In [1]:

```
import pandas as pd
from sklearn import preprocessing
```

In [2]:

```
from keras.models import Sequential
from keras.layers import Dense
from keras.utils import to_categorical
from keras.callbacks import EarlyStopping
```

```
Using TensorFlow backend.
```

In [3]:

```
import matplotlib.pyplot as plt
import seaborn as sns
```

In [4]:

```
from sklearn.decomposition import PCA
```

In [5]:

```
train_df = pd.read_csv("wat-time-interval-100.csv")
```

In [6]:

```
train_df.head()
```

Out[6]:

	time_intervals	r0	r1	r2	r3	r4	r5	r6	r7	r8	...	inport_east	inport_west	outport_local
0	100	6	1	1	1	5	5	5	6	0	...	1	3	6
1	100	7	8	1	1	5	11	5	6	0	...	4	3	12
2	200	6	6	0	0	7	5	0	0	12	...	4	5	12
3	200	6	0	0	0	6	0	0	0	6	...	2	0	6
4	300	6	1	1	0	5	5	6	0	0	...	1	2	6

5 rows × 34 columns

In [7]:

```
train_X = train_df.drop(columns=['time_intervals','target'])
```

In [8]:

```
train_X
```

Out[8]:

	r0	r1	r2	r3	r4	r5	r6	r7	r8	r9	...	inport_south	inport_east	inport_west	outport
0	6	1	1	1	5	5	5	6	0	0	...	15	1	3	
1	7	8	1	1	5	11	5	6	0	6	...	16	4	3	
2	6	6	0	0	7	5	0	0	12	5	...	0	4	5	
3	6	0	0	0	6	0	0	0	6	0	...	0	2	0	
4	6	1	1	0	5	5	6	0	0	0	...	10	1	2	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
63041	6	6	0	0	0	0	0	0	0	0	...	6	0	0	
63042	7	7	7	0	0	0	0	0	0	0	...	2	0	12	
63043	11	11	11	0	0	0	6	0	0	0	...	10	6	12	
63044	6	0	0	0	6	6	6	0	0	0	...	12	0	0	
63045	6	6	6	6	0	0	0	6	0	0	...	0	12	18	

63046 rows × 32 columns

In [9]:

```
x = train_X.values #returns a numpy array
min_max_scaler = preprocessing.MinMaxScaler()
x_scaled = min_max_scaler.fit_transform(x)
train_X = pd.DataFrame(x_scaled)
```

In [10]:

```
corr_df = pd.concat([train_X, train_df[['target']]], axis = 1)
corr_df.corr()
```

Out[10]:

	0	1	2	3	4	5	6	7	
0	1.000000	0.653889	0.507160	0.340777	0.661897	0.379204	0.325477	0.262398	0.5
1	0.653889	1.000000	0.750968	0.526911	0.068831	0.317016	0.276059	0.269963	0.0
2	0.507160	0.750968	1.000000	0.701754	0.027827	0.037948	0.343140	0.352931	0.0
3	0.340777	0.526911	0.701754	1.000000	0.004526	0.022720	0.011406	0.508766	-0.0
4	0.661897	0.068831	0.027827	0.004526	1.000000	0.349372	0.248422	0.151741	0.7
5	0.379204	0.317016	0.037948	0.022720	0.349372	1.000000	0.398816	0.250118	0.0
6	0.325477	0.276059	0.343140	0.011406	0.248422	0.398816	1.000000	0.267795	0.0
7	0.262398	0.269963	0.352931	0.508766	0.151741	0.250118	0.267795	1.000000	0.0
8	0.503852	0.031960	0.004822	-0.010799	0.778664	0.042090	0.018470	0.014883	1.0
9	0.333070	0.215330	0.027869	0.013805	0.353675	0.349830	0.035910	0.036135	0.4
10	0.282609	0.198526	0.257074	0.010885	0.260424	0.006967	0.401698	0.038666	0.3
11	0.208082	0.194196	0.262244	0.387462	0.145682	-0.002734	-0.031493	0.509019	0.1
12	0.327811	0.019444	-0.001196	-0.011293	0.513167	0.024180	0.011610	0.005052	0.6
13	0.272327	0.133172	0.020437	0.005685	0.325217	0.205143	0.030821	0.023551	0.4
14	0.215413	0.124177	0.153264	0.011667	0.244657	0.014015	0.237256	0.031726	0.3
15	0.155611	0.114480	0.154060	0.231545	0.144339	0.004544	-0.021738	0.313594	0.1
16	0.580321	0.656525	0.621837	0.497079	0.296395	0.327642	0.387119	0.481620	0.2
17	0.703021	0.348643	0.273474	0.214634	0.750577	0.363267	0.275876	0.207605	0.6
18	0.632444	0.692807	0.669859	0.513947	0.247464	0.277194	0.373895	0.481462	0.2
19	0.591077	0.344848	0.338829	0.291311	0.599954	0.295722	0.298517	0.284951	0.5
20	0.928790	0.714592	0.554794	0.387384	0.625095	0.437670	0.367544	0.306522	0.4
21	0.582999	0.117839	0.047455	0.013963	0.812326	0.256095	0.158409	0.100088	0.8
22	0.594529	0.417066	0.407609	0.361163	0.512715	0.340630	0.323630	0.292109	0.4
23	0.482721	0.429038	0.376271	0.251580	0.354242	0.313552	0.351047	0.379882	0.3
24	0.529766	0.750239	0.806957	0.688074	0.068810	0.179492	0.337869	0.538782	0.0
25	0.939613	0.699315	0.538435	0.377834	0.665000	0.436844	0.361399	0.301783	0.5
26	0.491962	0.448013	0.392350	0.262995	0.335248	0.309670	0.348219	0.378665	0.3
27	0.548348	0.736740	0.786028	0.660620	0.068637	0.170792	0.321891	0.520082	0.0
28	0.566045	0.118581	0.049323	0.015933	0.815389	0.268189	0.169694	0.109496	0.8
29	0.576227	0.429987	0.433140	0.390455	0.488248	0.339310	0.337808	0.313705	0.3
30	0.867312	0.667443	0.602962	0.463569	0.671029	0.430830	0.435779	0.434737	0.6
31	0.867312	0.667443	0.602962	0.463569	0.671029	0.430830	0.435779	0.434737	0.6
target	-0.003530	-0.031543	-0.009298	-0.006043	-0.003693	-0.026236	-0.006352	-0.004051	-0.0

33 rows × 33 columns

In [11]:

```
train_Y = train_df['target']
```

In [12]:

```
train_Y
```

Out[12]:

```
0        1
1        0
2        0
3        1
4        1
        ..
63041    1
63042    1
63043    1
63044    1
63045    1
Name: target, Length: 63046, dtype: int64
```

In [13]:

```
model = Sequential()
```

In [14]:

```
n_cols = train_X.shape[1]
n_cols
```

Out[14]:

```
32
```

In [22]:

```
model.add(Dense(32, activation='relu', input_shape=(n_cols,)))
model.add(Dense(16, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(4, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

In [23]:

```
model.compile(optimizer='sgd', loss='mean_squared_error', metrics=['accuracy'])
```

In [24]:

```
early_stopping_monitor = EarlyStopping(patience=5)
```

In [25]:

```
model.fit(train_X, train_Y, epochs=30, validation_split=0.2, callbacks=[early_stopping_mon
itor])
```

```
Train on 50436 samples, validate on 12610 samples
Epoch 1/30
50436/50436 [==============================] - 6s 122us/step - loss: 0.2500 -
accuracy: 0.5016 - val_loss: 0.2500 - val_accuracy: 0.4995
Epoch 2/30
50436/50436 [==============================] - 5s 96us/step - loss: 0.2500 -
accuracy: 0.4959 - val_loss: 0.2500 - val_accuracy: 0.5000
Epoch 3/30
50436/50436 [==============================] - 5s 94us/step - loss: 0.2500 -
accuracy: 0.4951 - val_loss: 0.2500 - val_accuracy: 0.4999
Epoch 4/30
50436/50436 [==============================] - 5s 105us/step - loss: 0.2500 -
accuracy: 0.4968 - val_loss: 0.2500 - val_accuracy: 0.5009
Epoch 5/30
50436/50436 [==============================] - 5s 99us/step - loss: 0.2500 -
accuracy: 0.4955 - val_loss: 0.2500 - val_accuracy: 0.5014
Epoch 6/30
50436/50436 [==============================] - 5s 95us/step - loss: 0.2500 -
accuracy: 0.4943 - val_loss: 0.2500 - val_accuracy: 0.4995
Epoch 7/30
50436/50436 [==============================] - 5s 101us/step - loss: 0.2500 -
accuracy: 0.4986 - val_loss: 0.2500 - val_accuracy: 0.4995
Epoch 8/30
50436/50436 [==============================] - 6s 121us/step - loss: 0.2500 -
accuracy: 0.5013 - val_loss: 0.2500 - val_accuracy: 0.5005
Epoch 9/30
50436/50436 [==============================] - 5s 98us/step - loss: 0.2500 -
accuracy: 0.4991 - val_loss: 0.2500 - val_accuracy: 0.5013
Epoch 10/30
50436/50436 [==============================] - 5s 96us/step - loss: 0.2500 -
accuracy: 0.4935 - val_loss: 0.2500 - val_accuracy: 0.5012
Epoch 11/30
50436/50436 [==============================] - 5s 104us/step - loss: 0.2500 -
accuracy: 0.4907 - val_loss: 0.2500 - val_accuracy: 0.5005
Epoch 12/30
50436/50436 [==============================] - 5s 95us/step - loss: 0.2500 -
accuracy: 0.4982 - val_loss: 0.2500 - val_accuracy: 0.5005
Epoch 13/30
50436/50436 [==============================] - 5s 93us/step - loss: 0.2500 -
accuracy: 0.4986 - val_loss: 0.2500 - val_accuracy: 0.5005
Epoch 14/30
50436/50436 [==============================] - 5s 104us/step - loss: 0.2500 -
accuracy: 0.4990 - val_loss: 0.2500 - val_accuracy: 0.5005
Epoch 15/30
50436/50436 [==============================] - 5s 94us/step - loss: 0.2500 -
accuracy: 0.4931 - val_loss: 0.2500 - val_accuracy: 0.5005
Epoch 16/30
50436/50436 [==============================] - 5s 93us/step - loss: 0.2500 -
accuracy: 0.4995 - val_loss: 0.2500 - val_accuracy: 0.5005
Epoch 17/30
50436/50436 [==============================] - 5s 95us/step - loss: 0.2500 -
accuracy: 0.4965 - val_loss: 0.2500 - val_accuracy: 0.5005
Epoch 18/30
50436/50436 [==============================] - 5s 94us/step - loss: 0.2500 -
accuracy: 0.4999 - val_loss: 0.2500 - val_accuracy: 0.5005
```

Out[25]:

```
<keras.callbacks.callbacks.History at 0x1ec06f9ab08>
```

In [26]:

```
pred = model.predict(train_X)
```

In [27]:

```
for i in range(100):
    print("%s, %s" % (pred[i], train_Y[i]))
```

```
[0.50215954], 1
[0.5028052], 0
[0.50223315], 0
[0.50216657], 1
[0.5021685], 1
[0.50255543], 0
[0.50215954], 1
[0.50238997], 0
[0.5021734], 1
[0.5023394], 0
[0.5023353], 0
[0.5021743], 1
[0.5030036], 0
[0.50215405], 1
[0.50241387], 0
[0.5021628], 1
[0.5023081], 0
[0.5021545], 1
[0.50307477], 0
[0.5021524], 1
[0.5021736], 0
[0.50216967], 1
[0.50243264], 0
[0.5021558], 1
[0.5022059], 1
[0.5027382], 0
[0.5021507], 1
[0.5025965], 0
[0.5021715], 1
[0.50281006], 0
[0.502714], 0
[0.5021461], 1
[0.5021684], 1
[0.50216615], 0
[0.5021685], 1
[0.50235415], 0
[0.5021856], 1
[0.50258654], 0
[0.50217587], 1
[0.50258553], 0
[0.5021544], 1
[0.5025753], 0
[0.50216156], 0
[0.5021754], 1
[0.5021575], 1
[0.50272596], 0
[0.502313], 0
[0.5021557], 1
[0.5021639], 1
[0.5025128], 0
[0.502178], 1
[0.50217724], 0
[0.5021564], 1
[0.5026738], 0
[0.50233626], 0
[0.5021608], 1
[0.5021702], 0
```

```
[0.50217736], 1
[0.50217], 1
[0.5023443], 0
[0.50249153], 0
[0.5021555], 1
[0.5021472], 0
[0.50214857], 1
[0.50218004], 1
[0.5021759], 0
[0.50256735], 0
[0.5021592], 1
[0.50214994], 0
[0.50216657], 1
[0.5021632], 0
[0.50216657], 1
[0.50254625], 0
[0.50214857], 1
[0.5025163], 0
[0.5022039], 1
[0.50214934], 1
[0.50245136], 0
[0.5025124], 0
[0.50214785], 1
[0.5021721], 0
[0.50218034], 1
[0.50218034], 1
[0.50219923], 0
[0.50217503], 1
[0.50220627], 0
[0.5021767], 1
[0.50217676], 0
[0.5021449], 1
[0.50214064], 0
[0.502165], 1
[0.50247407], 0
[0.50253916], 0
[0.5021628], 1
[0.5025979], 0
[0.502175], 1
[0.50265145], 0
[0.5021464], 1
[0.50230044], 0
[0.5021635], 1
```

In [28]:

```
pca = PCA(n_components = 2)
```

In [29]:

```
pca.fit(train_X)
```

Out[29]:

```
PCA(copy=True, iterated_power='auto', n_components=2, random_state=None,
    svd_solver='auto', tol=0.0, whiten=False)
```

In [30]:

```
principal_components = pca.transform(train_X)
principal_components
```

Out[30]:

```
array([[ 0.70750416, -0.11419311],
       [ 1.50044365, -0.26110977],
       [ 1.04977862, -0.34850538],
       ...,
       [ 1.00089985,  0.54701531],
       [ 0.550526  , -0.3960446 ],
       [ 0.78607618,  0.96350408]])
```

In [31]:

```
pca.explained_variance_ratio_
```

Out[31]:

```
array([0.5244298 , 0.15771428])
```

In [32]:

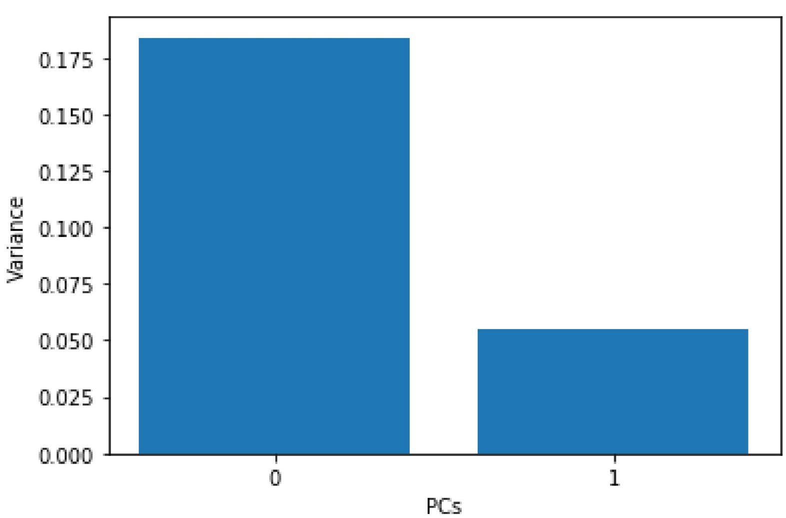
```
features = range(pca.n_components_)
```

In [33]:

```
plt.bar(features, pca.explained_variance_)
plt.xticks(features)
plt.xlabel("PCs")
plt.ylabel("Variance")
```

Out[33]:

```
Text(0, 0.5, 'Variance')
```



In [34]:

```
principal_df = pd.DataFrame(data = principal_components , columns = ['pc 1', 'pc 2'])
principal_df
```

Out[34]:

	pc 1	pc 2
0	0.707504	-0.114193
1	1.500444	-0.261110
2	1.049779	-0.348505
3	0.375046	-0.268697
4	0.531222	-0.148766
...	...	...
63041	0.214155	-0.079290
63042	0.499020	0.428835
63043	1.000900	0.547015
63044	0.550526	-0.396045
63045	0.786076	0.963504

63046 rows × 2 columns

In [35]:

```
final_df = pd.concat([principal_df, train_df[['target']]], axis = 1)
final_df
```

Out[35]:

	pc 1	pc 2	target
0	0.707504	-0.114193	1
1	1.500444	-0.261110	0
2	1.049779	-0.348505	0
3	0.375046	-0.268697	1
4	0.531222	-0.148766	1
...	...	...	...
63041	0.214155	-0.079290	1
63042	0.499020	0.428835	1
63043	1.000900	0.547015	1
63044	0.550526	-0.396045	1
63045	0.786076	0.963504	1

63046 rows × 3 columns

In [36]:

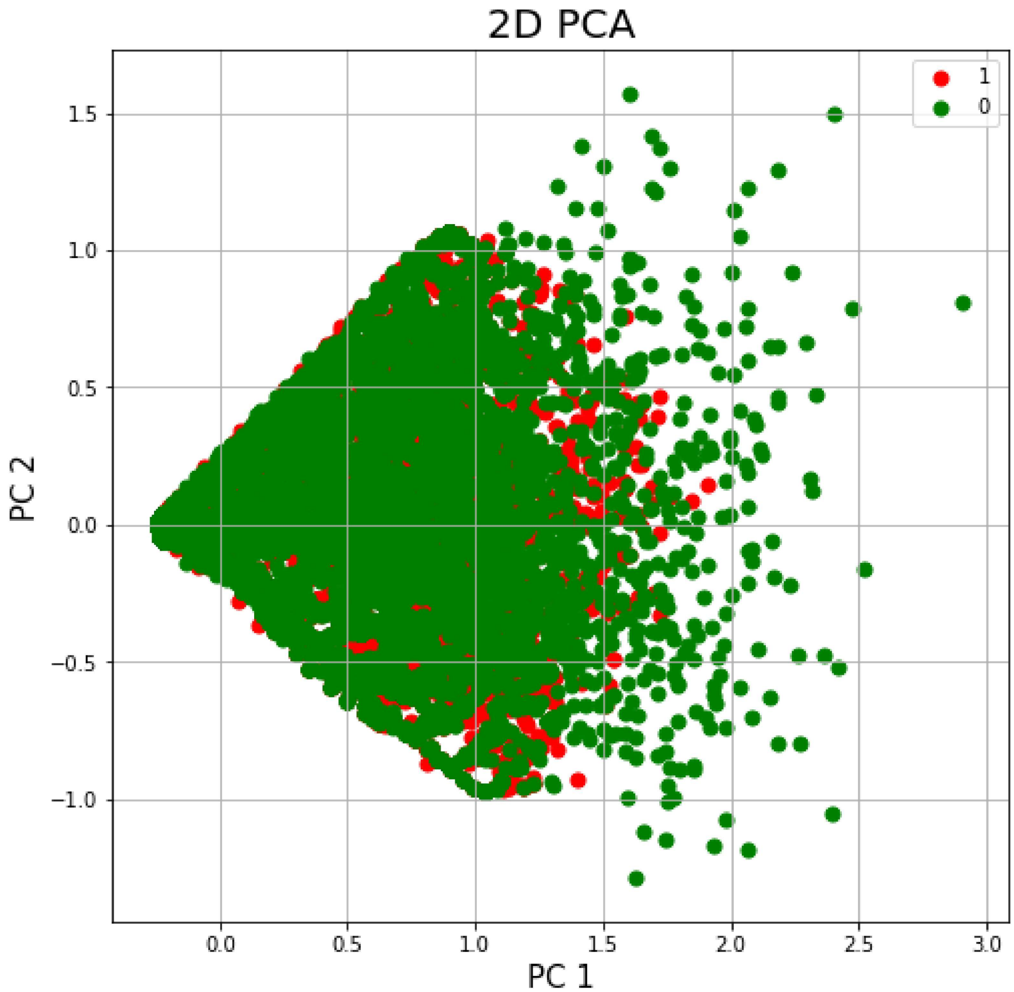
```
final_df.corr()
```

Out[36]:

	pc 1	pc 2	target
pc 1	1.000000e+00	1.321278e-14	-0.015262
pc 2	1.321278e-14	1.000000e+00	-0.003208
target	-1.526209e-02	-3.208231e-03	1.000000

In [37]:

```
fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('PC 1', fontsize = 15)
ax.set_ylabel('PC 2', fontsize = 15)
ax.set_title('2D PCA', fontsize = 20)
targets = [1, 0]
colors = ['r', 'g']
for target, color in zip(targets,colors):
    indicesToKeep = final_df['target'] == target
    ax.scatter(final_df.loc[indicesToKeep, 'pc 1']
               , final_df.loc[indicesToKeep, 'pc 2']
               , c = color
               , s = 50)
ax.legend(targets)
ax.grid()
```

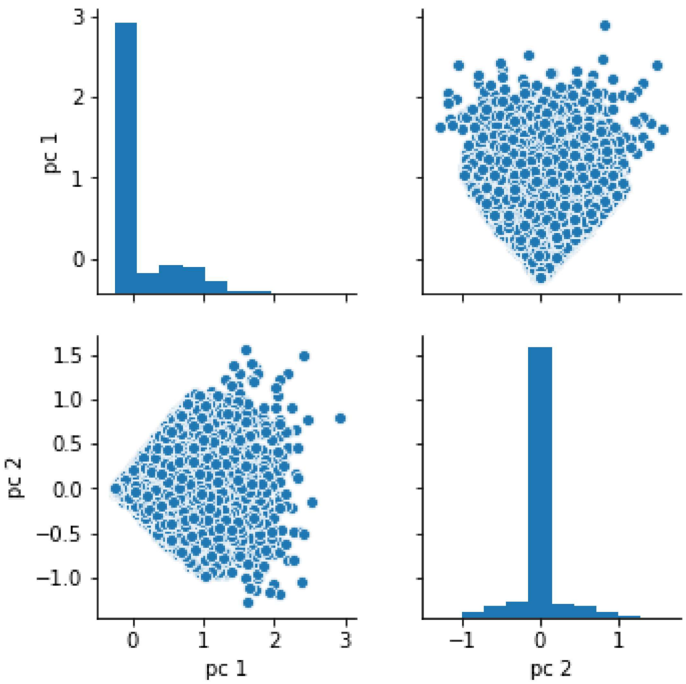

2D PCA
1
0
PC 2
PC 1
1.5
1.0
0.5
0.0
−0.5
−1.0
0.0
0.5
1.0
1.5
2.0
2.5
3.0

In [38]:

```
sns.pairplot(final_df.loc[:,final_df.dtypes == 'float64'])
```

Out[38]:

```
<seaborn.axisgrid.PairGrid at 0x1ec070f7c48>
```



In [39]:

```
corr_df
```

Out[39]:

	0	1	2	3	4	5	6	7	8
0	0.272727	0.041667	0.052632	0.055556	0.277778	0.277778	0.3125	0.461538	0.000000
1	0.318182	0.333333	0.052632	0.055556	0.277778	0.611111	0.3125	0.461538	0.000000
2	0.272727	0.250000	0.000000	0.000000	0.388889	0.277778	0.0000	0.000000	0.705882
3	0.272727	0.000000	0.000000	0.000000	0.333333	0.000000	0.0000	0.000000	0.352941
4	0.272727	0.041667	0.052632	0.000000	0.277778	0.277778	0.3750	0.000000	0.000000
...	...	...	...	...	...	...	...	...	...
63041	0.272727	0.250000	0.000000	0.000000	0.000000	0.000000	0.0000	0.000000	0.000000
63042	0.318182	0.291667	0.368421	0.000000	0.000000	0.000000	0.0000	0.000000	0.000000
63043	0.500000	0.458333	0.578947	0.000000	0.000000	0.000000	0.3750	0.000000	0.000000
63044	0.272727	0.000000	0.000000	0.000000	0.333333	0.333333	0.3750	0.000000	0.000000
63045	0.272727	0.250000	0.315789	0.333333	0.000000	0.000000	0.0000	0.461538	0.000000

63046 rows × 33 columns

In [40]:

```
corr_df[corr_df.duplicated()].shape
```

Out[40]:

```
(56581, 33)
```

In [45]:

```
dup_df = train_df.drop(columns=['time_intervals'])
dup_df
```

Out[45]:

	r0	r1	r2	r3	r4	r5	r6	r7	r8	r9	...	inport_east	inport_west	outport_local	outpor
0	6	1	1	1	5	5	5	6	0	0	...	1	3	6	
1	7	8	1	1	5	11	5	6	0	6	...	4	3	12	
2	6	6	0	0	7	5	0	0	12	5	...	4	5	12	
3	6	0	0	0	6	0	0	0	6	0	...	2	0	6	
4	6	1	1	0	5	5	6	0	0	0	...	1	2	6	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
63041	6	6	0	0	0	0	0	0	0	0	...	0	0	6	
63042	7	7	7	0	0	0	0	0	0	0	...	0	12	7	
63043	11	11	11	0	0	0	6	0	0	0	...	6	12	11	
63044	6	0	0	0	6	6	6	0	0	0	...	0	0	6	
63045	6	6	6	6	0	0	0	6	0	0	...	12	18	6	

63046 rows × 33 columns

In [46]:

```
dup_df[dup_df.duplicated()].shape
```

Out[46]:

```
(56581, 33)
```

In [47]:

```
dup_df[dup_df.duplicated()].count()
```

Out[47]:

```
r0               56581
r1               56581
r2               56581
r3               56581
r4               56581
r5               56581
r6               56581
r7               56581
r8               56581
r9               56581
r10              56581
r11              56581
r12              56581
r13              56581
r14              56581
r15              56581
pkt_get          56581
pkt_data         56581
pkt_put          56581
pkt_ack          56581
inport_local     56581
inport_north     56581
inport_south     56581
inport_east      56581
inport_west      56581
outport_local    56581
outport_north    56581
outport_south    56581
outport_east     56581
outport_west     56581
target           56581
tot_packets      56581
tot_mean         56581
dtype: int64
```

In [37]:

```
print ((56581/63046)*100)
```

```
89.74558259048948
```