rahul-project-1

August 5, 2024

1 NAME:- RAHUL ASHOK PATIL

2 PROJECT:-

SOLVING CLASSIFICATION PREDICTION FOR "MACHINE FAILURE PREDICTION USING SENSOR DATA" DATASET USING LOGISTIC REGRESSION, NAIVES BAYES, CLASSIFICATION ,SUPPORT VECTOR CLASSIFIER, K NEAREST NEIGHBOUR, DECISION TREE CLASSIFIER.

3 ABOUT PROJECT:-

THIS DATASET CONTAINS SENSOR DATA COLLECTED FROM VARIOUS MACHINES, WITH THE AIM OF PREDICTING MACHINE FAILURES IN ADVANCE. IT INCLUDES A VARIETY OF SENSOR READINGS AS WELL AS THE RECORDED MACHINE FAILURES.

4 DATA:-

```
FOOTFALL: The number of people or objects passing by the machine.
TEMPMODE: The temperature mode or setting of the machine.
AQ: Air quality index near the machine.
USS: Ultrasonic sensor data, indicating proximity measurements.
CS: Current sensor readings, indicating the electrical current usage of the machine.
VOC: Volatile organic compounds level detected near the machine.
RP: Rotational position or RPM (revolutions per minute) of the machine parts.
IP: Input pressure to the machine.
TEMPERATURE: The operating temperature of the machine.
FAIL: Binary indicator of machine failure (1 for failure, 0 for no failure).
```

5 APPROACH:-

1.LOAD THE REQUIRED LIBRARIES SUCH AS PANDAS , MATPLOTLIB, SEABORN , NUMPY, ALONG WITH THE GIVEN DATASET.

2.PERFORM EDA ON THE GIVEN DATASET.

3.IMPORT 'LOGISTIC REGRESSION , NAIVES BAYES, CLASSIFICATION ,SUPPORT VECTOR CLASSIFIER, K NEAREST NEIGHBOUR, DECISION TREE CLASSIFIER'.AND SPLIT THE GIVEN DATASET INTO TRAINING AND TESTING DATA USING

TRAIN_TEST_SPLIT.THEN CALCULATE ACCURACY SCORE USING SKLEARN LIBRARY BY IMPORTING METRICS.

4.ONCE WE GET ACCURACY SCORE OF ALL MODELS FOR BOTH TRAINING AND TESTING DATA, CREATE A DATAFRAME AND LOAD ALL THE ACCURACY OF ALL MODEL.

5.VISUALIZATION: ONCE THE DATASET IS CREATED PLOT THE ACCURACY OF ALL THE MODELS USING BARPLOT

[130]:
```
import pandas as pd
import matplotlib.pyplot as plt                #LOADING ALL THE REQURIED␣
 ↪LIBRARIES.
import seaborn as sns
import numpy as np
```

[131]:
```
D=pd.read_csv(r"C:\Users\RAHUL PATIL\Downloads\data.csv")  #LOADING THE GIVEN␣
 ↪DATASET
D
```

[131]:
```
     footfall  tempMode  AQ  USS  CS  VOC  RP  IP  Temperature  fail
0           0         7   7    1   6    6  36   3            1     1
1         190         1   3    3   5    1  20   4            1     0
2          31         7   2    2   6    1  24   6            1     0
3          83         4   3    4   5    1  28   6            1     0
4         640         7   5    6   4    0  68   6            1     0
..        …         …  ..  …  ..  …  ..  ..          …   …
939         0         7   7    1   6    4  73   6           24     1
940         0         7   5    2   6    6  50   6           24     1
941         0         3   6    2   7    5  43   6           24     1
942         0         6   6    2   5    6  46   7           24     1
943        18         7   4    2   6    3  61   7           24     1

[944 rows x 10 columns]
```

[132]:
```
D.isna().sum() #CHECKING NULL VALUES
```

[132]:
```
footfall       0
tempMode       0
AQ             0
USS            0
CS             0
VOC            0
RP             0
IP             0
Temperature    0
fail           0
dtype: int64
```

[133]:
```
D.info() #SHOWS ALL INFORMATION REGARDING THE DATA SUCH AS NULL VALUE,COLUMNS
↪,DATATYPES
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 944 entries, 0 to 943
Data columns (total 10 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   footfall     944 non-null    int64
 1   tempMode     944 non-null    int64
 2   AQ           944 non-null    int64
 3   USS          944 non-null    int64
 4   CS           944 non-null    int64
 5   VOC          944 non-null    int64
 6   RP           944 non-null    int64
 7   IP           944 non-null    int64
 8   Temperature  944 non-null    int64
 9   fail         944 non-null    int64
dtypes: int64(10)
memory usage: 73.9 KB
```

[134]:
```
D.describe() #SHOWS THE ALL DETAILS REGARDING  ALL NUMERICAL COLUMNS
```

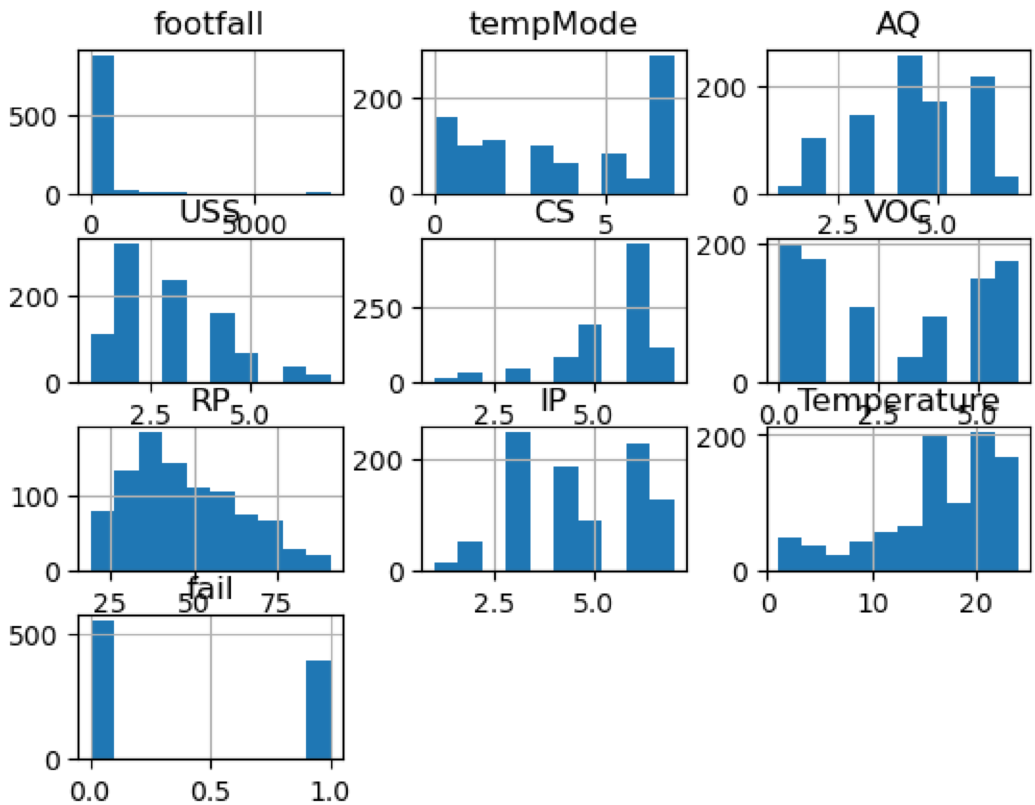
[134]:
```
          footfall    tempMode          AQ         USS          CS  \
count   944.000000  944.000000  944.000000  944.000000  944.000000
mean    306.381356    3.727754    4.325212    2.939619    5.394068
std    1082.606745    2.677235    1.438436    1.383725    1.269349
min       0.000000    0.000000    1.000000    1.000000    1.000000
25%       1.000000    1.000000    3.000000    2.000000    5.000000
50%      22.000000    3.000000    4.000000    3.000000    6.000000
75%     110.000000    7.000000    6.000000    4.000000    6.000000
max    7300.000000    7.000000    7.000000    7.000000    7.000000

              VOC          RP          IP  Temperature        fail
count  944.000000  944.000000  944.000000   944.000000  944.000000
mean     2.842161   47.043432    4.565678    16.331568    0.416314
std      2.273337   16.423130    1.599287     5.974781    0.493208
min      0.000000   19.000000    1.000000     1.000000    0.000000
25%      1.000000   34.000000    3.000000    14.000000    0.000000
50%      2.000000   44.000000    4.000000    17.000000    0.000000
75%      5.000000   58.000000    6.000000    21.000000    1.000000
max      6.000000   91.000000    7.000000    24.000000    1.000000
```

[135]:
```
D.shape #shows no. of rows and columns
```

[135]: (944, 10)

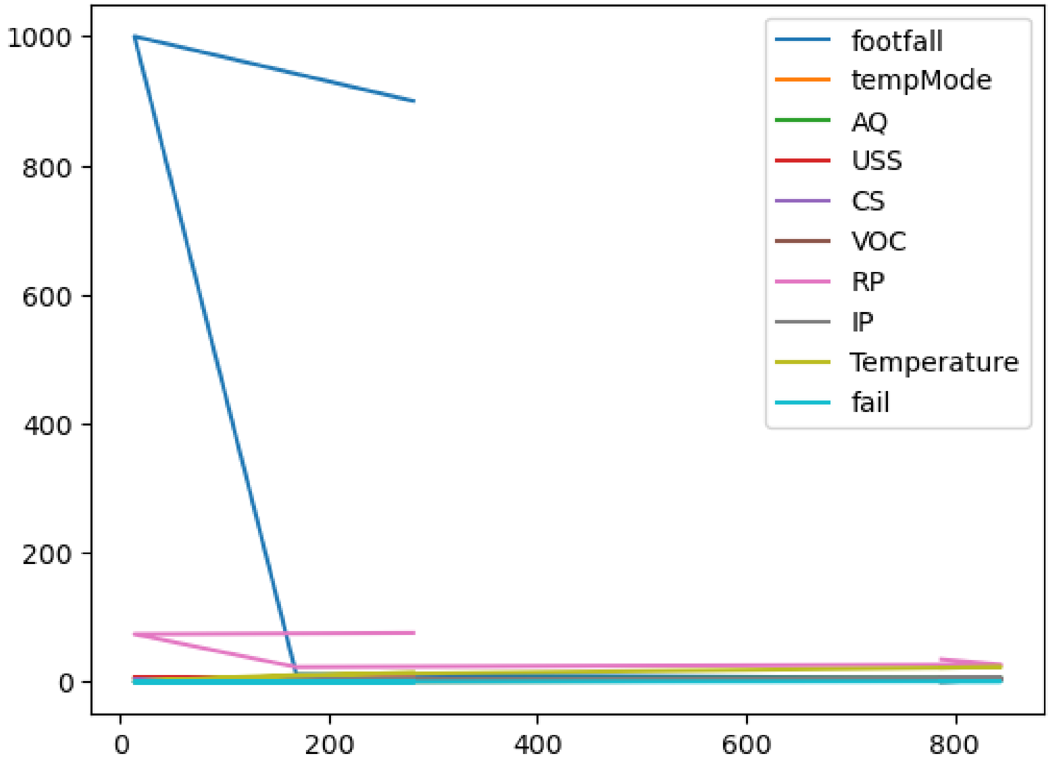
[136]:
```
plt.figure(figsize=(20,15)) #PLOT HISTPLOT TO SEE DATA DISTRIBUTION
D.hist()
plt.show()
```

```
<Figure size 2000x1500 with 0 Axes>
```



[137]:
```
D.sample(5).plot()  #PLOT SAMPLE DATA
```

[137]:
```
<Axes: >
```



[138]:
```
D.corr()*100 #SHOWS CORRELATION
```

[138]:

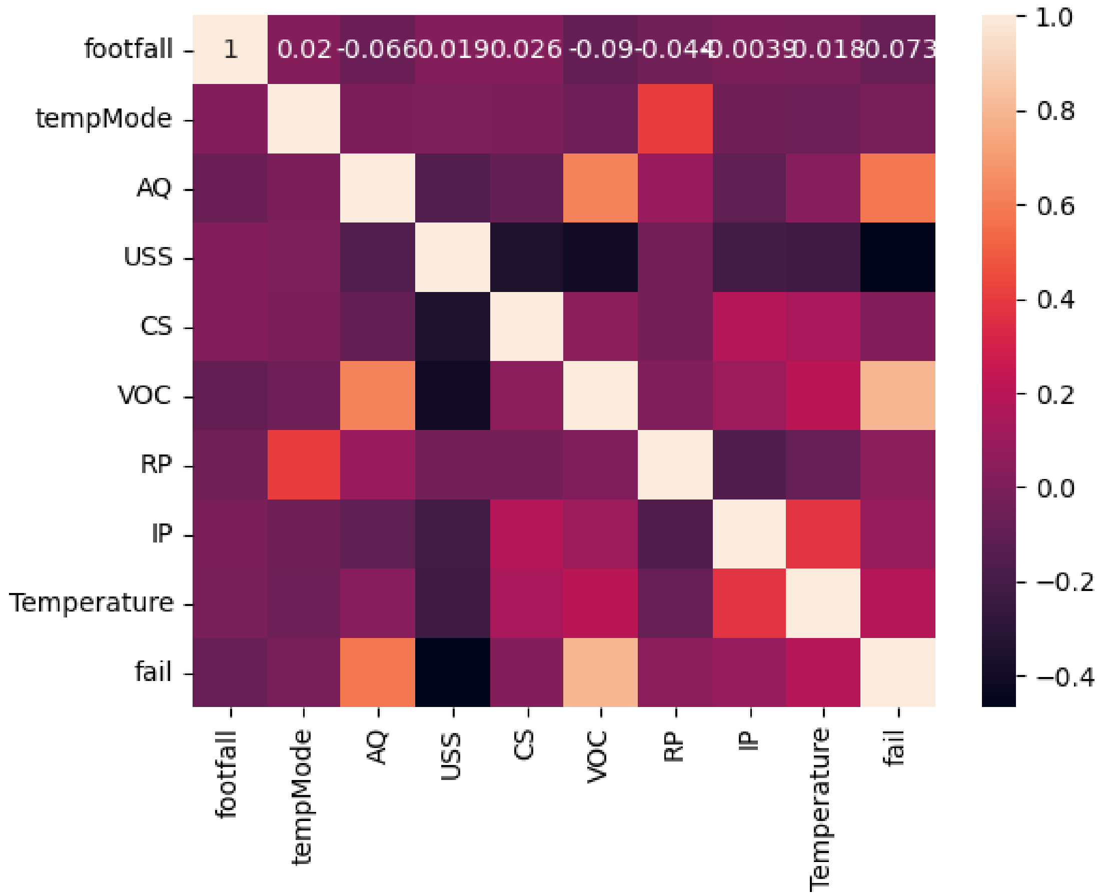
	footfall	tempMode	AQ	USS	CS
footfall	100.000000	2.045710	-6.581633	1.945272	2.563835
tempMode	2.045710	100.000000	-1.085510	0.214175	-1.395619
AQ	-6.581633	-1.085510	100.000000	-15.688392	-9.000961
USS	1.945272	0.214175	-15.688392	100.000000	-35.291496
CS	2.563835	-1.395619	-9.000961	-35.291496	100.000000
VOC	-8.959027	-5.236919	61.856955	-39.947697	4.803661
RP	-4.371965	40.878426	9.465632	-3.254931	-2.696842
IP	-0.386942	-5.810881	-10.586751	-20.641620	18.573905
Temperature	-1.800898	-6.256824	3.432784	-22.512226	14.397186
fail	-7.306605	-1.446182	58.323765	-46.657375	1.885493

\

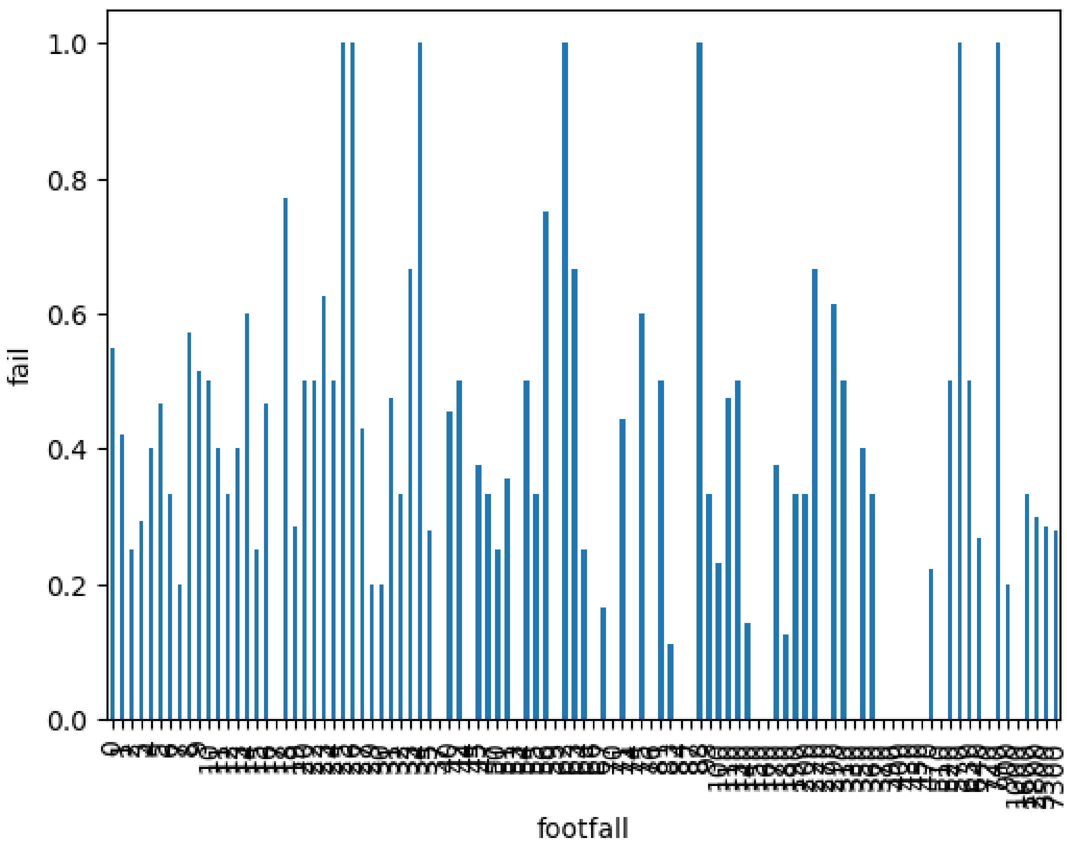
	VOC	RP	IP	Temperature	fail
footfall	-8.959027	-4.371965	-0.386942	-1.800898	-7.306605
tempMode	-5.236919	40.878426	-5.810881	-6.256824	-1.446182
AQ	61.856955	9.465632	-10.586751	3.432784	58.323765
USS	-39.947697	-3.254931	-20.641620	-22.512226	-46.657375
CS	4.803661	-2.696842	18.573905	14.397186	1.885493
VOC	100.000000	0.802311	10.362780	20.895564	79.732915
RP	0.802311	100.000000	-15.884066	-7.849861	5.366771

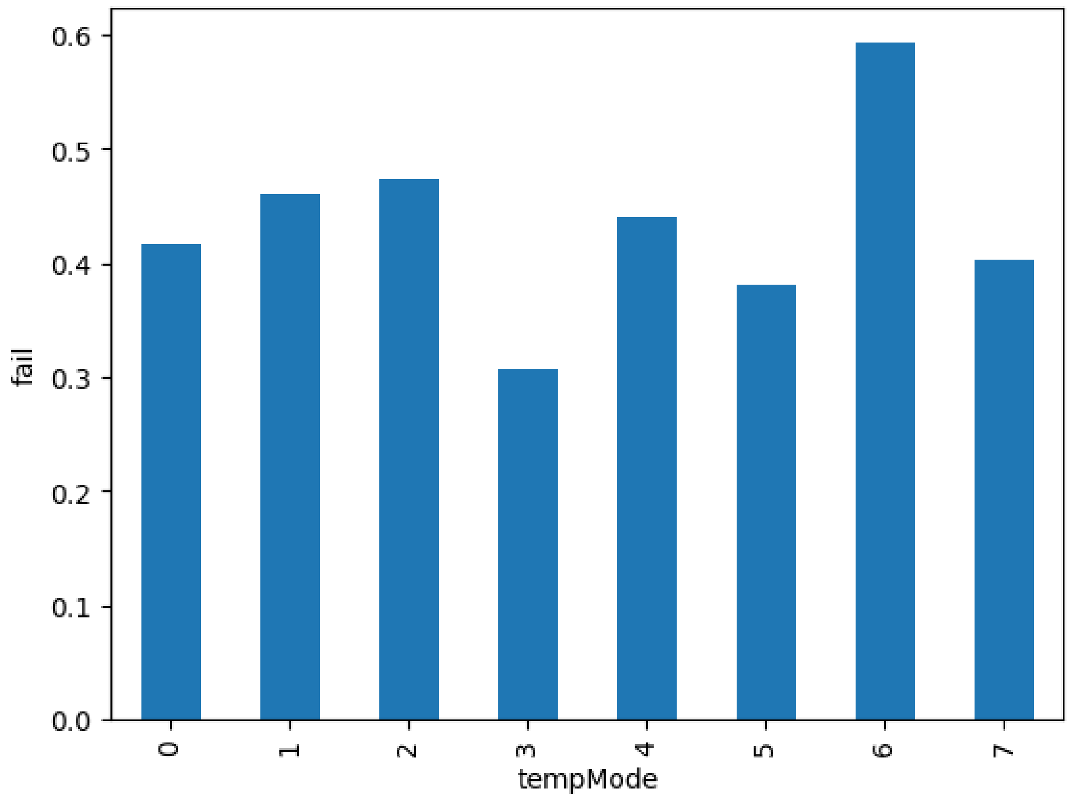
```
IP              10.362780  -15.884066  100.000000   37.277143    8.562354
Temperature     20.895564   -7.849861   37.277143  100.000000   19.025688
fail            79.732915    5.366771    8.562354   19.025688  100.000000
```

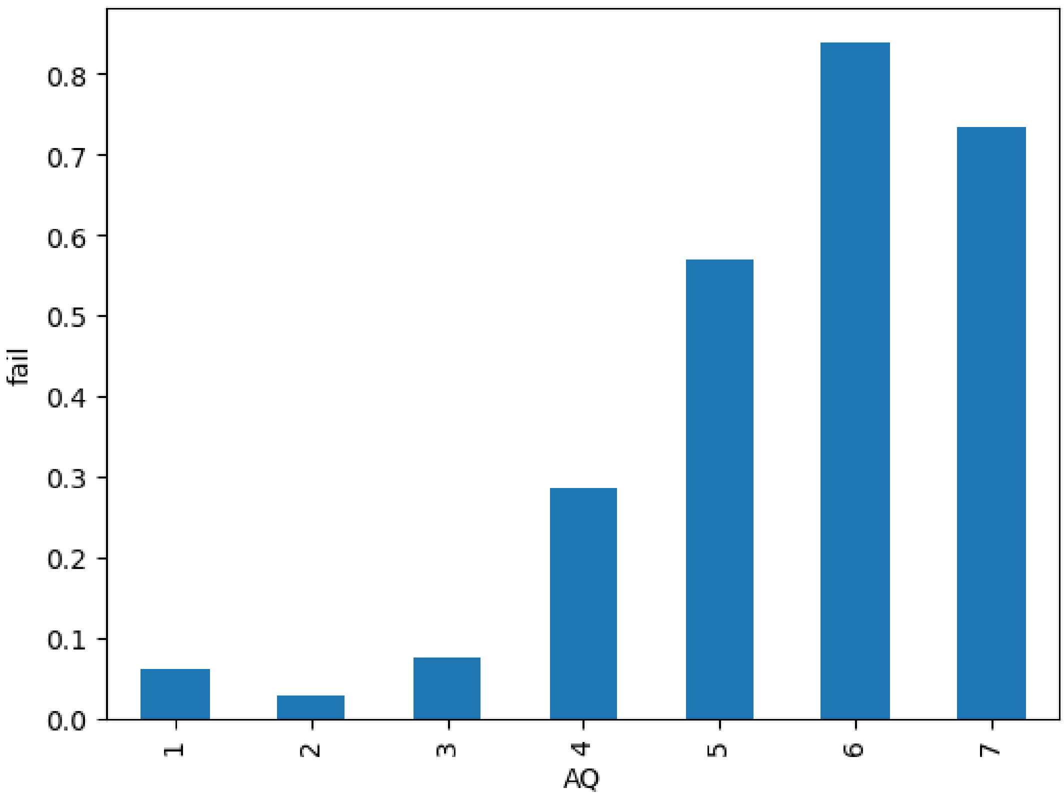
[139]:
```
sns.heatmap(D.corr(),annot=True)
plt.show()
```



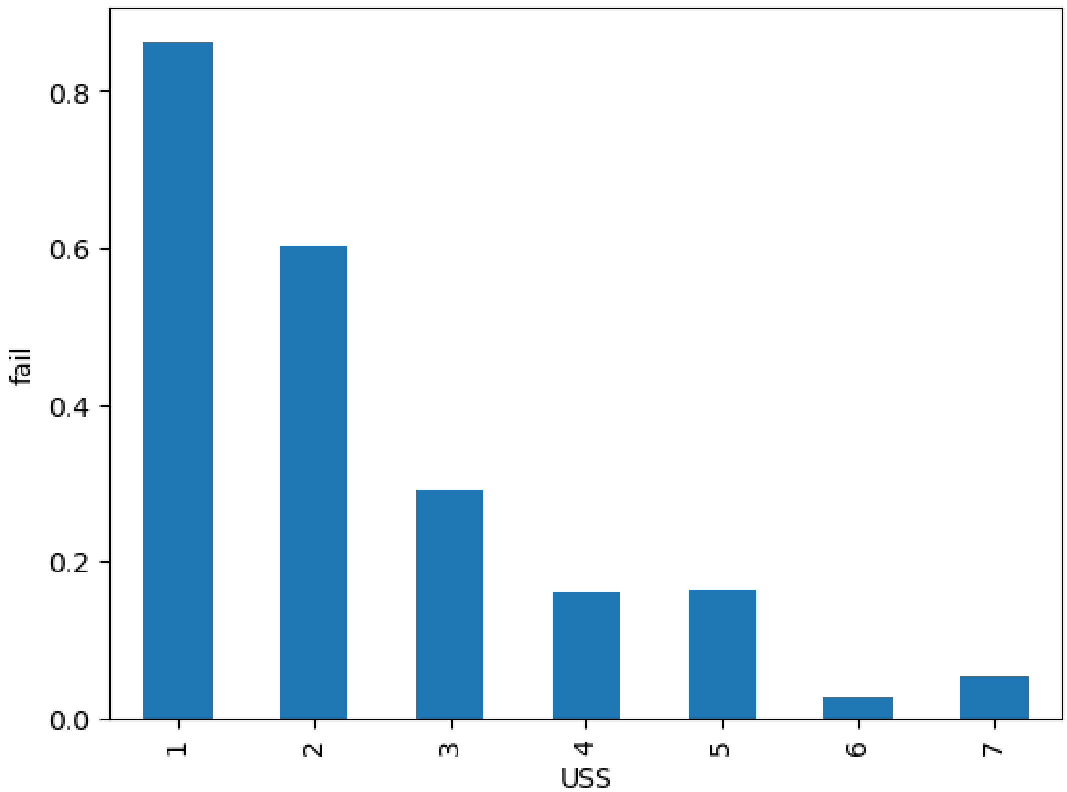
[ ]:
```
for i in D.columns:
    D.groupby(i)['fail'].mean().plot.bar()
    plt.xlabel(i)
    plt.ylabel('fail')
    plt.show()
```


1.0
0.8
0.6
0.4
0.2
0.0
fail
footfall

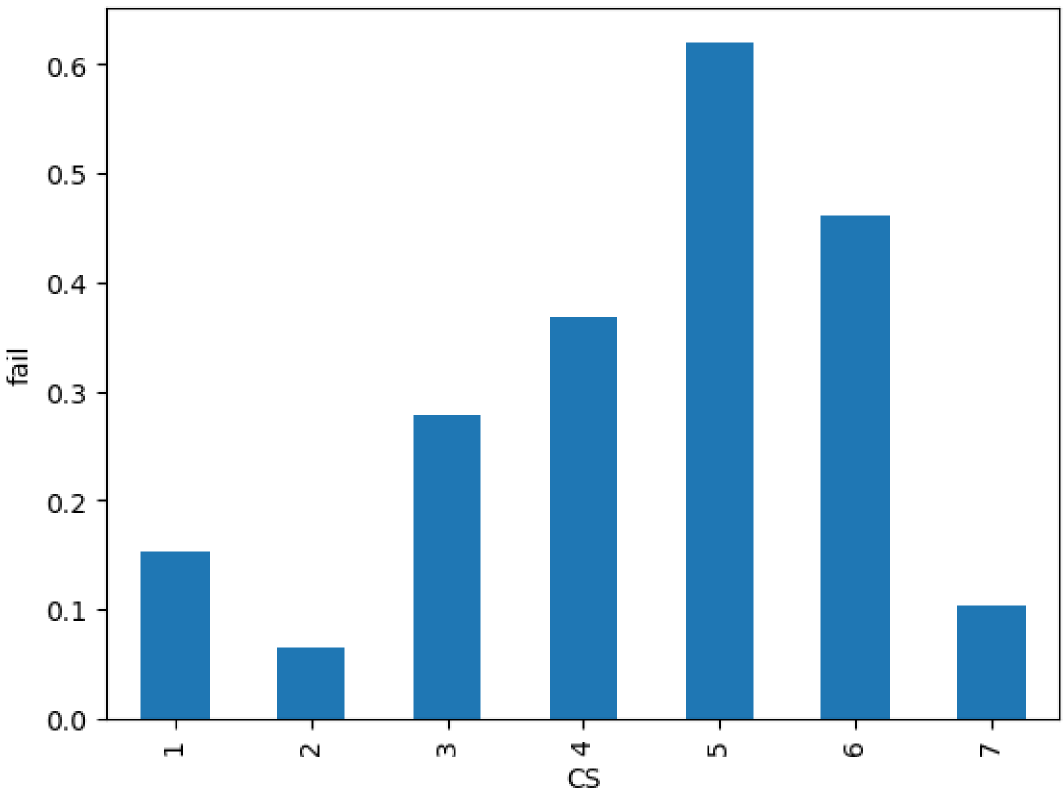

0.6
0.5
0.4
0.3
0.2
0.1
0.0
fail
0
1
2
3
4
5
6
7
tempMode

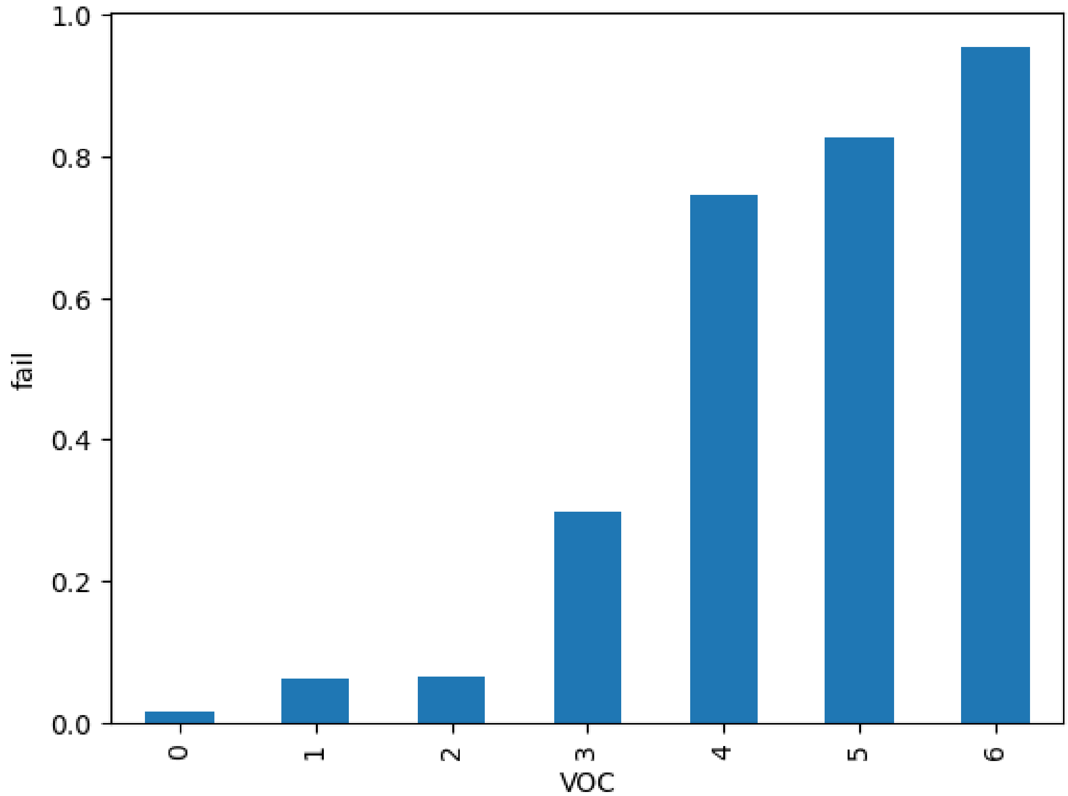


0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
fail
1
2
3
4
5
6
7
AQ

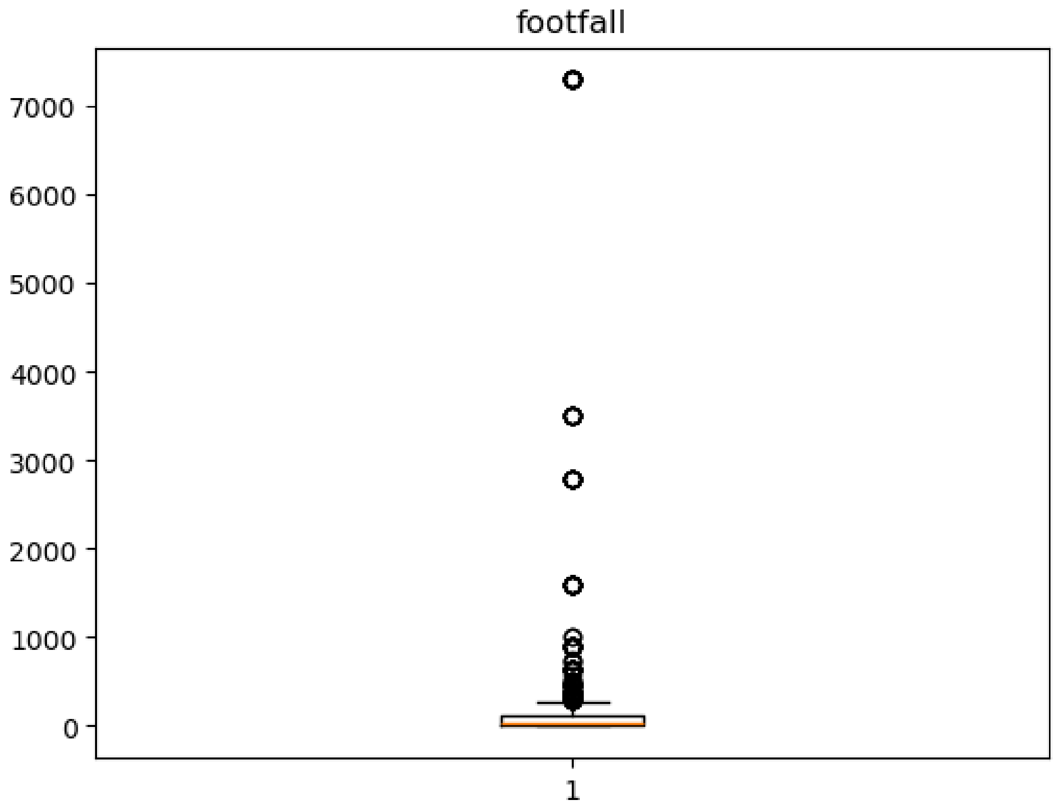


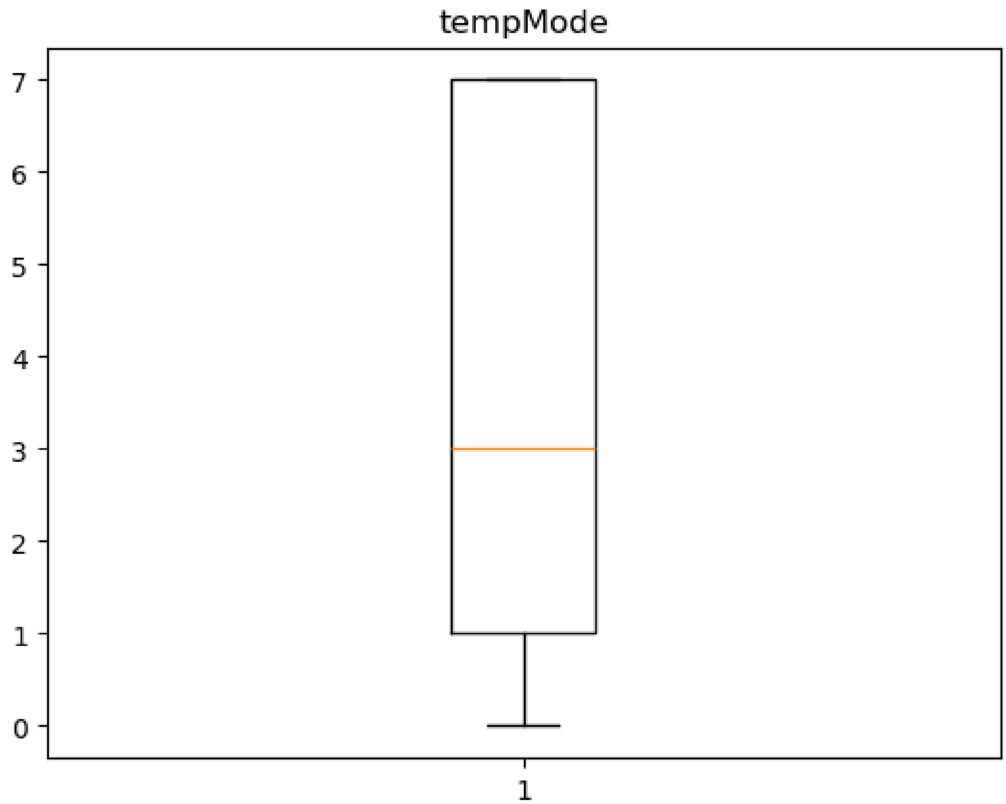
0.8
0.6
0.4
0.2
0.0
fail
1
2
3
4
5
6
7
USS

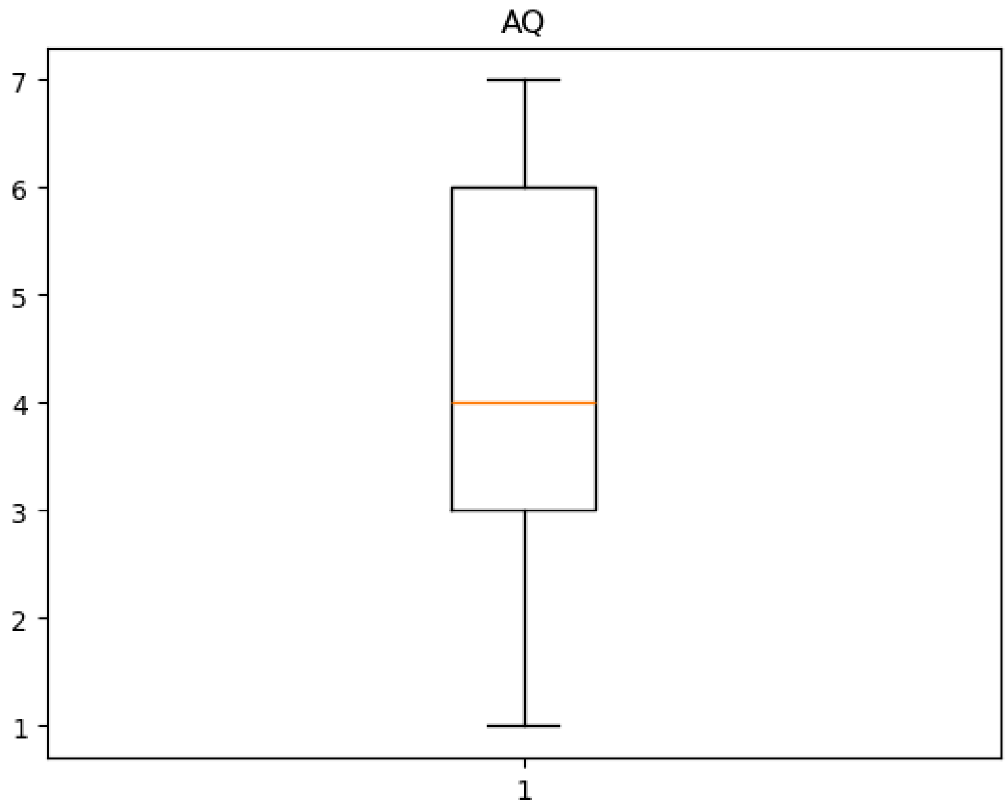

0.6
0.5
0.4
0.3
0.2
0.1
0.0
fail
1
2
3
4
5
6
7
CS

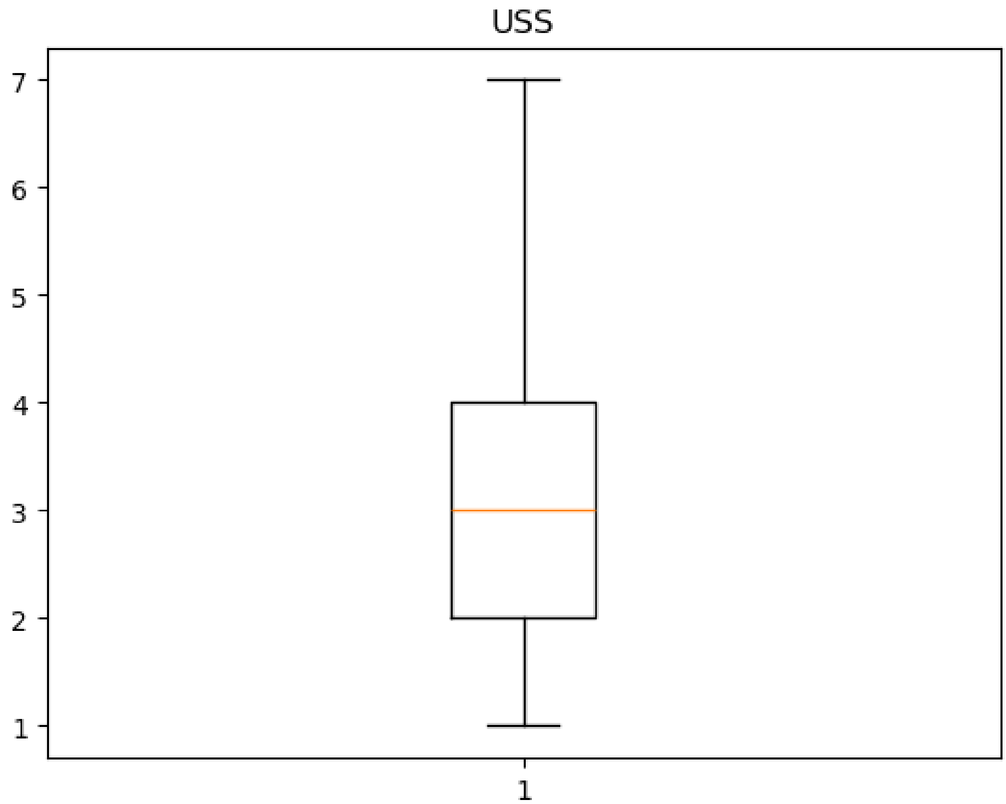


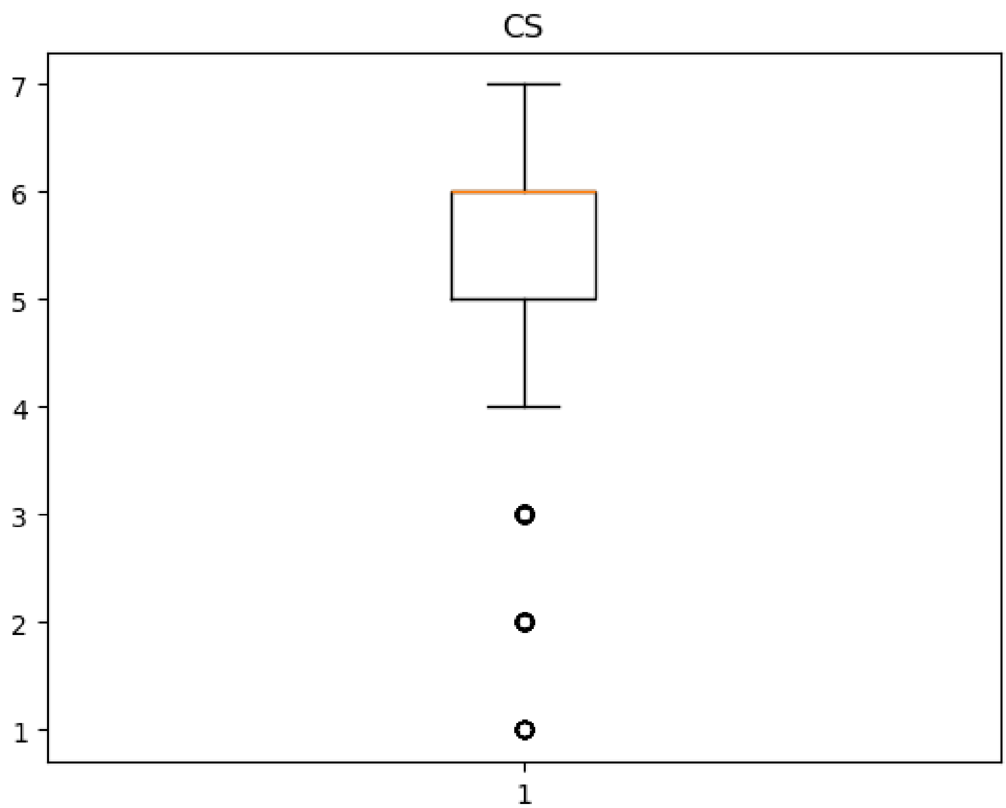
[72]:
```
for i in D.columns:
    plt.boxplot(D[i])
    plt.title(i)
    plt.show()
```

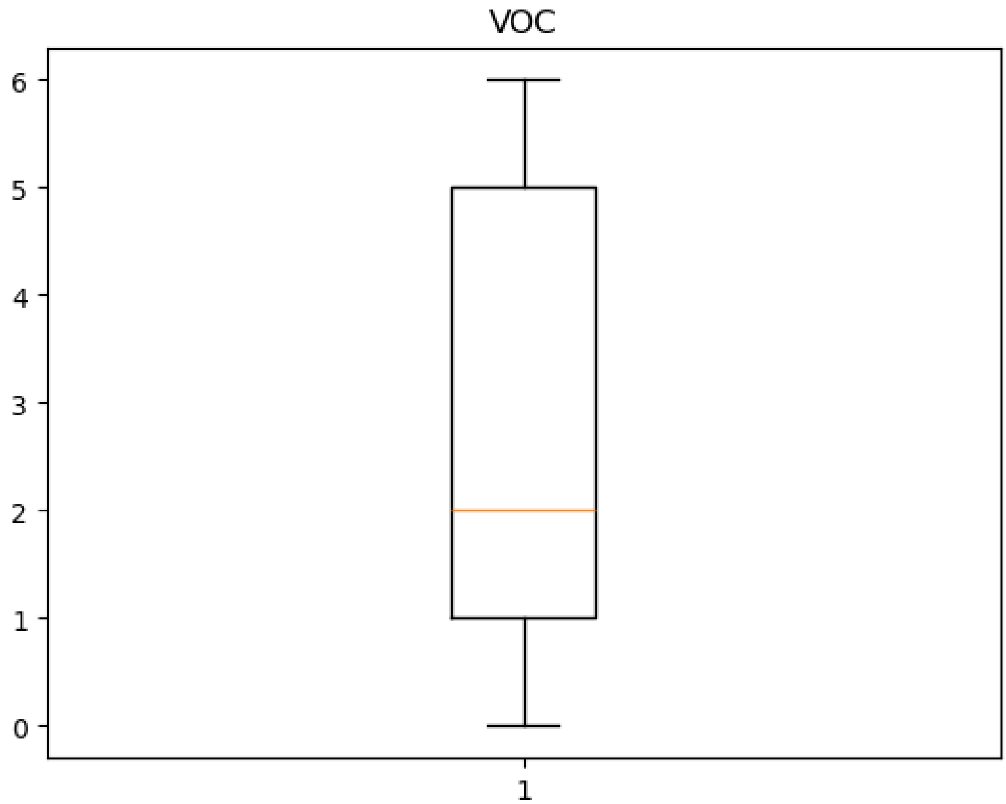

footfall
7000
6000
5000
4000
3000
2000
1000
0
1

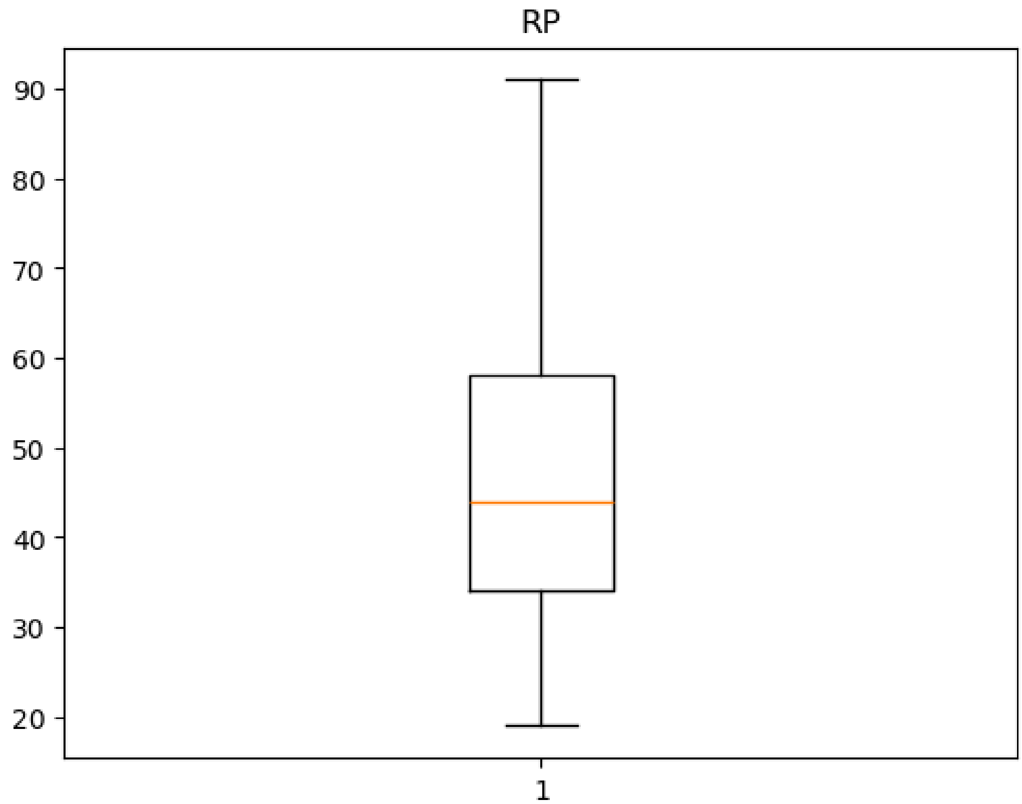

tempMode
7
6
5
4
3
2
1
0
1

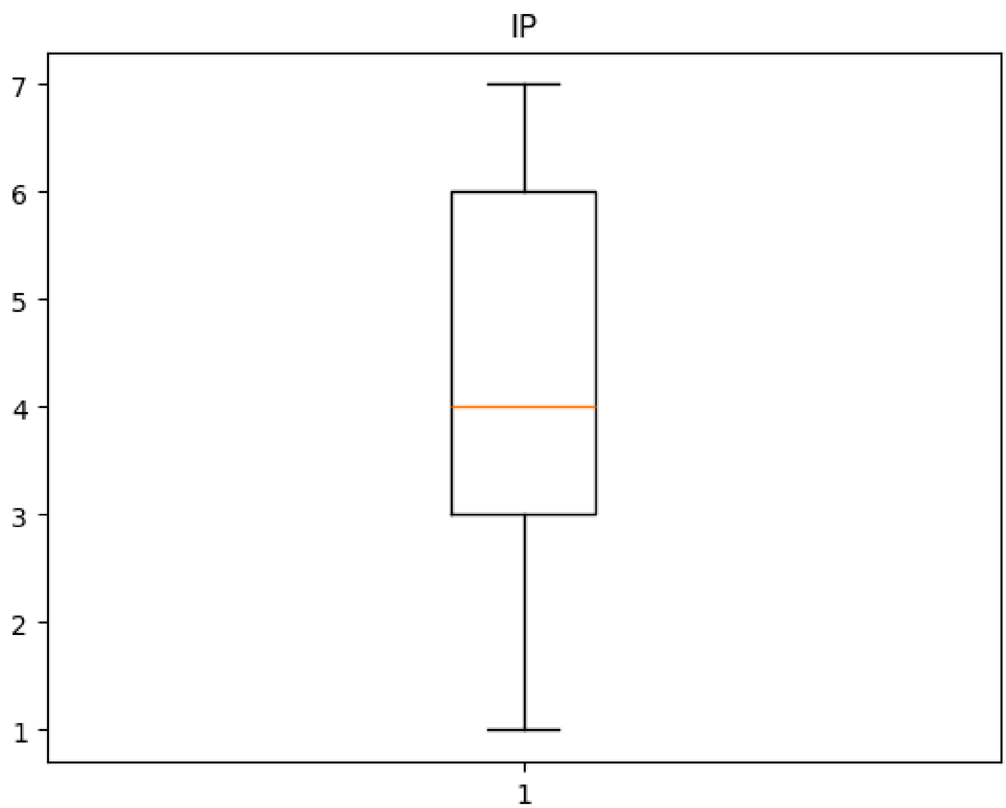

AQ
7
6
5
4
3
2
1
1

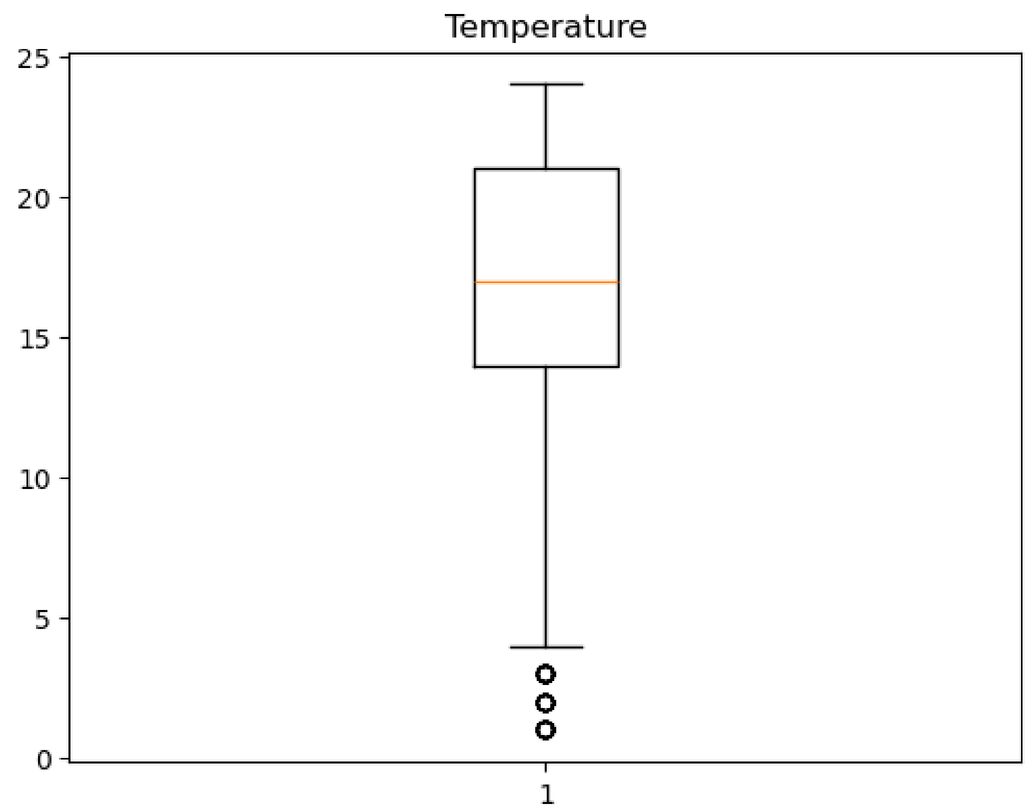

USS
7
6
5
4
3
2
1
1

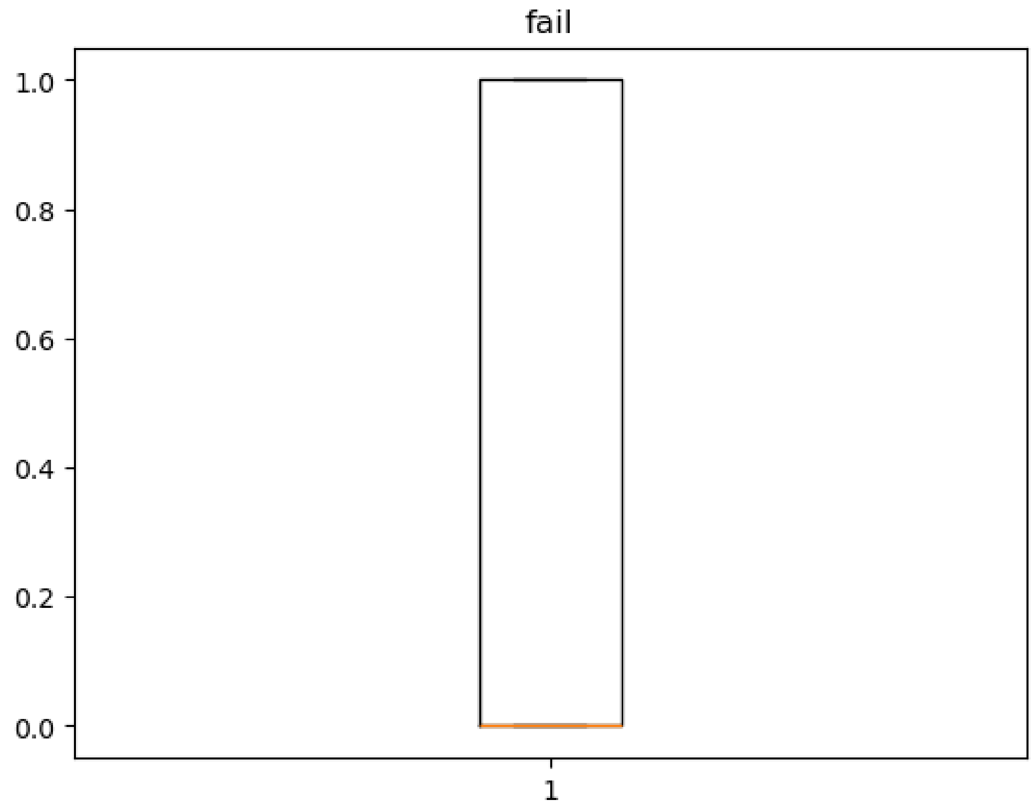

CS
7
6
5
4
3
2
1
1


VOC
6
5
4
3
2
1
0
1


RP
90
80
70
60
50
40
30
20
1

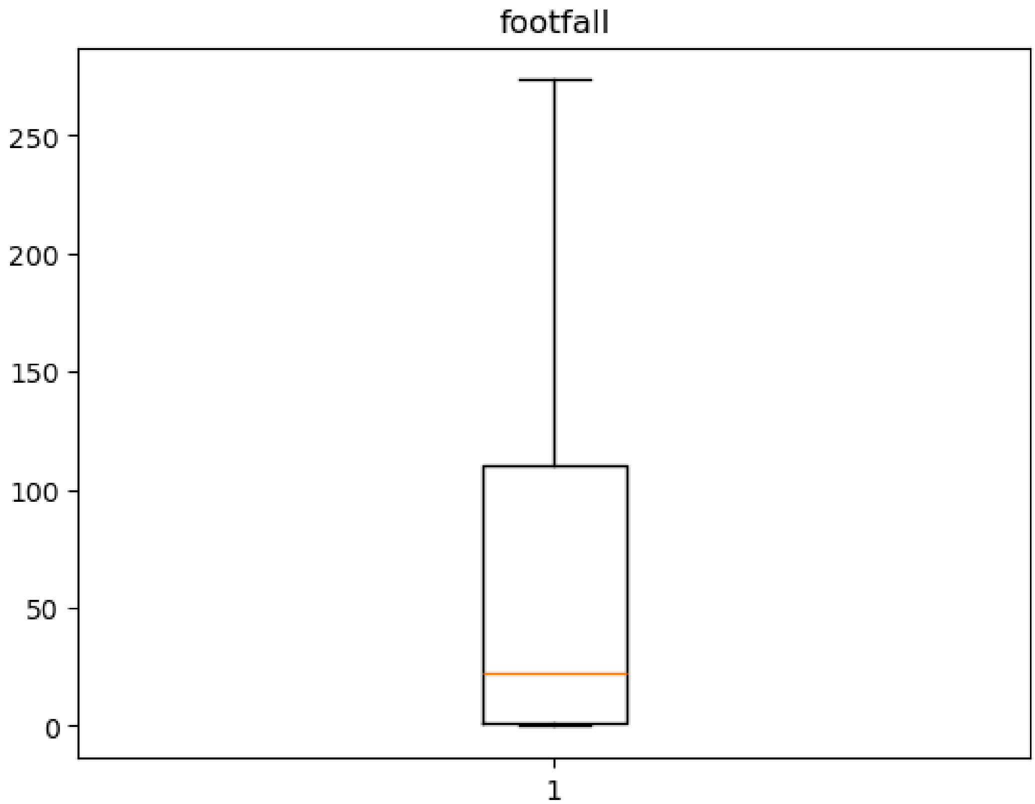

IP
7
6
5
4
3
2
1
1

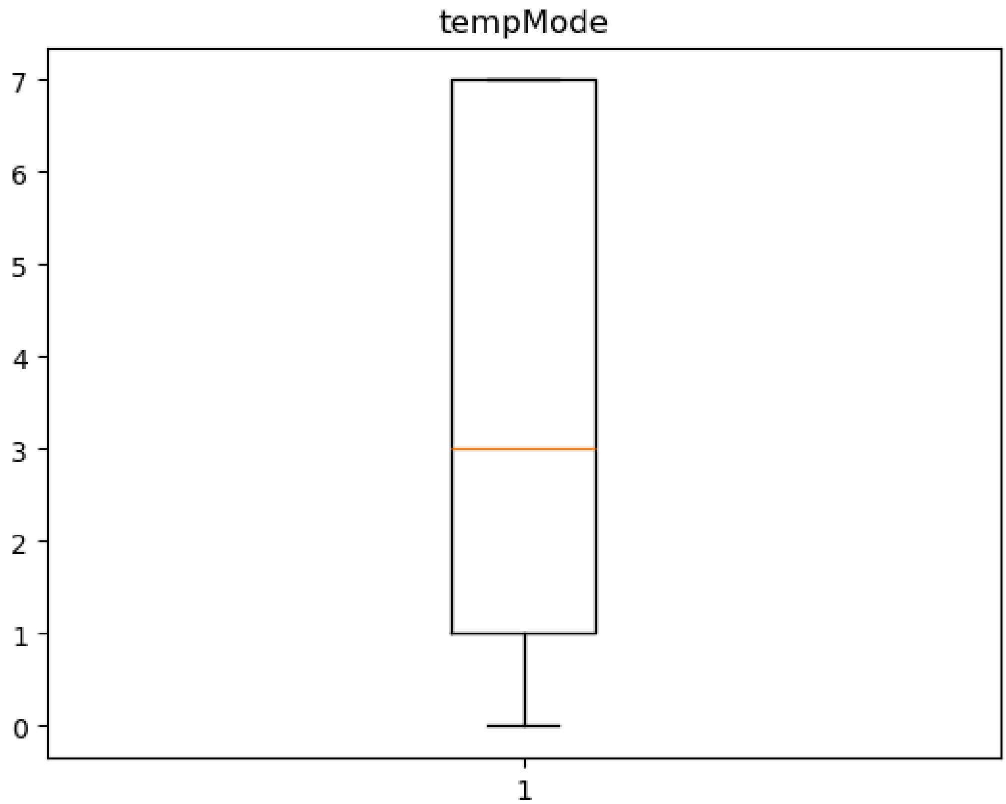

Temperature
25
20
15
10
5
0
1



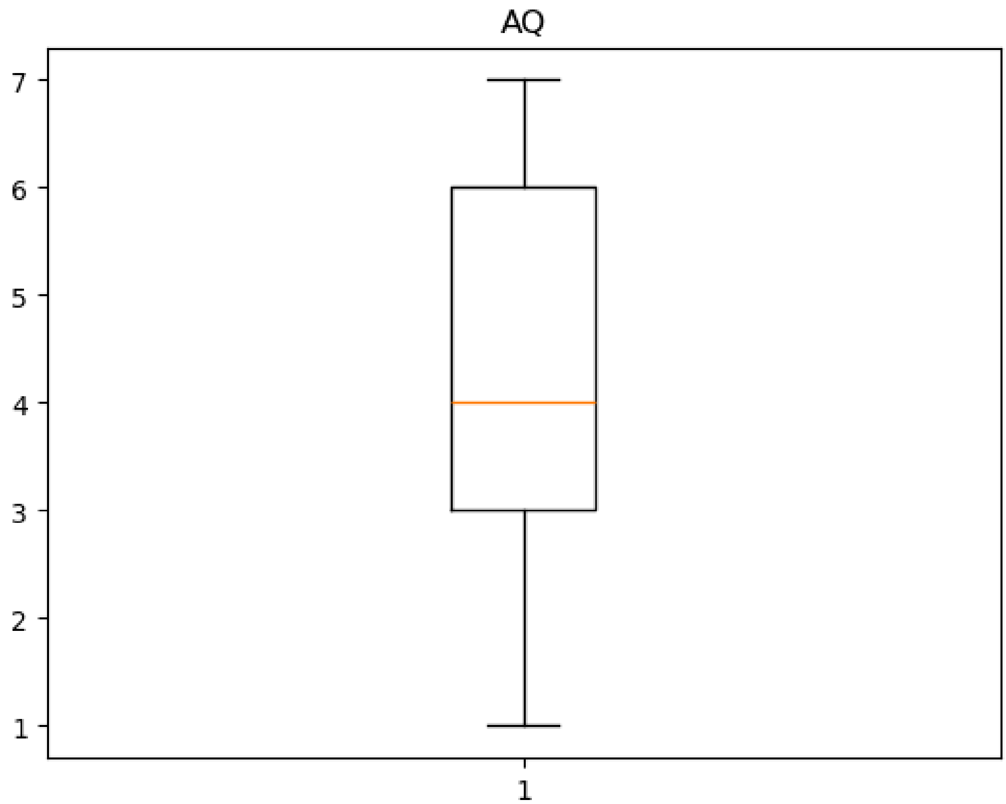
[73]:
```
for i in D.columns:
    q1=D[i].quantile(.25)
    q3=D[i].quantile(.75)
    ub=q3+(1.5*(q3-q1))
    lb=q1-(1.5*(q3-q1))
    D.loc[D[i]>ub,i]=ub
    D.loc[D[i]<lb,i]=lb
    plt.boxplot(D[i])
    plt.title(i)
    plt.show()
```

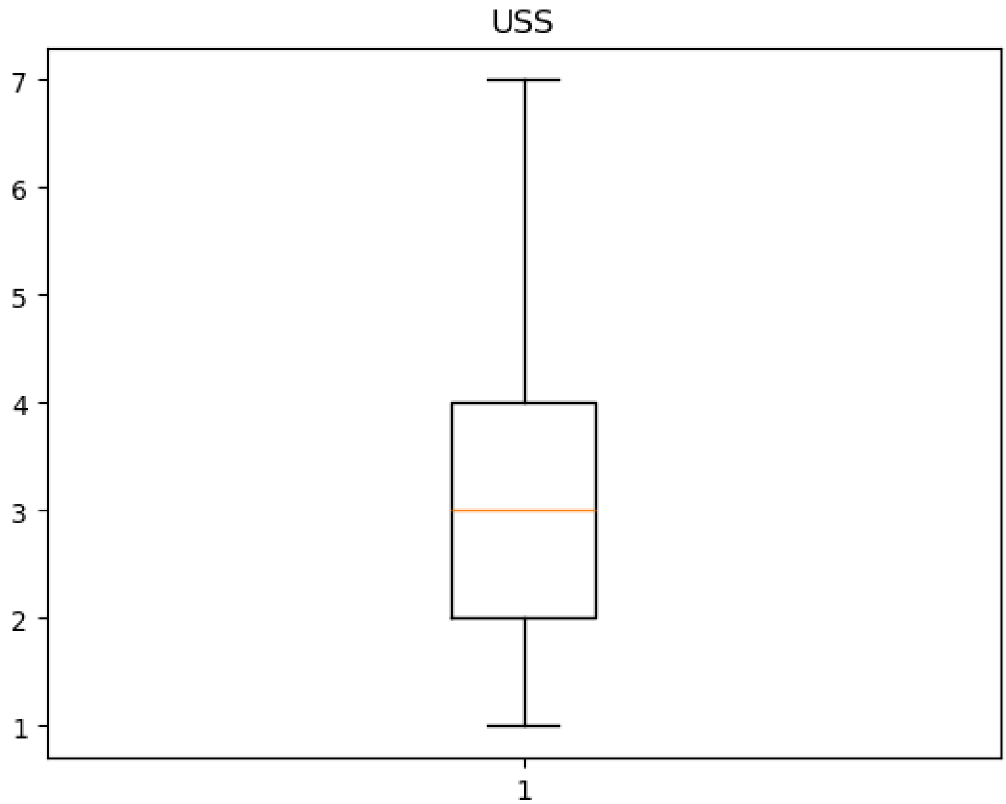
```
C:\Users\RAHUL PATIL\AppData\Local\Temp\ipykernel_11052\2536566110.py:6:
FutureWarning: Setting an item of incompatible dtype is deprecated and will
raise in a future error of pandas. Value '273.5' has dtype incompatible with
int64, please explicitly cast to a compatible dtype first.
  D.loc[D[i]>ub,i]=ub
```


footfall
250
200
150
100
50
0
1

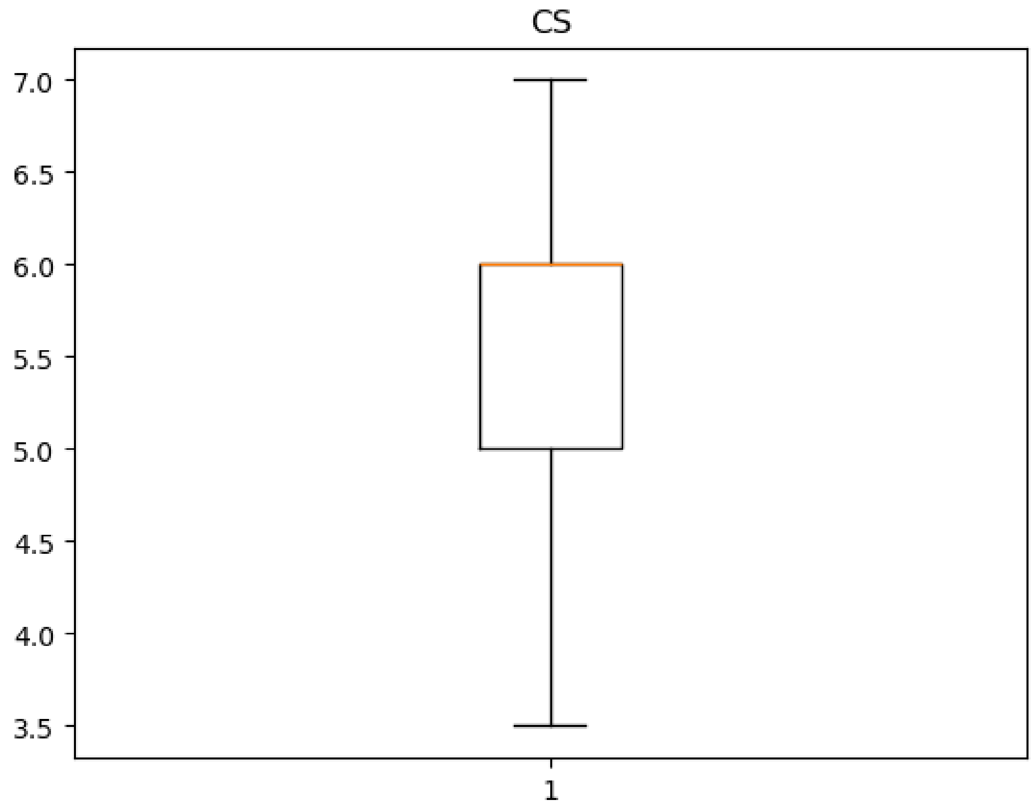


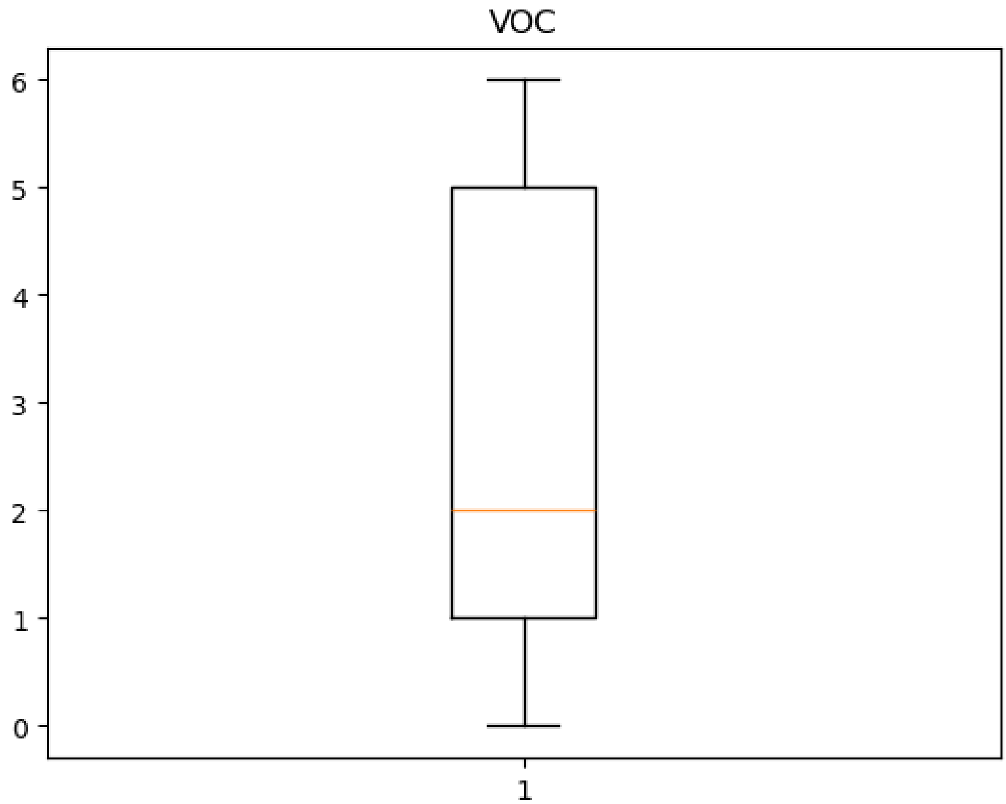

```
C:\Users\RAHUL PATIL\AppData\Local\Temp\ipykernel_11052\2536566110.py:6:
FutureWarning: Setting an item of incompatible dtype is deprecated and will
raise in a future error of pandas. Value '10.5' has dtype incompatible with
int64, please explicitly cast to a compatible dtype first.
  D.loc[D[i]>ub,i]=ub
```

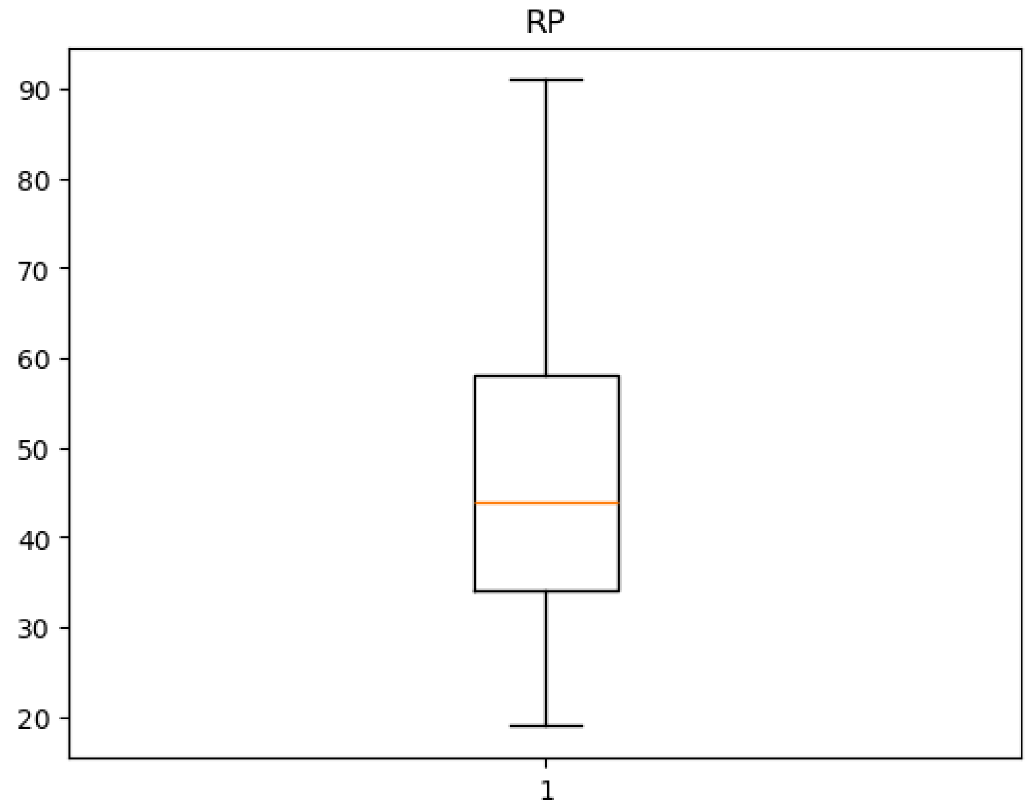

AQ
7
6
5
4
3
2
1
1



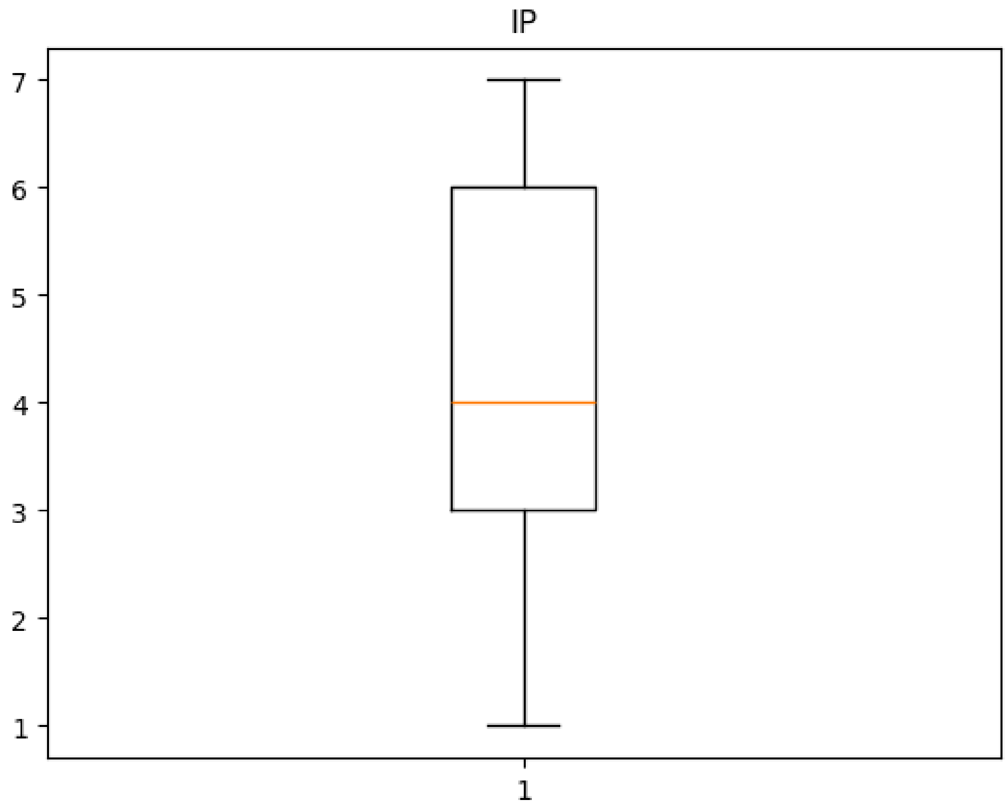

```
C:\Users\RAHUL PATIL\AppData\Local\Temp\ipykernel_11052\2536566110.py:6:
FutureWarning: Setting an item of incompatible dtype is deprecated and will
raise in a future error of pandas. Value '7.5' has dtype incompatible with
int64, please explicitly cast to a compatible dtype first.
  D.loc[D[i]>ub,i]=ub
```


CS
7.0
6.5
6.0
5.5
5.0
4.5
4.0
3.5
1

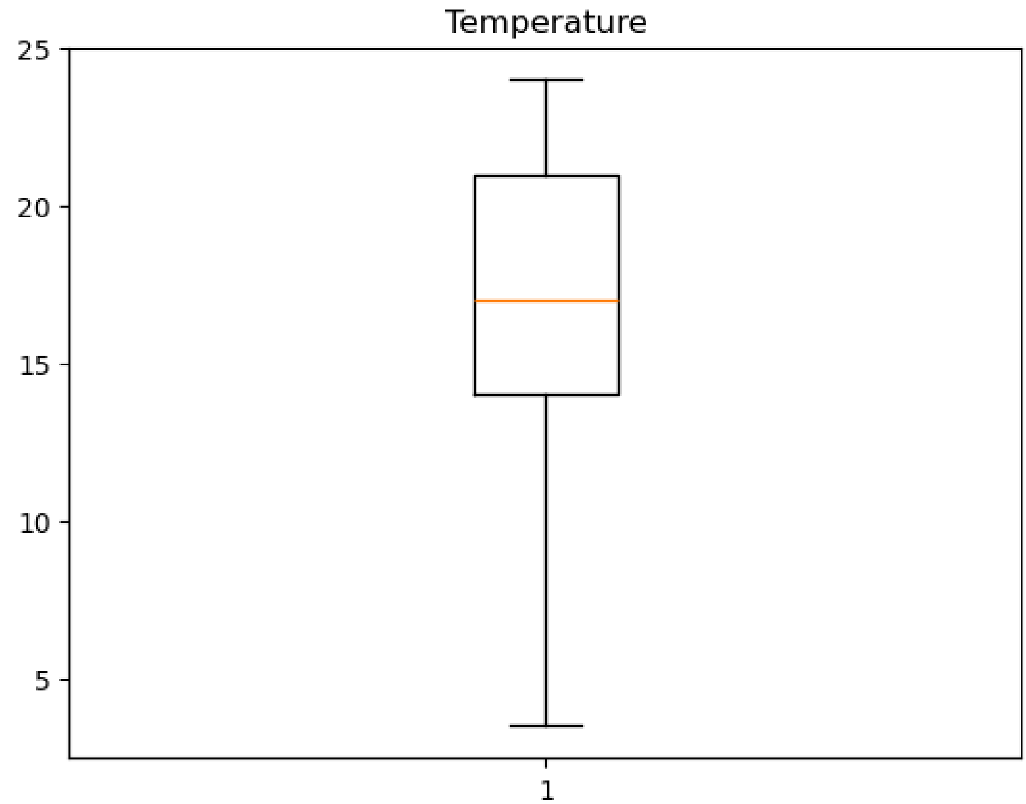

VOC
6
5
4
3
2
1
0
1



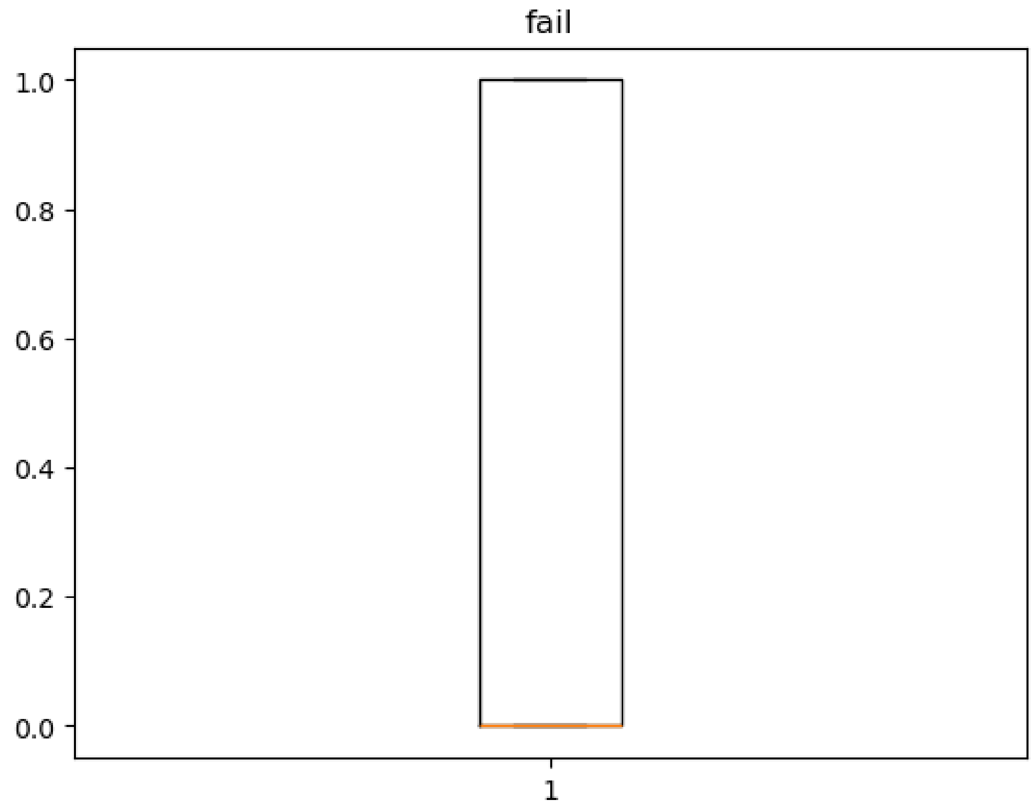

```
C:\Users\RAHUL PATIL\AppData\Local\Temp\ipykernel_11052\2536566110.py:6:
FutureWarning: Setting an item of incompatible dtype is deprecated and will
raise in a future error of pandas. Value '10.5' has dtype incompatible with
int64, please explicitly cast to a compatible dtype first.
  D.loc[D[i]>ub,i]=ub
```




```
C:\Users\RAHUL PATIL\AppData\Local\Temp\ipykernel_11052\2536566110.py:6:
FutureWarning: Setting an item of incompatible dtype is deprecated and will
raise in a future error of pandas. Value '31.5' has dtype incompatible with
int64, please explicitly cast to a compatible dtype first.
  D.loc[D[i]>ub,i]=ub
```




```
C:\Users\RAHUL PATIL\AppData\Local\Temp\ipykernel_11052\2536566110.py:6:
FutureWarning: Setting an item of incompatible dtype is deprecated and will
raise in a future error of pandas. Value '2.5' has dtype incompatible with
int64, please explicitly cast to a compatible dtype first.
  D.loc[D[i]>ub,i]=ub
```



[74]:
```
D.columns
```

[74]:
```
Index(['footfall', 'tempMode', 'AQ', 'USS', 'CS', 'VOC', 'RP', 'IP',
       'Temperature', 'fail'],
      dtype='object')
```

[75]:
```
F=D.drop('fail',axis=1) #STORE DATA INTO FEATURES AND TARGET
T=D['fail']
```

[76]:
```
from sklearn.model_selection import train_test_split  #SPLIT THE DATASET INTO␣
↪TRAIN AND TESTING DATA
x_train,x_test,y_train,y_test=train_test_split(F,T)
```

[77]:
```
from sklearn.preprocessing import MinMaxScaler
M=MinMaxScaler()
```

[78]:
```
x_train
```

[78]:
```
     footfall  tempMode   AQ  USS   CS  VOC  RP   IP  Temperature
58        0.0         0  6.0    2  6.0    6  37  6.0          4.0
472       9.0         7  3.0    4  6.0    1  53  2.0         17.0
```

```
853        19.0          3  5.0    3  7.0    3  46  4.0          23.0
857         0.0          3  3.0    3  6.0    1  39  6.0          23.0
650        54.0          7  4.0    2  6.0    4  62  5.0          20.0
..          …       …   …   …   …   …   ..  …           …
324       110.0          0  5.0    3  6.0    5  26  4.0          15.0
19         19.0          2  2.0    1  4.0    0  36  3.0           3.5
15          0.0          7  6.0    7  5.0    0  62  3.0           3.5
283        35.0          4  6.0    2  5.0    4  38  2.0          15.0
925         0.0          0  3.0    4  4.0    0  48  6.0          24.0

[708 rows x 9 columns]
```

[79]:

```
x_train=M.fit_transform(x_train) #fit the data into model
x_test=M.transform(x_test)
```

[80]:

```
x_train
```

[80]:

```
array([[0.        , 0.        , 0.83333333, …, 0.25      , 0.83333333,
        0.02439024],
       [0.03290676, 1.        , 0.33333333, …, 0.47222222, 0.16666667,
        0.65853659],
       [0.06946984, 0.42857143, 0.66666667, …, 0.375     , 0.5       ,
        0.95121951],
       …,
       [0.        , 1.        , 0.83333333, …, 0.59722222, 0.33333333,
        0.        ],
       [0.12797075, 0.57142857, 0.83333333, …, 0.26388889, 0.16666667,
        0.56097561],
       [0.        , 0.        , 0.33333333, …, 0.40277778, 0.83333333,
        1.        ]])
```

[81]:

```
x_test
```

[81]:

```
array([[1.        , 0.57142857, 0.33333333, …, 0.40277778, 0.33333333,
        0.75609756],
       [0.        , 0.28571429, 0.5       , …, 0.36111111, 0.33333333,
        0.65853659],
       [0.03290676, 1.        , 0.33333333, …, 0.29166667, 0.83333333,
        0.75609756],
       …,
       [0.12065814, 1.        , 0.83333333, …, 0.44444444, 0.5       ,
        0.        ],
       [1.        , 1.        , 0.33333333, …, 0.26388889, 0.5       ,
        0.80487805],
       [0.        , 0.        , 0.66666667, …, 0.25      , 0.66666667,
        0.70731707]])
```

6 LOGISTIC REGRESSION:-

[82]:
```
from sklearn.linear_model import LogisticRegression
L=LogisticRegression()
L.fit(x_train,y_train)
```

[82]:
```
LogisticRegression()
```

[83]:
```
L1=L.score(x_train,y_train)*100 #for training accuracy
L1
```

[83]:
```
90.3954802259887
```

[84]:
```
L2=L.score(x_test,y_test)*100 #for testing accuracy
L2
```

[84]:
```
93.64406779661016
```

7 SVC:-

[85]:
```
from sklearn.svm import SVC
S=SVC()
S.fit(x_train,y_train)
```

[85]:
```
SVC()
```

[86]:
```
S1=S.score(x_train,y_train)*100
S1
```

[86]:
```
92.37288135593221
```

[87]:
```
S2=S.score(x_test,y_test)*100
S2
```

[87]:
```
93.22033898305084
```

8 NAIVES BAYES:-

[88]:
```
from sklearn.naive_bayes import␣
↪GaussianNB,ComplementNB,MultinomialNB,BernoulliNB
G=GaussianNB()
C=ComplementNB()
M=MultinomialNB()
B=BernoulliNB()
```

9 GaussianNB:-

[89]:
```
G.fit(x_train,y_train)
```

[89]: GaussianNB()

[90]:
```
G1=G.score(x_train,y_train)*100
G1
```

[90]: 90.5367231638418

[91]:
```
G2=G.score(x_test,y_test)*100
G2
```

[91]: 93.22033898305084

10 BernoulliNB:-

[92]:
```
B.fit(x_train,y_train)
```

[92]: BernoulliNB()

[93]:
```
B1=B.score(x_train,y_train)*100
B1
```

[93]: 70.90395480225989

[94]:
```
B2=B.score(x_test,y_test)*100
B2
```

[94]: 68.22033898305084

11 ComplementNB:-

[95]:
```
C.fit(x_train,y_train)
```

[95]: ComplementNB()

[96]:
```
C1=C.score(x_train,y_train)*100
C1
```

[96]: 88.70056497175142

[97]:
```
C2=C.score(x_test,y_test)*100
C2
```

[97]: 90.2542372881356

12 MultinomialNB:-

[98]:
```
M.fit(x_train,y_train)
```

[98]: MultinomialNB()

[99]:
```
M1=M.score(x_train,y_train)*100
M1
```

[99]: 88.2768361581921

[100]:
```
M2=M.score(x_test,y_test)*100
M2
```

[100]: 89.83050847457628

13 K NEAREST NEIGHBOR:-

[101]:
```
from sklearn.neighbors import KNeighborsClassifier
K=KNeighborsClassifier()
```

[102]:
```
K.fit(x_train,y_train)
```

[102]: KNeighborsClassifier()

[103]:
```
K1=K.score(x_train,y_train)*100
K1
```

[103]: 90.67796610169492

[104]:
```
K2=K.score(x_test,y_test)*100
K2
```

[104]: 91.52542372881356

14 DECISION TREE CLASSIFIER:-

[105]:
```
from sklearn.tree import DecisionTreeClassifier
D=DecisionTreeClassifier()
```

[106]:
```
D.fit(x_train,y_train)
```

[106]: DecisionTreeClassifier()

[107]:
```
D1=D.score(x_train,y_train)*100
D1
```

[107]: 100.0

[108]:
```
D2=D.score(x_test,y_test)*100
D2
```

[108]: 86.4406779661017

15 RANDOM FOREST:-

[111]:
```
from sklearn.ensemble import RandomForestClassifier
f=RandomForestClassifier()
f.fit(x_train,y_train)
```

[111]: RandomForestClassifier()

[121]:
```
F1=f.score(x_train,y_train)*100
F1
```

[121]: 100.0

[122]:
```
F2=f.score(x_test,y_test)*100
F2
```

[122]: 90.67796610169492

16 ADA BOOST:-

[123]:
```
from sklearn.ensemble import AdaBoostClassifier
A=AdaBoostClassifier()
```

[124]:
```
A.fit(x_train,y_train)
```

[124]: AdaBoostClassifier()

[125]:
```
A1=A.score(x_train,y_train)*100
A1
```

[125]: 91.94915254237289

[126]:
```
A2=A.score(x_test,y_test)*100
A2
```

[126]: 92.37288135593221

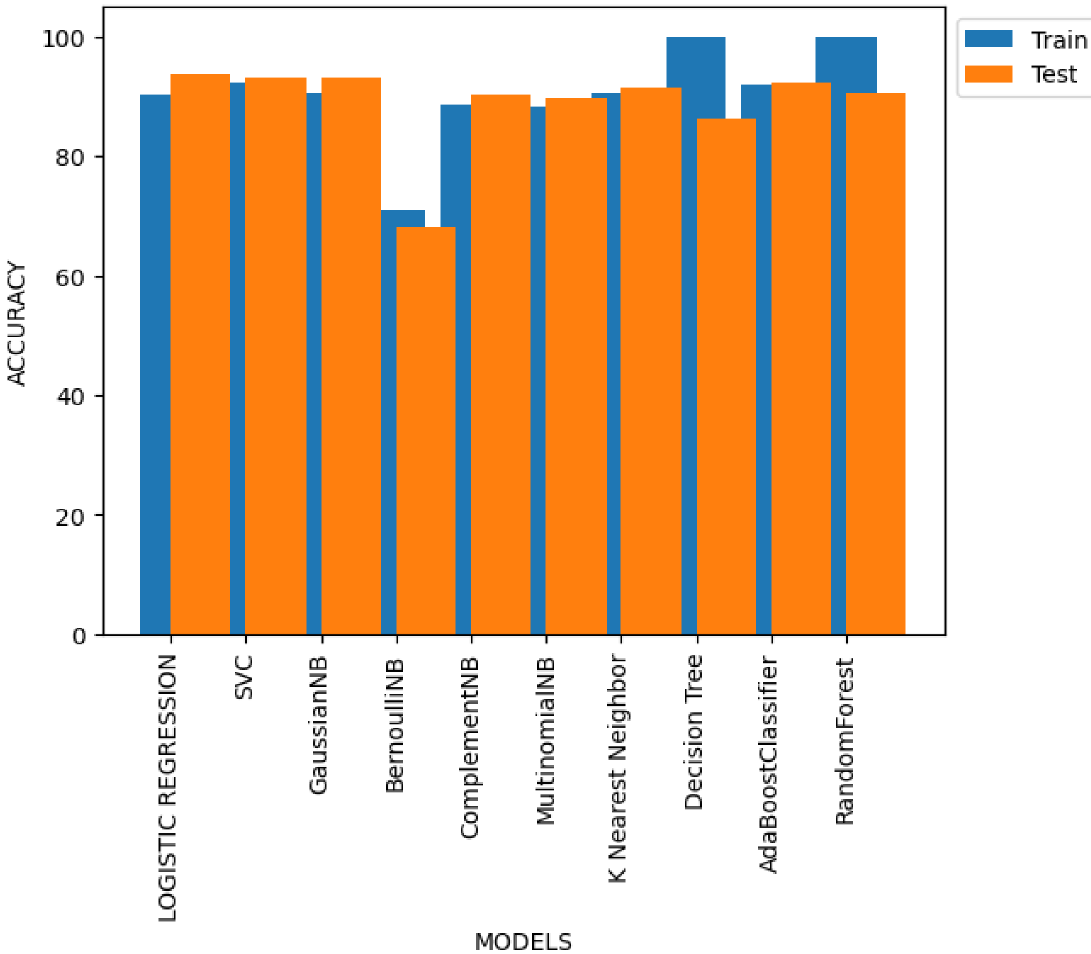
17 ACCURACY GRAPH :-

[127]:
```
AC={'Models':['LOGISTIC␣
↪REGRESSION','SVC','GaussianNB','BernoulliNB','ComplementNB','MultinomialNB','K␣
↪Nearest Neighbor','Decision␣
↪Tree','AdaBoostClassifier','RandomForest'],'Train Accuracy':
↪[L1,S1,G1,B1,C1,M1,K1,D1,A1,F1],'Test Accuracy':
↪[L2,S2,G2,B2,C2,M2,K2,D2,A2,F2]}
AC=pd.DataFrame(AC)
AC=np.around(AC,2)
AC
```

[127]:

	Models	Train Accuracy	Test Accuracy
0	LOGISTIC REGRESSION	90.40	93.64
1	SVC	92.37	93.22
2	GaussianNB	90.54	93.22
3	BernoulliNB	70.90	68.22
4	ComplementNB	88.70	90.25
5	MultinomialNB	88.28	89.83
6	K Nearest Neighbor	90.68	91.53
7	Decision Tree	100.00	86.44
8	AdaBoostClassifier	91.95	92.37
9	RandomForest	100.00	90.68

[128]:
```
plt.bar(AC['Models'],AC['Train Accuracy'],label='Train')
plt.bar(AC['Models'],AC['Test Accuracy'],align='edge',label='Test')
plt.legend(bbox_to_anchor=[1,0,0,1])
plt.xlabel('MODELS')
plt.ylabel('ACCURACY')
plt.xticks(rotation=90)
plt.show()
```



18 CONCLUSION :-

ABOVE THE BAR CHART IT IS CLEAR THAT SVC IS BEST FOR CLASSIFICATION FOR THIS DATASET

[ ]: