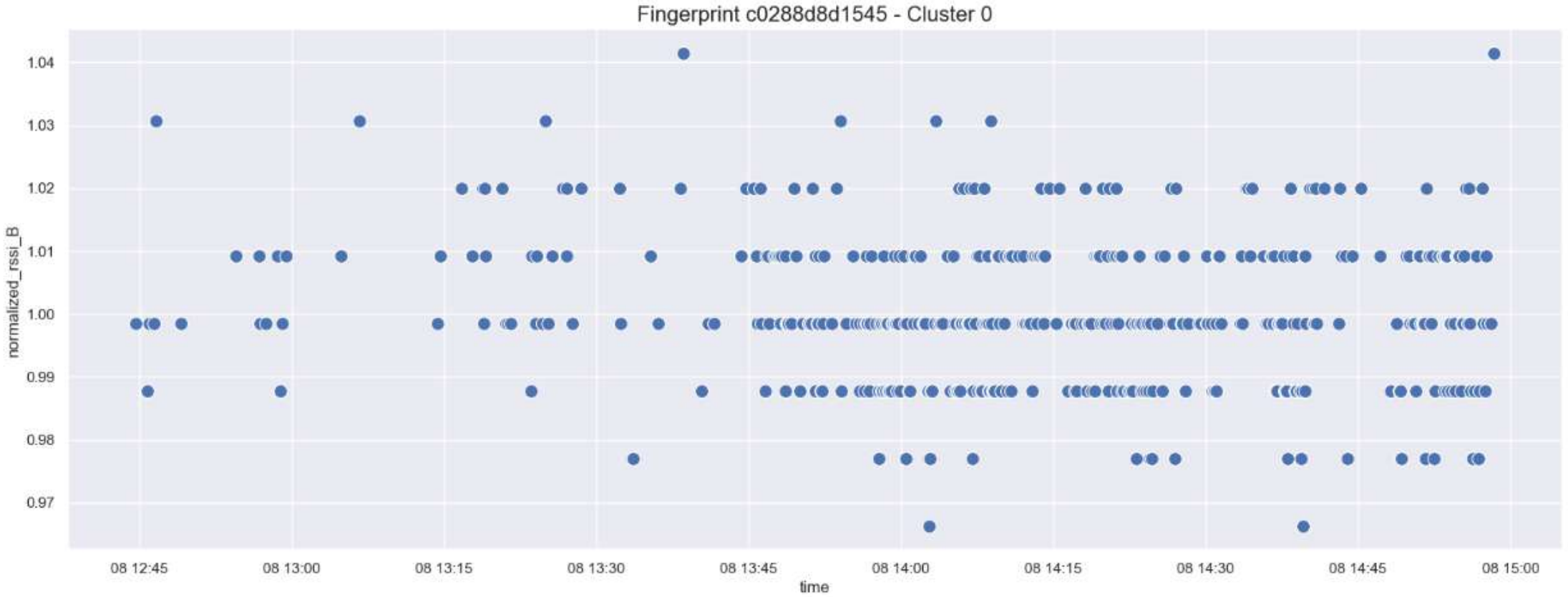
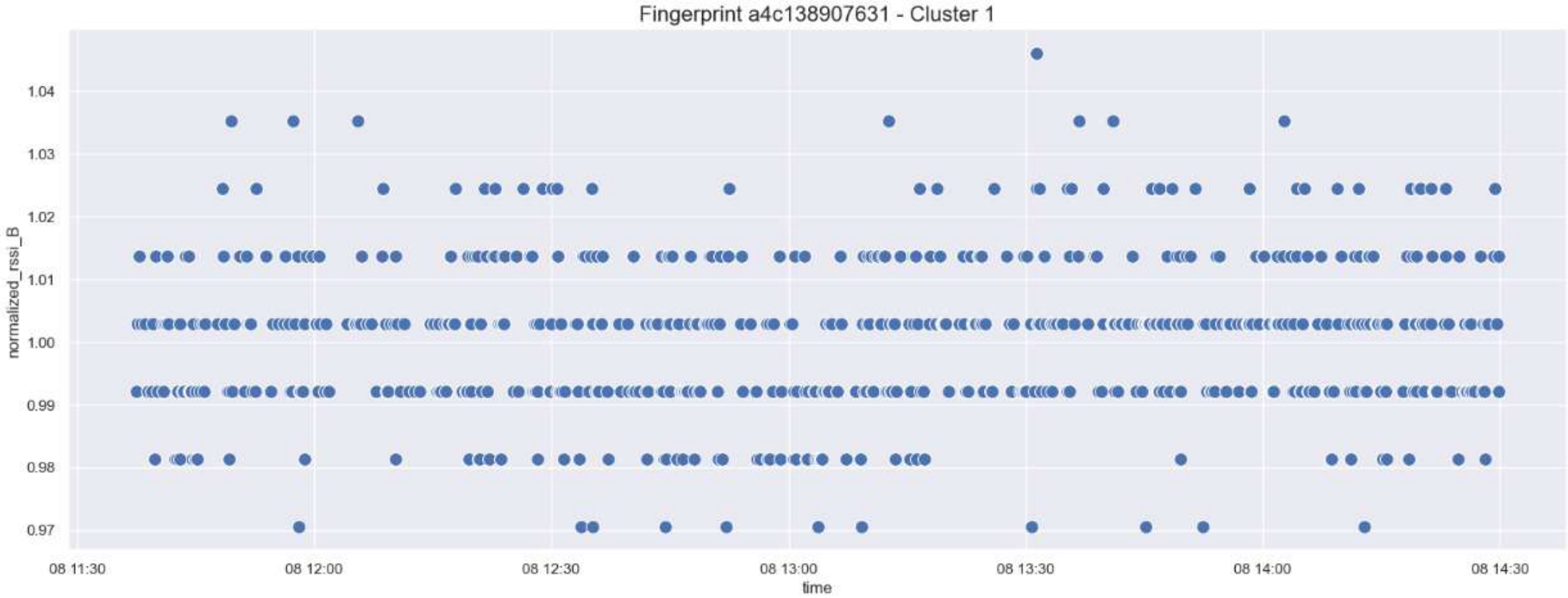
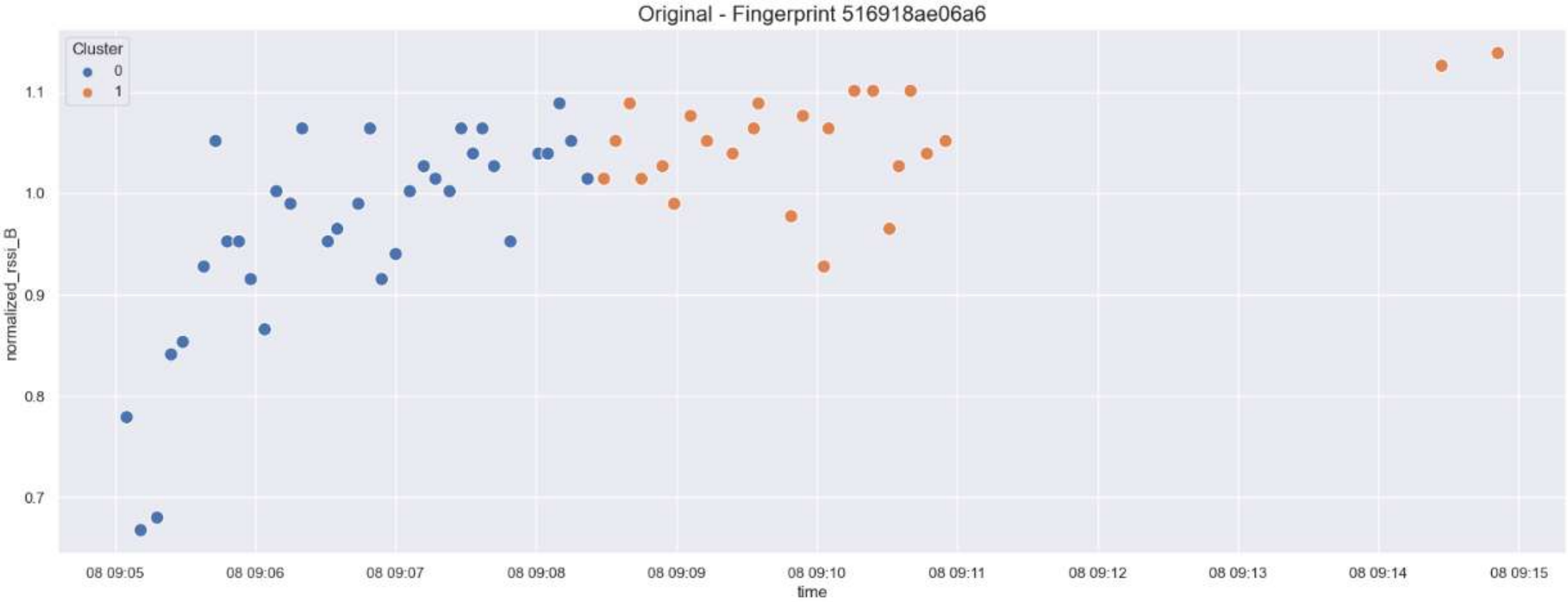

Fingerprint c0288d8d1545 - Cluster 0
normalized_rssi_B
1.04
1.03
1.02
1.01
1.00
0.99
0.98
0.97
08 12:45
08 13:00
08 13:15
08 13:30
08 13:45
08 14:00
08 14:15
08 14:30
08 14:45
08 15:00
time

```
df_1=df_2
# Create the time column for df1
df_1['time'] = pd.to_datetime(df_1['timestamp'], unit='s')
df_1['time'] = df_1['time'].dt.strftime('%H:%M:%S')
df_1['time'] = df_1['time'] - datetime.timedelta(hours=4)
df_1['time'] = df_1.time.astype(str).str.replace('0 days ', '')
df_1['time'] = pd.to_datetime(df_1['time']).dt.strftime('%H:%M:%S')

# Create the mode column : it reflects the times the signals captured the object moving
df_1['mode'] = df_1.groupby('fingerprint', sort=False).cumcount() + 1

#Sort data by fingerptint and time
df =df_1.sort_values(by=["fingerprint", "time"], ascending=[True, True])

#reset the index
df=df.reset_index()
df=df.drop(columns='index')


#Group the fingerprint based on max mode
df_mode=df.groupby('fingerprint')['mode'].agg('max')
df_mode=pd.DataFrame(df_mode)
df_mode=df_mode.reset_index()


#Group the fingerprint based on average rssi
df_rssi=df.groupby('fingerprint').mean()
df_rssi=df_rssi.drop(columns=["type","timestamp","mode"])
df_rssi=df_rssi.reset_index()


#Group the fingerprint based on standard deviation rssi
df_rssi_std=df.groupby('fingerprint').std()
df_rssi_std=df_rssi_std.reset_index()
```


Fingerprint a4c138907631 - Cluster 1
normalized_rssi_B
1.04
1.03
1.02
1.01
1.00
0.99
0.98
0.97
08 11:30
08 12:00
08 12:30
08 13:00
08 13:30
08 14:00
08 14:30
time

```
## Logistic Regression
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(random_state=32)

lr_accuracy = cross_val_score(lr,X_train,y_train.values.ravel(), cv=5, scoring ='accuracy')
lr_f1 = cross_val_score(lr,X_train,y_train.values.ravel(), cv=5, scoring ='f1')

print('lr_accuracy: ' +str(lr_accuracy))

print('lr_accuracy_avg: ' + str(lr_accuracy.mean()))

## Decision Tree
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier()

dt_accuracy = cross_val_score(dt,X_train,y_train.values.ravel(), cv=5, scoring ='accuracy')
dt_f1 = cross_val_score(dt,X_train,y_train.values.ravel(), cv=5, scoring ='f1')


print('dt_accuracy: ' +str(dt_accuracy))

print('dt_accuracy_avg: ' + str(dt_accuracy.mean()))

## SVM - Requires feature scaling (more on features scaling in this notebook: )
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.pipeline import make_pipeline, Pipeline #creates chained events

svc = Pipeline([('scale',StandardScaler()), ('svc',SVC())])

svc_accuracy = cross_val_score(svc,X_train,y_train.values.ravel(), cv=5, scoring ='accuracy')
svc_f1 = cross_val_score(svc,X_train,y_train.values.ravel(), cv=5, scoring ='f1')
```


Original - Fingerprint 516918ae06a6
Cluster
0
1
normalized_rssi_B
1.1
1.0
0.9
0.8
0.7
08 09:05
08 09:06
08 09:07
08 09:08
08 09:09
08 09:10
08 09:11
08 09:12
08 09:13
08 09:14
08 09:15
time