In [607…

```
import pandas as pd
import os
import matplotlib.pyplot as plt
import seaborn as sns
import datetime
import numpy as np
from scipy.io import loadmat
from mpl_toolkits.mplot3d import Axes3D
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
```

In [608…

```
#Creation of dataframe

def load_data(nm,battery): # Example of input load_data('B0006.mat','B0006')
  mat = loadmat( nm)
  #print('Total data in dataset: ', len(mat[battery][0, 0]['cycle'][0]))
  counter = 0
  dataset = []
  capacity_data = []

  for i in range(len(mat[battery][0, 0]['cycle'][0])):
    row = mat[battery][0, 0]['cycle'][0, i]
    if row['type'][0] == 'discharge' :
      ambient_temperature = row['ambient_temperature'][0][0]
      date_time = datetime.datetime(int(row['time'][0][0]),
                               int(row['time'][0][1]),
                               int(row['time'][0][2]),
                               int(row['time'][0][3]),
                               int(row['time'][0][4])) + datetime.timedelta(seconds=int(row['time'][0][5]))
      data = row['data']
      capacity = data[0][0]['Capacity'][0][0]
      for j in range(len(data[0][0]['Voltage_measured'][0])):
        voltage_measured = data[0][0]['Voltage_measured'][0][j]
        current_measured = data[0][0]['Current_measured'][0][j]
        temperature_measured = data[0][0]['Temperature_measured'][0][j]
        current_load = data[0][0]['Current_load'][0][j]
        voltage_load = data[0][0]['Voltage_load'][0][j]
        time = data[0][0]['Time'][0][j]
        dataset.append([counter + 1, ambient_temperature, date_time, capacity,
                        voltage_measured, current_measured,
                        temperature_measured, current_load,
                        voltage_load, time])
      capacity_data.append([counter + 1, ambient_temperature, date_time, capacity])
      counter = counter + 1
  print(dataset[1])
  return [pd.DataFrame(data=dataset,
                       columns=['cycle', 'ambient_temperature', 'datetime',
                                'capacity', 'voltage_measured',
                                'current_measured', 'temperature_measured',
                                'current_load', 'voltage_load', 'time']),
          pd.DataFrame(data=capacity_data,
                       columns=['cycle', 'ambient_temperature', 'datetime',
                                'capacity'])]
```

In [609…

```
df1 = load_data('B0005.mat','B0005')
```

```
[1, 24, datetime.datetime(2008, 4, 2, 15, 25, 41), 1.8564874208181574, 4.190749067776103, -0.0014780055516425076,
24.325993424022467, -0.0006, 4.206, 16.781]
```

In [610…

```
type(df1)
```

Out[610…

```
list
```

In [611…

```
len(df1)
```

Out[611…

```
2
```

In [612…

```
df1[0].head()
```

Out[612…

	cycle	ambient_temperature	datetime	capacity	voltage_measured	current_measured	temperature_measured	current_load	voltage_load	ti
			2008-04-							

0	1	24	02 15:25:41	1.856487	4.191492	-0.004902	24.330034	-0.0006	0.000	0.(
1	1	24	2008-04-02 15:25:41	1.856487	4.190749	-0.001478	24.325993	-0.0006	4.206	16.:
2	1	24	2008-04-02 15:25:41	1.856487	3.974871	-2.012528	24.389085	-1.9982	3.062	35.:
3	1	24	2008-04-02 15:25:41	1.856487	3.951717	-2.013979	24.544752	-1.9982	3.030	53.:
4	1	24	2008-04-02 15:25:41	1.856487	3.934352	-2.011144	24.731385	-1.9982	3.011	71.!

In [613…

```
df1[1].head()
```

Out[613…

	cycle	ambient_temperature	datetime	capacity
0	1	24	2008-04-02 15:25:41	1.856487
1	2	24	2008-04-02 19:43:48	1.846327
2	3	24	2008-04-03 00:01:06	1.835349
3	4	24	2008-04-03 04:16:37	1.835263
4	5	24	2008-04-03 08:33:25	1.834646

In [614…

```
#Adding flag for Battery 1
df1[0]['flag'] = 1
```

In [ ]:

In [615…

```
df2 = load_data('B0006.mat','B0006')
```

```
[1, 24, datetime.datetime(2008, 4, 2, 15, 25, 41), 2.035337591005598, 4.179823027658306, 0.00043376246575117864,
24.27707330832413, -0.0006, 4.195, 16.781]
```

In [616…

```
#Adding flag for Battery 2
df2[0]['flag'] = 2
```

In [617…

```
df3 = load_data('B0007.mat','B0007')
```

```
[1, 24, datetime.datetime(2008, 4, 2, 15, 25, 41), 1.89105229539079, 4.199497433806136, -0.0021394269898071224, 2
3.92407356409745, -0.0004, 4.215, 16.781]
```

In [618…

```
#Adding flag for battery 3
df3[0]['flag'] = 3
```

In [619…

```
df4 = load_data('B0018.mat','B0018')
```

```
[1, 24, datetime.datetime(2008, 7, 7, 15, 15, 28), 1.8550045207910817, 4.188195942647034, 0.001459080605681204, 2
3.82880715958107, 0.0006, 4.203, 9.421999999999997]
```

In [620…

```
#Adding flag for battery 4
df4[0]['flag'] = 4
```

In [621…

```
frames = [df1[0], df2[0], df3[0], df4[0]]
```

In [622…

```
df = pd.concat(frames)
```

In [623…

```
#Battery 1, 2 and 3 have most values provided
```

```
df.groupby('flag').count()
```

Out[623…

flag	cycle	ambient_temperature	datetime	capacity	voltage_measured	current_measured	temperature_measured	current_load	voltage_load
1	50285	50285	50285	50285	50285	50285	50285	50285	50285
2	50285	50285	50285	50285	50285	50285	50285	50285	50285
3	50285	50285	50285	50285	50285	50285	50285	50285	50285
4	34866	34866	34866	34866	34866	34866	34866	34866	34866

In [624…

```
df.head()
```

Out[624…

	cycle	ambient_temperature	datetime	capacity	voltage_measured	current_measured	temperature_measured	current_load	voltage_load	ti
0	1	24	2008-04-02 15:25:41	1.856487	4.191492	-0.004902	24.330034	-0.0006	0.000	0.(
1	1	24	2008-04-02 15:25:41	1.856487	4.190749	-0.001478	24.325993	-0.0006	4.206	16.'
2	1	24	2008-04-02 15:25:41	1.856487	3.974871	-2.012528	24.389085	-1.9982	3.062	35.'
3	1	24	2008-04-02 15:25:41	1.856487	3.951717	-2.013979	24.544752	-1.9982	3.030	53.'
4	1	24	2008-04-02 15:25:41	1.856487	3.934352	-2.011144	24.731385	-1.9982	3.011	71.!

In [625…

```
#11 columns and 185721 rows..
df.shape
```

Out[625…

```
(185721, 11)
```

In [626…

```
df.info()
#data type of all attributes
#There are no null values
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 185721 entries, 0 to 34865
Data columns (total 11 columns):
 #   Column                Non-Null Count   Dtype
---  ------                --------------   -----
 0   cycle                 185721 non-null  int64
 1   ambient_temperature   185721 non-null  int8
 2   datetime              185721 non-null  datetime64[ns]
 3   capacity              185721 non-null  float64
 4   voltage_measured      185721 non-null  float64
 5   current_measured      185721 non-null  float64
 6   temperature_measured  185721 non-null  float64
 7   current_load          185721 non-null  float64
 8   voltage_load          185721 non-null  float64
 9   time                  185721 non-null  float64
 10  flag                  185721 non-null  int64
dtypes: datetime64[ns](1), float64(7), int64(2), int8(1)
memory usage: 15.8 MB
```

In [627…

```
#Statistical summary
df.describe()
```

Out[627…

	cycle	ambient_temperature	capacity	voltage_measured	current_measured	temperature_measured	current_load	voltag
count	185721.000000	185721.0	185721.000000	185721.000000	185721.000000	185721.000000	185721.000000	185721.
mean	82.838758	24.0	1.574863	3.497219	-1.832569	32.378997	1.465434	2.
std	45.692247	0.0	0.190633	0.251691	0.561405	4.027737	1.226874	0.
min	1.000000	24.0	1.153818	1.737030	-2.029098	22.350256	-2.000000	0.
25%	45.000000	24.0	1.426025	3.377653	-2.011418	29.570621	1.998200	2.

50%	81.000000	24.0	1.559634	3.500859	-2.009015	32.355737	1.998800	2.
75%	120.000000	24.0	1.741850	3.655751	-1.989974	35.420677	1.999000	2.
max	168.000000	24.0	2.035338	4.233325	0.014306	42.332522	2.000000	4.

In [628…

```
df.nunique()
```

Out[628…

```
cycle                      168
ambient_temperature          1
datetime                   300
capacity                   636
voltage_measured        185721
current_measured        185721
temperature_measured    185721
current_load                21
voltage_load              1835
time                     62016
flag                         4
dtype: int64
```

In [629…

```
# Ambient temperature has just one value so we can delete it. It cannot have any outliers

# Flag is a categorical vairable added by me

# Voltage measured, current measured and temperature measured have unique values for each row since they are
# measured till 6 decemal places
```

In [630…

```
#We drop ambient temperature
df.drop('ambient_temperature', inplace = True, axis =1)
```

In [631…

```
#No null values
df.isna().value_counts()
```

Out[631…

```
cycle  datetime  capacity  voltage_measured  current_measured  temperature_measured  current_load  voltage_load
time   flag
False  False      False     False             False             False                 False         False
False  False     185721
dtype: int64
```

In [632…

```
df['cycle'].value_counts()
```

Out[632…

```
31    1413
32    1393
33    1385
34    1381
35    1372
      ...
27     857
28     855
29     853
30     848
43     818
Name: cycle, Length: 168, dtype: int64
```

In [633…

```
df['capacity'].value_counts()
```

Out[633…

```
1.851803    371
1.883468    371
1.924776    371
1.855005    366
1.882232    365
           ...
1.802778    182
1.804077    182
1.767617    179
1.754677    179
1.813204    179
Name: capacity, Length: 636, dtype: int64
```

In [634…

```
df['current_load'].value_counts()
```

```
df['current_load'].value_counts()
```

Out[634…

```
 2.0000    42650
 1.9982    30516
 1.9990    29675
 1.9988    18817
 1.9986    17968
 0.0006    12603
 1.9980     8624
-2.0000     5649
-1.9982     4963
-1.9990     4551
 1.9992     3229
 0.0008     2317
 1.9984     1477
-1.9992      805
-0.0006      585
-1.9984      436
-1.9988      390
 0.0004      370
-0.0004       46
-0.0008       34
-1.9980       16
Name: current_load, dtype: int64
```

In [635…

```
df['voltage_load'].value_counts()
```

Out[635…

```
0.000    13626
0.001     1693
2.536      552
2.542      544
2.540      534
         ...
1.469        1
1.277        1
1.592        1
1.441        1
1.515        1
Name: voltage_load, Length: 1835, dtype: int64
```

In [636…

```
df['time'].value_counts()
```

Out[636…

```
0.000       636
9.375        63
9.391        55
263.078      42
9.360        39
           ...
2850.563      1
673.360       1
650.907       1
639.703       1
2728.750      1
Name: time, Length: 62016, dtype: int64
```

In [637…

```
df['current_measured'].value_counts()
```

Out[637…

```
-0.004902    1
-1.991226    1
-1.989353    1
-1.988887    1
-1.989830    1
            ..
-2.011567    1
-2.008766    1
-2.008298    1
-2.009321    1
-0.001940    1
Name: current_measured, Length: 185721, dtype: int64
```

In [638…

```
df['current_load'].value_counts()
```

Out[638…

```
 2.0000    42650
 1.9982    30516
 1.9990    29675
 1.9988    18817
```

```
 1.9986    17968
 0.0006    12603
 1.9980     8624
-2.0000     5649
-1.9982     4963
-1.9990     4551
 1.9992     3229
 0.0008     2317
 1.9984     1477
-1.9992      805
-0.0006      585
-1.9984      436
-1.9988      390
 0.0004      370
-0.0004       46
-0.0008       34
-1.9980       16
Name: current_load, dtype: int64
```

In [639…

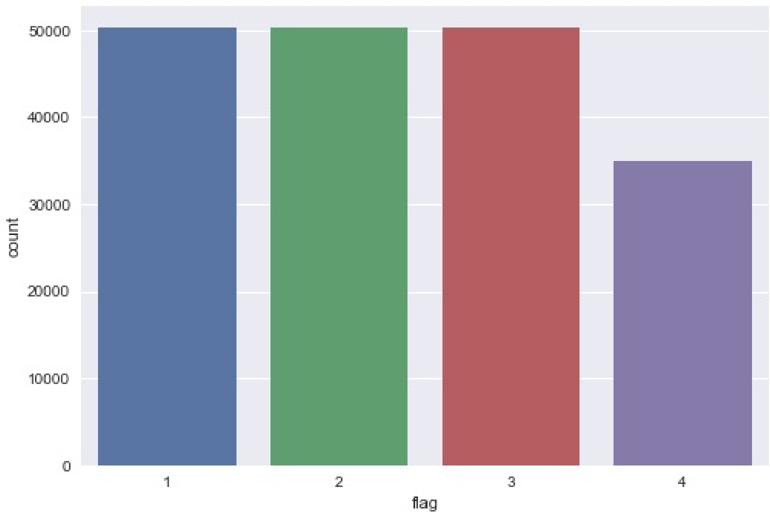
```
#Univariate Analysis

#Categorical vairable is flag
```

In [640…

```
sns.countplot(x="flag", data=df)
```

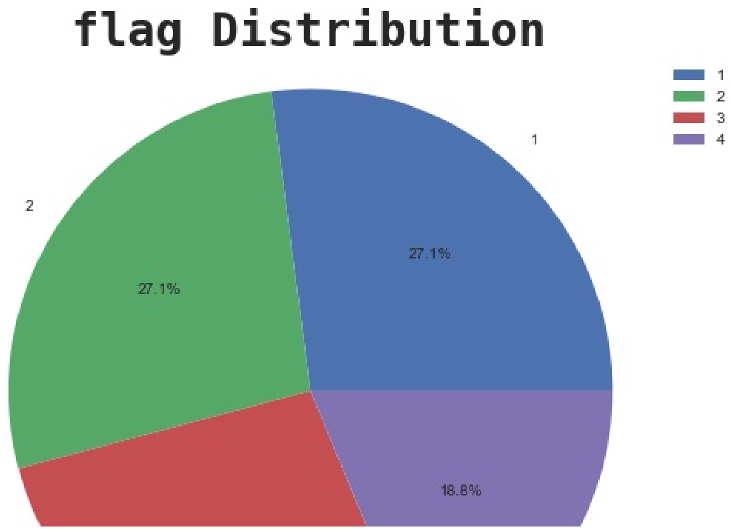
Out[640…

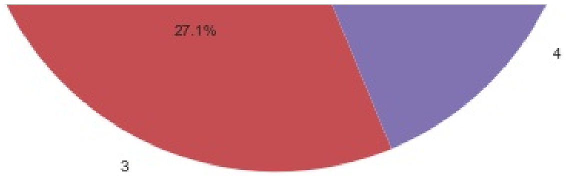
```
<AxesSubplot:xlabel='flag', ylabel='count'>
```



In [641…

```
#Fuel cell 1,2,3 make up 27.% of the data and 4 makes up the remainder 18.8%
ssn = df['flag'].value_counts()
plt.style.use('seaborn')
plt.figure(figsize = (10, 8))
plt.pie(ssn.values, labels = ssn.index, autopct = '%1.1f%%')
plt.title('flag Distribution', fontdict = {'fontname' : 'Monospace','fontsize' : 30, 'fontweight' : 'bold'})
plt.legend()
plt.axis('equal')
plt.show()
```





In [642…

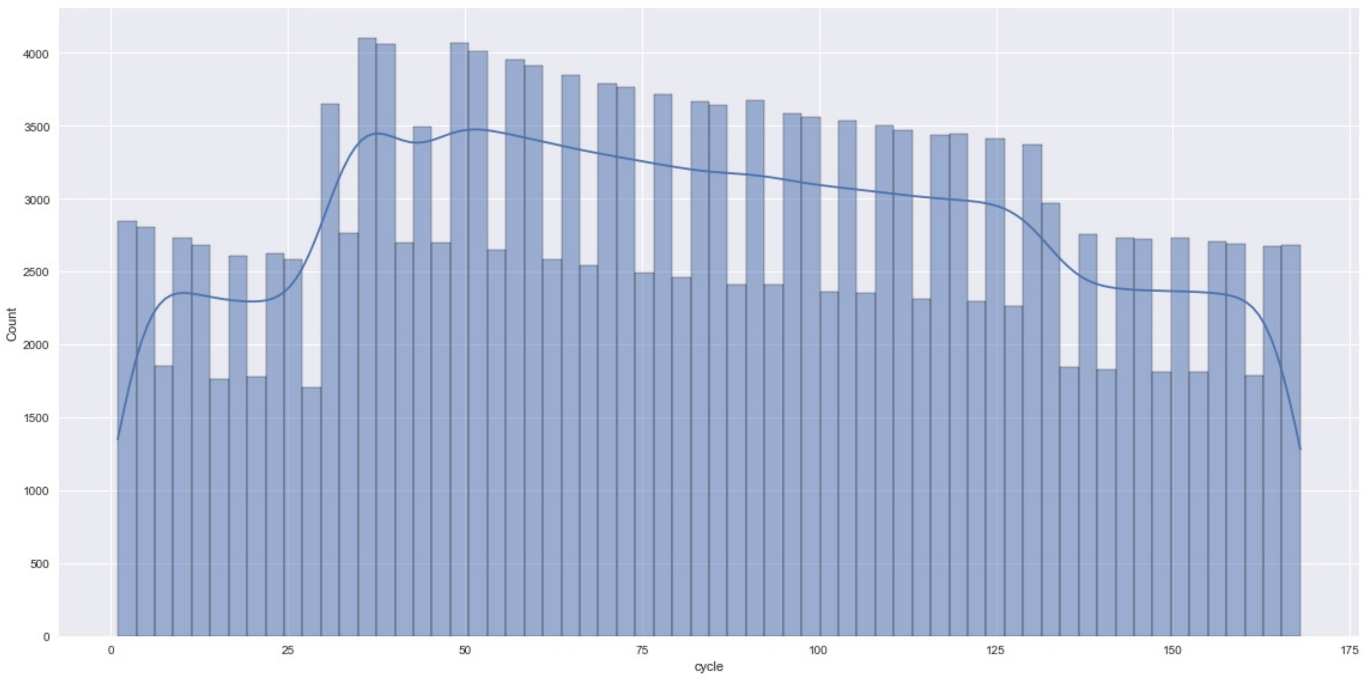
```
#Continous variables are cycle, datetime, capacity,voltage_measured, current_measured, temperature_measured
# current_laod, voltage_load, time
```

In [643…

```
#Most of the cycle values lie between 25 and 125 both included
sns.displot( df['cycle'], kde = True, height = 8, aspect = 2)
```

Out[643…

```
<seaborn.axisgrid.FacetGrid at 0x22b61c39370>
```

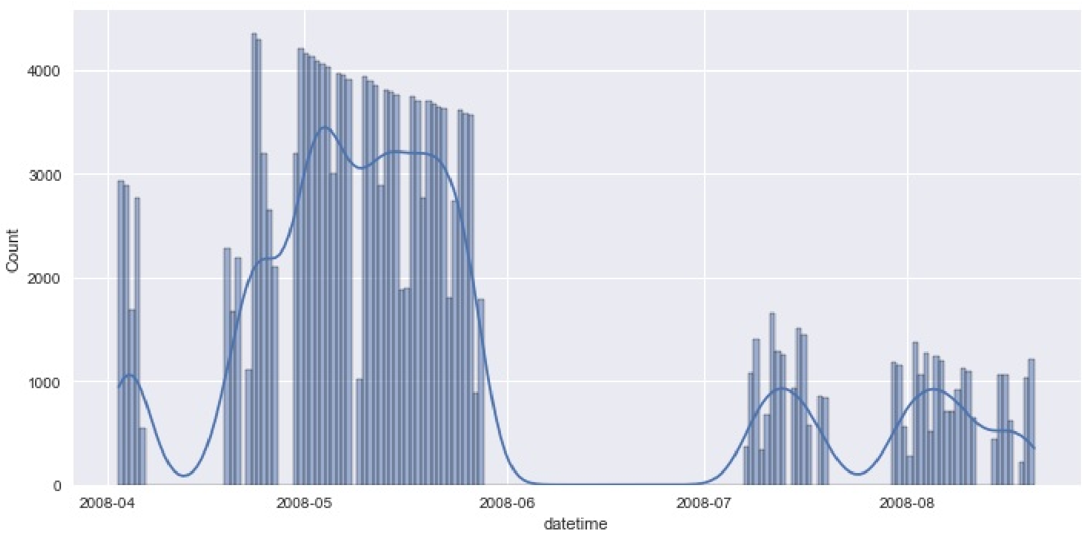


In [644…

```
#Not informative at all
#Most values lie in 2008 . Month no 4,5,7 and 8 have most of the entries
sns.displot( df['datetime'], kde = True, height = 5, aspect=2)
```

Out[644…

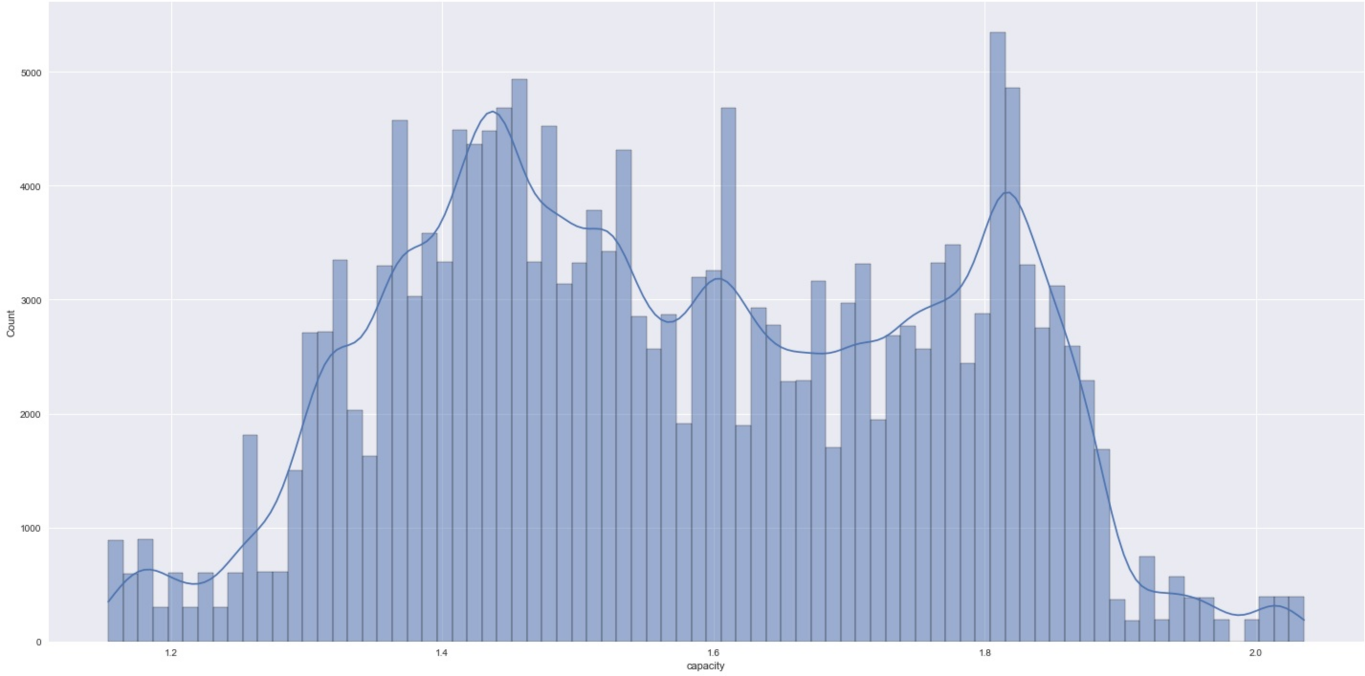
```
<seaborn.axisgrid.FacetGrid at 0x22b78f127f0>
```



In [645…

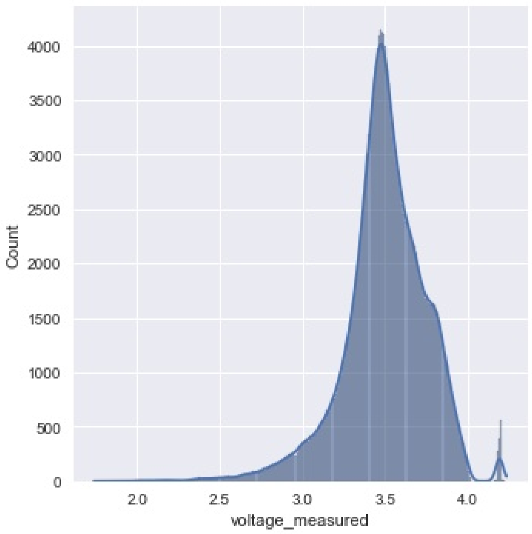
```
#Capacity : Mostly lies between 1.4 and 1.8
sns.displot( df['capacity'], kde = True, height = 10, aspect =2)
```

Out[645… <seaborn.axisgrid.FacetGrid at 0x22bc8d58850>



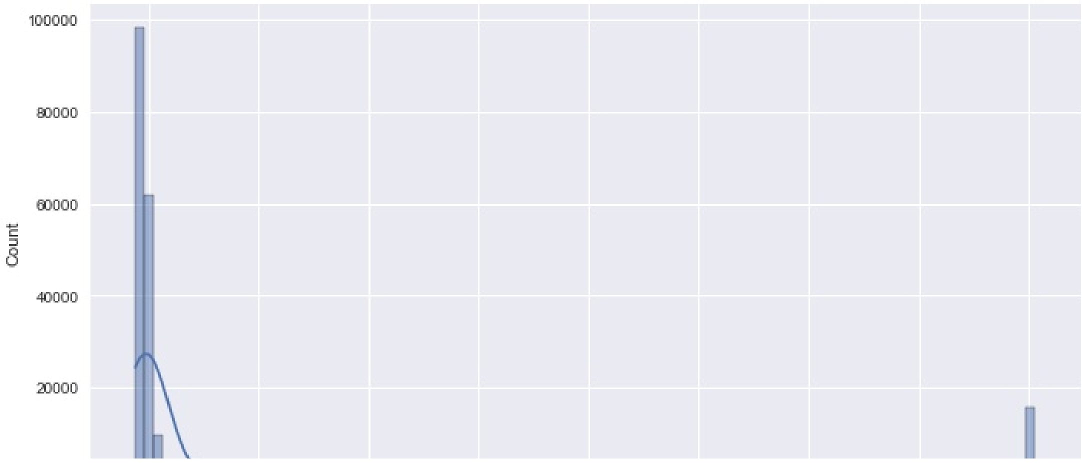
In [646…
```
#Voltage : Most values lie between 3 and 4. Peak around 3.5
sns.displot( df['voltage_measured'], kde = True) #height = 10, aspect =2)
```

Out[646… <seaborn.axisgrid.FacetGrid at 0x22bc4599910>



In [647…
```
#Current measured : Values are heavily clustered around 0 and -2
sns.displot( df['current_measured'],bins = 100, kde = True, height = 5, aspect =2)
```

Out[647… <seaborn.axisgrid.FacetGrid at 0x22bc92c9a00>

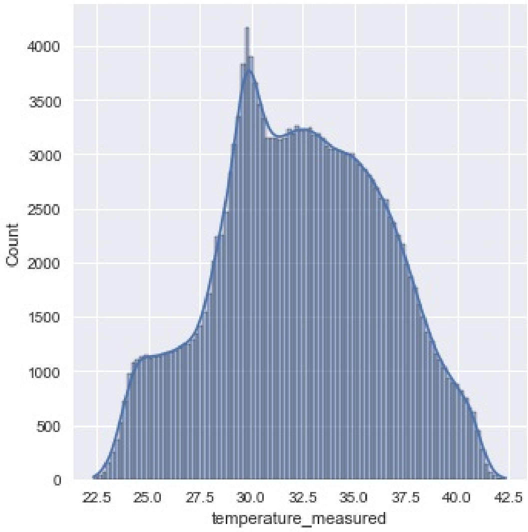


In [648…

```
#Temperature measured :
sns.displot( df['temperature_measured'], kde = True)
```

Out[648…

```
<seaborn.axisgrid.FacetGrid at 0x22bc92c7a30>
```

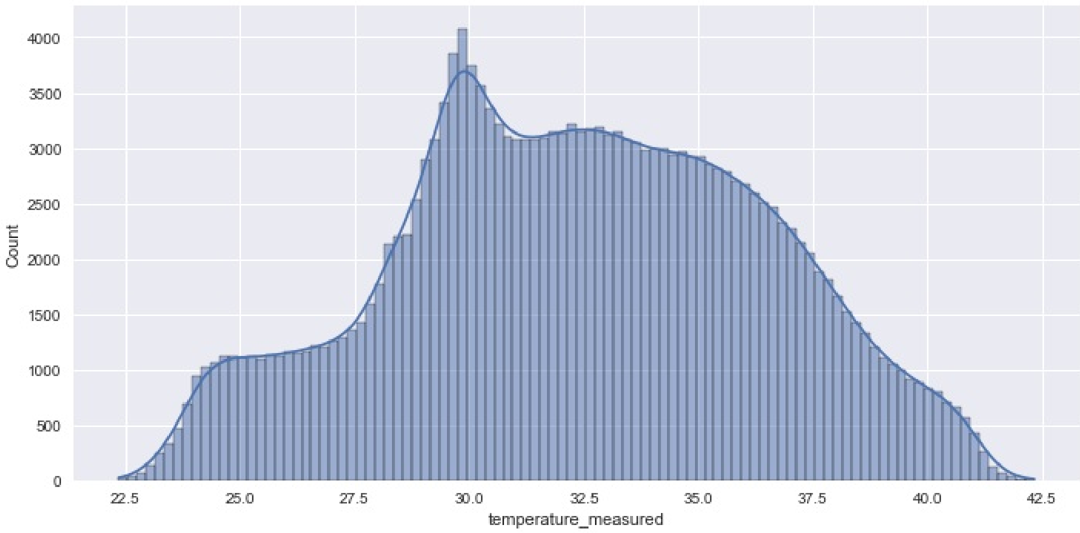


In [649…

```
#
sns.displot( df['temperature_measured'],bins = 100, kde = True, height = 5, aspect =2)
```

Out[649…

```
<seaborn.axisgrid.FacetGrid at 0x22a4880ad90>
```

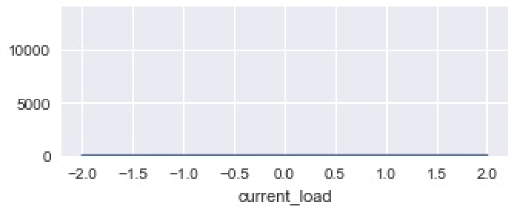


In [650…

```
#displot doen't reveal much for Current load
sns.displot( df['current_load'],kde = True)
```

Out[650…

```
<seaborn.axisgrid.FacetGrid at 0x22a4880aeb0>
```

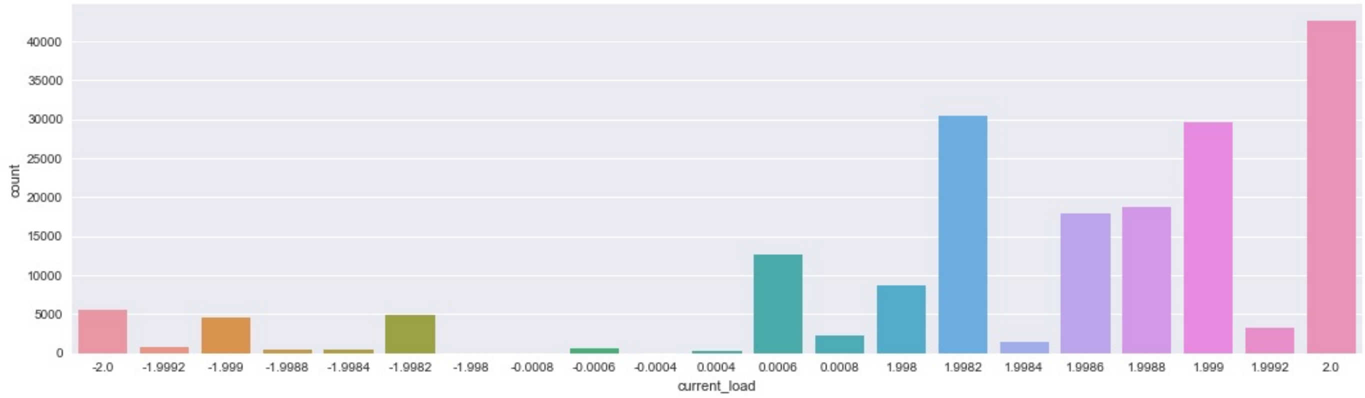


In [651…

```
#Current load : Most values heavily clustered around 2, then -2 and then 0. (Decreasing number of values acorss 2
#,-2,0)
fig, ax = plt.subplots(figsize=(18, 5))
sns.countplot(x= 'current_load', data = df, ax= ax)
```

Out[651…

```
<AxesSubplot:xlabel='current_load', ylabel='count'>
```

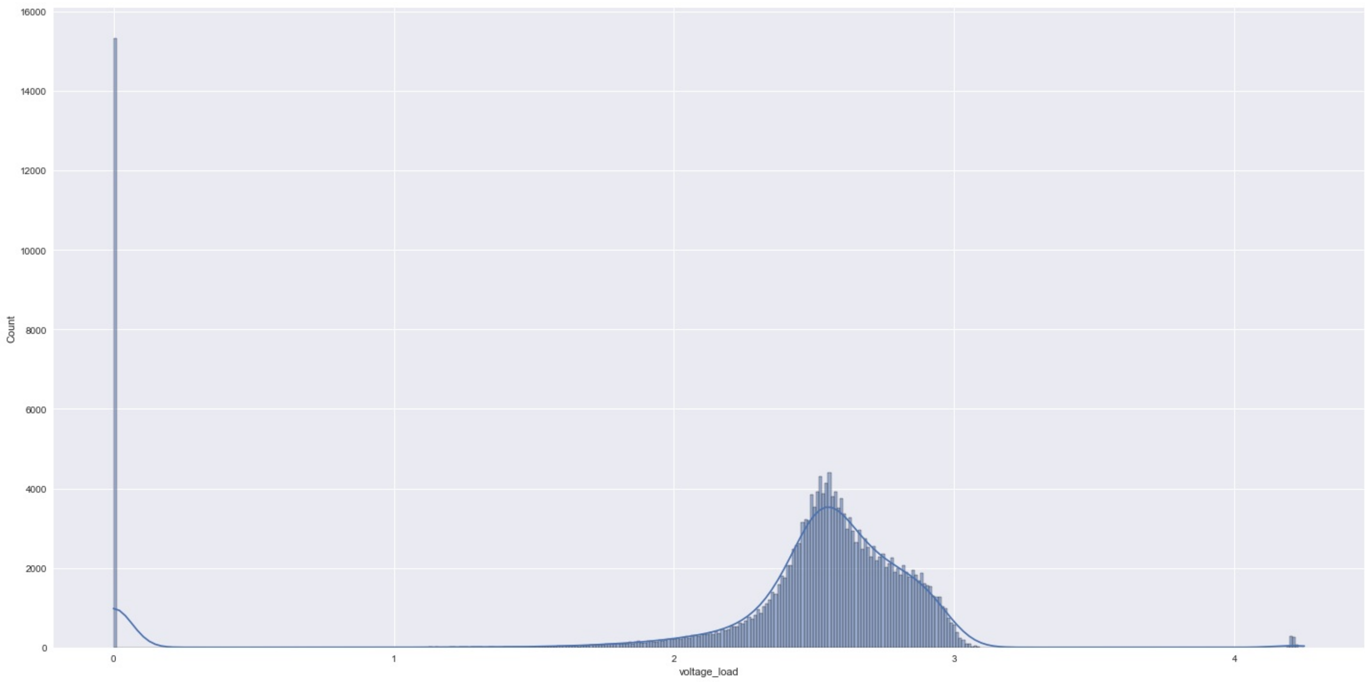


In [652…

```
sns.displot( df['voltage_load'],bins = 400,kde = True, height = 10, aspect = 2)
```

Out[652…

```
<seaborn.axisgrid.FacetGrid at 0x22c0529e490>
```

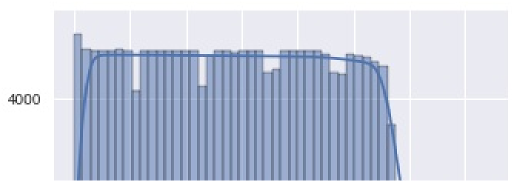


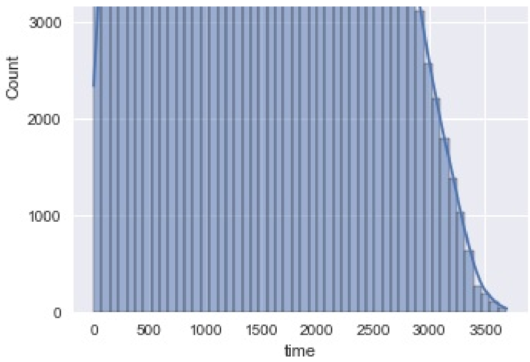
In [653…

```
#Time : Seem to follow a uniform distribution 0 and 3000 and then decreases sharply till 3500
sns.displot(df['time'],bins = 50,kde = True)
```

Out[653…

```
<seaborn.axisgrid.FacetGrid at 0x22bfcef3760>
```





In [ ]:

In [654…

```
df.index
```

Out[654…

```
Int64Index([     0,      1,      2,      3,      4,      5,      6,      7,      8,
                 9,
            ...
            34856, 34857, 34858, 34859, 34860, 34861, 34862, 34863, 34864,
            34865],
           dtype='int64', length=185721)
```

In [655…

```
#Changing the index of the dataframe
#Since we merged 3 dataframes the index is faulty

index = []
for i in range(0,185721):
    index.append(i)
print(index[0],index[185720])


df = df.set_index(pd.Index(index))
```

```
0 185720
```

In [656…

```
df.head()
```

Out[656…

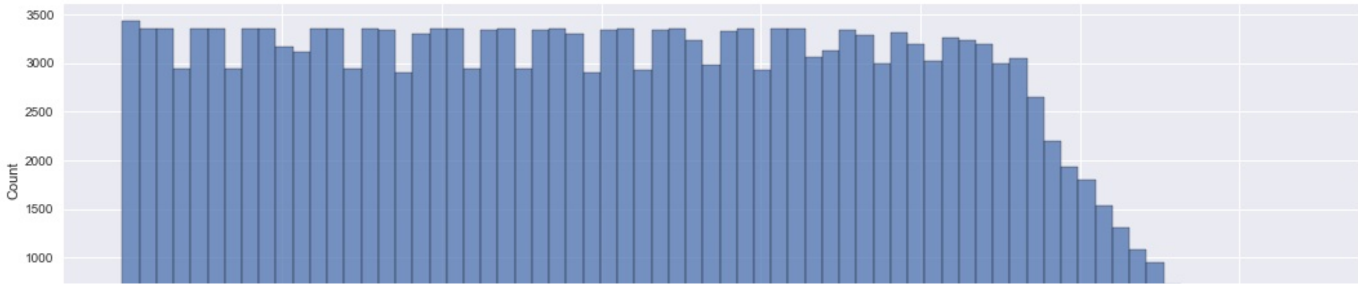
	cycle	datetime	capacity	voltage_measured	current_measured	temperature_measured	current_load	voltage_load	time	flag
0	1	2008-04-02 15:25:41	1.856487	4.191492	-0.004902	24.330034	-0.0006	0.000	0.000	1
1	1	2008-04-02 15:25:41	1.856487	4.190749	-0.001478	24.325993	-0.0006	4.206	16.781	1
2	1	2008-04-02 15:25:41	1.856487	3.974871	-2.012528	24.389085	-1.9982	3.062	35.703	1
3	1	2008-04-02 15:25:41	1.856487	3.951717	-2.013979	24.544752	-1.9982	3.030	53.781	1
4	1	2008-04-02 15:25:41	1.856487	3.934352	-2.011144	24.731385	-1.9982	3.011	71.922	1

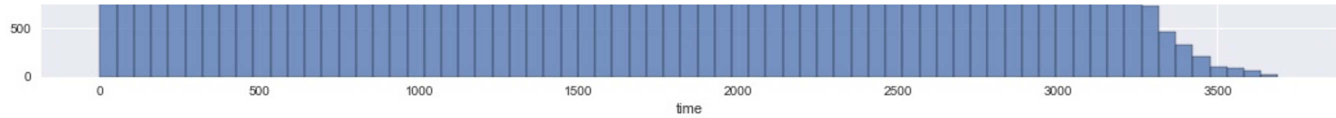
In [657…

```
fig, ax = plt.subplots(figsize=(18, 5))
sns.histplot(data = df, x = 'time', ax=ax)
```

Out[657…

```
<AxesSubplot:xlabel='time', ylabel='Count'>
```



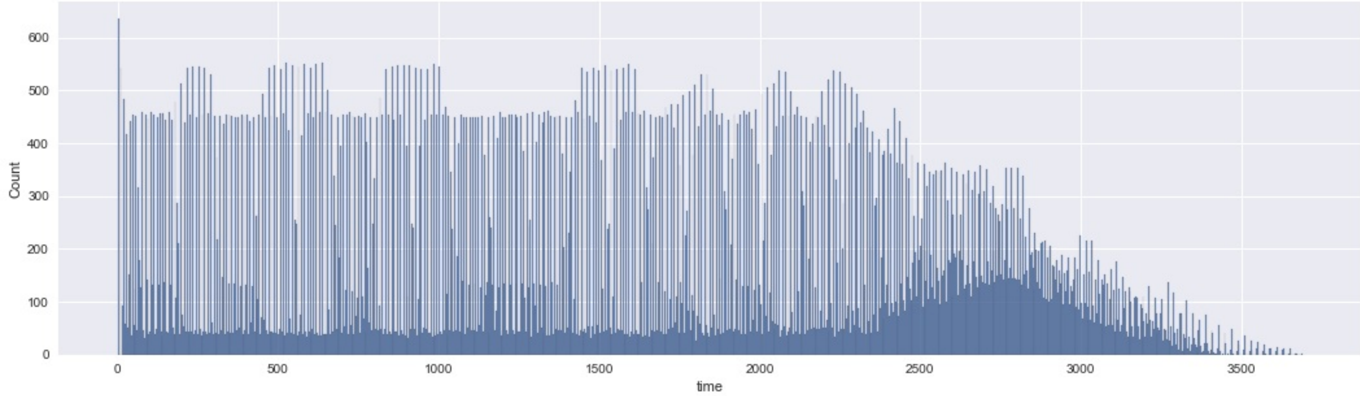


In [658…

```
fig, ax = plt.subplots(figsize=(18, 5))
sns.histplot(data = df, x = 'time', bins = 1000, ax=ax)
```

Out[658…

```
<AxesSubplot:xlabel='time', ylabel='Count'>
```

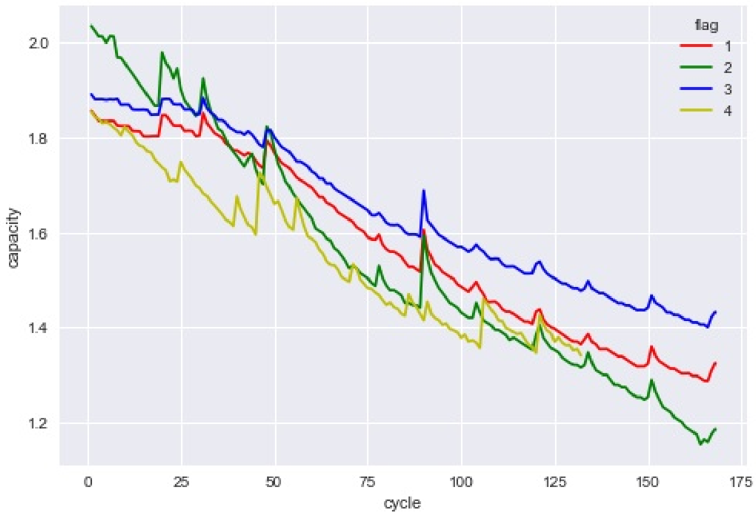


In [659…

```
#Bivariate analysis

sns.lineplot(x='cycle',y='capacity', data=df, palette = ['r', 'g', 'b', 'y'], hue='flag')
```

Out[659…

```
<AxesSubplot:xlabel='cycle', ylabel='capacity'>
```

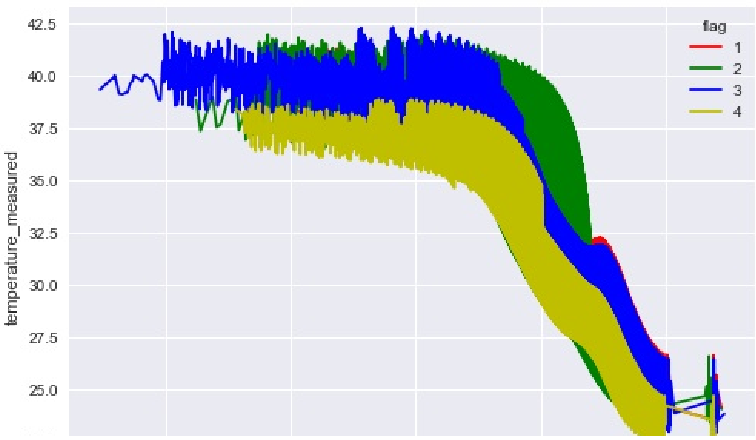


In [660…

```
sns.lineplot(x='voltage_measured',y='temperature_measured', data=df, palette = ['r', 'g', 'b', 'y'], hue='flag')
```

Out[660…

```
<AxesSubplot:xlabel='voltage_measured', ylabel='temperature_measured'>
```

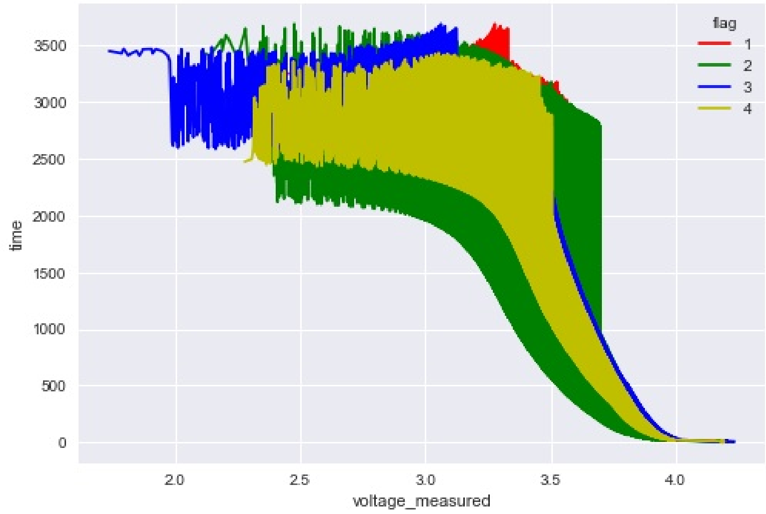


In [661…

```python
sns.lineplot(x='voltage_measured',y='time', data=df, palette = ['r', 'g', 'b', 'y'], hue='flag')
```

Out[661…

```
<AxesSubplot:xlabel='voltage_measured', ylabel='time'>
```

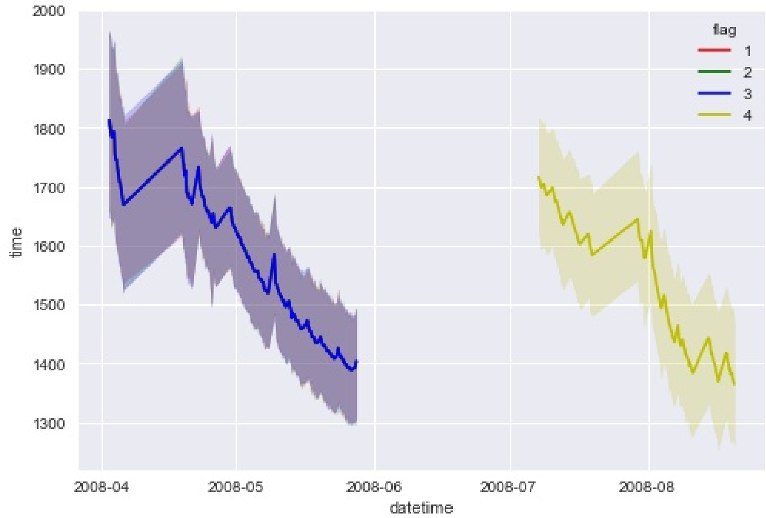


In [662…

```python
#Datetime : When we started the charging of the cycle
#Time : Time taken to complete one cycle

sns.lineplot(x='datetime',y='time', data=df, palette = ['r', 'g', 'b', 'y'], hue='flag')
```

Out[662…

```
<AxesSubplot:xlabel='datetime', ylabel='time'>
```

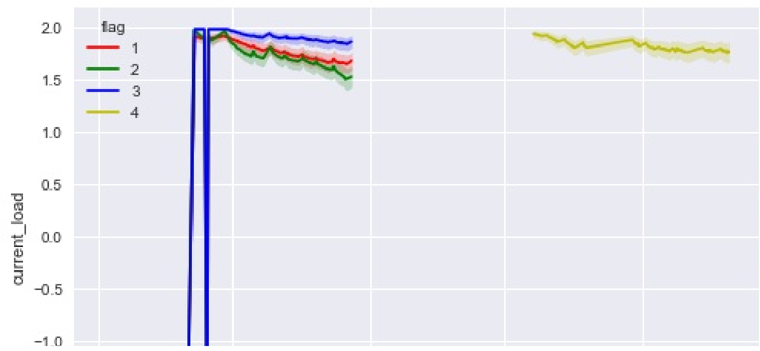


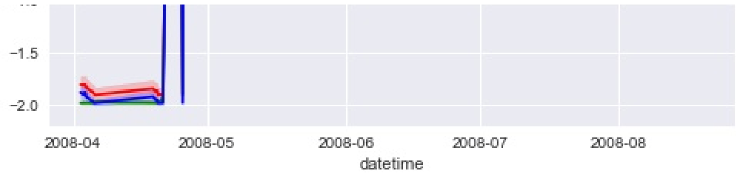
In [663…

```python
sns.lineplot(x='datetime',y='current_load', data=df, palette = ['r', 'g', 'b', 'y'], hue='flag')
```

Out[663…

```
<AxesSubplot:xlabel='datetime', ylabel='current_load'>
```



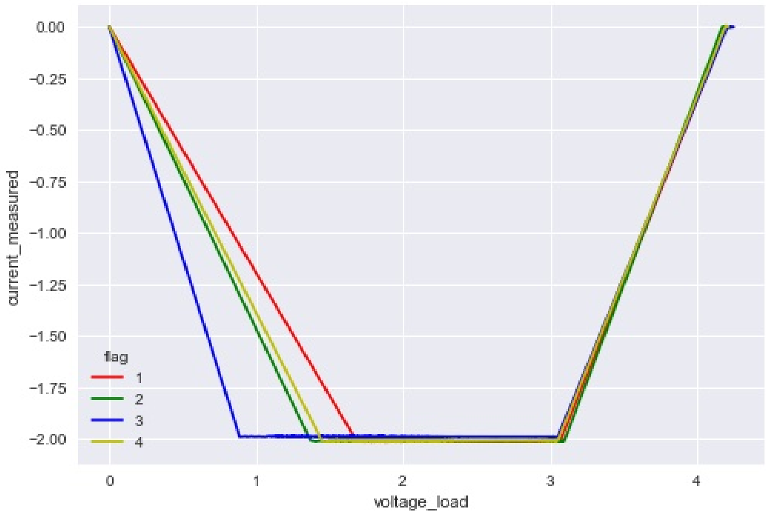


In [664…

```
sns.lineplot(x='voltage_load',y='current_measured', data=df, palette = ['r', 'g', 'b', 'y'], hue='flag')
```

Out[664…

```
<AxesSubplot:xlabel='voltage_load', ylabel='current_measured'>
```

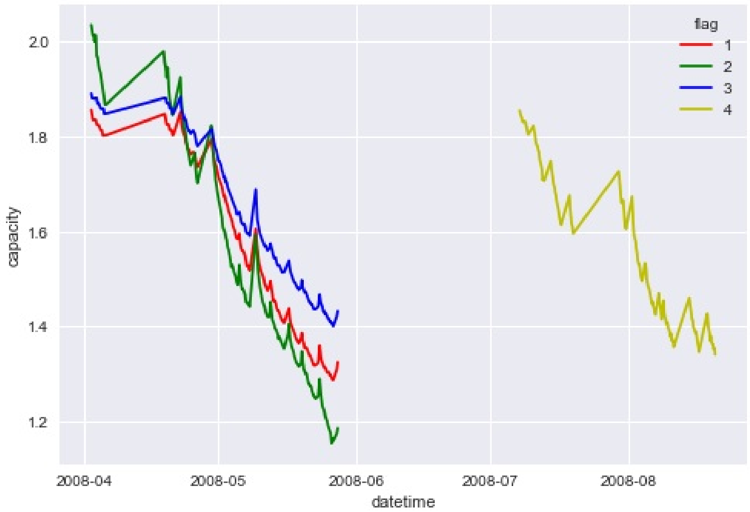


In [665…

```
sns.lineplot(x='datetime',y='capacity', data=df, palette = ['r', 'g', 'b', 'y'], hue='flag')
```

Out[665…

```
<AxesSubplot:xlabel='datetime', ylabel='capacity'>
```

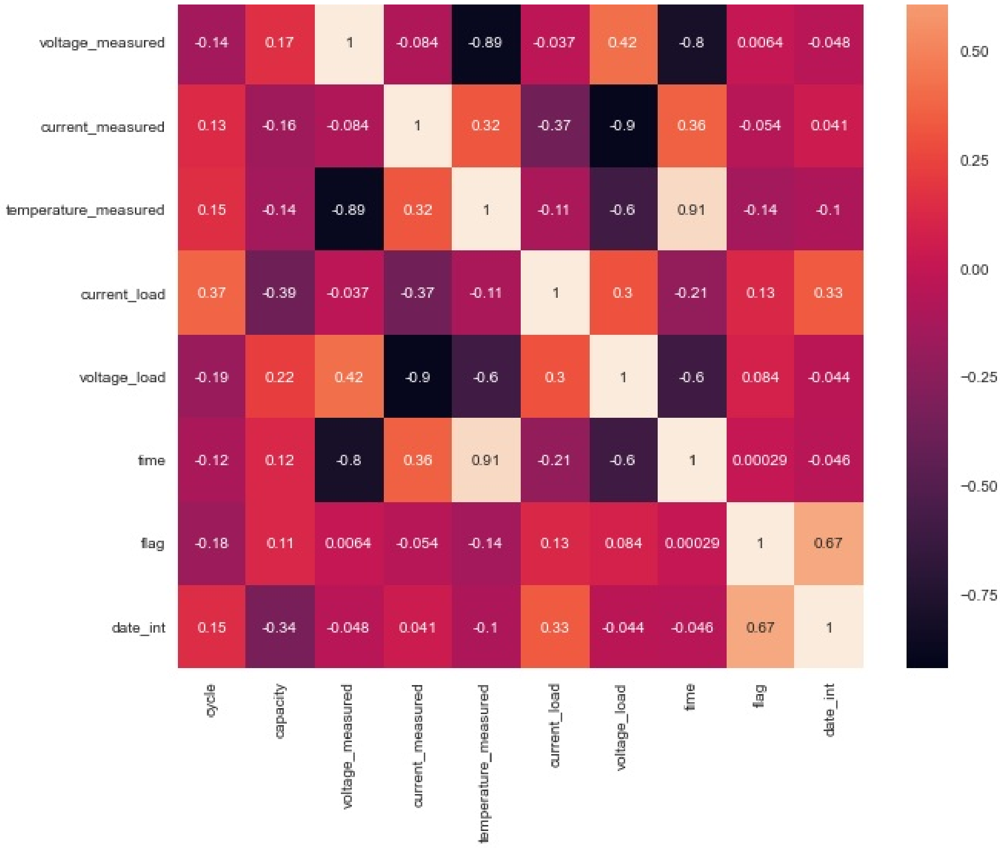


In [666…

```
#Heatmap

df['date_int'] =  df['datetime'].apply(lambda x: x.value)

#Since variables aren't normally distributed we do not consider Pearson correlation
fig, ax = plt.subplots(figsize=(10, 10))
Var_Corr = df.corr(method = 'pearson')
sns.heatmap(Var_Corr, xticklabels=Var_Corr.columns, yticklabels=Var_Corr.columns, annot=True, ax=ax)
```

Out[666…

```
<AxesSubplot:>
```

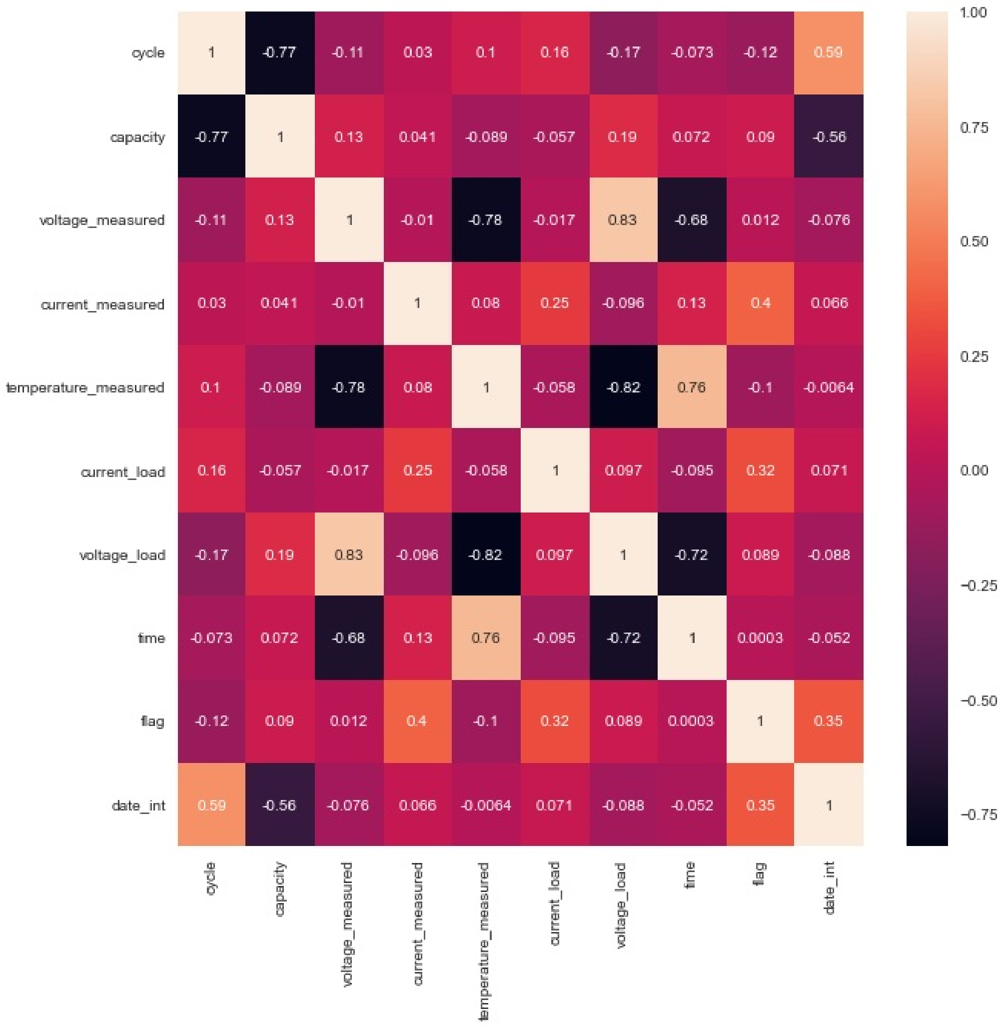


In [667…

```
fig, ax = plt.subplots(figsize=(10, 10))
Var_Corr = df.corr(method = 'kendall')
sns.heatmap(Var_Corr, xticklabels=Var_Corr.columns, yticklabels=Var_Corr.columns, annot=True, ax=ax)
```

Out[667…

```
<AxesSubplot:>
```

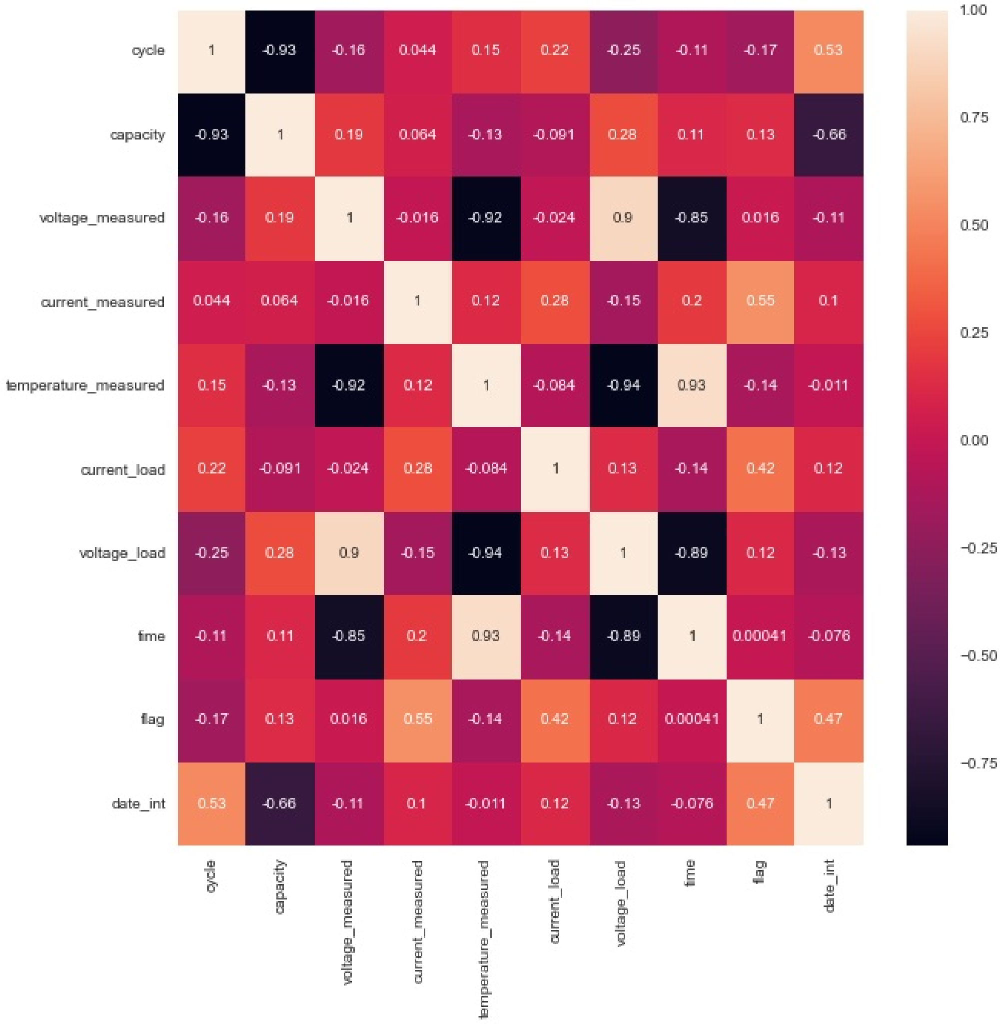


In [668…

```
fig, ax = plt.subplots(figsize=(10, 10))
Var_Corr = df.corr(method = 'spearman')
sns.heatmap(Var_Corr, xticklabels=Var_Corr.columns, yticklabels=Var_Corr.columns, annot=True, ax=ax)
```

Out[668…

```
<AxesSubplot:>
```

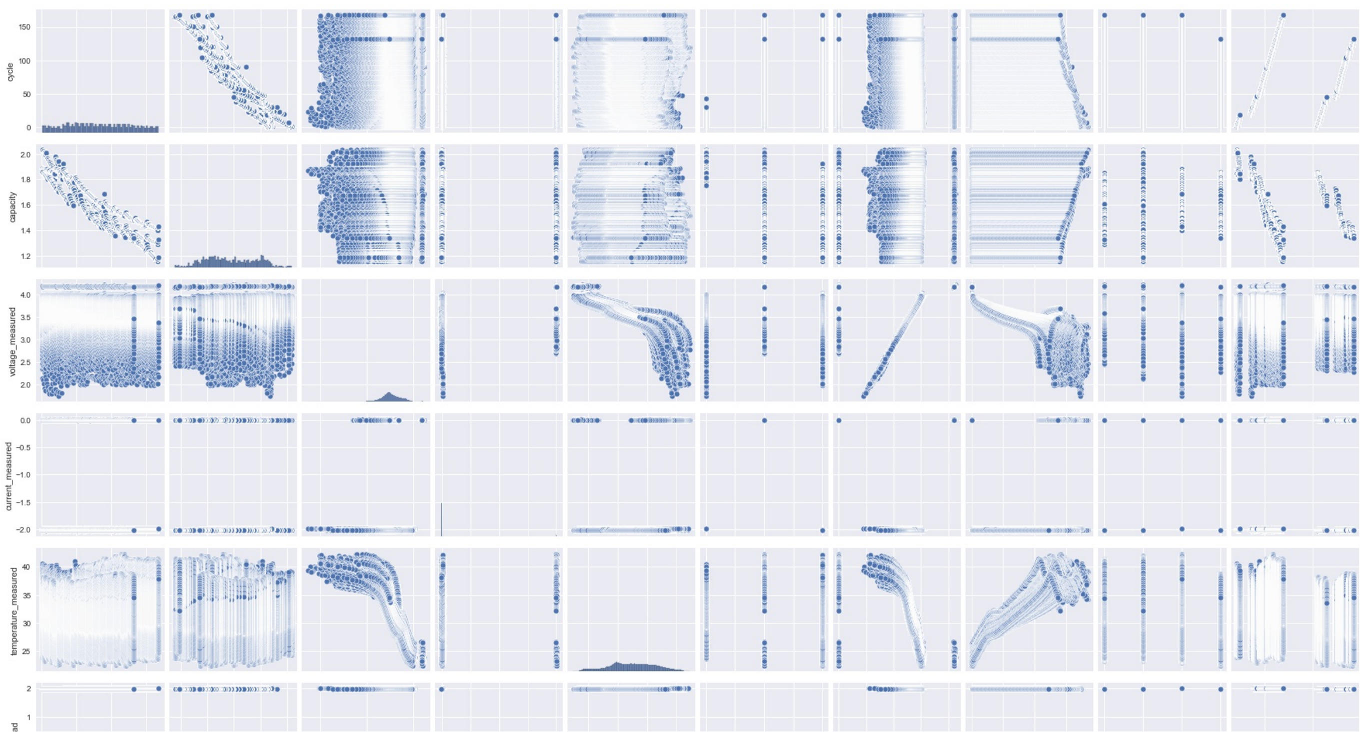


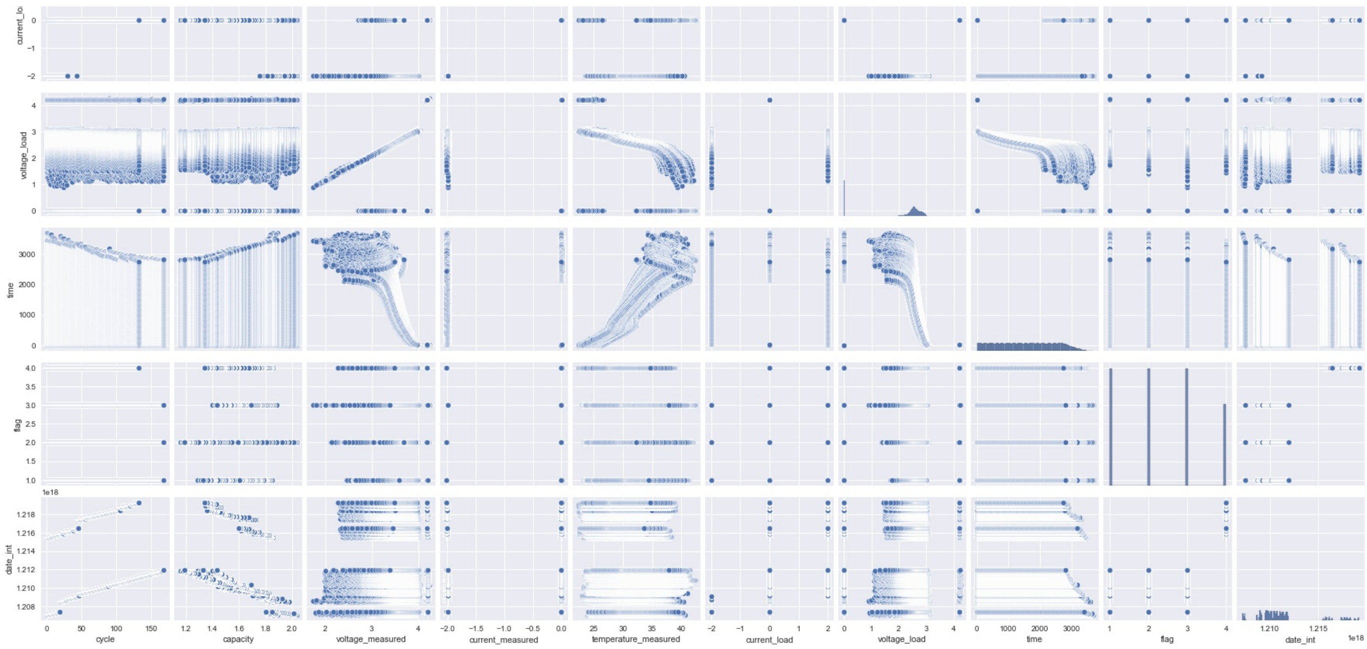
In [669…

```
sns.pairplot(df)
```

Out[669…

```
<seaborn.axisgrid.PairGrid at 0x22c0a166a00>
```





In [670…

```
#We can also use Maximum Information Coefficient to study non linear correaltions betweeen various features / co
```

In [671…

```
#DATA PREPROCESSING

# 1) Conversion of files using the helper code : Already done
# 2) Data concat and use of flags to identify different fuel cells : Already done
```

In [672…

```
# Checking for duplicates in the dataset and dropping them
print(df.shape)
df.drop_duplicates(keep = 'first',inplace=True)
print(df.shape)
```

```
(185721, 11)
(185721, 11)
```

In [673…

```
#There are no duplicate rows
```

In [674…

```
df.dropna(inplace=True)
print(df.shape)
```

```
(185721, 11)
```

In [675…

```
#Standardization and encoding : As such right now I don't see any use for either standardization or encoding
```

In [676…

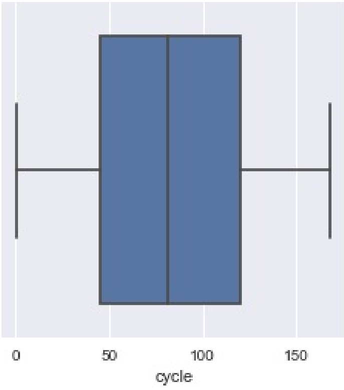
```
#Outlier detection with IQR
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1

print(IQR)
```

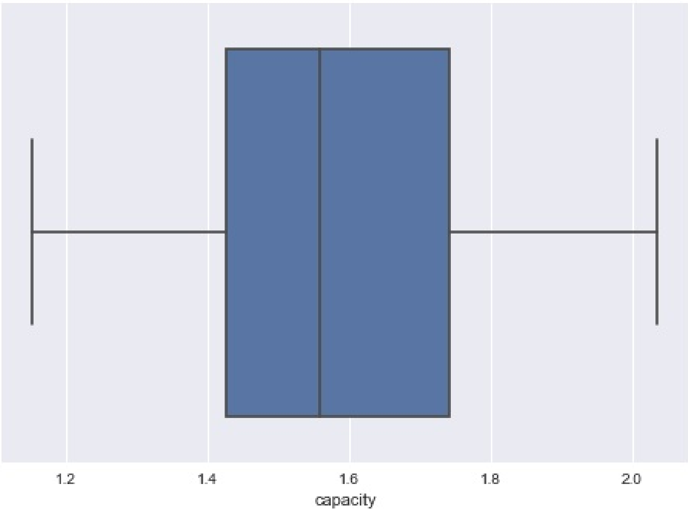
```
cycle                   7.500000e+01
capacity                3.158249e-01
voltage_measured        2.780981e-01
current_measured        2.144362e-02
temperature_measured    5.850056e+00
current_load            8.000000e-04
voltage_load            3.080000e-01
time                    1.542688e+03
flag                    2.000000e+00
date_int                1.995003e+15
dtype: float64
```

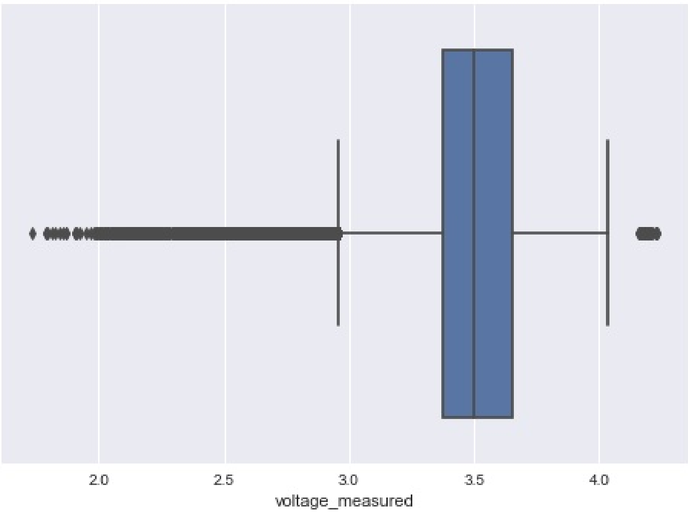
In [677…

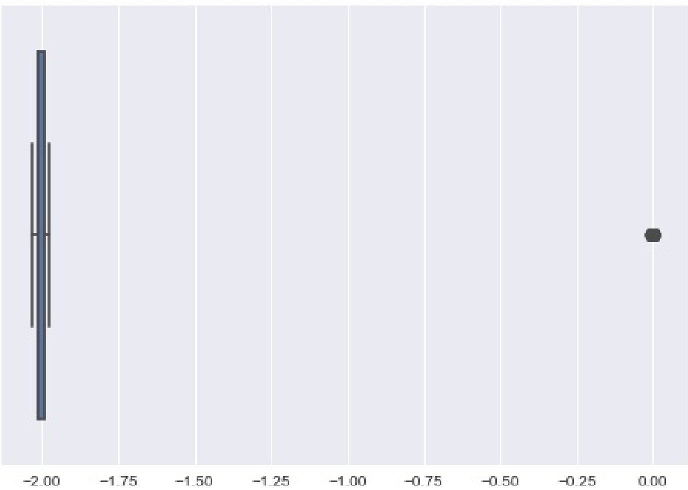
```
df.drop('datetime', axis=1, inplace=True)
```

```
plt.figure(figsize=(4,4))
for i in df.columns:
    sns.boxplot(x=i, data=df)
    plt.show()
```

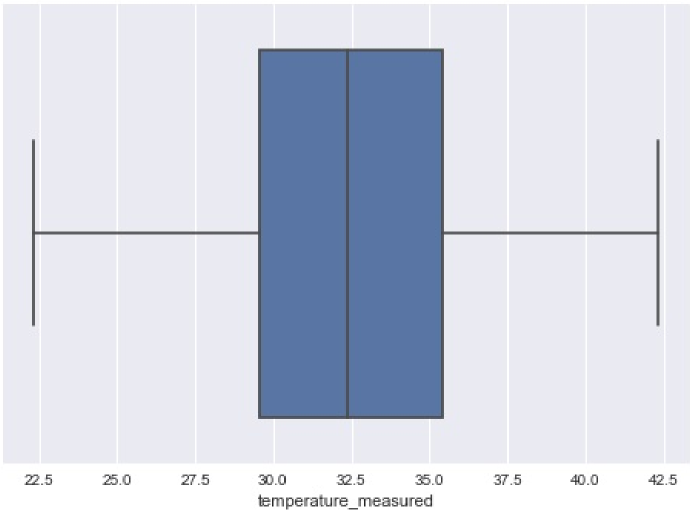


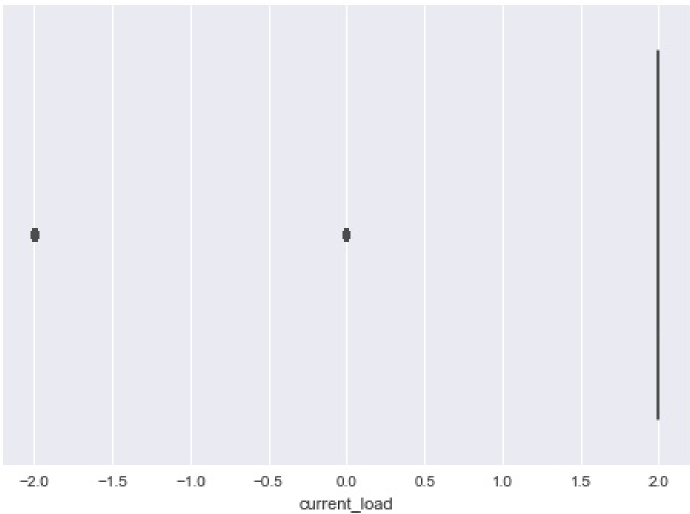


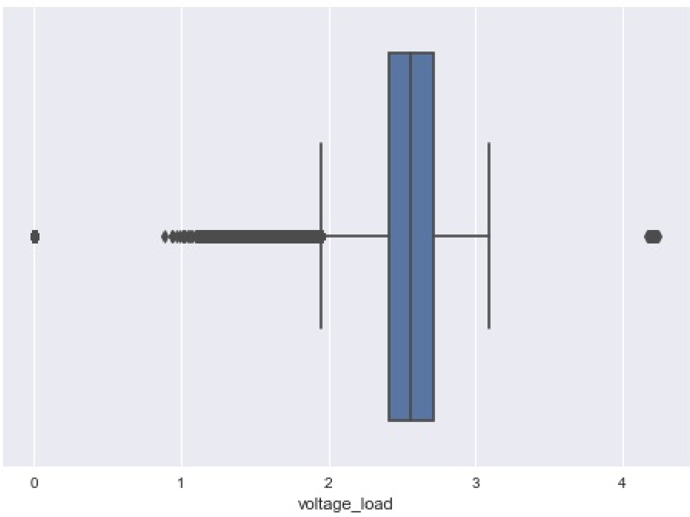


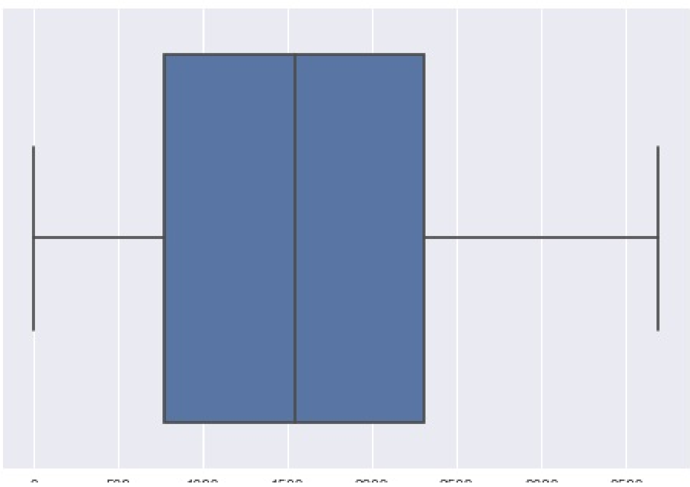


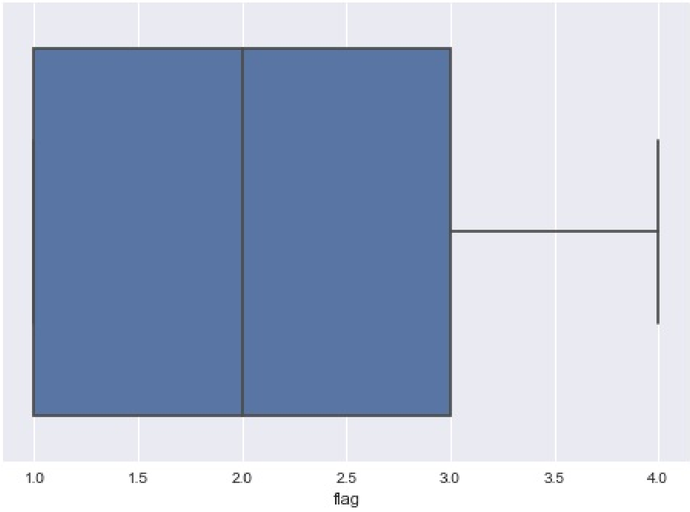
current_measured

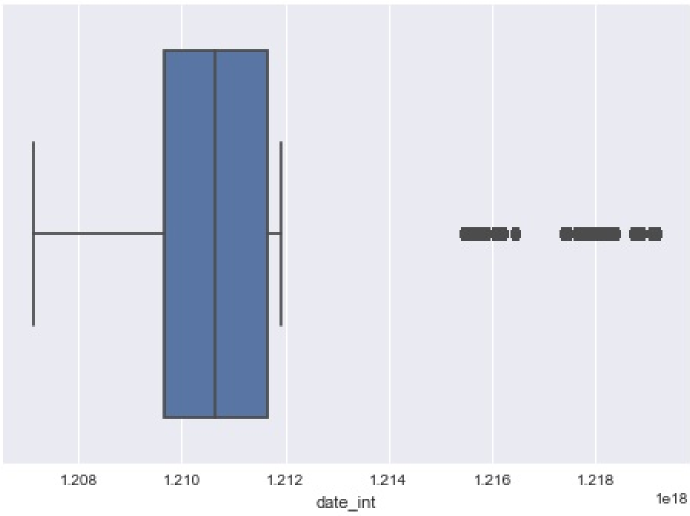












In [679…

```
#inner quartile . Detecting outliers with 1.5 IQR range
df_in = df[ ((df < (Q1 -  1.5 * IQR)) | (df > (Q3 + 1.5 * IQR))).any(axis=1) ]
print(df_in.shape)
```

```
(68810, 10)
```

In [680…

```
#Outer quartile. Detecting outliers with 3 IQR range
df_out = df[ ((df < (Q1 -  3 * IQR)) | (df > (Q3 + 3 * IQR))).any(axis=1) ]
print(df_out.shape)
```

```
(48586, 10)
```

In [681…

```
#Lets try 2
df_med = df[ ((df < (Q1 -  2 * IQR)) | (df > (Q3 + 2 * IQR))).any(axis=1) ]
print(df_med.shape)
```

```
(63872, 10)
```

In [682…

```
#copying the data for experimentation purpose
df_c1 = df.copy(deep=True)
df_c2 = df.copy(deep=True)
```

In [683…

```
print(df_c1.columns)
```

```
Index(['cycle', 'capacity', 'voltage_measured', 'current_measured',
```

```
       'temperature_measured', 'current_load', 'voltage_load', 'time', 'flag',
       'date_int'],
      dtype='object')
```

In [684…

```
cols = ['cycle', 'capacity', 'voltage_measured', 'current_measured',
        'temperature_measured', 'current_load', 'voltage_load', 'time',
        'date_int']
```

In [685…

```
df_c1[cols]
```

Out[685…

	cycle	capacity	voltage_measured	current_measured	temperature_measured	current_load	voltage_load	time	date_ir
0	1	1.856487	4.191492	-0.004902	24.330034	-0.0006	0.000	0.000	120714994100000000
1	1	1.856487	4.190749	-0.001478	24.325993	-0.0006	4.206	16.781	120714994100000000
2	1	1.856487	3.974871	-2.012528	24.389085	-1.9982	3.062	35.703	120714994100000000
3	1	1.856487	3.951717	-2.013979	24.544752	-1.9982	3.030	53.781	120714994100000000
4	1	1.856487	3.934352	-2.011144	24.731385	-1.9982	3.011	71.922	120714994100000000
...	...	...	...	...	...	...	...	...	.
185716	132	1.341051	3.443760	-0.002426	35.383979	0.0006	0.000	2686.359	121922143900000000
185717	132	1.341051	3.453271	-0.000981	35.179732	0.0006	0.000	2700.546	121922143900000000
185718	132	1.341051	3.461963	0.000209	34.977000	0.0006	0.000	2714.640	121922143900000000
185719	132	1.341051	3.469907	0.001516	34.785943	0.0006	0.000	2728.750	121922143900000000
185720	132	1.341051	3.477277	-0.001940	34.581660	0.0006	0.000	2742.843	121922143900000000

185721 rows × 9 columns

In [686…

```
#Normalization
#MinMax Scaler x = x = xmin / xmax - xmin
from sklearn.preprocessing import MinMaxScaler
ms = MinMaxScaler()

df_c1[cols] = ms.fit_transform(df_c1[cols])
df_c1.head(10)
```

Out[686…

	cycle	capacity	voltage_measured	current_measured	temperature_measured	current_load	voltage_load	time	flag	date_int
0	0.0	0.797111	0.983242	0.990600	0.099077	0.49985	0.000000	0.000000	1	0.0
1	0.0	0.797111	0.982944	0.992276	0.098875	0.49985	0.989880	0.004547	1	0.0
2	0.0	0.797111	0.896465	0.008109	0.102032	0.00045	0.720640	0.009675	1	0.0
3	0.0	0.797111	0.887189	0.007399	0.109822	0.00045	0.713109	0.014574	1	0.0
4	0.0	0.797111	0.880233	0.008787	0.119162	0.00045	0.708637	0.019490	1	0.0
5	0.0	0.797111	0.874507	0.007875	0.128092	0.00045	0.703930	0.024414	1	0.0
6	0.0	0.797111	0.869638	0.007193	0.137904	0.00045	0.700635	0.029343	1	0.0
7	0.0	0.797111	0.865284	0.008562	0.148470	0.00045	0.698282	0.034267	1	0.0
8	0.0	0.797111	0.861455	0.005424	0.158099	0.00045	0.696399	0.039196	1	0.0
9	0.0	0.797111	0.858043	0.007812	0.167816	0.00045	0.694516	0.044128	1	0.0

In [687…

```
#Standardized Sclaer x= x - mean / standard deviation
from sklearn.preprocessing import StandardScaler
ss = StandardScaler()

df_c2[cols] = ss.fit_transform(df_c2[cols])
df_c2.head(10)
```

Out[687…

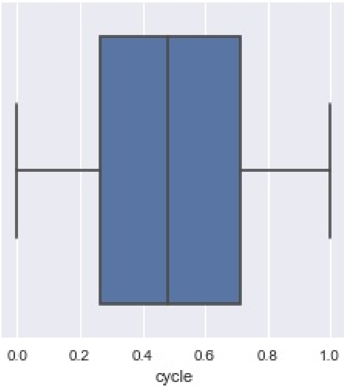
	cycle	capacity	voltage_measured	current_measured	temperature_measured	current_load	voltage_load	time	flag	date_int
0	-1.791091	1.477315	2.758442	3.255533	-1.998389	-1.194938	-3.149552	-1.705022	1	-1.424022
1	-1.791091	1.477315	2.755491	3.261632	-1.999392	-1.194938	2.448187	-1.686519	1	-1.424022
2	-1.791091	1.477315	1.897777	-0.320553	-1.983728	-2.823146	0.925645	-1.665656	1	-1.424022
3	-1.791091	1.477315	1.805782	-0.323138	-1.945079	-2.823146	0.883056	-1.645723	1	-1.424022
4	-1.791091	1.477315	1.736792	-0.318086	-1.898742	-2.823146	0.857769	-1.625721	1	-1.424022
5	-1.791091	1.477315	1.679999	-0.321405	-1.854441	-2.823146	0.831151	-1.605685	1	-1.424022

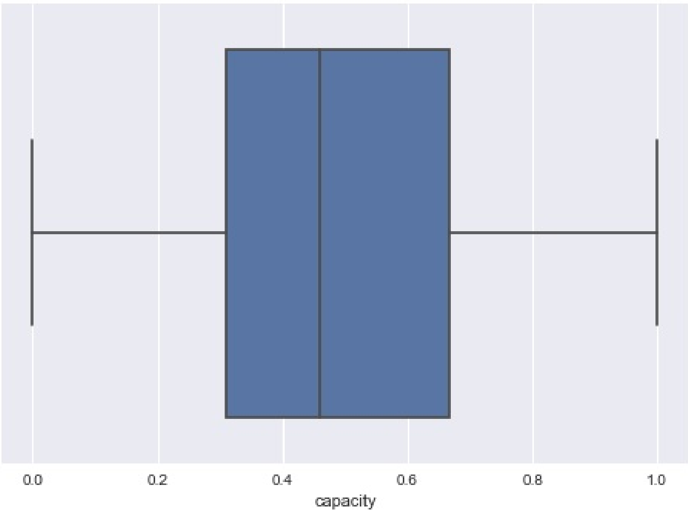
6	-1.791091	1.477315	1.631706	-0.323886	-1.805762	-2.823146	0.812519	-1.585632	1	-1.424022
7	-1.791091	1.477315	1.588527	-0.318904	-1.753341	-2.823146	0.799210	-1.565596	1	-1.424022
8	-1.791091	1.477315	1.550547	-0.330325	-1.705571	-2.823146	0.788563	-1.545542	1	-1.424022
9	-1.791091	1.477315	1.516704	-0.321633	-1.657360	-2.823146	0.777916	-1.525472	1	-1.424022

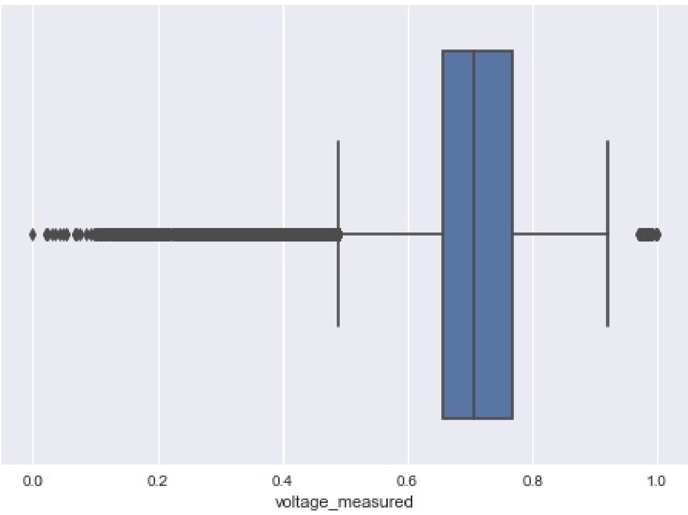
We should ideally remove and detect outliers first and then scale the data. If we first scale and then remove outliers then anyways the outliers would have been scaled. But as we can see from the boxplots below EVEN AFTER SCALING OUTLIERS REMAIN IN THE DATA.

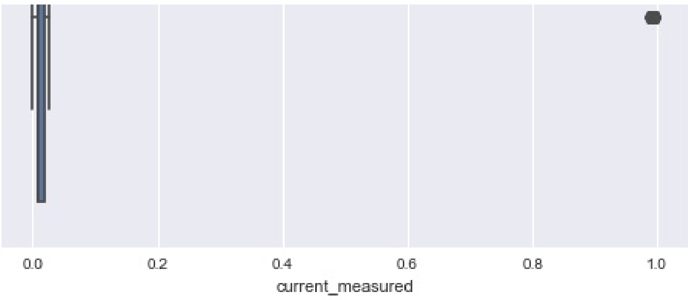
In [688…

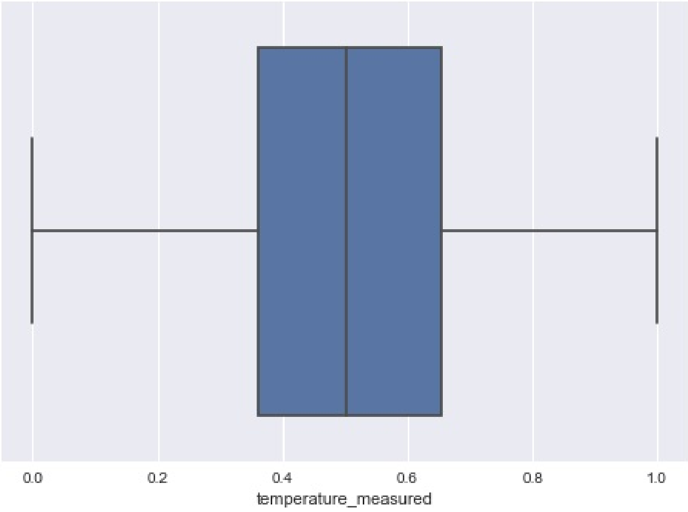
```
plt.figure(figsize=(4,4))
for i in df_c1.columns:
    sns.boxplot(x=i, data=df_c1)
    plt.show()
```

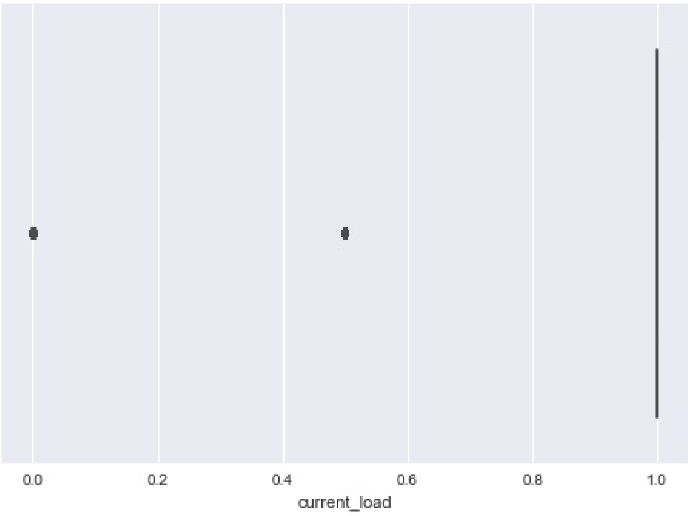


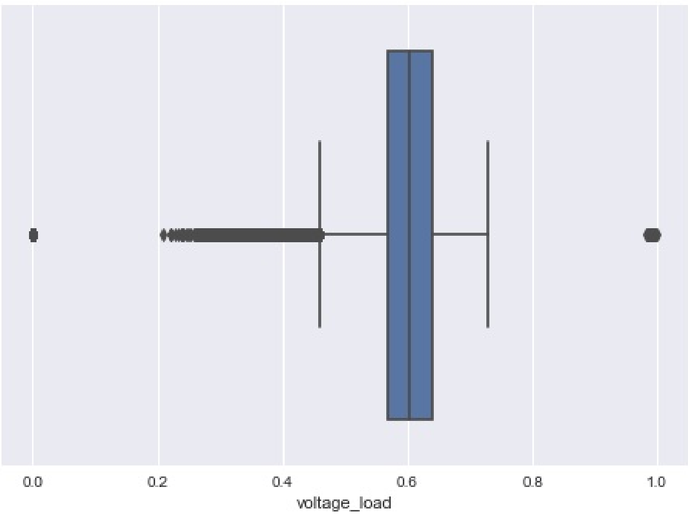


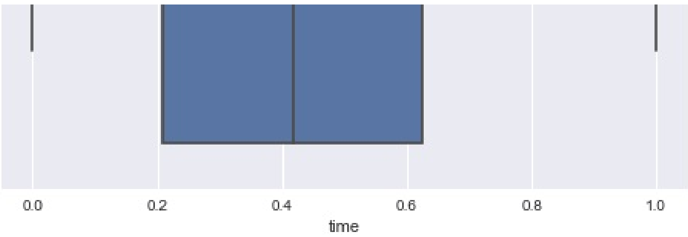


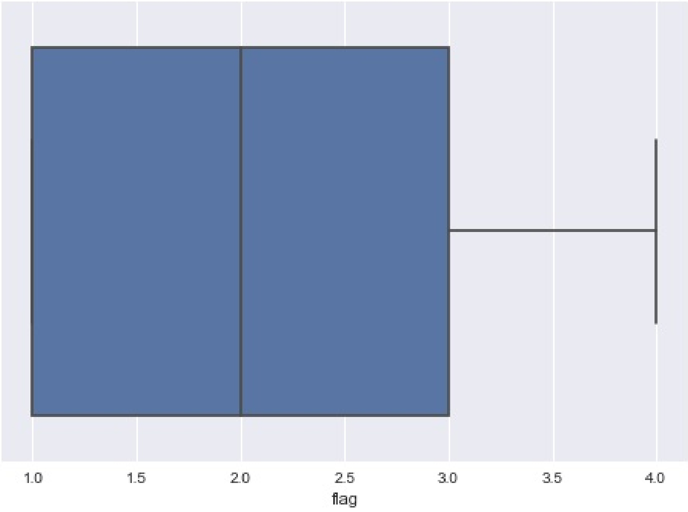

0.0
0.2
0.4
0.6
0.8
1.0
current_measured

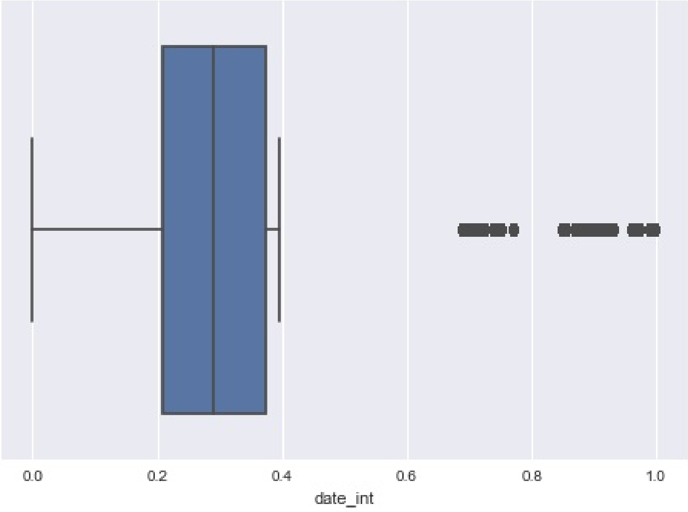

0.0
0.2
0.4
0.6
0.8
1.0
temperature_measured


0.0
0.2
0.4
0.6
0.8
1.0
current_load


0.0
0.2
0.4
0.6
0.8
1.0
voltage_load

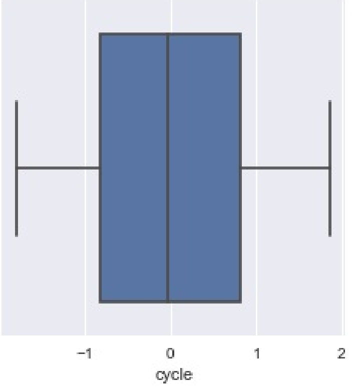


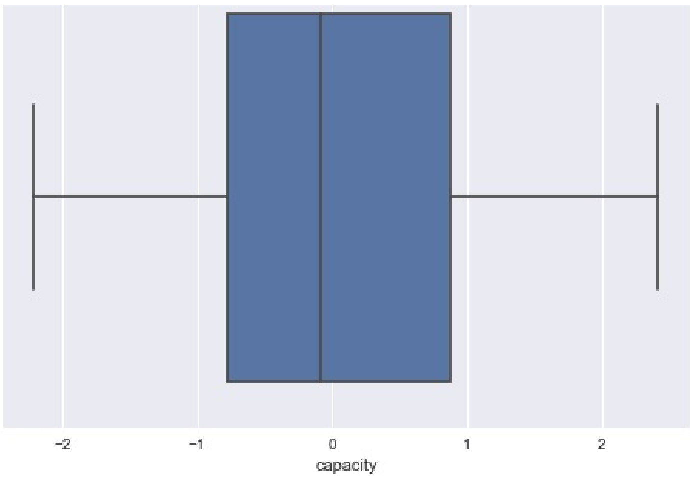


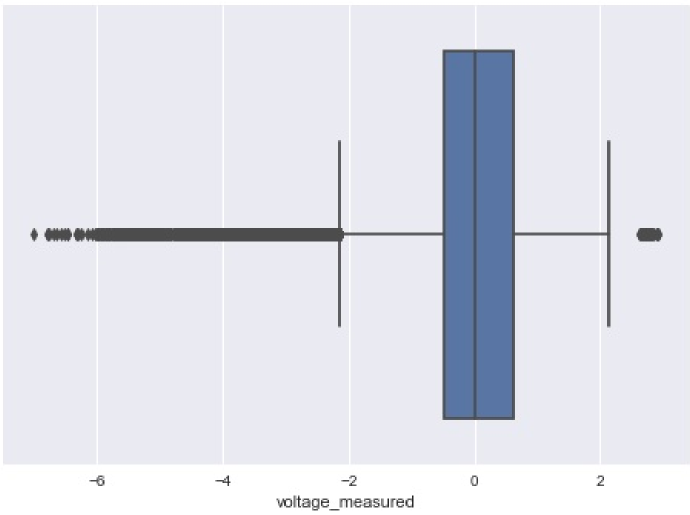


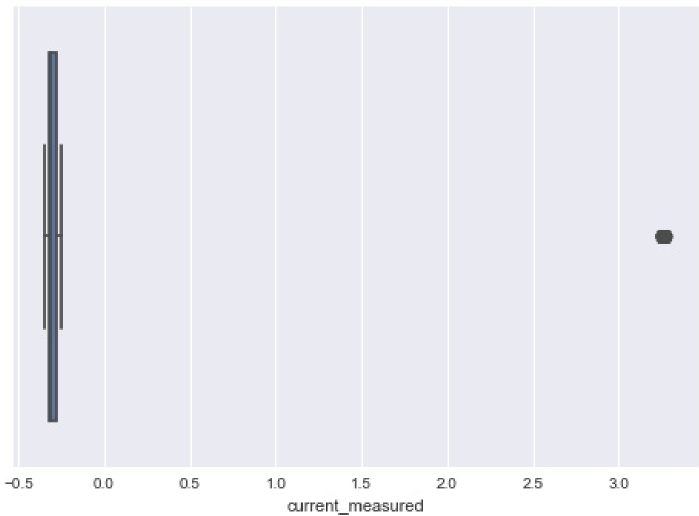
In [697…

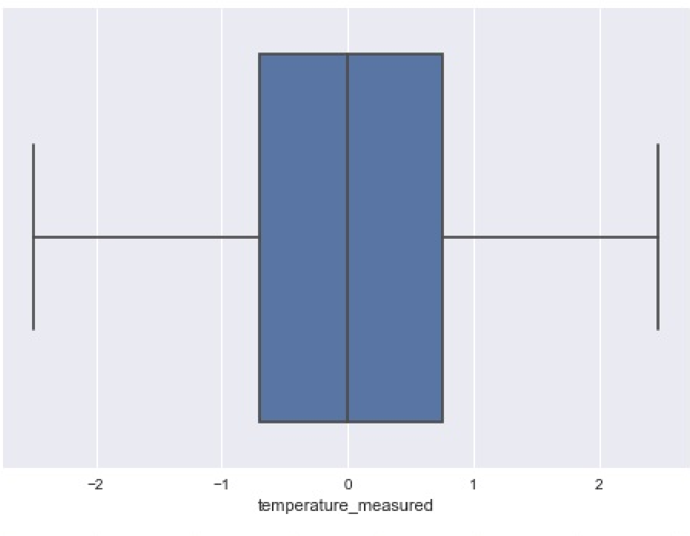
```
plt.figure(figsize=(4,4))
for i in df_c2.columns:
    sns.boxplot(x=i, data=df_c2)
    plt.show()
```

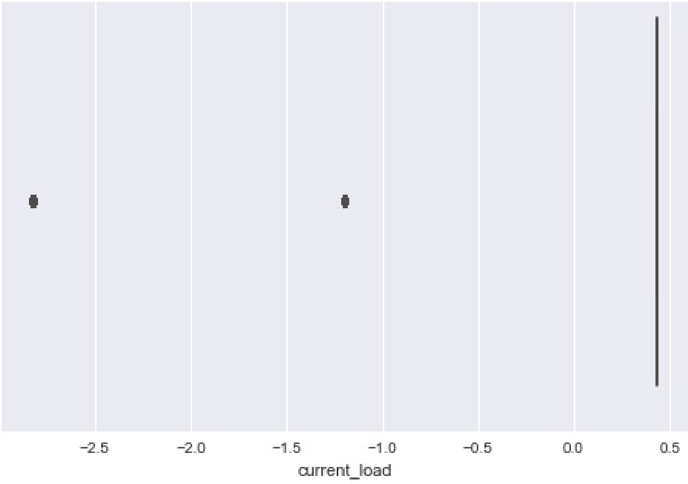


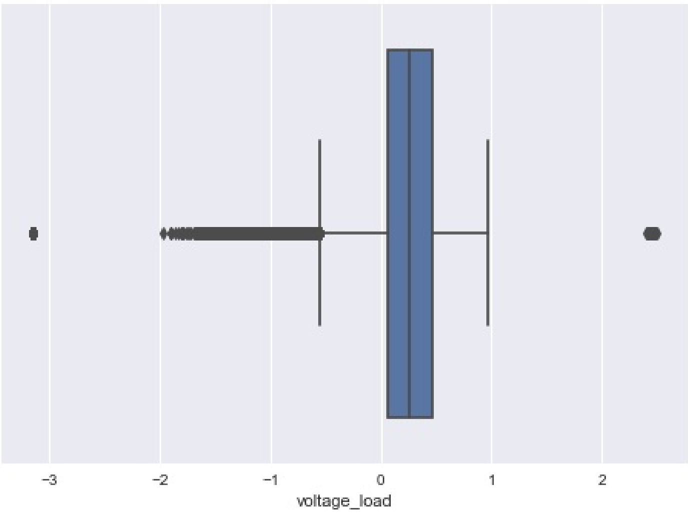

−2
−1
0
1
2
capacity

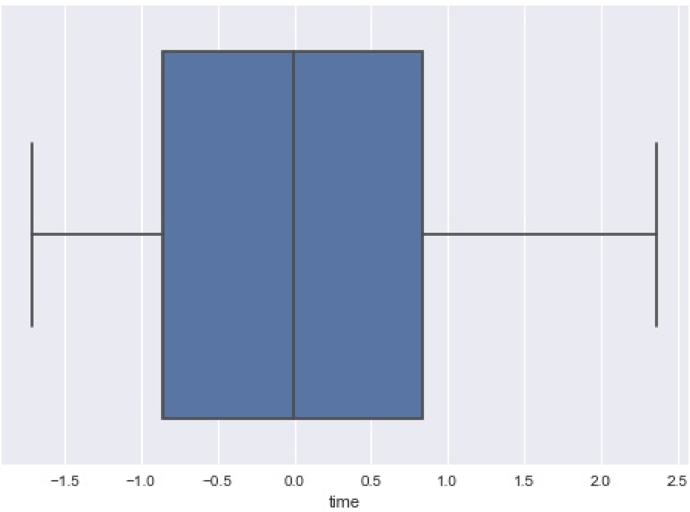

−6
−4
−2
0
2
voltage_measured

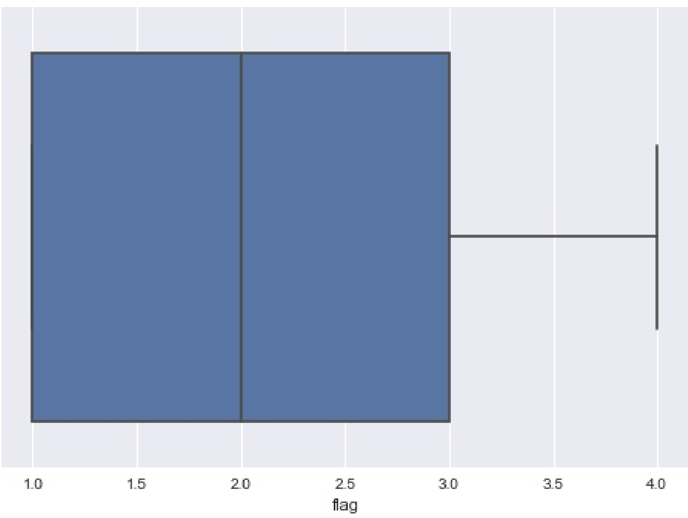

−0.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0
current_measured

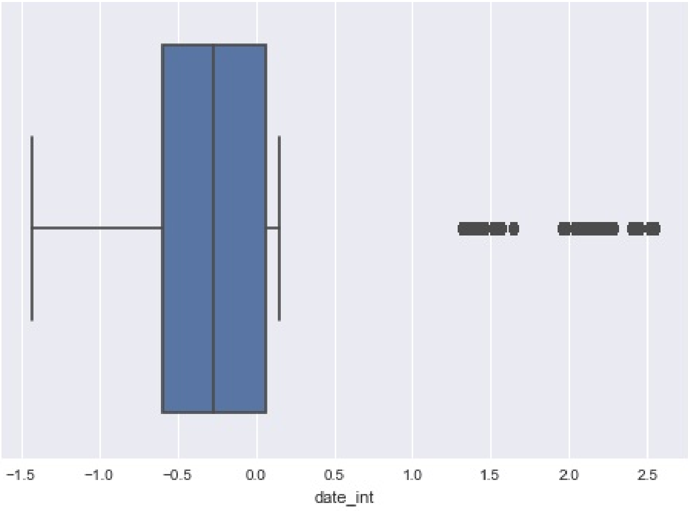

−2
−1
0
1
2
temperature_measured


−2.5 −2.0 −1.5 −1.0 −0.5 0.0 0.5
current_load


−3 −2 −1 0 1 2
voltage_load


−1.5 −1.0 −0.5 0.0 0.5 1.0 1.5 2.0 2.5
time


1.0 1.5 2.0 2.5 3.0 3.5 4.0
flag



With the above two boxplots it is proves that both standardization and normalization preserve outliers. They scale the data but outliers remain.

Unlike Normalization, standardization maintains useful information about outliers and makes the algorithm less sensitive to them in contrast to min-max scaling.

Anomaly detection using isolation forest

The range of output from sickit-learn IsolationForest decision_function is between -0.5 abd 0.5, where smaller values mean more anomalous. The predict function then applies a threshold to this function to get either -1 or +1

In [698…

```
from sklearn.ensemble import IsolationForest

#lets ee what we get with contamination = auto. self_offset = -0.5 with this setting
#https://stackoverflow.com/questions/63073951/what-does-setting-the-contamination-parameter-to-auto-in-sklearn-ou
#we don't use flag to classify outliers  hence remove it

clf = IsolationForest(n_estimators = 200, random_state=0).fit_predict(df_c1.drop('flag', axis=1))
non_anomaly = (clf== 1).sum()
anomaly = (clf== -1).sum()
print(non_anomaly)
print(anomaly)
print("% of outliers approx : " +  str(anomaly/non_anomaly*100) )

#36 percent are outliers across all batteries
```

```
136090
49631
% of outliers approx : 36.46924829157175
```

In [699…

```
import warnings
warnings.filterwarnings('ignore')

#Calculating outliers for each battery seperately
flagger=pd.DataFrame()
new=pd.DataFrame()
for j in df['flag'].unique():
    contamination = [0.01, 0.05, 0.15, 0.25, 0.40 ]
    #seperate flag / battery wise
    #We can try amd remove this falg and then do isolation tree but since all flags for a battery anyways have
    #one value it won't matter in outlier detection
    flagger = df_c1[df_c1['flag'] == j]
    for i in contamination :
        model = IsolationForest(n_estimators=200, max_samples='auto', contamination=i, random_state=0)
        flagger['anomaly_score' + '_'+str(i)] = model.fit_predict(flagger)
        flagger['scores'+'_'+str(i)] = model.decision_function(flagger[flagger.columns.difference(['anomaly_score
    new = pd.concat([flagger, new])
```

In [700…

```
#Caculating outliers for all batteries together
df_nf = df_c1.copy(deep=True)
df_nf =  df_c1.drop('flag',axis=1)
contamination = [0.01, 0.05, 0.15, 0.25, 0.40 ]
for i in contamination :
    model = IsolationForest(n_estimators=200, max_samples='auto', contamination=i, random_state=0)
    df_nf['anomaly_score' + '_'+str(i)] = model.fit_predict(df_nf)
```

```
    df_nf['scores'+'_'+str(i)] = model.decision_function(df_nf[df_nf.columns.difference(['anomaly_score' + '_'+st
```

In [701…

```
new.head()
```

Out[701…

	cycle	capacity	voltage_measured	current_measured	temperature_measured	current_load	voltage_load	time	flag	date_int	anoı
150855	0.0	0.795429	0.981886	0.993063	0.073528	0.50015	0.000000	0.000000	4	0.687055	
150856	0.0	0.795429	0.981921	0.993713	0.073993	0.50015	0.989174	0.002553	4	0.687055	
150857	0.0	0.795429	0.897491	0.011464	0.074801	0.99970	0.712874	0.005305	4	0.687055	
150858	0.0	0.795429	0.891298	0.008267	0.078836	0.99970	0.712168	0.007863	4	0.687055	
150859	0.0	0.795429	0.886436	0.008365	0.083092	0.99970	0.709579	0.010429	4	0.687055	

In [702…

```
new['flag'].value_counts()
```

Out[702…

```
3    50285
2    50285
1    50285
4    34866
Name: flag, dtype: int64
```

In [703…

```
df['flag'].value_counts()
```

Out[703…

```
1    50285
2    50285
3    50285
4    34866
Name: flag, dtype: int64
```

In [704…

```
#We merged correctly
```

In [709…

```
new[new['flag']==1].head(10)
```

Out[709…

	cycle	capacity	voltage_measured	current_measured	temperature_measured	current_load	voltage_load	time	flag	date_int	anomaly_s
0	0.0	0.797111	0.983242	0.990600	0.099077	0.49985	0.000000	0.000000	1	0.0	
1	0.0	0.797111	0.982944	0.992276	0.098875	0.49985	0.989880	0.004547	1	0.0	
2	0.0	0.797111	0.896465	0.008109	0.102032	0.00045	0.720640	0.009675	1	0.0	
3	0.0	0.797111	0.887189	0.007399	0.109822	0.00045	0.713109	0.014574	1	0.0	
4	0.0	0.797111	0.880233	0.008787	0.119162	0.00045	0.708637	0.019490	1	0.0	
5	0.0	0.797111	0.874507	0.007875	0.128092	0.00045	0.703930	0.024414	1	0.0	
6	0.0	0.797111	0.869638	0.007193	0.137904	0.00045	0.700635	0.029343	1	0.0	
7	0.0	0.797111	0.865284	0.008562	0.148470	0.00045	0.698282	0.034267	1	0.0	
8	0.0	0.797111	0.861455	0.005424	0.158099	0.00045	0.696399	0.039196	1	0.0	
9	0.0	0.797111	0.858043	0.007812	0.167816	0.00045	0.694516	0.044128	1	0.0	

In [710…

```
print("Finding outliers for all batteries seperately")
for i in contamination:
    print(f'Anomalies with contamination {i}: ', len(new[new['anomaly_score' + '_'+str(i)] == -1]))
```

```
Finding outliers for all batteries seperately
Anomalies with contamination 0.01:  1858
Anomalies with contamination 0.05:  9289
Anomalies with contamination 0.15:  27859
Anomalies with contamination 0.25:  46430
Anomalies with contamination 0.4:  74288
```

In [711…

```
print("Finding outliers for all batteries together")
for i in contamination:
    print(f'Anomalies with contamination {i}: ', len(df_nf[df_nf['anomaly_score' + '_'+str(i)] == -1]))
```

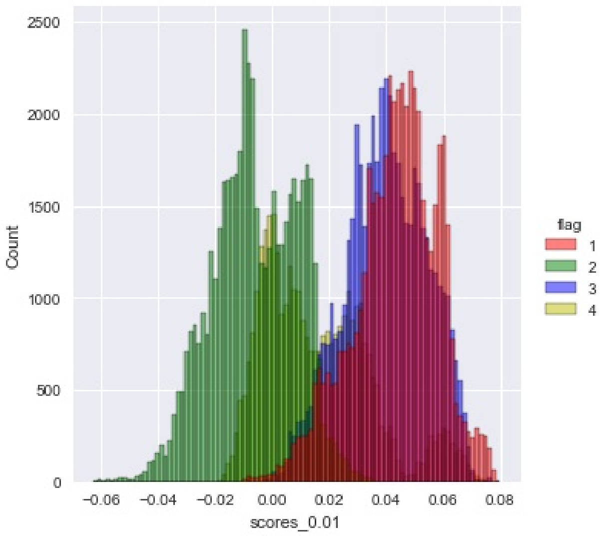
```
Finding outliers for all batteries together
Anomalies with contamination 0.01:  1858
Anomalies with contamination 0.05:  9286
Anomalies with contamination 0.15:  27858
Anomalies with contamination 0.25:  46430
Anomalies with contamination 0.4:  74288
```

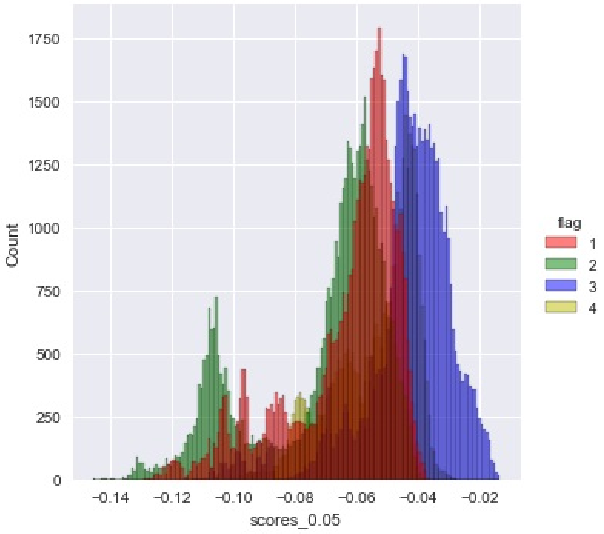
Surprisingly the results are almost same excet for contamination level 0.05, 0.15 where they differ very little

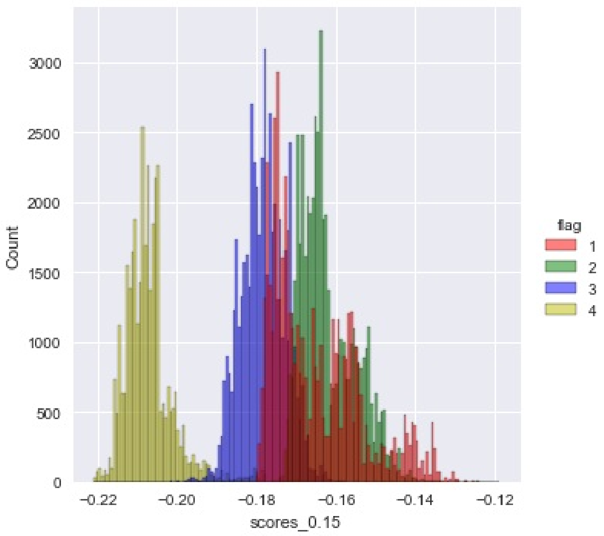
Plotting the anomaly scores for vairous batteries

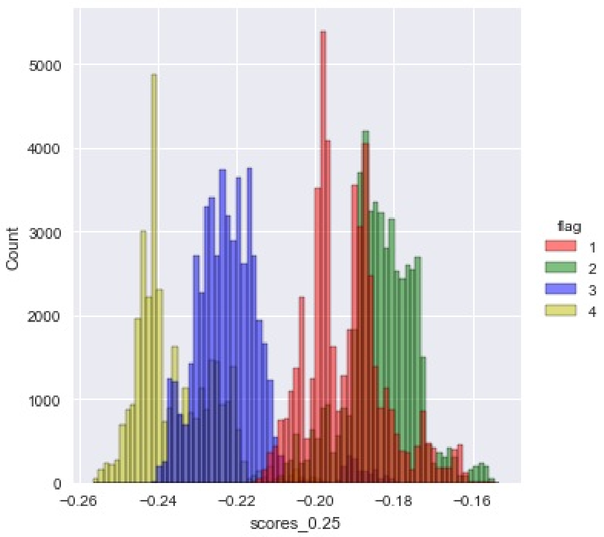
In [712…

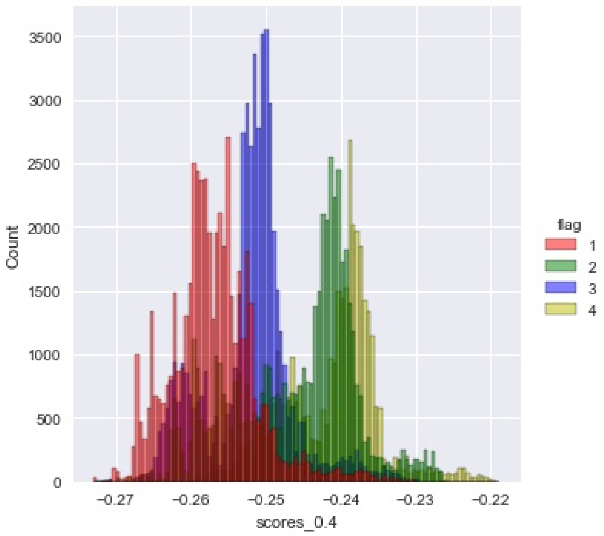
```
a =  ['scores_0.01', 'scores_0.05', 'scores_0.15', 'scores_0.25', 'scores_0.4']

for i in range(0, len(a)):
    sns.displot(x=a[i], hue='flag', palette=['r','g','b','y'], data = new)
```











In [713…

```
def count_anomaly(series):
    return (series == 1).sum()

def count_nonanomaly(series):
    return (series == -1).sum()

b =  ['score_0.01', 'score_0.05', 'score_0.15', 'score_0.25', 'score_0.4']


for i in range(0, len(b)):
    #Best battery is the one with the least anomalies
    anomaly_apply = 'anomaly_' + b[i]
    print("Contamination: "+ b[i])
    print(new.groupby('flag')[anomaly_apply].apply(count_anomaly).reset_index(name='count_anomaly'))
    print(new.groupby('flag')[anomaly_apply].apply(count_nonanomaly).reset_index(name='count_nonanomaly'))
```

```
Contamination: score_0.01
   flag  count_anomaly
0     1          49782
1     2          49782
2     3          49782
3     4          34517
   flag  count_nonanomaly
0     1               503
1     2               503
2     3               503
3     4               349
Contamination: score_0.05
   flag  count_anomaly
0     1          47770
1     2          47770
2     3          47770
3     4          33122
   flag  count_nonanomaly
0     1              2515
1     2              2515
2     3              2515
3     4              1744
Contamination: score_0.15
```

```
   flag  count_anomaly
0     1          42742
1     2          42742
2     3          42742
3     4          29636
   flag  count_nonanomaly
0     1              7543
1     2              7543
2     3              7543
3     4              5230
Contamination: score_0.25
   flag  count_anomaly
0     1          37714
1     2          37714
2     3          37714
3     4          26149
   flag  count_nonanomaly
0     1             12571
1     2             12571
2     3             12571
3     4              8717
Contamination: score_0.4
   flag  count_anomaly
0     1          30171
1     2          30171
2     3          30171
3     4          20920
   flag  count_nonanomaly
0     1             20114
1     2             20114
2     3             20114
3     4             13946
```

In [714…

```
#Battery with the most anomalies will the one for which we have the most entries as for every contamination leve
# that % of entries from a battery are marked as anomalies
```

Just seeing for conatimatnion level 0.05 how many extreme outliers we have for battery 1 and 2

In [715…

```
len(new.loc[ (new['flag']==1) & (new['anomaly_score_0.05']==-1) & (new['scores_0.05'] < -0.012 )])
```

Out[715…

```
2515
```

In [716…

```
len(new.loc[ (new['flag']==2) & (new['anomaly_score_0.05']==-1) & (new['scores_0.05'] < -0.12 )])
```

Out[716…

```
638
```

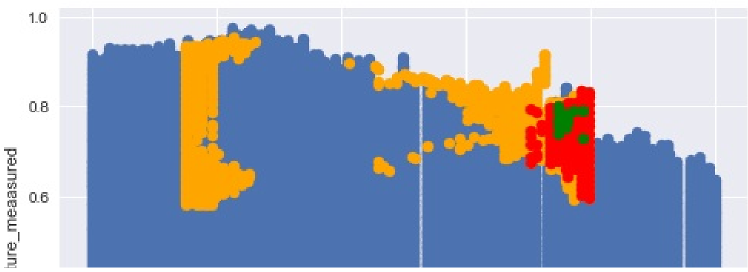
In [717…

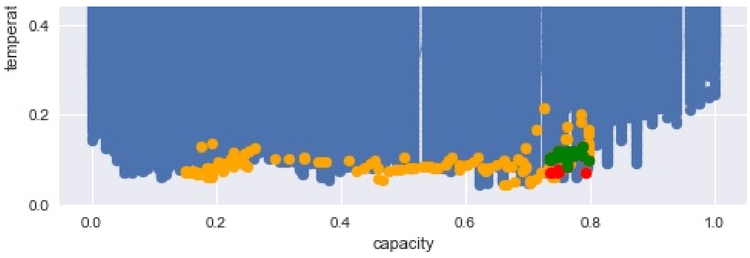
```
#For battery 2 for contamination level lets retreieve some extreme outliers
b11 = new.loc[ (new['flag']==1) & (new['anomaly_score_0.05']==-1) & (new['scores_0.05'] < -0.1 ) ]
b22 = new.loc[ (new['flag']==1) & (new['anomaly_score_0.05']==-1) & (new['scores_0.05'] < -0.12 )]
b23 = new.loc[ (new['flag']==1) & (new['anomaly_score_0.05']==-1) & (new['scores_0.05'] <= -0.125 )]
```

Plot between Temperature and Capacity measured

In [718…

```
plt.scatter( new[ (new['flag']==2) & (new['anomaly_score_0.05']==1) ]['capacity'] ,
             new[ (new['flag']==2) & (new['anomaly_score_0.05']==1) ]['temperature_measured'])

plt.scatter (b11['capacity'], b11['temperature_measured'], c="orange", marker="o", )
plt.scatter (b22['capacity'], b22['temperature_measured'], c='red', marker='o', )
plt.scatter (b23['capacity'], b23['temperature_measured'], c='green', marker='o', )

plt.xlabel('capacity')
plt.ylabel('temperature_meaasured')
plt.show()
```

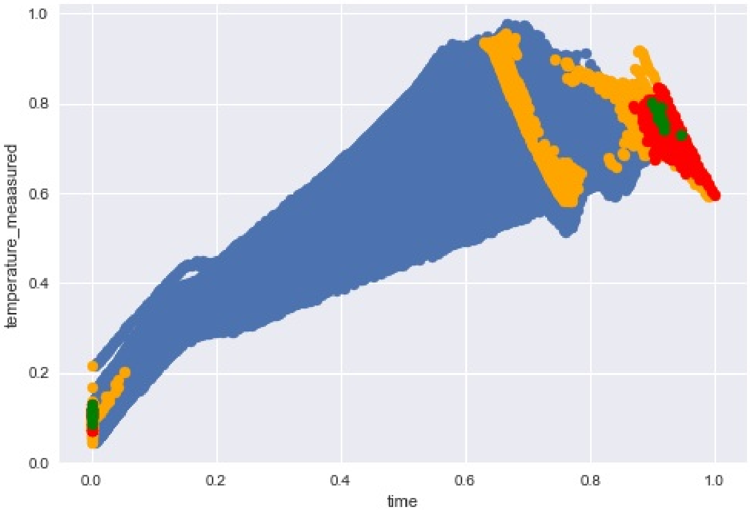




Plot between Time and Temeprature measured

In [719…

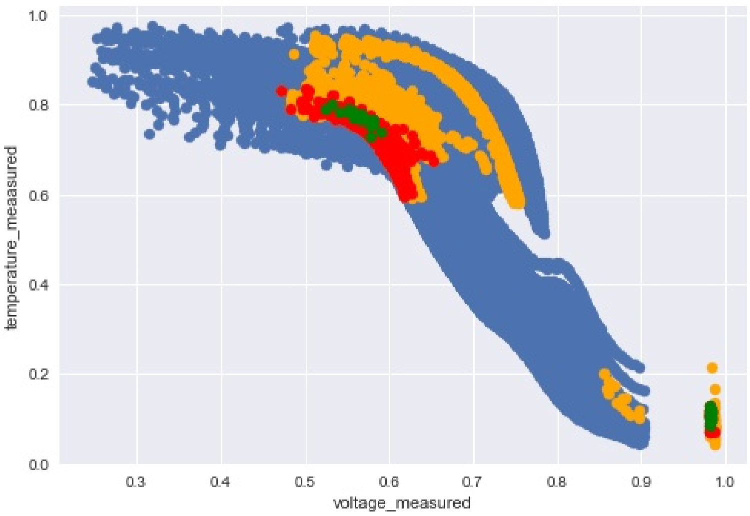
```
plt.scatter( new[ (new['flag']==2) & (new['anomaly_score_0.05']==1) ]['time'] ,
             new[ (new['flag']==2) & (new['anomaly_score_0.05']==1) ]['temperature_measured'])

plt.scatter (b11['time'], b11['temperature_measured'], c="orange", marker="o", )
plt.scatter (b22['time'], b22['temperature_measured'], c='red', marker='o', )
plt.scatter (b23['time'], b23['temperature_measured'], c='green', marker='o', )

plt.xlabel('time')
plt.ylabel('temperature_meaasured')
plt.show()
```



Plot between Voltage and Temeprature measured

In [720…

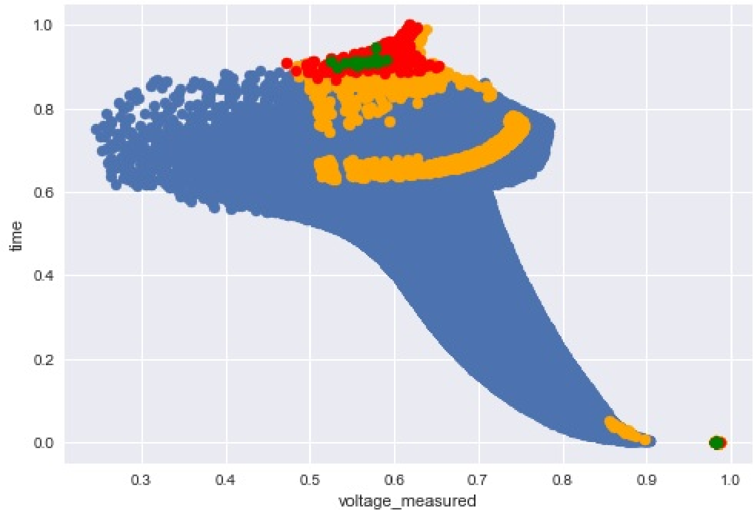
```
plt.scatter( new[ (new['flag']==2) & (new['anomaly_score_0.05']==1) ]['voltage_measured'] ,
             new[ (new['flag']==2) & (new['anomaly_score_0.05']==1) ]['temperature_measured'])

plt.scatter (b11['voltage_measured'], b11['temperature_measured'], c="orange", marker="o", )
plt.scatter (b22['voltage_measured'], b22['temperature_measured'], c='red', marker='o', )
plt.scatter (b23['voltage_measured'], b23['temperature_measured'], c='green', marker='o', )

plt.xlabel('voltage_measured')
plt.ylabel('temperature_meaasured')
plt.show()
```



Plot between Voltage Measured and Time

In [721...

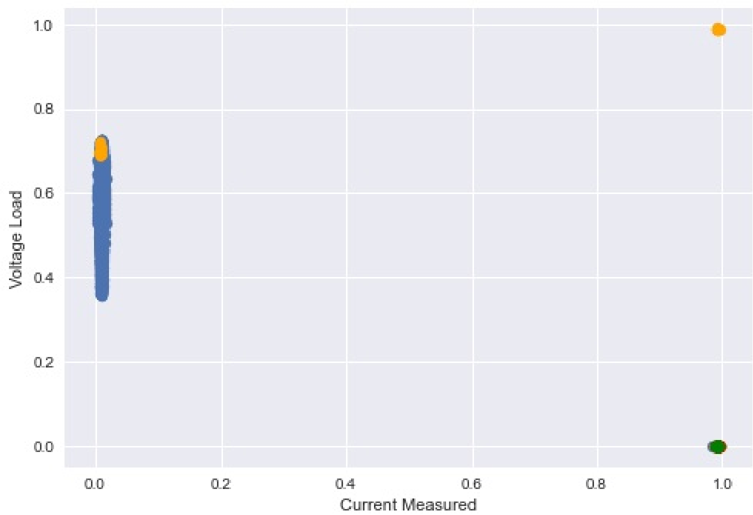
```
plt.scatter( new[ (new['flag']==2) & (new['anomaly_score_0.05']==1) ]['voltage_measured'] ,
             new[ (new['flag']==2) & (new['anomaly_score_0.05']==1) ]['time'])

plt.scatter (b11['voltage_measured'], b11['time'], c="orange", marker="o", )
plt.scatter (b22['voltage_measured'], b22['time'], c='red', marker='o', )
plt.scatter (b23['voltage_measured'], b23['time'], c='green', marker='o', )

plt.xlabel('voltage_measured')
plt.ylabel('time')
plt.show()
```



Plot between Current Measured and Voltage Load

In [722...

```
plt.scatter( new[ (new['flag']==2) & (new['anomaly_score_0.05']==1) ]['current_measured'] ,
             new[ (new['flag']==2) & (new['anomaly_score_0.05']==1) ]['voltage_load'])

plt.scatter (b11['current_measured'], b11['voltage_load'], c="orange", marker="o", )
plt.scatter (b22['current_measured'], b22['voltage_load'], c='red', marker='o', )
plt.scatter (b23['current_measured'], b23['voltage_load'], c='green', marker='o', )

plt.xlabel('Current Measured')
plt.ylabel('Voltage Load')
plt.show()
```



In [723...

```
from sklearn.neighbors import LocalOutlierFactor
```

To note that negative_outlier*factor* is the oppsoite of LOF scores. The higher, the more normal. Inliers tend to have a LOF score close to 1 (negative_outlier*factor* close to -1), while outliers tend to have a larger LOF score.

Hence one the array for negative outlier factor is calcualted when need to negate it.

In [724…
```python
flagger1=pd.DataFrame()
new1=pd.DataFrame()
for j in df['flag'].unique():

    #Higher contamination values take too much RAM
    contamination = [0.01, 0.05, 0.15, 0.25]

    flagger1= df_c1[df_c1['flag'] == j]
    for i in contamination :
        model = LocalOutlierFactor(n_neighbors = 25, leaf_size = 25, contamination=i)
        flagger1['anomaly_score' + '_'+str(i)] = model.fit_predict(flagger1)
        lof_scores =  np.negative(model.negative_outlier_factor_)
        flagger1['scores'+'_'+str(i)] = lof_scores
    new1 = pd.concat([flagger1, new1])
```

In [725…
```python
new1.head()
```

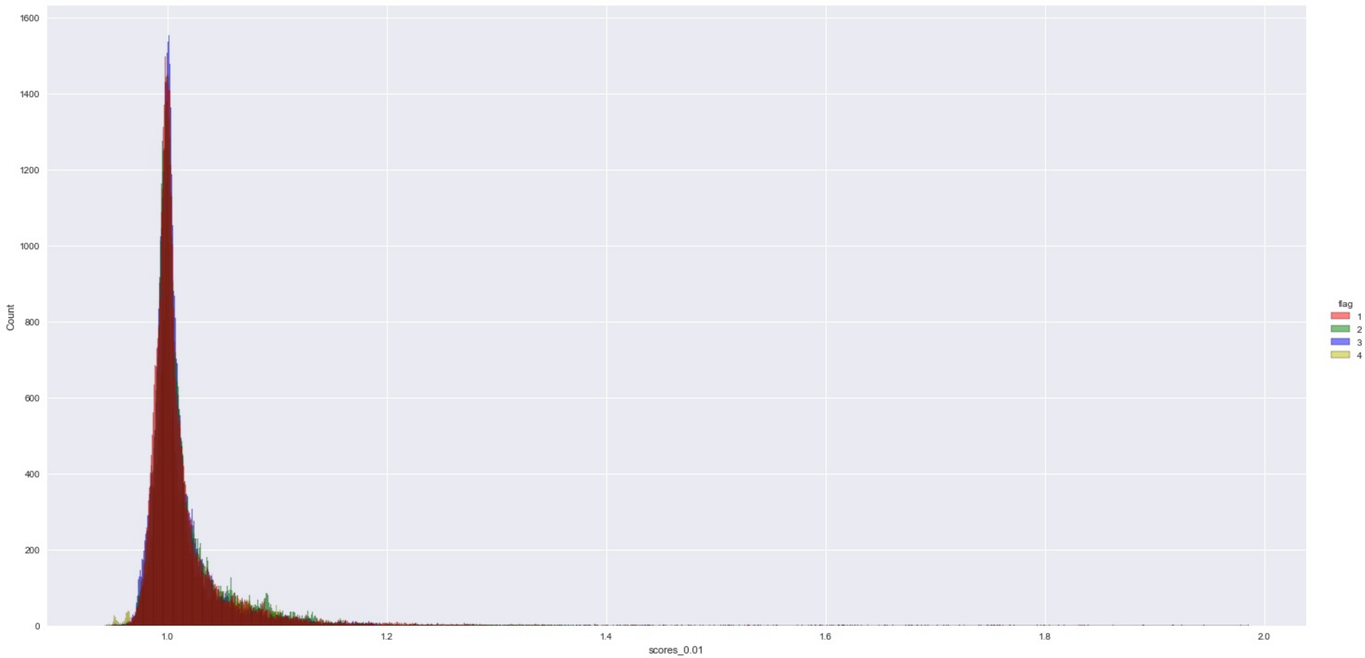
Out[725…

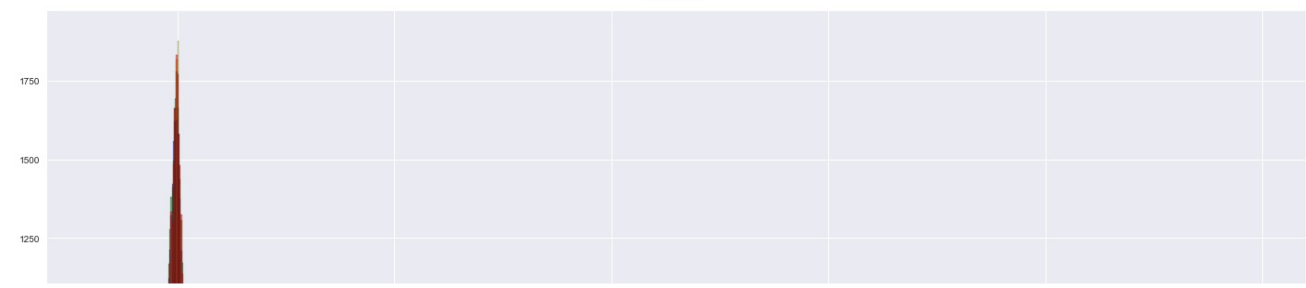
	cycle	capacity	voltage_measured	current_measured	temperature_measured	current_load	voltage_load	time	flag	date_int	anoı
150855	0.0	0.795429	0.981886	0.993063	0.073528	0.50015	0.000000	0.000000	4	0.687055	
150856	0.0	0.795429	0.981921	0.993713	0.073993	0.50015	0.989174	0.002553	4	0.687055	
150857	0.0	0.795429	0.897491	0.011464	0.074801	0.99970	0.712874	0.005305	4	0.687055	
150858	0.0	0.795429	0.891298	0.008267	0.078836	0.99970	0.712168	0.007863	4	0.687055	
150859	0.0	0.795429	0.886436	0.008365	0.083092	0.99970	0.709579	0.010429	4	0.687055	

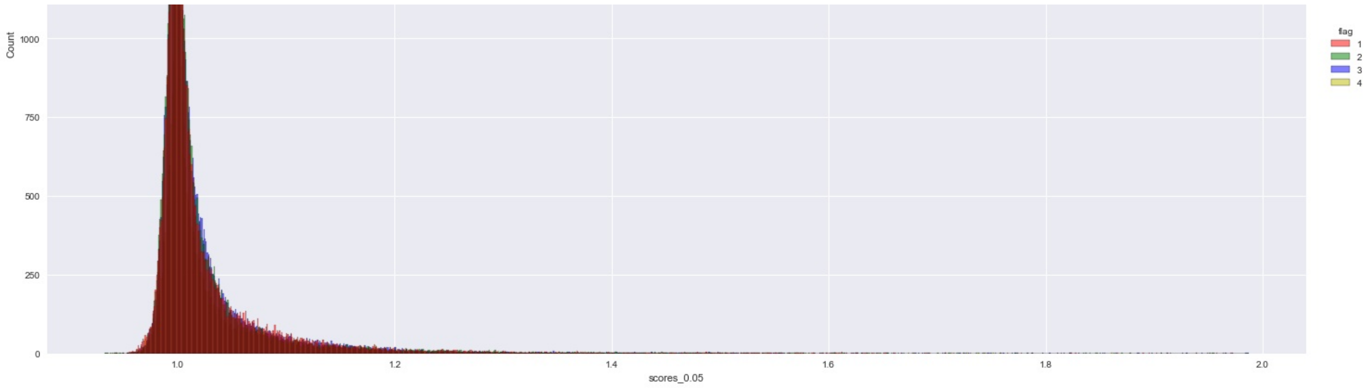
In [726…
```python
print("Finding outliers for all batteries seperately with nearest neighbors 25")
for i in contamination:
    print(f'Anomalies with contamination {i}: ', len(new1[new1['anomaly_score' + '_'+str(i)] == -1]))
```

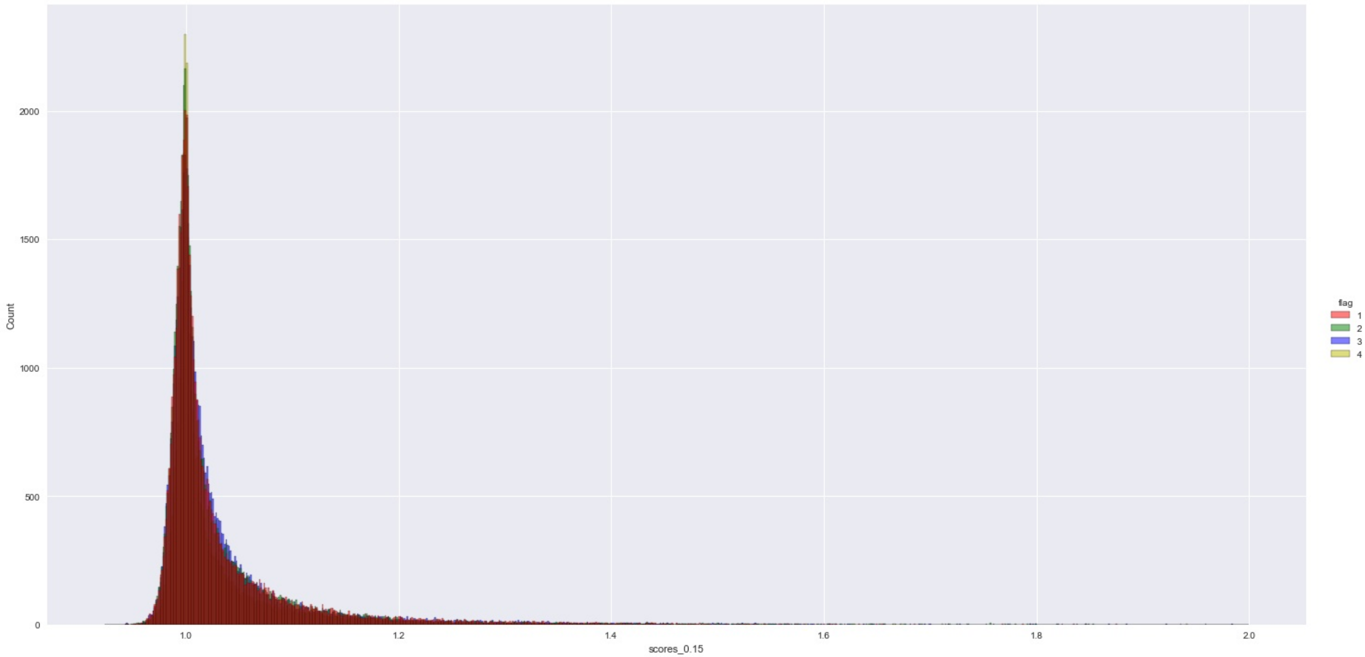
```
Finding outliers for all batteries seperately with nearest neighbors 25
Anomalies with contamination 0.01:  1858
Anomalies with contamination 0.05:  9289
Anomalies with contamination 0.15:  27859
Anomalies with contamination 0.25:  46430
```

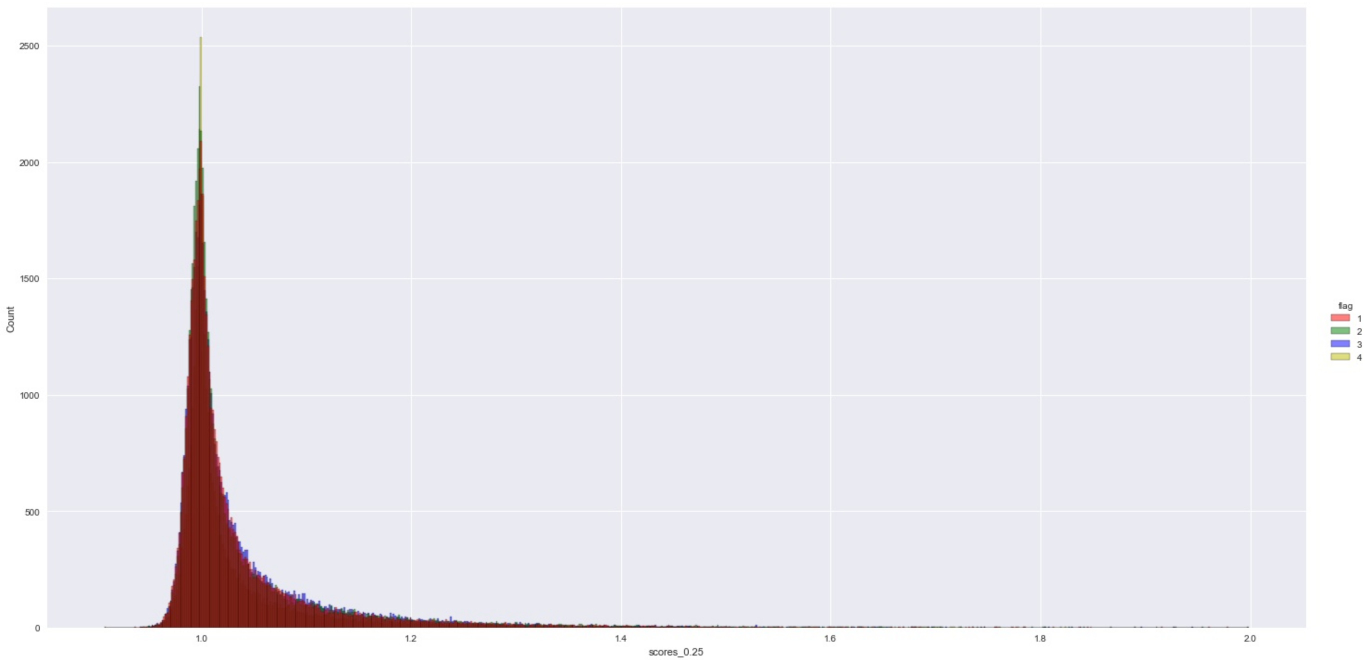
In [727…
```python
a = ['scores_0.01', 'scores_0.05', 'scores_0.15', 'scores_0.25']

for i in range(0, len(a)):
    sns.displot(x=a[i], hue='flag', palette=['r','g','b','y'], data = new1[new1[a[i]]<=2], height =10, aspect=2)
```











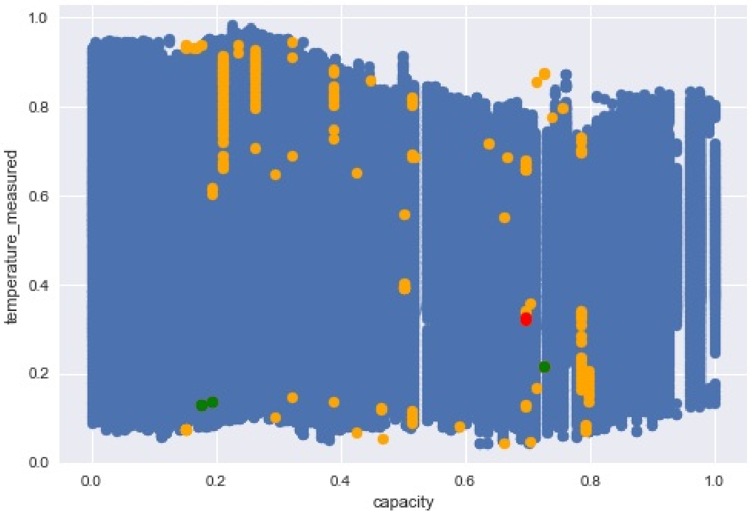
In [728…

```
#For battery 2 for contamination level lets retreieve some extreme outliers
b11 = new1.loc[ (new1['flag']==1) & (new1['anomaly_score_0.05']==-1) & (new1['scores_0.05'] > 1.5 ) ]
b22 = new1.loc[ (new1['flag']==1) & (new1['anomaly_score_0.05']==-1) & (new1['scores_0.05'] > 2.5 )]
b23 = new1.loc[ (new1['flag']==1) & (new1['anomaly_score_0.05']==-1) & (new1['scores_0.05'] > 3 )]
```

In [729…

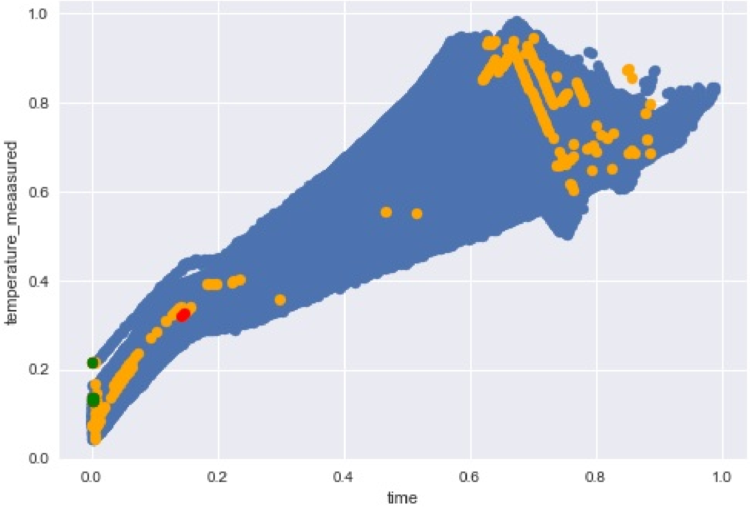
```
plt.scatter( new1[ (new1['flag']==2) & (new1['anomaly_score_0.05']==1) ]['capacity'] ,
             new1[ (new1['flag']==2) & (new1['anomaly_score_0.05']==1) ]['temperature_measured'])

plt.scatter (b11['capacity'], b11['temperature_measured'], c="orange", marker="o", )
plt.scatter (b22['capacity'], b22['temperature_measured'], c='red', marker='o', )
plt.scatter (b23['capacity'], b23['temperature_measured'], c='green', marker='o', )
```

```python
plt.xlabel('capacity')
plt.ylabel('temperature_measured')
plt.show()
```



In [730…

```python
plt.scatter( new1[ (new1['flag']==2) & (new1['anomaly_score_0.05']==1) ]['time'] ,
             new1[ (new1['flag']==2) & (new1['anomaly_score_0.05']==1) ]['temperature_measured'])

plt.scatter (b11['time'], b11['temperature_measured'], c="orange", marker="o", )
plt.scatter (b22['time'], b22['temperature_measured'], c='red', marker='o', )
plt.scatter (b23['time'], b23['temperature_measured'], c='green', marker='o', )

plt.xlabel('time')
plt.ylabel('temperature_meaasured')
plt.show()
```



In [731…

```python
#Optional Elliptic envelope
```

In [732…

```python
from sklearn.covariance import EllipticEnvelope
```

In [733…

```python
flagger2=pd.DataFrame()
new2=pd.DataFrame()
for j in df['flag'].unique():

    #Higher contamination values take too much RAM
    contamination = [0.01, 0.05, 0.15, 0.25]

    flagger2= df_c1[df_c1['flag'] == j]
    for i in contamination :
        model = EllipticEnvelope(contamination = i)
        flagger2['anomaly_score' + '_'+str(i)] = model.fit_predict(flagger2)
        flagger2['scores'+'_'+str(i)] = model.decision_function(flagger2[flagger2.columns.difference(['anomaly_sc
    new2 = pd.concat([flagger2, new2])
```

In [734…

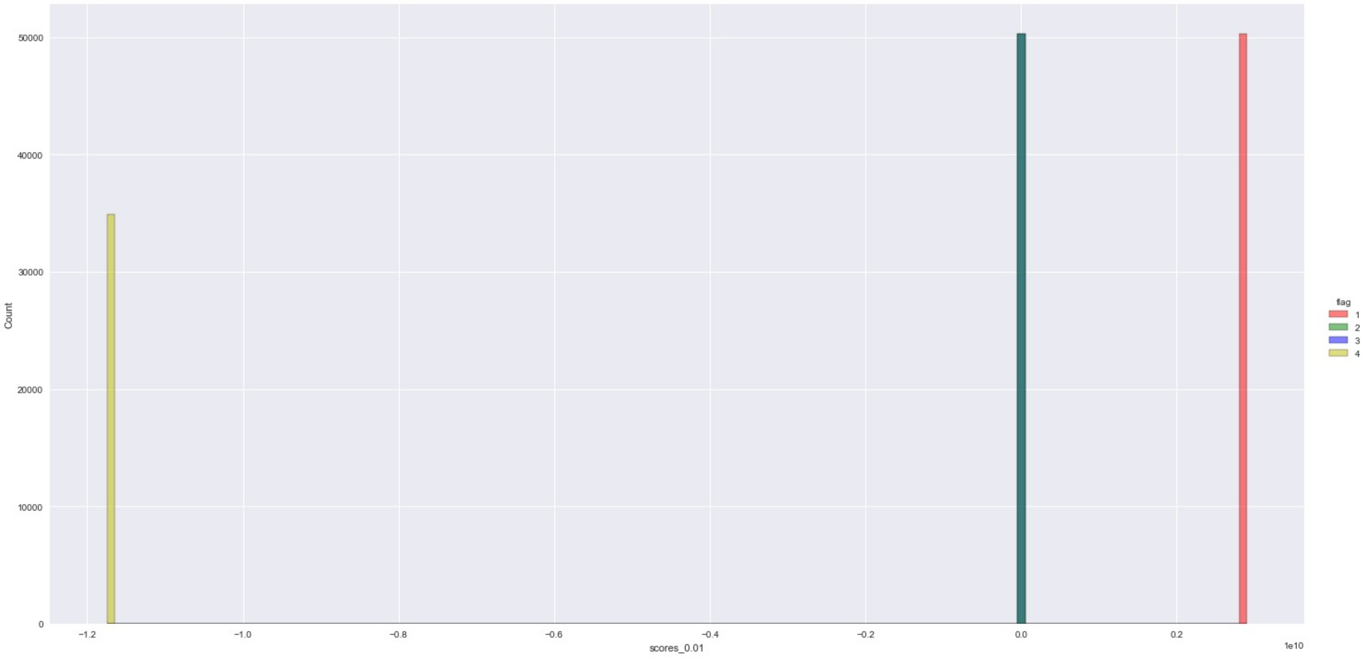
```python
new2.head()
```

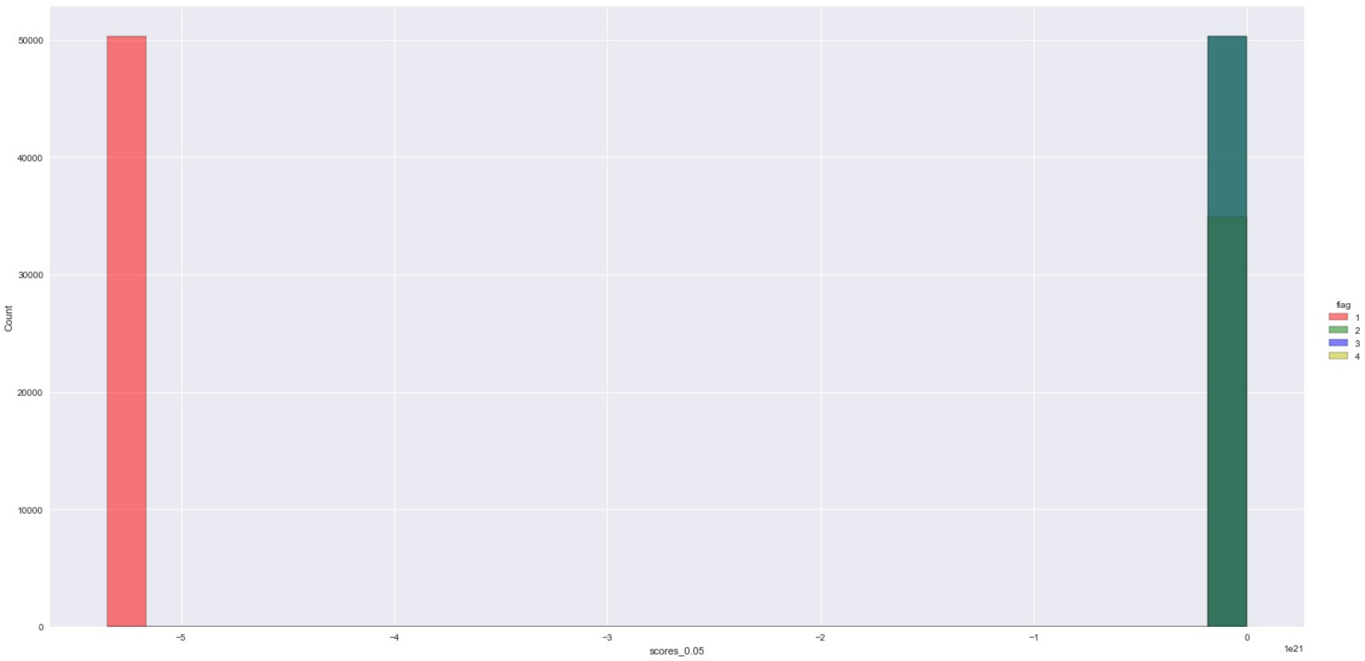
Out[734…

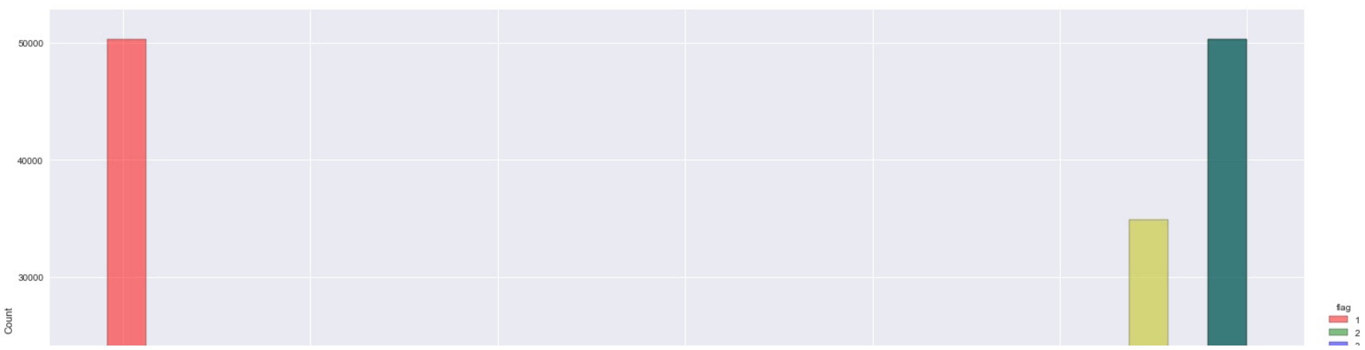
	cycle	capacity	voltage_measured	current_measured	temperature_measured	current_load	voltage_load	time	flag	date_int	anoi
150855	0.0	0.795429	0.981886	0.993063	0.073528	0.50015	0.000000	0.000000	4	0.687055	
150856	0.0	0.795429	0.981921	0.993713	0.073993	0.50015	0.989174	0.002553	4	0.687055	
150857	0.0	0.795429	0.897491	0.011464	0.074801	0.99970	0.712874	0.005305	4	0.687055	
150858	0.0	0.795429	0.891298	0.008267	0.078836	0.99970	0.712168	0.007863	4	0.687055	
150859	0.0	0.795429	0.886436	0.008365	0.083092	0.99970	0.709579	0.010429	4	0.687055	

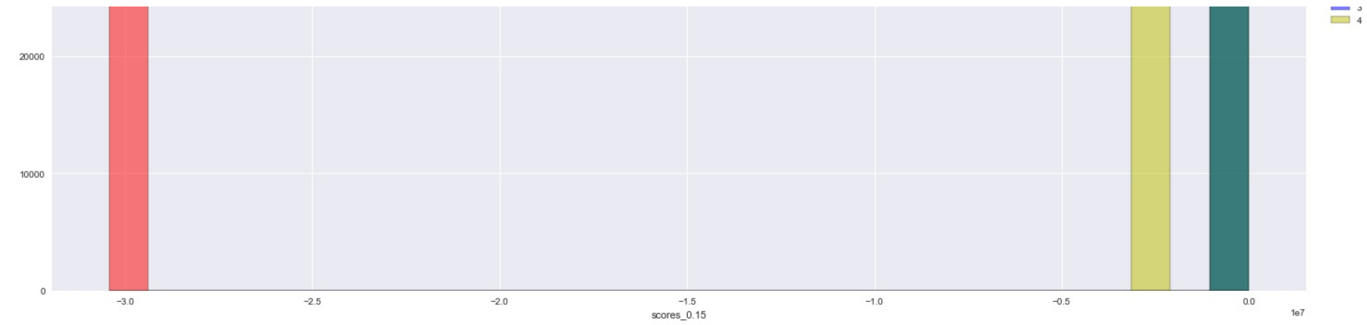
In [735…

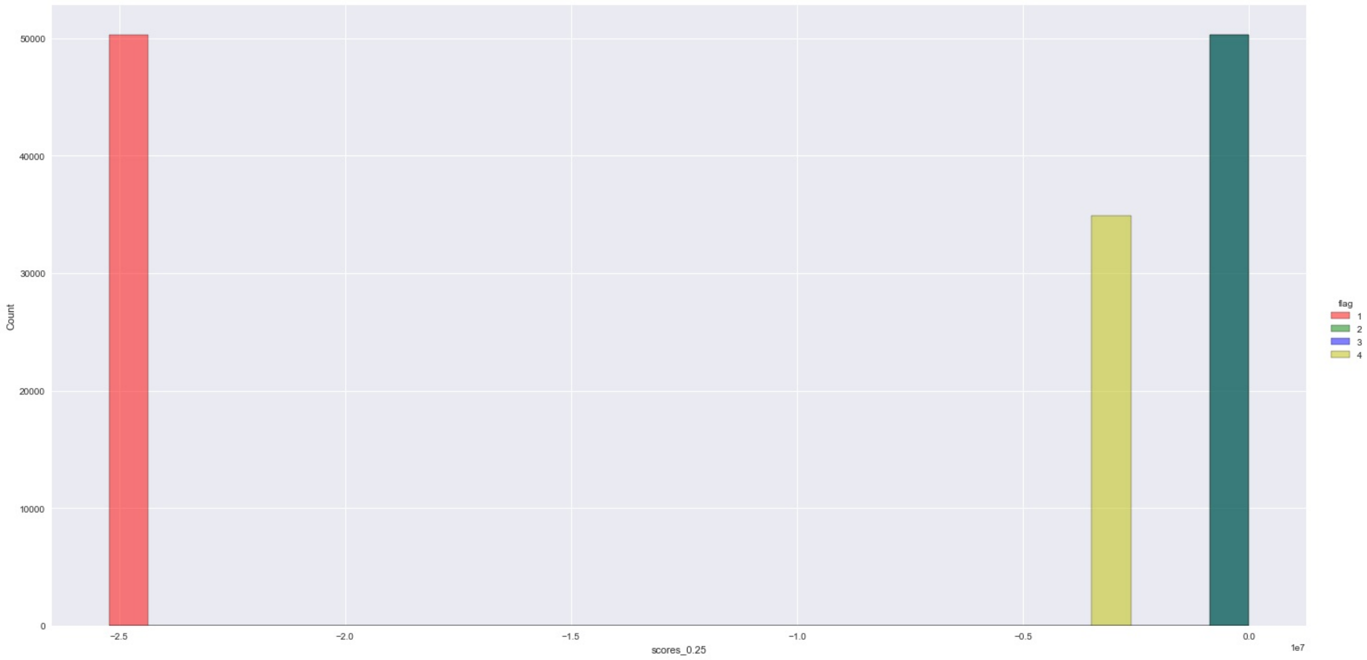
```
a = ['scores_0.01', 'scores_0.05', 'scores_0.15', 'scores_0.25']

for i in range(0, len(a)):
    sns.displot(x=a[i], hue='flag', palette=['r','g','b','y'], data = new2, height =10, aspect=2)
```











In [ ]:

In [ ]:

In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js