_PV	1_LS_001_AL	_FLOW	Attack LABLE (1:No Attack, -1:Attack)
1	1	10/9/17	00:00.0	164.21	0.529486	11.9972	482.48	0.331167	0.00127323	0	0.39	1
2	2	10/9/17	00:01.0	164.21	0.529486	11.9972	482.48	0.331167	0.00127323	0	0.39	1
3	3	10/9/17	00:02.0	164.21	0.529486	11.9972	482.48	0.331167	0.00127323	0	0.39	1
4	4	10/9/17	00:03.0	164.21	0.529486	11.9972	482.48	0.331167	0.00127323	0	0.39	1
...	...	...	...	...	...	...	...	...	...	...	...	...
172799	172799.0	10/11/17	59:58.0	172.915	0.583479	11.9211	466.051	0.318317	0.00126	0.0	0.0	1
172800	172800.0	10/11/17	59:59.0	172.915	0.583479	11.9211	466.051	0.318317	0.00126	0.0	0.0	1
172801	172801.0	10/11/17	00:00.0	172.915	0.583479	11.9211	466.051	0.318317	0.00126	0.0	0.0	1
172802	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1
172803	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1

172804 rows × 131 columns

Fig6 WADI2019

2. Water Distribution (WADI2019)
   In CISS 2020, dataset get more complex to make it closer to real Water Distribution system.
   Dataset contain 130 features and 172,801 row.
   Dataset contain 2 classes. Attack = 162824, No Attack =9977

   All features are collected from the system that was built within iTrust lab. The data has no details of each feature. The data has already labeled which are ready to use in Machine learning to find relation between features and classes.

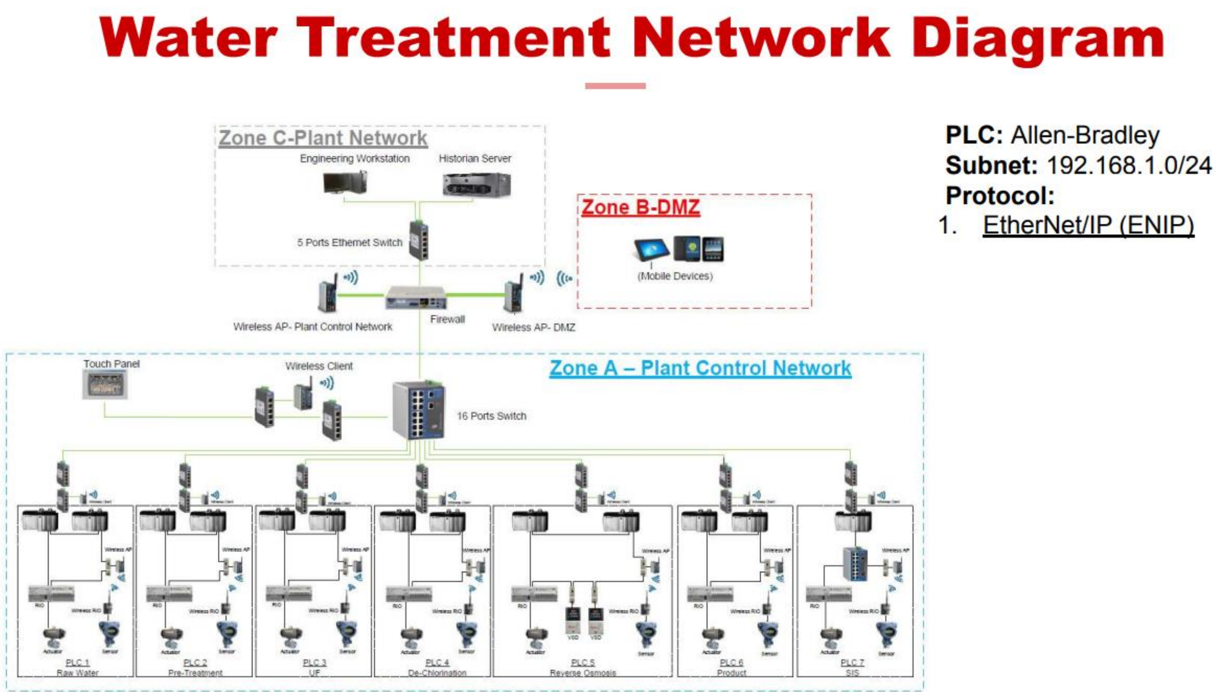


Fig7. WaDi2020 Network Diagram

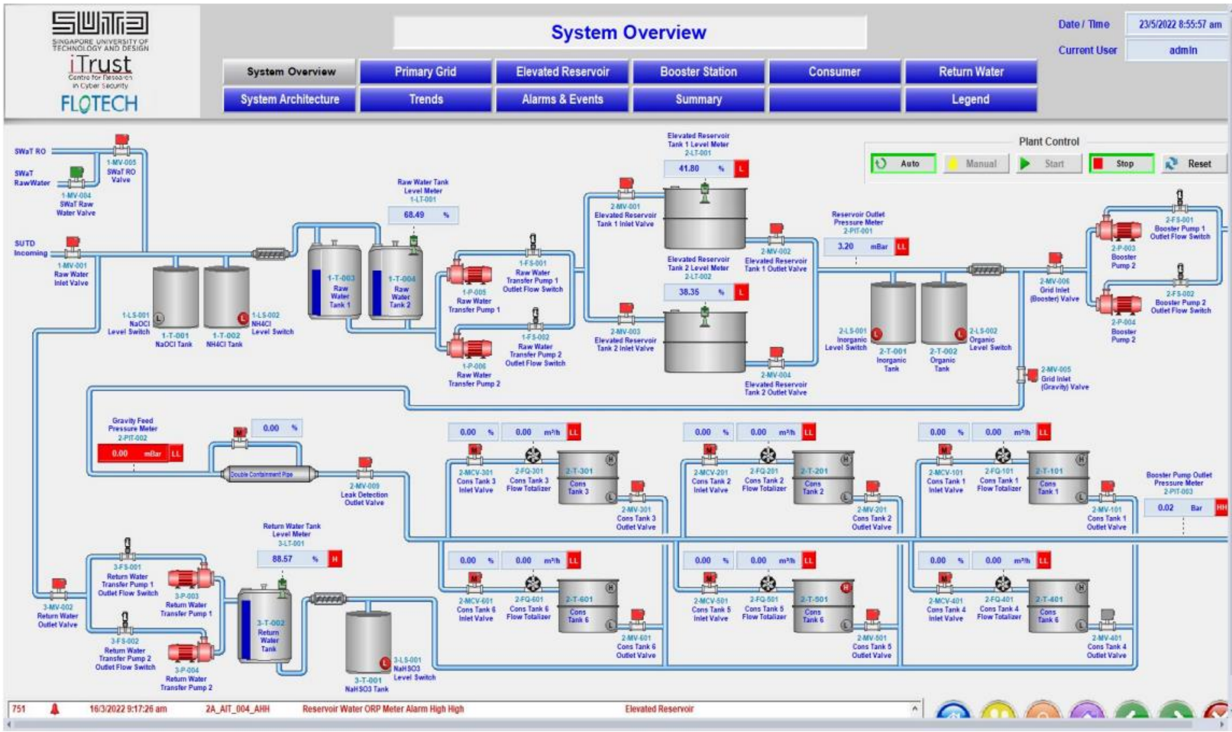


Fig8. WaDi2020 Water distribution diagram

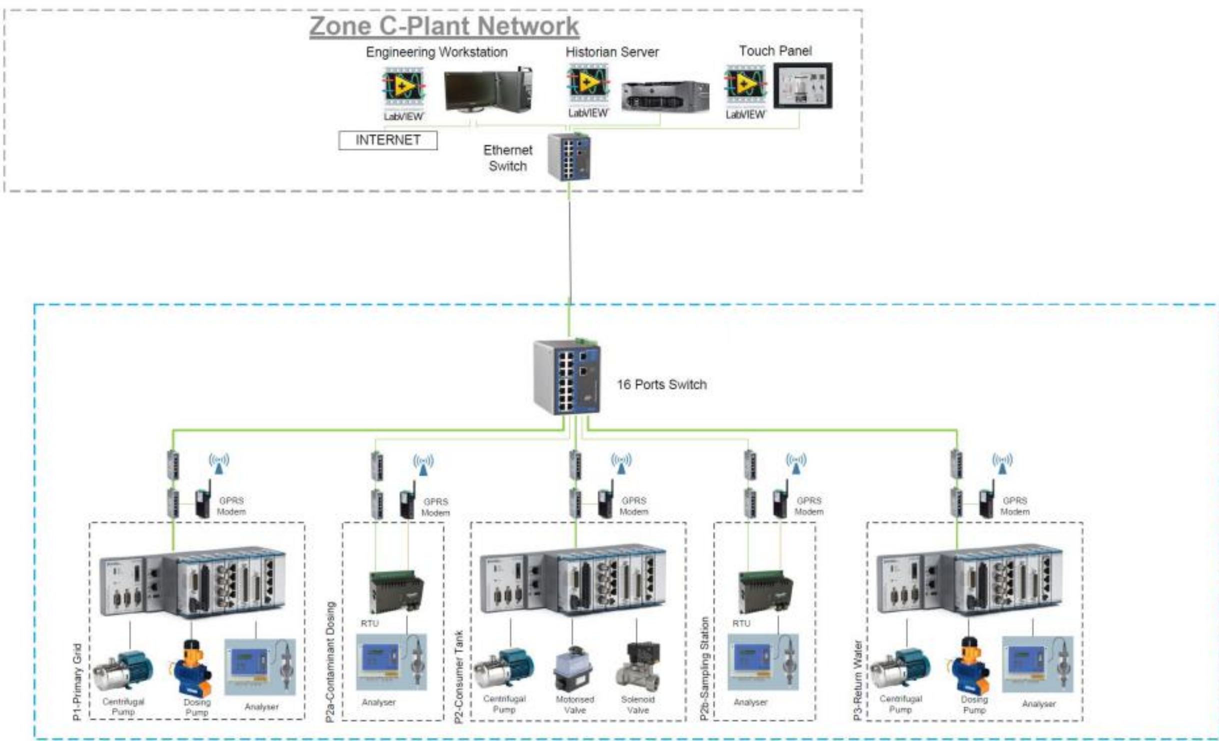


Fig9. Wadi2020 Control diagram

Example of Attack on WADI2019

Attack Identifier	Starting Time	Ending Time	Duration (minutes)	Attack description
1	9/10/17 19:25:00	9/10/17 19:50:16	25.16	Motorized valve 1_MV_001 is maliciously turned on, this causes an overflow on primary tank **should reflect on 1LT001 and 1FIT001**
2	10/10/17 10:24:10	10/10/17 10:34:00	9.50	Flow Indication Transmitter 1_FIT_001 is tuned off, a false reading is seen by PLC for 1_FIT_001. This will turn chemical dosing pump on while leaving the water level in primary tank constant. Consequently the attacker is increasing the level of chemicals inside water.
3-4	10/10/17 10:55:00	10/10/17 11:24:00	29.0	Stealthy attack. Attacker aims to drain elevated reservoir 2_LT_002. This is done controlling manipulating tank level draining and filling speed. 1_AIT_001 Moreover the attacker changes the reading seen by water quality sensor, this causes the raw water tank drain.

Fig10. Example attack on WADI2019

This project mainly focused on WADI2019. Most methods will be done with WADI2019

BATADAL2017 will be use only to give more understanding on feature, e.g., what kind of feature has high correlation to classes.

Method

Data Preparation:

Visualize data, contains 172084 rows and 131 columns

```
Shape =  (172804, 131)
```

Inspect first 3 columns is Row, Date, Time. These're not correlated to classes, remove them.

Check negative value and create list, check if value in this list columns contain positive value. We will get list of columns contain both positive and negative values, omit column '130' as class

```
The following columns contain negative values: ['88', '90', '115', '130']
The following columns contain both negative and positive values: ['88', '90', '115', '130']
```

Inspect columns contain both positive and negative values.

```
        85  86  87       88   89        90
47125  1.0 NaN NaN  23.5320  2.0  0.001922
47126  1.0 NaN NaN  23.5320  2.0  0.001922
47127  1.0 NaN NaN  23.5091  2.0 -0.002741
47128  1.0 NaN NaN  23.5091  2.0 -0.002741
47129  1.0 NaN NaN  23.5091  2.0 -0.002741
47130  1.0 NaN NaN  23.5091  2.0 -0.002741
47131  1.0 NaN NaN  23.5091  2.0 -0.002741
47132  1.0 NaN NaN  23.5622  2.0  0.001947
47133  1.0 NaN NaN  23.5622  2.0  0.001947
```

Create new column based on value on columns which contains both positive and negative, if value is positive, assign 1 to new column, unless assign 2 to new column. Make the value on column '88', '90', '115' to all positive by multiply all negative with (-1). Inspect results at column '90' and '90-sign'

```
        89        90  90-sign
47125  2.0  0.001922        1
47126  2.0  0.001922        1
47127  2.0  0.002741        2
47128  2.0  0.002741        2
47129  2.0  0.002741        2
47130  2.0  0.002741        2
47131  2.0  0.002741        2
47132  2.0  0.001947        1
47133  2.0  0.001947        1
```

Find empty columns. Inspect them.

```
The following columns have all values empty (except the header): ['50', '51', '86', '87']
         49  50  51   52
0  0.051166 NaN NaN  0.0
1  0.051166 NaN NaN  0.0
2  0.051166 NaN NaN  0.0
3  0.051166 NaN NaN  0.0
4  0.051166 NaN NaN  0.0

    85  86  87        88
0  2.0 NaN NaN  0.016157
1  2.0 NaN NaN  0.016157
2  2.0 NaN NaN  0.016157
3  2.0 NaN NaN  0.016157
4  2.0 NaN NaN  0.016157
```

Drop empty columns. Inspect the result.

```
         49   52   53   54
0  0.051166  0.0  0.0  0.0
1  0.051166  0.0  0.0  0.0
2  0.051166  0.0  0.0  0.0
3  0.051166  0.0  0.0  0.0
4  0.051166  0.0  0.0  0.0

    85        88  88-sign
0  2.0  0.016157      2.0
1  2.0  0.016157      2.0
2  2.0  0.016157      2.0
3  2.0  0.016157      2.0
4  2.0  0.016157      2.0
```

Find single empty value. Inspect row 172801 and 172802 are empty in many columns but not all columns.

```
Column '3' has missing values:
172801   NaN
172802   NaN
Name: 3, dtype: float64


Column '4' has missing values:
172801   NaN
172802   NaN
Name: 4, dtype: float64


Column '5' has missing values:
172801   NaN
172802   NaN
Name: 5, dtype: float64
```

```
Column '127' has missing values:
172801   NaN
172802   NaN
Name: 127, dtype: float64


Column '128' has missing values:
172801   NaN
172802   NaN
Name: 128, dtype: float64


Column '129' has missing values:
172801   NaN
172802   NaN
Name: 129, dtype: float64
```

Remove row 172801 and 172802. Check and found no missing value left.

```
No missing values found in the DataFrame.
```

Inspect Data description before normalize.

	3	4	5	6	7	8	9	10	11	12	...
count	172801.000000	172801.000000	172801.000000	172801.000000	172801.000000	172801.000000	172801.0	172801.0	172801.000000	172801.000000	..
mean	176.210422	0.648910	11.928407	453.784271	0.274574	0.542569	0.0	0.0	55.539636	1.274287	..
std	18.669165	0.351526	0.139214	18.862597	0.037848	0.862086	0.0	0.0	8.706924	0.452633	..
min	0.000000	0.000000	0.000000	0.000000	0.201966	0.000605	0.0	0.0	37.002300	0.000000	..
25%	170.866000	0.589479	11.911300	440.867000	0.241040	0.001102	0.0	0.0	47.829700	1.000000	..
50%	177.234000	0.631472	11.927600	454.977000	0.273966	0.001186	0.0	0.0	55.932900	1.000000	..
75%	179.533000	0.661469	11.952000	468.240000	0.305849	1.872090	0.0	0.0	62.489400	2.000000	..
max	634.492000	6.000000	12.109800	484.871000	0.351282	2.495160	0.0	0.0	75.216100	2.000000	..

8 rows × 127 columns

Normalize data using MinMaxScaler, except last column for class. Inspect data after normalized. Data range inclusive between [0, 1]

	3	4	5	6	7	8	9	10	11	12	...
count	172801.000000	172801.000000	172801.000000	172801.000000	172801.000000	172801.000000	172801.0	172801.0	172801.000000	172801.000000	..
mean	0.277719	0.108152	0.985021	0.935887	0.486273	0.217259	0.0	0.0	0.485095	0.637143	..
std	0.029424	0.058588	0.011496	0.038902	0.253473	0.345587	0.0	0.0	0.227848	0.226316	..
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	0.0	0.000000	0.000000	..
25%	0.269296	0.098246	0.983608	0.909246	0.261687	0.000199	0.0	0.0	0.283337	0.500000	..
50%	0.279332	0.105245	0.984954	0.938346	0.482199	0.000233	0.0	0.0	0.495386	0.500000	..
75%	0.282955	0.110245	0.986969	0.965700	0.695726	0.750228	0.0	0.0	0.666961	1.000000	..
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	0.0	0.0	1.000000	1.000000	..

8 rows × 127 columns

Preprocess:

Train_test_split data, given test data = 30%. Specify stratify feature = y=[‘130’]

List all current features.

```
['3', '4', '5', '6', '7', '8', '9', '10', '11', '12', '13', '14', '15', '16', '17', '18', '19', '20', '21', '22', '23', '24', '25', '26', '27',
['3', '4', '5', '6', '7', '8', '9', '10', '11', '12', '13', '14', '15', '16', '17', '18', '19', '20', '21', '22', '23', '24', '25', '26', '27',
```

Rank each feature by scores. Inspect top 10 best and worst features.

```
--------------------------------------------------
Top 10 best features selected by this method are :
66 : 12054.955263772104
16 : 4538.045647704014
18 : 4487.315854347481
8 : 4350.468063608258
37 : 3387.726424933425
89 : 2191.9522378488746
90-sign : 2179.735695068156
69 : 1703.569038191479
34 : 1546.9967666436596
129 : 1538.0722013467896

--------------------------------------------------
Top 10 worst features selected by this method are :
94 : 0.006322622591439416
63 : 0.028701978538930605
5 : 0.03238054284976486
55 : 0.16195704792978696
116 : 0.35304909001758367
35 : 0.39265259199205804
85 : 0.4255473206577616
39 : 0.6160590566327536
49 : 0.6424503252307319
6 : 0.895688901082101
```

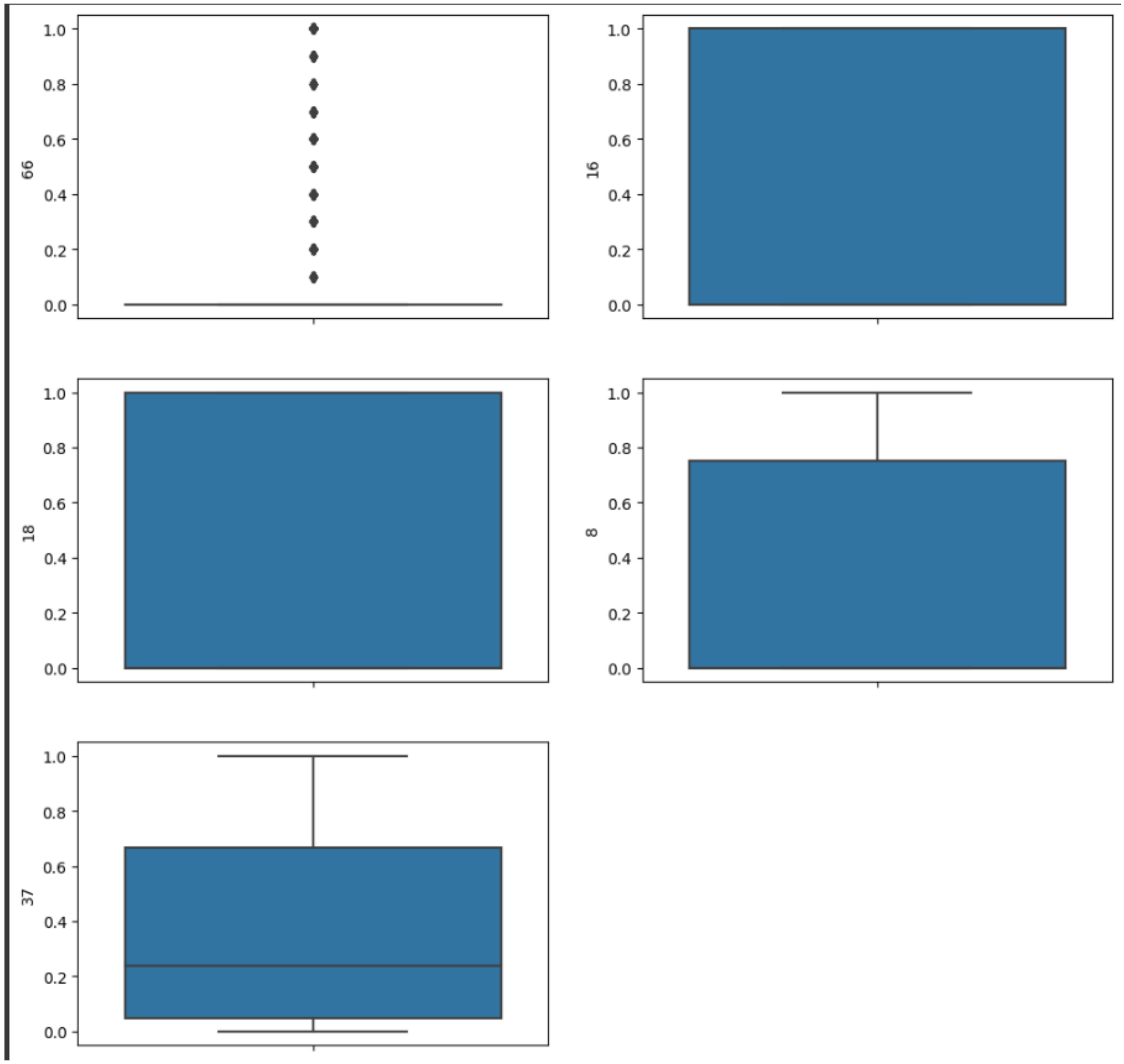
Select only features that score higher than 100. Inspect features count = 49

```
Feature score more than 100, count = 49
['66', '16', '18', '8', '37', '89', '90-sign', '69', '34', '129', '20', '64', '88', '72', '41', '43', '21', '70', '11', '12', '93', '23', '88-si
```

Inspect type of features, float64.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 172801 entries, 0 to 172800
Data columns (total 50 columns):
 #   Column   Non-Null Count   Dtype
---  ------   --------------   -----
 0   66       172801 non-null  float64
 1   16       172801 non-null  float64
 2   18       172801 non-null  float64
 3   8        172801 non-null  float64
 4   37       172801 non-null  float64
 5   89       172801 non-null  float64
 6   90-sign  172801 non-null  float64
 7   69       172801 non-null  float64
 8   34       172801 non-null  float64
 9   129      172801 non-null  float64
 10  20       172801 non-null  float64
 11  64       172801 non-null  float64
 12  88       172801 non-null  float64
 13  72       172801 non-null  float64
 14  41       172801 non-null  float64
 15  43       172801 non-null  float64
 16  21       172801 non-null  float64
 17  70       172801 non-null  float64
 18  11       172801 non-null  float64
 19  12       172801 non-null  float64
 20  93       172801 non-null  float64
 21  23       172801 non-null  float64
 22  88-sign  172801 non-null  float64
 23  90       172801 non-null  float64
 24  97       172801 non-null  float64
```

Boxplot on top 5 features, ‘66’, ‘16’, ‘18’, ‘8’, ‘37’. These features are highly correlated.



Inspect number of attack class = 9977, number of no attack class = 162824. Downsampling number of No Attack class from 162824 to 9977.

```
No Attack class, count =  162824
Attack class,    count =  9977
New total row          =  9977 * 2 = 19954
```

Inspect description after down sampling to make balance class.

	66	16	18	8	37	89	90-sign	69	34	129	...	78	31	71	32	119	
count	19954.000000	19954.000000	19954.000000	19954.000000	19954.000000	19954.000000	19954.000000	19954.000000	19954.000000	19954.000000	...	19954.000000	19954.000000	19954.000000	19954.000000	19954.000000	19954.00
mean	0.054285	0.471785	0.470482	0.383091	0.497435	0.248071	0.159467	0.250370	0.452602	0.341339	...	0.625739	0.352342	0.184584	0.542774	0.212132	0.15
std	0.206885	0.499216	0.499140	0.384585	0.364434	0.431904	0.366120	0.301613	0.343617	0.278215	...	0.226874	0.271429	0.240744	0.408076	0.230422	0.23
min	0.000000	0.000000	0.000000	0.000098	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.008524	0.003520	0.00
25%	0.000000	0.000000	0.000000	0.000219	0.071429	0.000000	0.000000	0.000000	0.097561	0.133047	...	0.500000	0.073171	0.000000	0.142649	0.070816	0.00
50%	0.000000	0.000000	0.000000	0.096444	0.666667	0.000000	0.000000	0.178334	0.390244	0.296137	...	0.500000	0.365854	0.156002	0.339241	0.088087	0.00
75%	0.000000	1.000000	1.000000	0.765028	0.809524	0.000000	0.000000	0.275087	0.731707	0.510730	...	1.000000	0.560976	0.271092	1.000000	0.274820	0.23
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	...	1.000000	1.000000	1.000000	1.000000	0.999942	1.00

8 rows × 50 columns

Learning Techniques:

Train models with train dataset. Using 3 machine learning models, Decision tree, K-Nearest Neighbor, Random Forest Classifier. Save model to model_<model name>.csv

Evaluate using test dataset. Using K-fold cross validation, K=10. Specify 4 criterias for each model to evaluates, Accuracy, Precision, Recall and F1-Score. Inspect result.

```
####### Features: count = 49 #######
['66', '16', '18', '8', '37', '89', '90-sign', '69',

=== [ DT ] Cross validation performance ===
  Accuracy : 98.998 +-(0.716)
  Precision: 99.613 +-(0.467)
  Recall   : 98.594 +-(1.028)
  F1-Score : 99.063 +-(0.646)

=== [ KNN ] Cross validation performance ===
  Accuracy : 98.680 +-(0.609)
  Precision: 99.077 +-(0.745)
  Recall   : 98.292 +-(1.003)
  F1-Score : 98.679 +-(0.589)

=== [ RF ] Cross validation performance ===
  Accuracy : 89.276 +-(1.241)
  Precision: 90.082 +-(1.536)
  Recall   : 88.349 +-(1.766)
  F1-Score : 89.186 +-(0.951)
```

With full features selection(only score higher than 100). All 3 models yield very good result. DT yields highest Accuracy, Precision, Recall and F1-Score. DT F1-Score = 99.063 +-(0.646)

In this project, we need to detect attack class. We need to focus on recall which will be discussed later.

Post-process:

Try 5 Ensemble modes. Inspect results compare to single ML model.
Ensemble1: Majority Votes
Ensemble2: Voting with adjusted weight, DT=2, KNN=1, RF=1
Ensemble3: Voting with adjusted weight, DT=2, KNN=2, RF=1
Ensemble4: Voting with adjusted weight, DT=4, KNN=3, RF=2
Ensemble5: Voting with adjusted weight, DT=2, KNN=1, RF=3

Inspect Accuracy, Precision, Recall and F1-Score

```
=== [Ensemble1: Majority Vote] Cross validation performance ===
  Accuracy : 99.048 +-(0.528)
  Precision: 99.379 +-(0.648)
  Recall   : 98.536 +-(0.937)
  F1-Score : 99.022 +-(0.539)


=== [Ensemble2: Weight211] Cross validation performance ===
  Accuracy : 98.931 +-(0.630)
  Precision: 99.757 +-(0.268)
  Recall   : 98.205 +-(1.134)
  F1-Score : 98.973 +-(0.625)

=== [Ensemble3: Weight221] Cross validation performance ===
  Accuracy : 98.948 +-(0.593)
  Precision: 99.413 +-(0.635)
  Recall   : 98.638 +-(0.980)
  F1-Score : 99.055 +-(0.585)

=== [Ensemble: Weight432] Cross validation performance ===
  Accuracy : 98.964 +-(0.568)
  Precision: 99.379 +-(0.648)
  Recall   : 98.601 +-(0.990)
  F1-Score : 99.007 +-(0.447)

=== [Ensemble5: Weight113] Cross validation performance ===
  Accuracy : 89.276 +-(1.241)
  Precision: 90.082 +-(1.536)
  Recall   : 88.349 +-(1.766)
  F1-Score : 89.186 +-(0.951)
```

Single Decision Tree model get higher score in every criteria compare to 5 Ensemble models. Ensemble4 with given weight DT=4, KNN=3, RF=2 get highest score among 5 ensemble models and the score is very close to Decision Tree model.

Experimental Results

Dataset size

Dataset contains 130 columns(features)
Dataset contains 172801 row
Attack = 162,802 rows, No Attack = 9977 rows
Data type is float64

Experiment detail(train_test_split, how to meature, measurement name selected)

Split train:test to 70:30.
Measure 3 models(DT, KNN, RF) using Accuracy. Precision, Recall.
We will focus on recall because this project aims to detect attack class.

Result

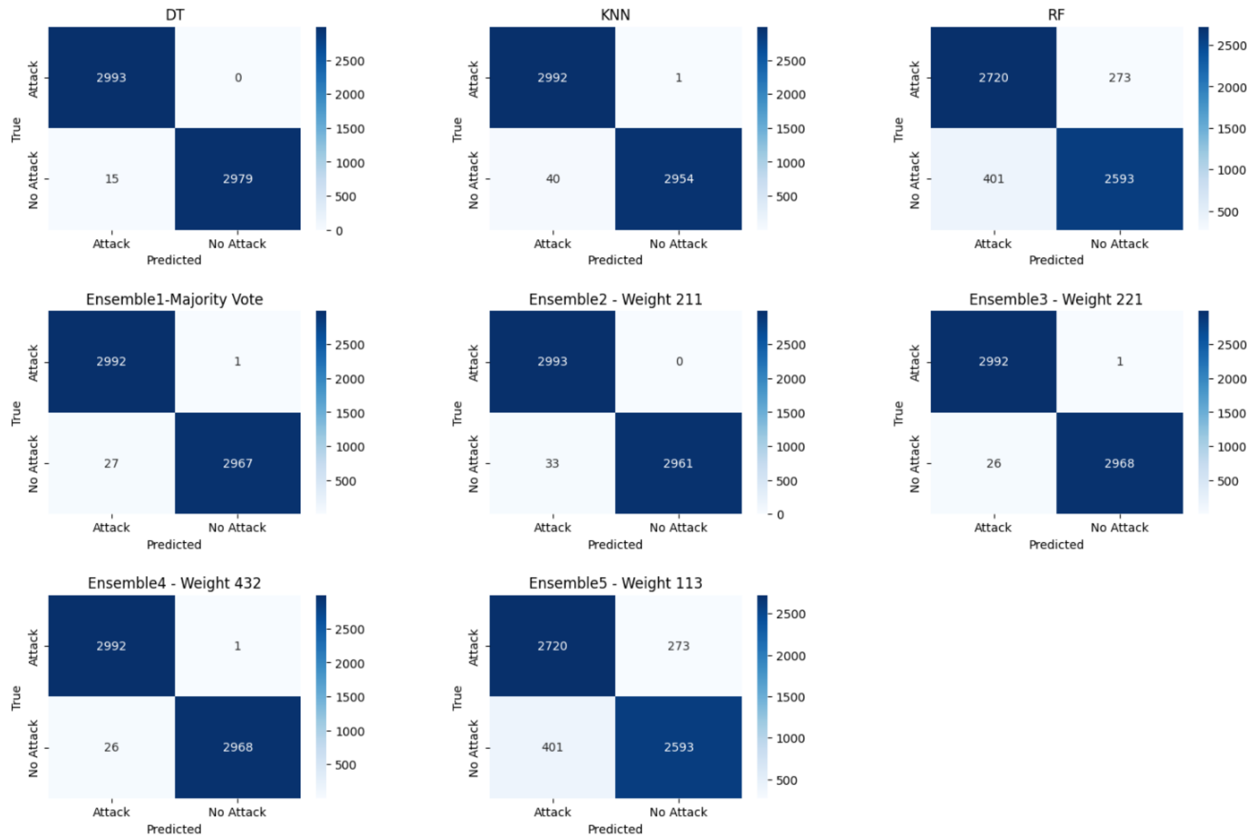
Decision tree yields highest on every categories(Accuracy, Precision, Recall, F1-score) over K-Nearest Neighbor and Random Forest

Decision tree also yields higher scores on every categories when compare to 5 ensemble modes with combination of DT, KNN and RF.

```
####### Features: count = 49 #######
['66', '16', '18', '8', '37', '89', '90-sign', '69',

=== [ DT ] Cross validation performance ===
  Accuracy : 98.998 +-(0.716)
  Precision: 99.613 +-(0.467)
  Recall   : 98.594 +-(1.028)
  F1-Score : 99.063 +-(0.646)

=== [ KNN ] Cross validation performance ===
  Accuracy : 98.680 +-(0.609)
  Precision: 99.077 +-(0.745)
  Recall   : 98.292 +-(1.003)
  F1-Score : 98.679 +-(0.589)

=== [ RF ] Cross validation performance ===
  Accuracy : 89.276 +-(1.241)
  Precision: 90.082 +-(1.536)
  Recall   : 88.349 +-(1.766)
  F1-Score : 89.186 +-(0.951)
```

```
=== [Ensemble1: Majority Vote] Cross validation performance ===
  Accuracy : 99.048 +-(0.528)
  Precision: 99.379 +-(0.648)
  Recall   : 98.536 +-(0.937)
  F1-Score : 99.022 +-(0.539)


=== [Ensemble2: Weight211] Cross validation performance ===
  Accuracy : 98.931 +-(0.630)
  Precision: 99.757 +-(0.268)
  Recall   : 98.205 +-(1.134)
  F1-Score : 98.973 +-(0.625)

=== [Ensemble3: Weight221] Cross validation performance ===
  Accuracy : 98.948 +-(0.593)
  Precision: 99.413 +-(0.635)
  Recall   : 98.638 +-(0.980)
  F1-Score : 99.055 +-(0.585)

=== [Ensemble: Weight432] Cross validation performance ===
  Accuracy : 98.964 +-(0.568)
  Precision: 99.379 +-(0.648)
  Recall   : 98.601 +-(0.990)
  F1-Score : 99.007 +-(0.447)

=== [Ensemble5: Weight113] Cross validation performance ===
  Accuracy : 89.276 +-(1.241)
  Precision: 90.082 +-(1.536)
  Recall   : 88.349 +-(1.766)
  F1-Score : 89.186 +-(0.951)
```

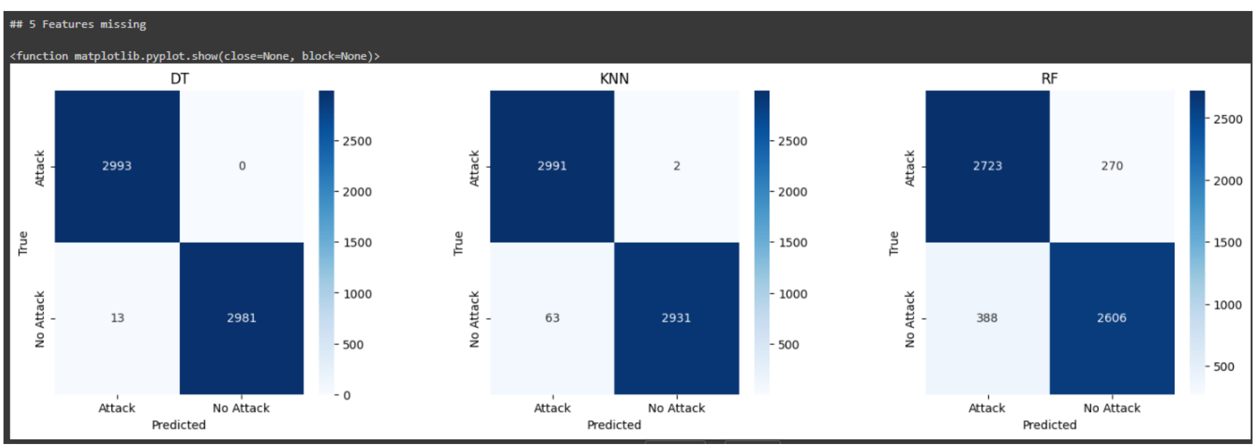
Inspect confusion matrix to compare DT, KNN, RF and 5 Ensemble models. With full features with feature score higher than 100. Random forest show higher false negative than DT and KNN.



In war era, some features might not be available but an army need to know if there's any attack happen within the Water Distribution System. We cut out some features and inspect results.

Take out top 5 features, 44 features left

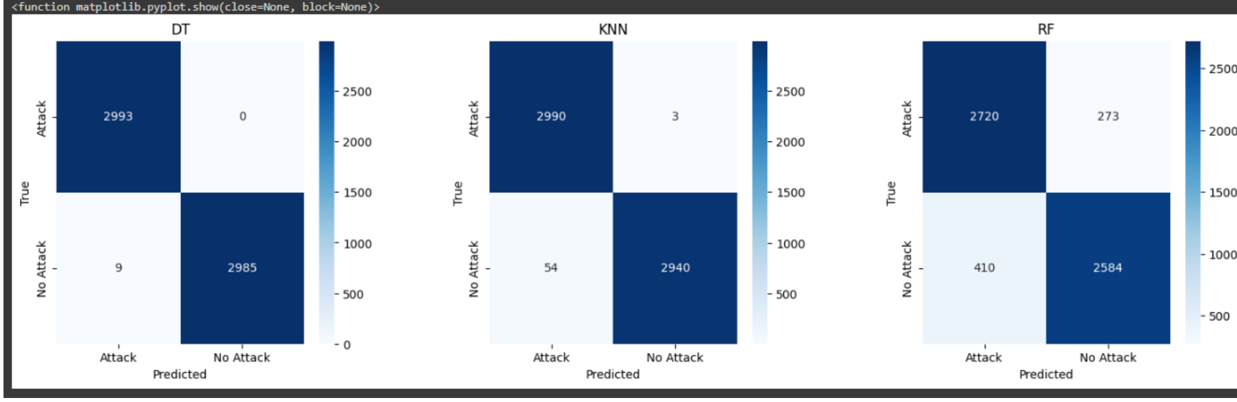
```
####### Features: count = 44 #######
['90-sign', '89', '69', '129', '34', '20', '64', '88', '72', '21', '41', '43', '70',
Index(['90-sign', '89', '69', '129', '34', '20', '64', '88', '72', '21', '41',
       '43', '70', '11', '12', '23', '93', '88-sign', '90', '97', '109', '110',
       '47', '33', '80', '108', '68', '44', '24', '25', '83', '4', '40', '107',
       '28', '78', '71', '31', '67', '119', '32', '127', '117', '104'],
      dtype='object')
=== [ DT ] Cross validation performance ===
  Accuracy : 98.931 +-(0.411)
  Precision: 99.307 +-(0.488)
  Recall   : 98.569 +-(0.367)
  F1-Score : 98.765 +-(0.293)

=== [ KNN ] Cross validation performance ===
  Accuracy : 97.929 +-(0.598)
  Precision: 99.395 +-(0.532)
  Recall   : 96.477 +-(0.858)
  F1-Score : 97.912 +-(0.539)

=== [ RF ] Cross validation performance ===
  Accuracy : 88.157 +-(1.251)
  Precision: 88.975 +-(1.656)
  Recall   : 87.049 +-(1.955)
  F1-Score : 87.990 +-(1.520)
```



Take out top 10 features, 39 features left

```
####### Features: count = 39 #######
['20', '64', '88', '72', '21', '41', '43', '70', '11', '12', '23', '93', '88-sign',
Index(['20', '64', '88', '72', '21', '41', '43', '70', '11', '12', '23', '93',
       '88-sign', '90', '97', '109', '110', '47', '33', '80', '108', '68',
       '44', '24', '25', '83', '4', '40', '107', '28', '78', '71', '31', '67',
       '119', '32', '127', '117', '104'],
      dtype='object')
=== [ DT ] Cross validation performance ===
  Accuracy : 98.881 +-(0.318)
  Precision: 99.228 +-(0.408)
  Recall   : 98.665 +-(0.425)
  F1-Score : 98.956 +-(0.411)

=== [ KNN ] Cross validation performance ===
  Accuracy : 98.380 +-(0.549)
  Precision: 99.498 +-(0.464)
  Recall   : 97.272 +-(0.729)
  F1-Score : 98.371 +-(0.516)

=== [ RF ] Cross validation performance ===
  Accuracy : 88.074 +-(1.320)
  Precision: 89.147 +-(1.712)
  Recall   : 86.642 +-(2.030)
  F1-Score : 87.865 +-(1.591)
```

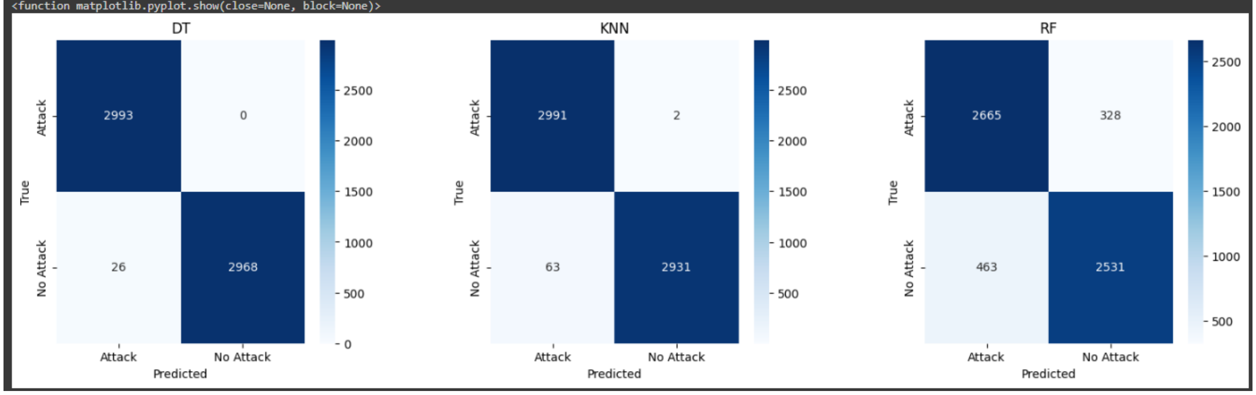
```
## 10 Features missing

<function matplotlib.pyplot.show(close=None, block=None)>
```



Take out top 30 features, 19 features left

```
####### Features: count = 19 #######
['108', '68', '44', '24', '25', '83', '4', '40', '107', '28', '78', '71', '31', '67', '119', '32', '127', '117', '104']
Index(['108', '68', '44', '24', '25', '83', '4', '40', '107', '28', '78', '71',
       '31', '67', '119', '32', '127', '117', '104'],
      dtype='object')
=== [ DT ] Cross validation performance ===
  Accuracy : 98.680 +-(0.493)
  Precision: 99.156 +-(0.430)
  Recall   : 98.328 +-(0.785)
  F1-Score : 98.774 +-(0.404)

=== [ KNN ] Cross validation performance ===
  Accuracy : 98.079 +-(0.320)
  Precision: 99.487 +-(0.392)
  Recall   : 96.648 +-(0.649)
  F1-Score : 98.046 +-(0.355)

=== [ RF ] Cross validation performance ===
  Accuracy : 86.838 +-(1.090)
  Precision: 88.444 +-(1.627)
  Recall   : 84.753 +-(1.734)
  F1-Score : 86.543 +-(1.182)
```

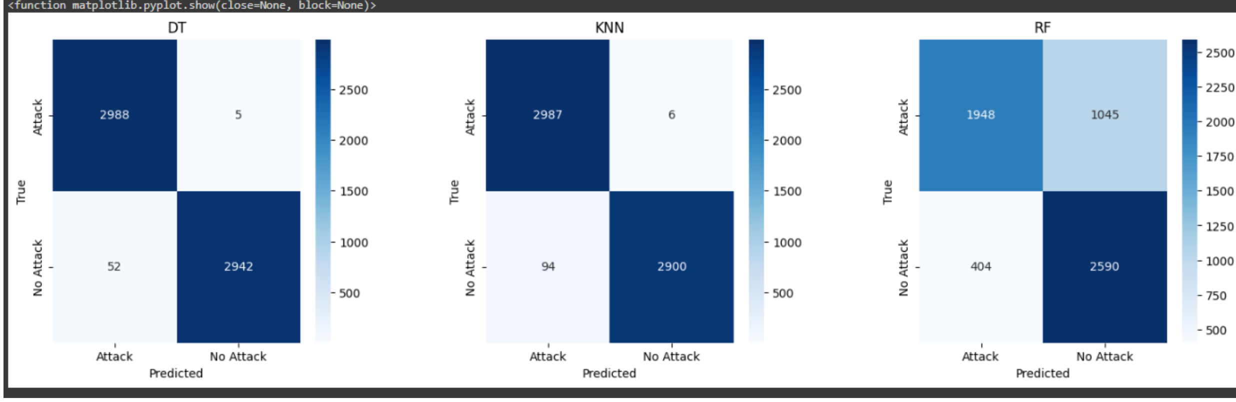
```
## 30 Features missing

<function matplotlib.pyplot.show(close=None, block=None)>
```



Take out top 40 features, 9 features left

```
####### Features: count = 9 #######
['78', '71', '31', '67', '119', '32', '127', '117', '104']
Index(['78', '71', '31', '67', '119', '32', '127', '117', '104'], dtype='object')
=== [ DT ] Cross validation performance ===
  Accuracy : 97.845 +-(0.609)
  Precision: 98.946 +-(0.578)
  Recall   : 96.896 +-(0.728)
  F1-Score : 97.876 +-(0.465)

=== [ KNN ] Cross validation performance ===
  Accuracy : 97.394 +-(0.561)
  Precision: 99.367 +-(0.495)
  Recall   : 95.394 +-(1.205)
  F1-Score : 97.334 +-(0.586)

=== [ RF ] Cross validation performance ===
  Accuracy : 79.071 +-(2.611)
  Precision: 75.712 +-(4.103)
  Recall   : 85.694 +-(1.336)
  F1-Score : 80.343 +-(2.596)
```

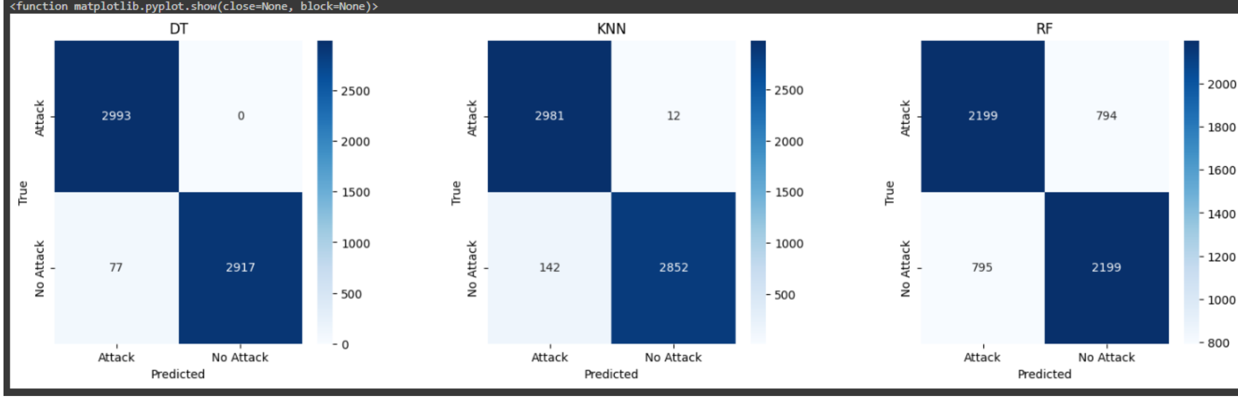
```
## 40 Features missing
<function matplotlib.pyplot.show(close=None, block=None)>
```



Take out top 45 features, 4 features left

```
####### Features: count = 4 #######
['67', '127', '117', '104']
Index(['67', '127', '117', '104'], dtype='object')
=== [ DT ] Cross validation performance ===
  Accuracy : 96.810 +-(0.637)
  Precision: 98.081 +-(0.987)
  Recall   : 95.620 +-(0.975)
  F1-Score : 96.883 +-(0.688)

=== [ KNN ] Cross validation performance ===
  Accuracy : 96.342 +-(0.730)
  Precision: 98.493 +-(0.483)
  Recall   : 94.109 +-(1.577)
  F1-Score : 96.243 +-(0.810)

=== [ RF ] Cross validation performance ===
  Accuracy : 73.276 +-(1.218)
  Precision: 73.131 +-(2.387)
  Recall   : 73.476 +-(2.369)
  F1-Score : 73.278 +-(1.919)
```

```
## 45 Features missing

<function matplotlib.pyplot.show(close=None, block=None)>
```



Analyze and discuss results

During features selection, when create another column beside column which contain both positive and negative value. We found that 90-sign(6th) get higher features score than 90(23rd). And 88(12th) get higher feature score than 88-sign(22nd). This mean for feature 90, sign(positive/negative) is more significant than it's value. But for feature 88, sign(positive/negative) is less significant than it's value.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 172801 entries, 0 to 172800
Data columns (total 50 columns):
 #   Column   Non-Null Count   Dtype  
---  ------   --------------   -----  
 0   66       172801 non-null  float64
 1   16       172801 non-null  float64
 2   18       172801 non-null  float64
 3   8        172801 non-null  float64
 4   37       172801 non-null  float64
 5   89       172801 non-null  float64
 6   90-sign  172801 non-null  float64
 7   69       172801 non-null  float64
 8   34       172801 non-null  float64
 9   129      172801 non-null  float64
 10  20       172801 non-null  float64
 11  64       172801 non-null  float64
 12  88       172801 non-null  float64
 13  72       172801 non-null  float64
 14  41       172801 non-null  float64
 15  43       172801 non-null  float64
 16  21       172801 non-null  float64
 17  70       172801 non-null  float64
 18  11       172801 non-null  float64
 19  12       172801 non-null  float64
 20  93       172801 non-null  float64
 21  23       172801 non-null  float64
 22  88-sign  172801 non-null  float64
 23  90       172801 non-null  float64
 24  97       172801 non-null  float64
```

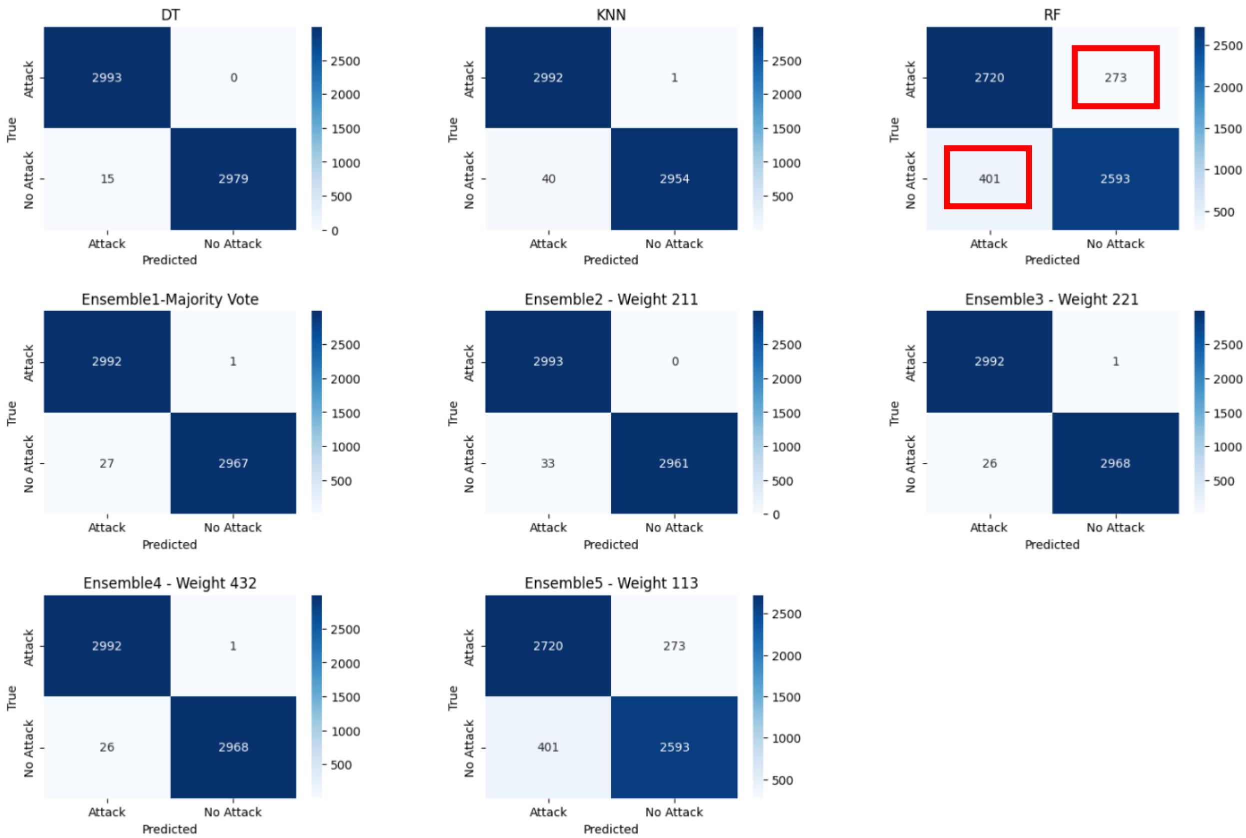
The result show that Decision Tree give the highest accuracy, precision, recall and F1-score and suitable for detect attack in Water Distribution system. KNN also give very high result on all criteria. RF is not good sine there're lots higher in False Positive and False negative.

Model	Accuracy	Precision	Recall	F1-Score
DT	**98.998**	**99.613**	**98.594**	**99.063**
KNN	98.680	99.077	98.292	98.679
RF	89.276	90.082	88.349	89.186

Comparing to Ensemble models Decision Tree yields higher score in ever criterias over 5 Ensemble models. Among 5 ensemble models. The model which given more weight to Decision tree get higher scores.

Model	Accuracy	Precision	Recall	F1-Score
DT	**98.998**	**99.613**	**98.594**	**99.063**
Ensemble1 – AVG	99.048	99.379	98.536	99.022
Ensemble2 – 211	98.931	99.757	98.205	98.973
Ensemble3 – 221	98.948	99.413	98.638	99.055
Ensemble4 – 432	98.964	99.379	96.601	99.007
Ensemble5 – 113	89.276	90.082	88.349	89.186

Inspect Confusion matrix on DT, KNN, RF an 5 Ensemble models



After cutting the most significant features, there’s little impact to accuracy and recall on DT and KNN model. But there’s huge impact on RF. This mean RF is not suitable for Water Distribution Attack detection.

There’re multiple stages of experiments. Saving .csv file for each stage can save time not to start over from the beginning.

Bonus track: Training with BATADAL Dataset

We need to know more on what kind of feature impact the prediction class the most. So we train the same method in BATADAL2017. Since BATADAL explain every features in details while WADI2019 doesn't

Inspect BATADAL

```
1 df_BATADAL
```

	DATETIME	L_T1	L_T2	L_T3	L_T4	L_T5	L_T6	L_T7	F_PU1	S_PU1	...	P_J256	P_J289	P_J415	P_J302	P_J306	P_J307	P_J317	P_J14	P_J422	ATT_FLAG
0	04/07/16 00	2.44	5.24	3.19	4.10	2.86	5.50	4.39	93.63	1.0	...	70.00	28.22	85.87	21.69	82.72	21.58	71.99	39.33	29.64	-999
1	04/07/16 01	2.66	4.53	3.20	4.18	3.29	5.44	4.53	89.41	1.0	...	87.73	24.45	84.87	29.81	86.62	29.81	59.76	42.17	26.15	-999
2	04/07/16 02	3.11	3.66	3.66	4.21	3.87	5.15	3.22	89.88	1.0	...	89.29	23.90	87.11	29.85	87.64	29.85	58.50	42.00	25.56	-999
3	04/07/16 03	3.62	3.04	4.17	4.04	3.56	4.98	2.40	88.10	1.0	...	91.98	27.10	68.75	31.60	64.25	31.47	72.30	43.24	28.38	-999
4	04/07/16 04	4.08	2.68	4.73	3.20	3.11	5.39	3.46	87.01	1.0	...	92.11	26.76	68.74	32.30	64.23	32.17	72.53	44.00	28.04	-999
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4172	24/12/16 20	2.65	2.37	3.85	3.04	3.82	4.94	2.19	120.08	1.0	...	70.03	27.38	84.14	18.45	81.67	18.34	66.04	29.88	28.98	-999
4173	24/12/16 21	2.24	2.56	3.42	2.92	3.69	5.02	1.97	119.12	1.0	...	68.60	27.66	83.46	25.40	60.85	25.28	66.89	30.19	29.29	-999
4174	24/12/16 22	1.91	2.76	2.95	2.49	2.70	5.14	1.87	120.71	1.0	...	85.63	26.84	82.82	24.46	59.56	24.34	66.08	29.68	28.78	-999
4175	24/12/16 23	1.52	2.52	3.33	2.03	1.69	5.10	1.39	120.02	1.0	...	86.15	25.78	103.63	24.77	59.01	24.65	66.42	28.98	28.08	-999
4176	25/12/16 00	1.10	2.12	3.73	2.73	1.58	5.13	1.19	120.33	1.0	...	89.08	25.64	105.91	18.16	80.65	18.06	67.32	28.75	27.85	-999

4177 rows × 45 columns

Before Normalize with MinMaxScaler

	L_T1	L_T2	L_T3	L_T4	L_T5	L_T6	L_T7	F_PU1	S_PU1	F_PU2	...	P_J256	P_J289	P_J415	P_J302	P_J306	P_J307	P_J317
count	4177.000000	4177.000000	4177.000000	4177.000000	4177.000000	4177.000000	4177.000000	4177.000000	4177.0	4177.000000	...	4177.000000	4177.000000	4177.000000	4177.000000	4177.000000	4177.000000	4177.000000
mean	2.707446	3.287965	4.204496	3.556232	2.748377	5.368429	3.323352	100.839069	1.0	69.385904	...	79.404214	27.772894	82.665293	24.574898	74.463383	24.482049	68.024829
std	1.209627	1.470462	0.681352	0.544905	0.735800	0.177560	0.838134	10.217831	0.0	42.180931	...	8.475271	2.740362	7.577549	4.712136	10.020099	4.734871	5.631171
min	0.320000	0.310000	2.880000	1.920000	1.280000	4.770000	0.430000	38.920000	1.0	0.000000	...	66.060000	19.460000	53.940000	13.910000	58.070000	13.640000	51.410000
25%	1.650000	2.150000	3.620000	3.130000	2.130000	5.240000	2.660000	94.170000	1.0	0.000000	...	71.960000	26.670000	83.860000	20.510000	62.930000	20.400000	66.680000
50%	2.820000	3.490000	4.230000	3.550000	2.750000	5.500000	3.280000	96.650000	1.0	93.860000	...	75.810000	28.140000	85.190000	23.160000	81.170000	23.050000	68.890000
75%	3.780000	4.580000	4.800000	4.000000	3.370000	5.500000	4.040000	112.140000	1.0	96.510000	...	88.170000	29.570000	86.380000	28.580000	82.650000	28.490000	71.360000
max	5.730000	5.670000	5.430000	4.700000	4.160000	5.500000	5.000000	122.570000	1.0	99.920000	...	94.870000	34.260000	107.950000	71.110000	95.270000	71.110000	93.730000

8 rows × 44 columns

After Normalized

	L_T1	L_T2	L_T3	L_T4	L_T5	L_T6	L_T7	F_PU1	S_PU1	F_PU2	...	P_J256	P_J289	P_J415	P_J302	P_J306	P_J307	P_J317
count	4177.000000	4177.000000	4177.000000	4177.000000	4177.000000	4177.000000	4177.000000	4177.000000	4177.0	4177.000000	...	4177.000000	4177.000000	4177.000000	4177.000000	4177.000000	4177.000000	4177.000000
mean	0.441302	0.555590	0.519410	0.588573	0.509853	0.819766	0.633119	0.740216	0.0	0.694415	...	0.463180	0.561682	0.531851	0.186449	0.440682	0.188656	0.392600
std	0.223591	0.274340	0.267197	0.196009	0.255486	0.243233	0.183399	0.122150	0.0	0.422147	...	0.294178	0.185160	0.140299	0.082380	0.269358	0.082389	0.133062
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	0.000000	...	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.245841	0.343284	0.290196	0.435252	0.295139	0.643836	0.487965	0.660490	0.0	0.000000	...	0.204790	0.487162	0.553971	0.115385	0.130645	0.117627	0.360822
50%	0.462107	0.593284	0.529412	0.586331	0.510417	1.000000	0.623632	0.690137	0.0	0.939351	...	0.338424	0.586486	0.578597	0.161713	0.620968	0.163738	0.413043
75%	0.639556	0.796642	0.752941	0.748201	0.725694	1.000000	0.789934	0.875314	0.0	0.965873	...	0.767442	0.683108	0.600630	0.256469	0.660753	0.258396	0.471408
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	0.0	1.000000	...	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

8 rows × 44 columns

Inspect Top 10 best feature scores. We can see feature related to Pump Flow and Pump status(F_PU6, S_PU6) get highest feature score.

```
----------------------------------------------------
Top 10 best features selected by this method are :
 F_PU6 : 695.6036391089294
 S_PU6 : 685.6517140148434
 S_PU11 : 313.42573044363075
 F_PU11 : 289.9837332248577
 F_PU7 : 17.728121058156763
 S_PU7 : 16.900190917996746
 L_T1 : 3.5553826455702273
 P_J14 : 2.0296320559887167
 S_PU2 : 1.1024679456696937
 P_J269 : 1.0682360601717962

----------------------------------------------------
Top 10 worst features selected by this method are :
 S_PU8 : 5.033210882264993e-05
 L_T2 : 0.0014803037674457193
 L_T3 : 0.010753601483845862
 P_J289 : 0.01148811139117855
 P_J300 : 0.01189105092839934
 L_T4 : 0.012425482745645273
 P_J422 : 0.012583932773734138
 F_V2 : 0.014477739198065338
 P_J415 : 0.01711877962242293
 L_T6 : 0.019690658313144066

----------------------------------------------------
 F_PU6 : 695.6036391089294
 S_PU6 : 685.6517140148434
 S_PU11 : 313.42573044363075
 F_PU11 : 289.9837332248577
 F_PU7 : 17.728121058156763
 S_PU7 : 16.900190917996746
 L_T1 : 3.5553826455702273
 P_J14 : 2.0296320559887167
 S_PU2 : 1.1024679456696937
 P_J269 : 1.0682360601717962
 P_J302 : 0.9862688963634857
 F_PU10 : 0.9816142720116781
 P_J307 : 0.9515680475375935
 S_PU10 : 0.7487900753704324
 S_V2 : 0.6867220151080162
 F_PU2 : 0.5688006200910656
 F_PU1 : 0.36763031727065476
 P_J317 : 0.30924594977135317
 S_PU4 : 0.2042008667805548
 F_PU4 : 0.15451564752145872
 P_J256 : 0.10149088076846922
 L_T7 : 0.08649795207526348
 P_J280 : 0.05922350930457521
 P_J306 : 0.035343920999454194
 L_T5 : 0.03391405625797637
 F_PU8 : 0.019703529502143328
 L_T6 : 0.019690658313144066
 P_J415 : 0.01711877962242293
 F_V2 : 0.014477739198065338
 P_J422 : 0.012583932773734138
```

Conclusion

In conclusion, the feature selection process revealed interesting insights into the significance of sign values and their impact on certain features. Notably, for feature 90, the sign (positive/negative) proved more significant than its numerical value, while for feature 88, the numerical value was more crucial than its sign.

The results of the model evaluation demonstrated that the Decision Tree exhibited the highest accuracy, precision, recall, and F1-score, making it a suitable choice for detecting attacks in Water Distribution systems. KNN also performed exceptionally well across all criteria. However, Random Forest did not fare as well due to a higher occurrence of False Positives and False Negatives.

Comparing individual models to ensemble models, the Decision Tree consistently outperformed the ensembles, particularly when more weight was assigned to the Decision Tree within the ensemble. Moreover, after pruning the most significant features, the impact on accuracy and recall was minimal for Decision Tree and KNN but substantial for Random Forest. This suggests that Random Forest may not be well-suited for Water Distribution Attack detection.

Throughout multiple stages of experiments, it became evident that saving .csv files at each stage could significantly save time by eliminating the need to start over from the beginning.

As a bonus track, training the models with the BATADAL Dataset provided further insights into feature importance. The analysis revealed that features related to Pump Flow and Pump Status, such as F_PU6 and S_PU6, received the highest feature scores. This additional experiment with BATADAL helped to better understand which features had the most impact on the prediction class, especially considering the detailed feature explanations provided by the dataset.

References

1. Paper: Battle of the Attack Detection Algorithms: Disclosing Cyber Attacks on Water Distribution Networks
   https://par.nsf.gov/servlets/purl/10104860
2. Dataset information:
   https://itrust.sutd.edu.sg/itrust-labs_datasets/dataset_info/
3. BATADAL2016 Dataset:
   4 Files already attached
4. WaDi2020 Dataset:
   https://drive.google.com/drive/folders/1c28Vfq4NF66Nchu8tQC4vsMQngK5xVTy
5. Secure Water Distribution, explained:
   https://youtu.be/8rk4hJvePFo