In [66]:
```
1
2  import numpy as np # linear algebra
3  import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
4
5
6  import os
7  for dirname, _, filenames in os.walk(r'C:\Users\pmoff\OneDrive\Desktop\PWSkills\MLPro
8      for filename in filenames:
9          print(os.path.join(dirname, filename))
10
11 # You can write up to 20GB to the current directory (/kaggle/working/) that gets pres
12 # You can also write temporary files to /kaggle/temp/, but they won't be saved outsid
```

```
C:\Users\pmoff\OneDrive\Desktop\PWSkills\MLProjects\EV_E_Mobility\.1_Anomaly_Detection_o
n_battery_health_nasa_dataset\Input_files\B0005.mat
C:\Users\pmoff\OneDrive\Desktop\PWSkills\MLProjects\EV_E_Mobility\.1_Anomaly_Detection_o
n_battery_health_nasa_dataset\Input_files\B0006.mat
C:\Users\pmoff\OneDrive\Desktop\PWSkills\MLProjects\EV_E_Mobility\.1_Anomaly_Detection_o
n_battery_health_nasa_dataset\Input_files\B0007.mat
C:\Users\pmoff\OneDrive\Desktop\PWSkills\MLProjects\EV_E_Mobility\.1_Anomaly_Detection_o
n_battery_health_nasa_dataset\Input_files\B0018.mat
```

In [ ]:
```
1
```

Introduction:

- In this notebook, I have **pre-processed ,analyzed, detected and treated several anomalies/outliers** using multiple techniques on the battery charge / discharge dataset obtained from NASA.
- I have used 5 most widely used anomalies/outliers detection techniques such as:
  - IQR based
  - DBSCAN (density based)
  - Isolation Forest
  - Local Outlier Factor (LOF)
  - Elliptical Envelope

Problem Statement:

- NASA is experimenting on making hydrogen Fuel Cell Electric Vehicle (FCEV) for space. An FCEV is powered by a hydrogen fuel cell, which generates electric power using on-board hydrogen. On a full tank, this FCEV can go up to a claimed 600 kilometers.
- They have reached out to the **Data Science team in NASA** for helping them in understanding some parameters related to Hydrogen Fuel cell sensors. As a data scientist working in the R&D team of NASA, **you need to analyze the aging process, charge, and discharge cycle of the cell and isolate some anomalies/ instances you see in the data.**

Data Dictionary:

- Voltage_measured: Fuel Cell terminal voltage (Volts)
- Current_measured: Fuel Cell output current (Amps)
- Temperature_measured: Fuel Cell temperature (degree C)
- Current_load: Current measured at load (Amps)
- Voltage_load: Voltage measured at load (Volts)
- Time: Time vector for the cycle (seconds)
- Capacity: Fuel Cell capacity (Ahr) for discharge till 2.7V
- Datetime: the date and time of the start of the cycle

- Ambient_temperature: Temperature in which the fuel cell is stored (degree C)
- Cycle: Charge cycle of the fuel cell

Data Description: ¶

- A set of four Fuel cells (# 5, 6, 7 and 18) were run through 2 different operational profiles (charge& discharge) at room temperature. Charging was carried out in a constant current (CC) mode at 1.5A until the Fuel cell voltage reached 4.2V and then continued in a constant voltage (CV) mode until the charge current dropped to 20mA. Discharge was carried out at a constant current (CC) level of 2A until the battery voltage fell to 2.7V, 2.5V, 2.2V and 2.5V for Fuel cell 5 6 7 and 18 respectively.
- Repeated charge and discharge cycles result in accelerated aging of the Fuel cell. The experiments were stopped when the Fuel cell reached end-of-life (EOL) criteria, which was a 30% fade in rated capacity (from 2Ahr to 1.4Ahr).
- This data can be further used for the prediction of both remaining charge (for a given discharge cycle) and remaining useful life (RUL).
- Files:
  - B0005.mat Data for Fuel cell #5
  - B0006.mat Data for Fuel cell #6
  - B0007.mat Data for Fuel cell #7
  - B0018.mat Data for Fuel cell #18

Importing libraries:

In [67]:
```
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
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

Converting the matlab files to csv:

In [68]:
```
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(secor
            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, ca
                counter = counter + 1
#     print(dataset[0])
    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 [69]:
```
B0005_dataset, B0005_capacity = load_data('../Input_files/B0005.mat','B0005')
B0005_dataset['flag'] = 1
```

In [70]:
```
B0006_dataset, B0006_capacity = load_data('../Input_files/B0006.mat','B0006')
B0006_dataset['flag'] = 2
```

In [71]:
```
B0007_dataset, B0007_capacity = load_data('../Input_files/B0007.mat','B0007')
B0007_dataset['flag'] = 3
```

In [72]:
```
B0018_dataset, B0018_capacity = load_data('../Input_files/B0018.mat','B0018')
B0018_dataset['flag'] = 4
```

In [73]:
```
fuel_cells_df = pd.concat([B0005_dataset,B0006_dataset,B0007_dataset,B0018_dataset],
```

In [74]:
```
1 fuel_cells_df
```

Out[74]:

	cycle	ambient_temperature	datetime	capacity	voltage_measured	current_measured	temperature_me
0	1	24	2008-04-02 15:25:41	1.856487	4.191492	-0.004902	24
1	2	24	2008-04-02 15:25:41	1.856487	4.190749	-0.001478	24
2	3	24	2008-04-02 15:25:41	1.856487	3.974871	-2.012528	24
3	4	24	2008-04-02 15:25:41	1.856487	3.951717	-2.013979	24
4	5	24	2008-04-02 15:25:41	1.856487	3.934352	-2.011144	24
...	...	...	...	...	...	...	
185716	34862	24	2008-08-20 08:37:19	1.341051	3.443760	-0.002426	35
185717	34863	24	2008-08-20 08:37:19	1.341051	3.453271	-0.000981	35
185718	34864	24	2008-08-20 08:37:19	1.341051	3.461963	0.000209	34
185719	34865	24	2008-08-20 08:37:19	1.341051	3.469907	0.001516	34
185720	34866	24	2008-08-20 08:37:19	1.341051	3.477277	-0.001940	34

185721 rows × 11 columns

Identification of variables and data types:

In [75]:
```
1 fuel_cells_df.shape
```

Out[75]:
```
(185721, 11)
```

In [76]:
```
1 fuel_cells_df.columns
```

Out[76]:
```
Index(['cycle', 'ambient_temperature', 'datetime', 'capacity',
       'voltage_measured', 'current_measured', 'temperature_measured',
       'current_load', 'voltage_load', 'time', 'flag'],
      dtype='object')
```

In [77]:
```
1 fuel_cells_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 185721 entries, 0 to 185720
Data columns (total 11 columns):
 #   Column                Non-Null Count   Dtype
---  ------                --------------   -----
 0   cycle                 185721 non-null  int64
 1   ambient_temperature   185721 non-null  uint8
 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), uint8(1)
memory usage: 14.3 MB
```

Unique values:

In [78]:
```
1 for i in fuel_cells_df.columns:
2     print(i, ":" , fuel_cells_df[i].nunique())
3
```

```
cycle : 50285
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
```

In [79]:
```
1 fuel_cells_df.drop('ambient_temperature', axis = 1, inplace = True)
```

In [80]:
```
1 fuel_cells_df
```

Out[80]:

	cycle	datetime	capacity	voltage_measured	current_measured	temperature_measured	current_load
0	1	2008-04-02 15:25:41	1.856487	4.191492	-0.004902	24.330034	-0.0006
1	2	2008-04-02 15:25:41	1.856487	4.190749	-0.001478	24.325993	-0.0006
2	3	2008-04-02 15:25:41	1.856487	3.974871	-2.012528	24.389085	-1.9982
3	4	2008-04-02 15:25:41	1.856487	3.951717	-2.013979	24.544752	-1.9982
4	5	2008-04-02 15:25:41	1.856487	3.934352	-2.011144	24.731385	-1.9982
...	...	...	...	...	...	...	...
185716	34862	2008-08-20 08:37:19	1.341051	3.443760	-0.002426	35.383979	0.0006
185717	34863	2008-08-20 08:37:19	1.341051	3.453271	-0.000981	35.179732	0.0006
185718	34864	2008-08-20 08:37:19	1.341051	3.461963	0.000209	34.977000	0.0006
185719	34865	2008-08-20 08:37:19	1.341051	3.469907	0.001516	34.785943	0.0006
185720	34866	2008-08-20 08:37:19	1.341051	3.477277	-0.001940	34.581660	0.0006

185721 rows × 10 columns

In [81]:
```
1 fuel_cells_df.describe(include=[np.number]).transpose()
```

Out[81]:

	count	mean	std	min	25%	50%	
cycle	185721.0	23695.670797	14115.216705	1.000000	11608.000000	23216.000000	34823.00
capacity	185721.0	1.574863	0.190633	1.153818	1.426025	1.559634	1.74
voltage_measured	185721.0	3.497219	0.251691	1.737030	3.377653	3.500859	3.65
current_measured	185721.0	-1.832569	0.561405	-2.029098	-2.011418	-2.009015	-1.98
temperature_measured	185721.0	32.378997	4.027737	22.350256	29.570621	32.355737	35.42
current_load	185721.0	1.465434	1.226874	-2.000000	1.998200	1.998800	1.99
voltage_load	185721.0	2.366494	0.751377	0.000000	2.410000	2.558000	2.71
time	185721.0	1546.379935	906.958628	0.000000	764.797000	1537.031000	2307.48
flag	185721.0	2.375466	1.073068	1.000000	1.000000	2.000000	3.00

In [82]:
```
# Missing values:

def missingValue(df):
    #Identifying Missing data.
    total_null = df.isnull().sum().sort_values(ascending = False)
    percent = ((df.isnull().sum()/len(df))*100).sort_values(ascending = False)
    print(f"Total records in our data =  {df.shape[0]} where missing values are as fo

    missing_data = pd.concat([total_null,percent.round(2)],axis=1,keys=['Total Missir
    return missing_data
```

In [83]:
```
missing_df = missingValue(fuel_cells_df)
missing_df[missing_df['Total Missing'] > 0]
```

```
Total records in our data =  185721 where missing values are as follows:
```

Out[83]:

	Total Missing	In Percent

In [ ]:
```

```

Univariate Analysis:

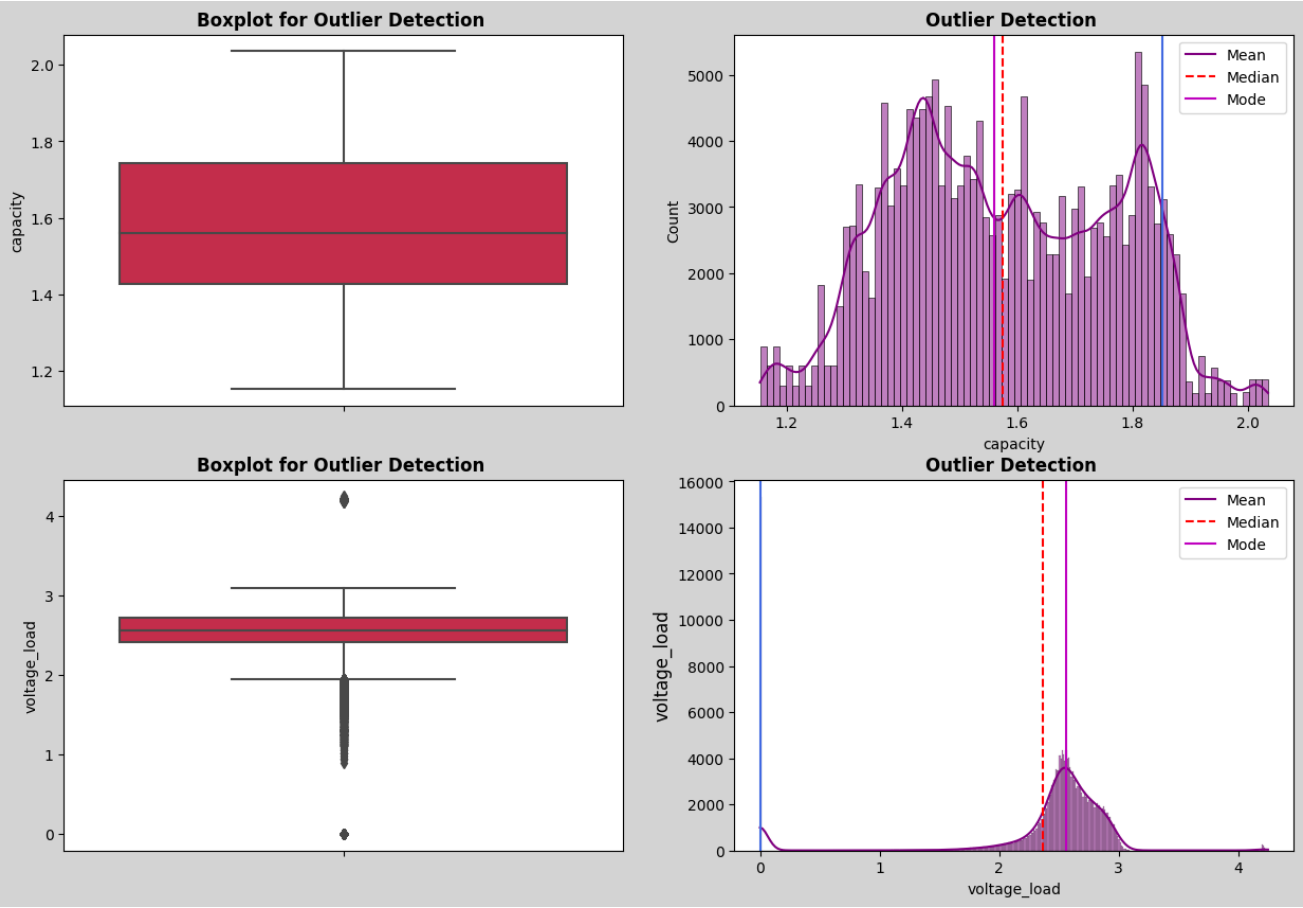
In [84]:
```
def numerical_feat(df,colname,nrows=2,mcols=2,width=15,height=70):
    fig , ax = plt.subplots(nrows,mcols,figsize=(width,height))
    fig.set_facecolor("lightgrey")
    rows = 0
    for var in colname:
        ax[rows][0].set_title("Boxplot for Outlier Detection ", fontweight="bold")
        plt.ylabel(var, fontsize=12)
        sns.boxplot(y = df[var],color='crimson',ax=ax[rows][0])

        sns.histplot(df[var],color='purple',ax=ax[rows][1],kde=True)
        ax[rows][1].axvline(df[var].mean(), color='r', linestyle='--', label="Mean")
        ax[rows][1].axvline(df[var].median(), color='m', linestyle='-', label="Median
        ax[rows][1].axvline(df[var].mode()[0], color='royalblue', linestyle='-', labe
        ax[rows][1].set_title("Outlier Detection ", fontweight="bold")
        ax[rows][1].legend({'Mean':df[var].mean(),'Median':df[var].median(),'Mode':df
        rows += 1
    plt.show()
```

In [85]:
```
fuel_cells_df.columns
```

Out[85]:
```
Index(['cycle', 'datetime', 'capacity', 'voltage_measured', 'current_measured',
       'temperature_measured', 'current_load', 'voltage_load', 'time', 'flag'],
      dtype='object')
```

In [86]:
```
numerical_cols = ['capacity', 'voltage_load']
```

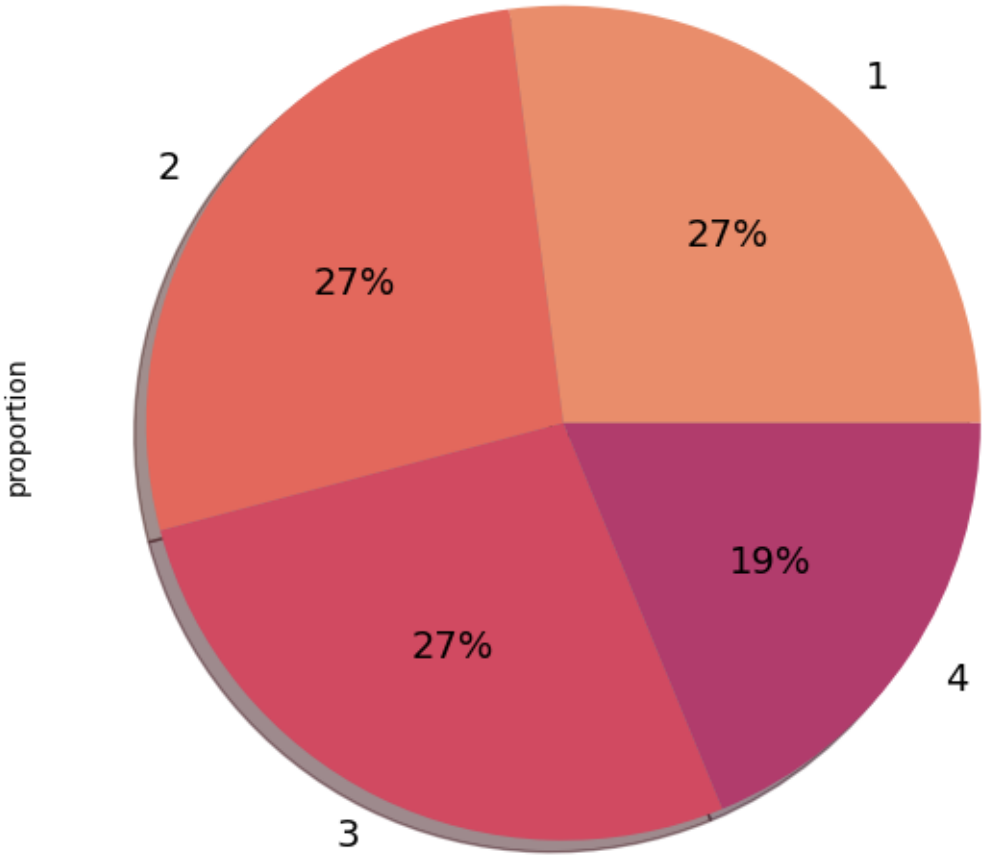
In [87]:
```
numerical_feat(fuel_cells_df,numerical_cols,len(numerical_cols),2,15,10)
```



- As we can see, the voltage load gas near normal distribution and has few outliers to the lower end.
- Whereas, capacity is multimodal distributed having almost zero outliers.

In [88]:
```
categorical_cols = ['flag']
```

In [89]:
```
plt.figure(figsize = (7,8))
count = (fuel_cells_df['flag'].value_counts(normalize=True)*100)
count.plot.pie(colors = sns.color_palette("flare"),autopct='%0.0f%%',
                        textprops={'fontsize': 14},shadow = True)
plt.show()
```



- Battery 4 has least no. of datapoints, whereas the other three have equal no. of dp.

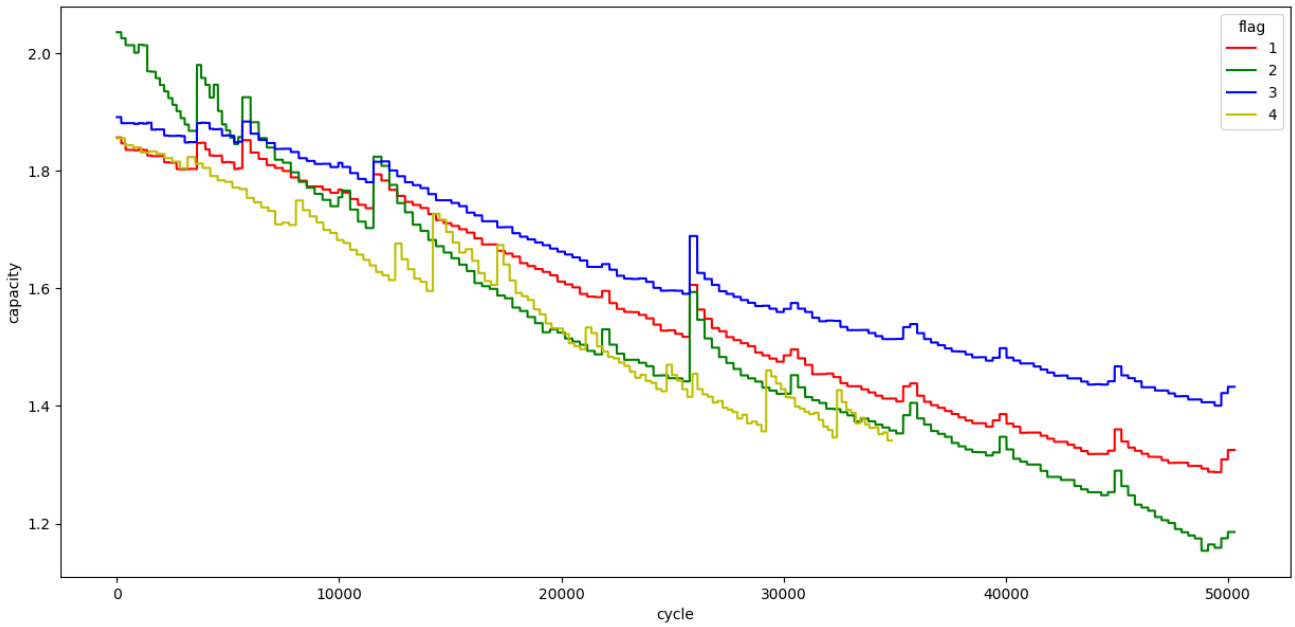
Bivariate Analysis:

In [90]:
```
fuel_cells_df.columns
```

Out[90]:
```
Index(['cycle', 'datetime', 'capacity', 'voltage_measured', 'current_measured',
       'temperature_measured', 'current_load', 'voltage_load', 'time', 'flag'],
      dtype='object')
```

1. Capacity w.r.t Cycle

In [91]:
```
plt.figure(figsize=(15, 7))
sns.lineplot(x = 'cycle', y = 'capacity', data = fuel_cells_df, palette = ['r','g','b
plt.show()
```

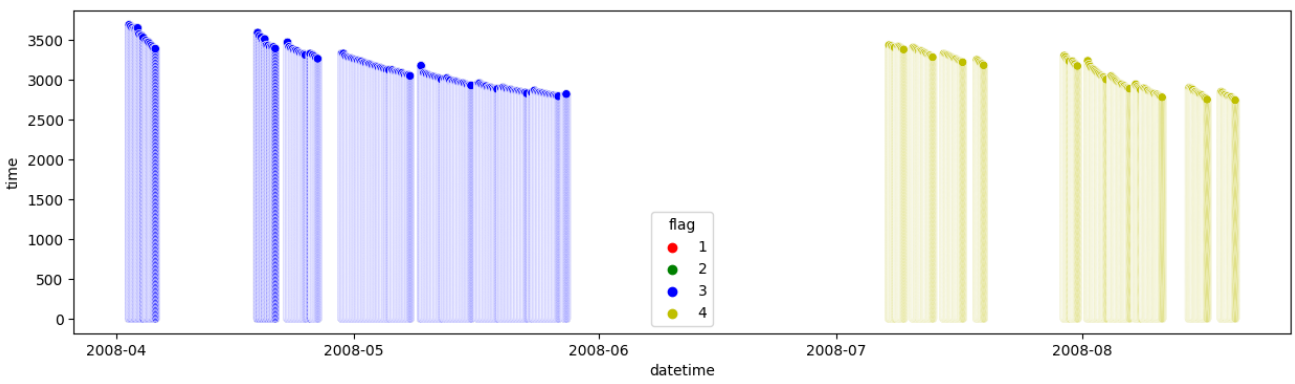


- Batteries:
  - B0005.mat Data for Fuel cell #5
  - B0006.mat Data for Fuel cell #6
  - B0007.mat Data for Fuel cell #7
  - B0018.mat Data for Fuel cell #18

- As the cycle increases, the batterry capacities is showing a decreasing trend.
- Capacity of batteries decrease as no. of cycles of battery usage increases
- Battery B0006 is having the largest capacity when it's new. Whereas B0018 and B0007 is having the least capacity when it's unused.
- Over a period of time, B0006 shows much more deteriorating trend than others , whereas B0007 has the most efficient capacity.

2. Datetime w.r.t time

In [92]:
```
plt.figure(figsize=(15, 4))
sns.scatterplot(x = 'datetime', y = 'time', data = fuel_cells_df, palette = ['r','g',
plt.show()
```

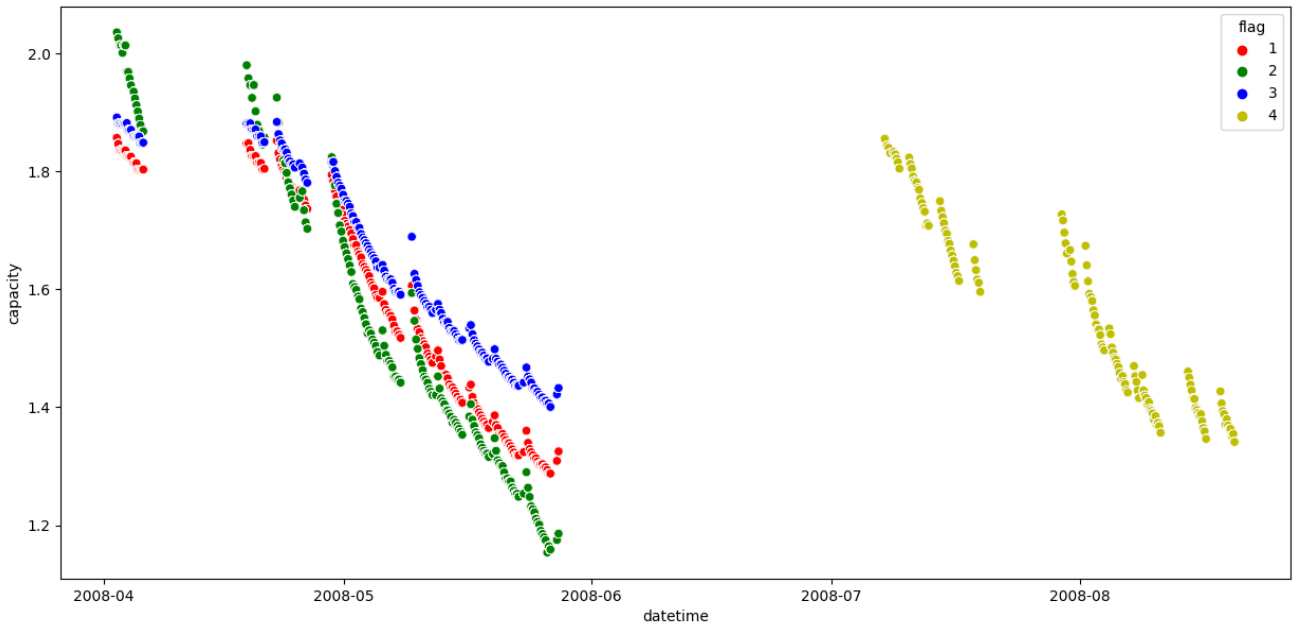


- There hasn't been a continuos testing og charging of batteries throught the cycle.

- First three batteries (5,6,7) have been charged/tested synchronously whereas the 4th battery (B0018) is charges/tested post July 2008 with couple of breaks in between.
- The gerenal trend is more or less the same which is as time of battery's cycle charging decreases over a period of time.

3.Datetime w.r.t capacity

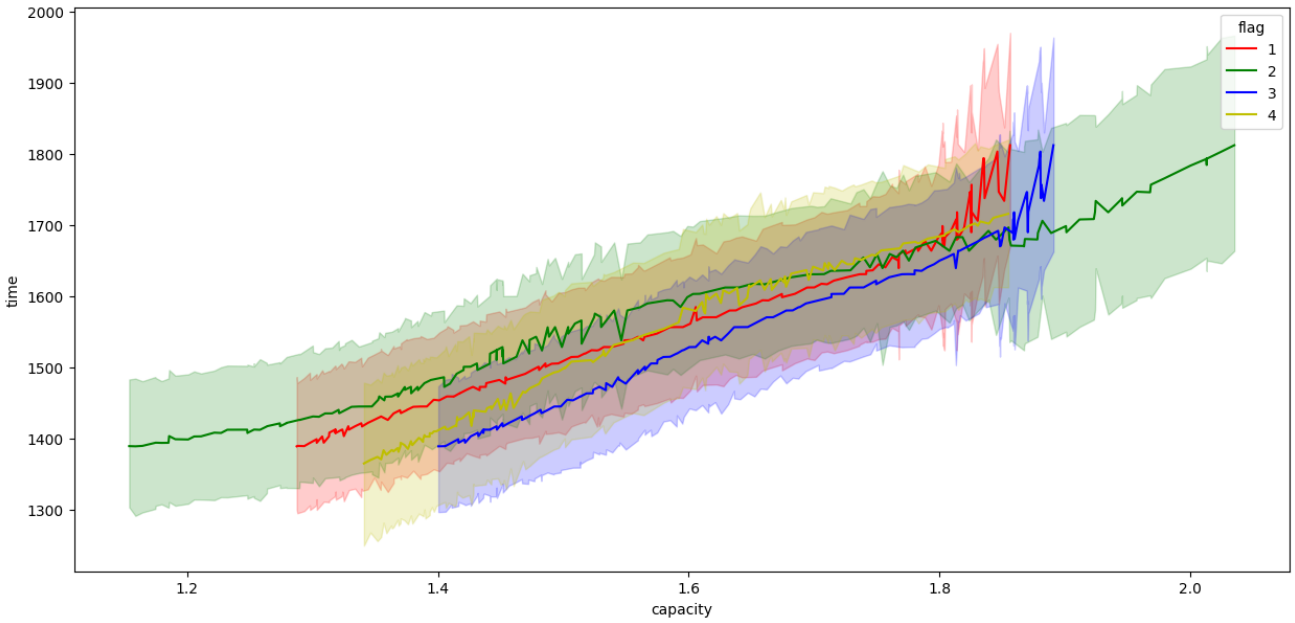
In [93]:
```
plt.figure(figsize=(15, 7))
sns.scatterplot(x = 'datetime', y = 'capacity', data = fuel_cells_df, palette = ['r',
plt.show()
```



- Capacity decreases over a period of time.
- We can clearly see that, Battery (B0018) has been launched recently and is having average capacity over time.
- Capacity of Battery B0002 deteriorates the most over time whereas for blue it's most efficient in terms of capacity over a period of time.

4. Capacity w.r.t time

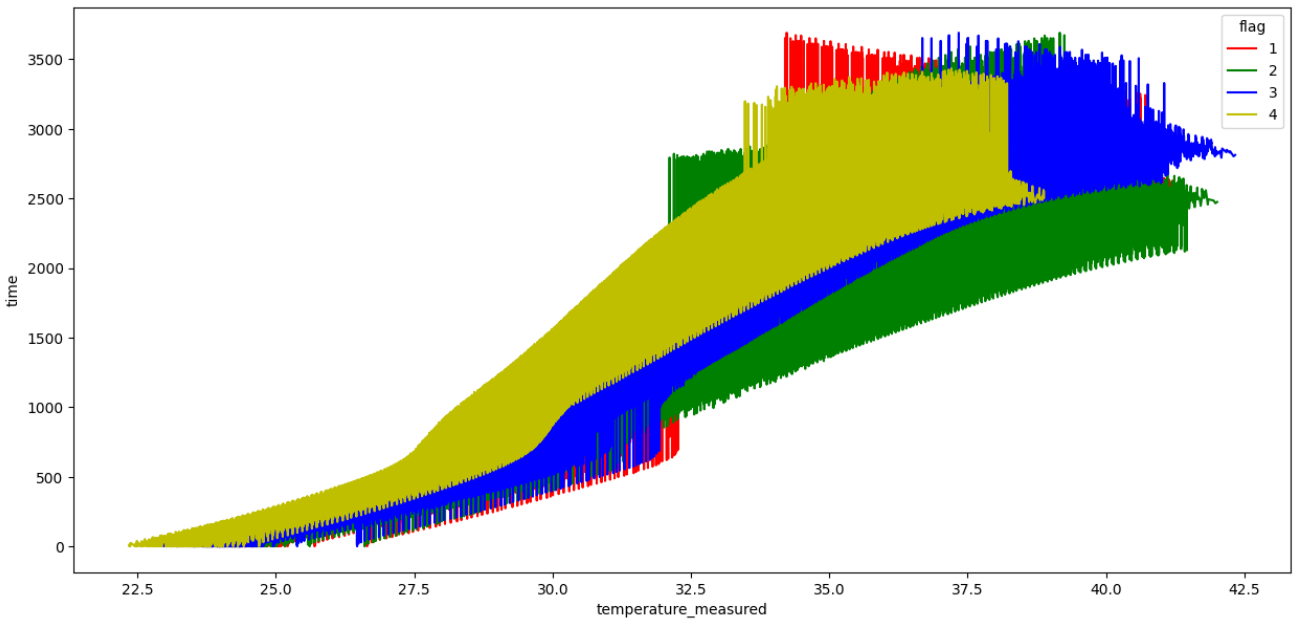
In [94]:
```
plt.figure(figsize=(15, 7))
sns.lineplot(x = 'capacity', y = 'time', data = fuel_cells_df, palette = ['r','g','b'
plt.show()
```



- As time to charge the batteries is increasing, the capacity is also increasing on an average for all four batteries

5. Temperature w.r.t time

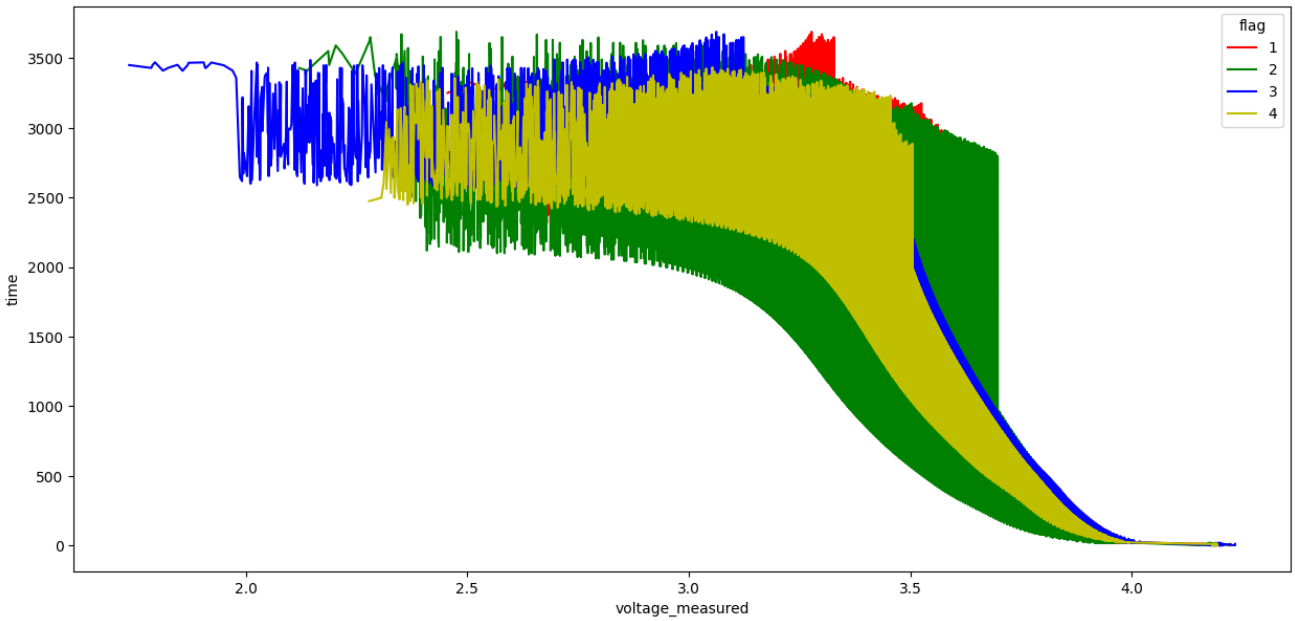
In [95]:
```
plt.figure(figsize=(15, 7))
sns.lineplot(x = 'temperature_measured', y = 'time', data = fuel_cells_df, palette =
plt.show()
```



- As time increases for batteries, the depreciation increases because, temeperature_measured increases in every cycle of charging.
- As clearly visible, Batter (B0006) with flag 2 has max no. of outliers

6. Voltage measured w.r.t time

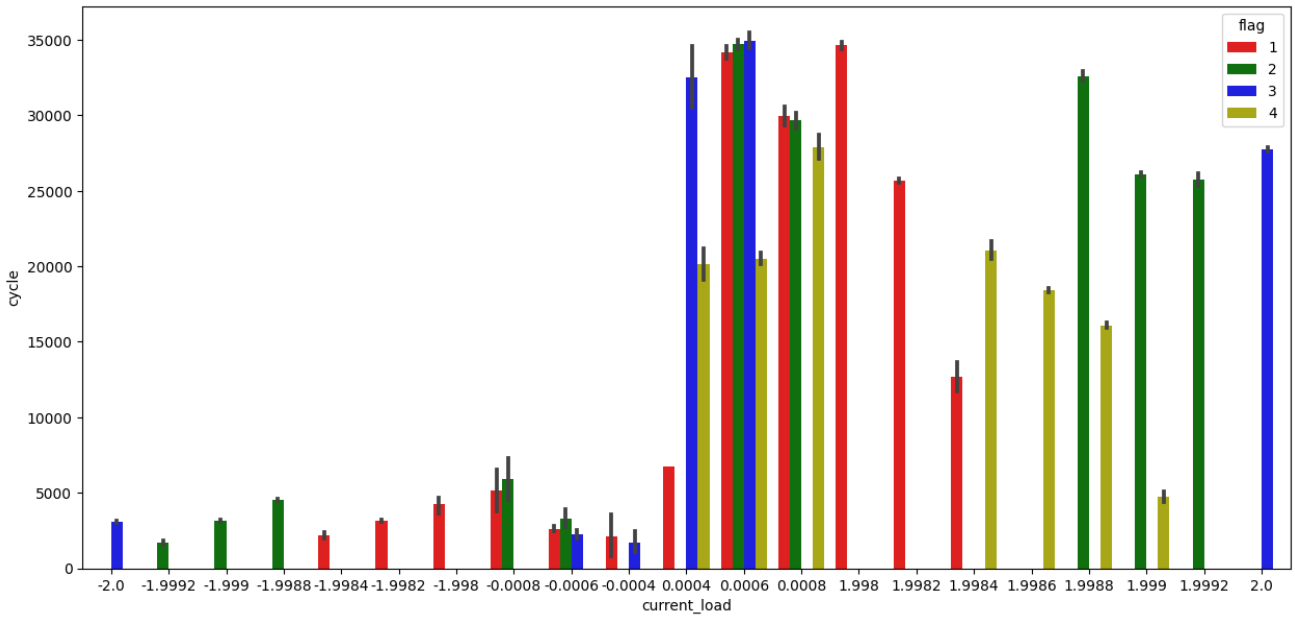
In [96]:
```
plt.figure(figsize=(15, 7))
sns.lineplot(x = 'voltage_measured', y = 'time', data = fuel_cells_df, palette = ['r'
plt.show()
```



- As clearly visible, Batter (B0006) with flag 2 has max no. of outliers

7. Current measured w.r.t time

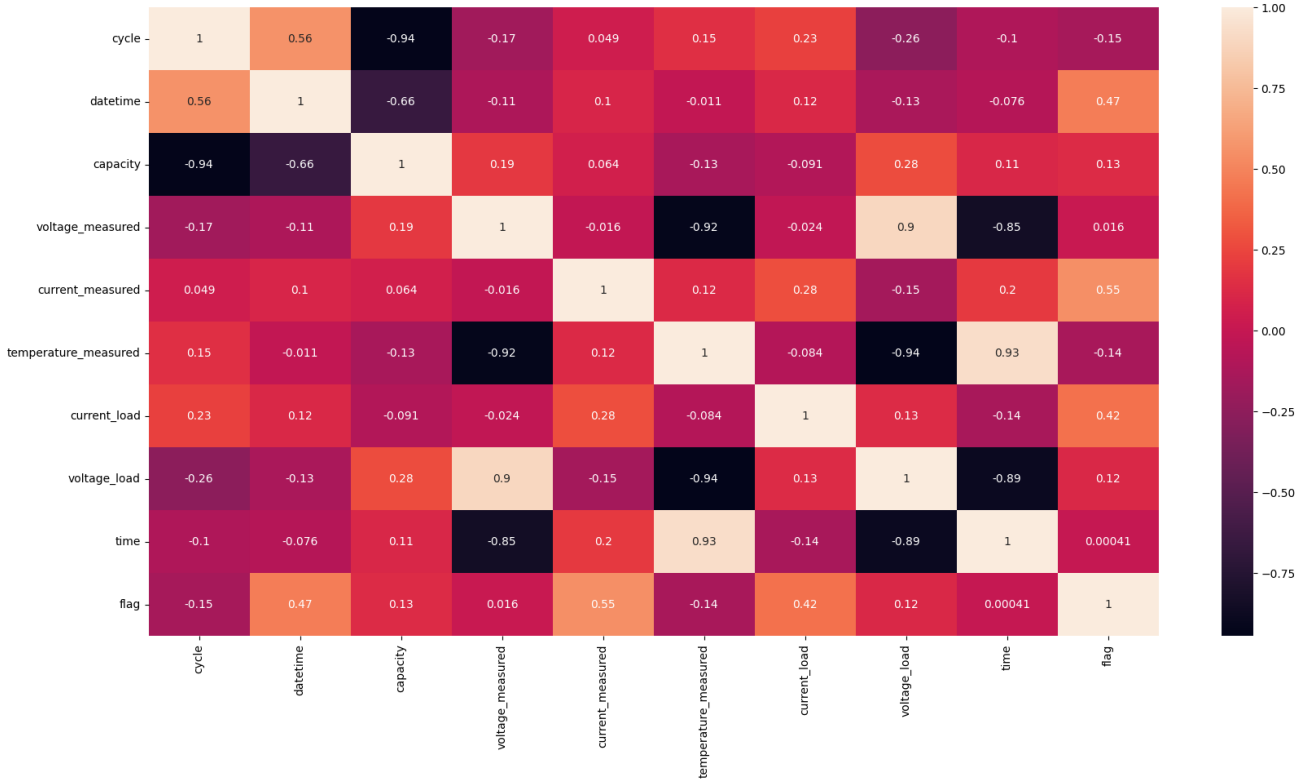
In [97]:
```
plt.figure(figsize=(15, 7))
sns.barplot(x = 'current_load', y = 'cycle', data = fuel_cells_df, palette = ['r','g'
plt.show()
```



- As clearly visible, Batter (B0006) with flag 2 has max no. of outliers

8. Correlation between various features

In [98]:
```
fig, ax = plt.subplots(figsize = (20,10))
sns.heatmap(fuel_cells_df.corr(method = 'spearman'), annot = True)
plt.show()
```



- cycle & capacity -> negatively correlated.
- volatage load and voltage_measured -> negatively correlated.
- voltage_measured and time -> positively correlated.
- temperature and time -> positively correlated.

In [99]:
```
fuel_cells_df.shape
```

Out[99]:
```
(185721, 10)
```

In [100]:
```
fuel_cells_df.drop_duplicates(keep = 'first', inplace = True)
fuel_cells_df.shape
```

Out[100]:
```
(185721, 10)
```

In [101]:
```
fuel_cells_df.dropna(inplace = True)
```

In [102]:
```
fuel_cells_df.shape
```

Out[102]:
```
(185721, 10)
```

In [103]:
```
fuel_cells_df.columns
```

Out[103]:
```
Index(['cycle', 'datetime', 'capacity', 'voltage_measured', 'current_measured',
       'temperature_measured', 'current_load', 'voltage_load', 'time', 'flag'],
      dtype='object')
```

Visual analysis using traditional methods of anomaly detection(IQR and DBSCAN)

1. IQR based anomaly detection

In [104]:
```
num_cols_1 = ['capacity', 'voltage_measured','voltage_load']
```

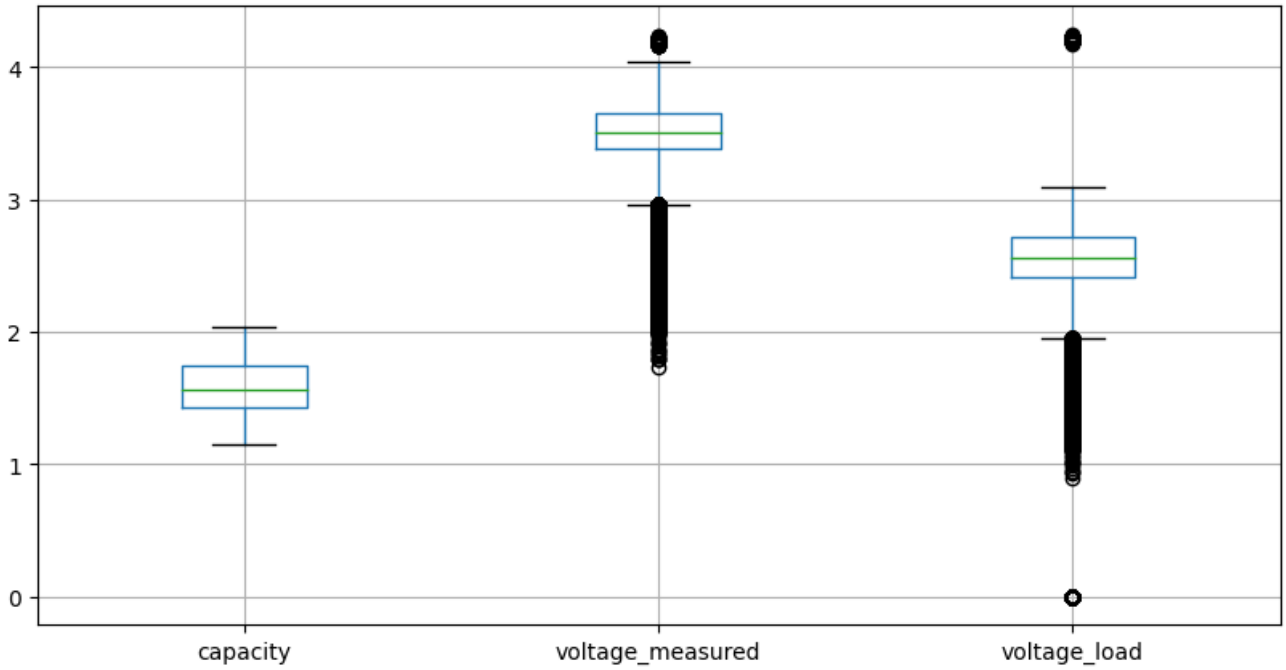
In [105]:
```
num_cols_2 = ['current_measured','current_load']
temp = ['temperature_measured']
```

In [106]:
```
num_cols_3 = ['cycle','time']
```

In [107]:
```
Q1 = fuel_cells_df.quantile(0.25)
Q3 = fuel_cells_df.quantile(0.75)
IQR = Q3 - Q1
print(IQR)
```

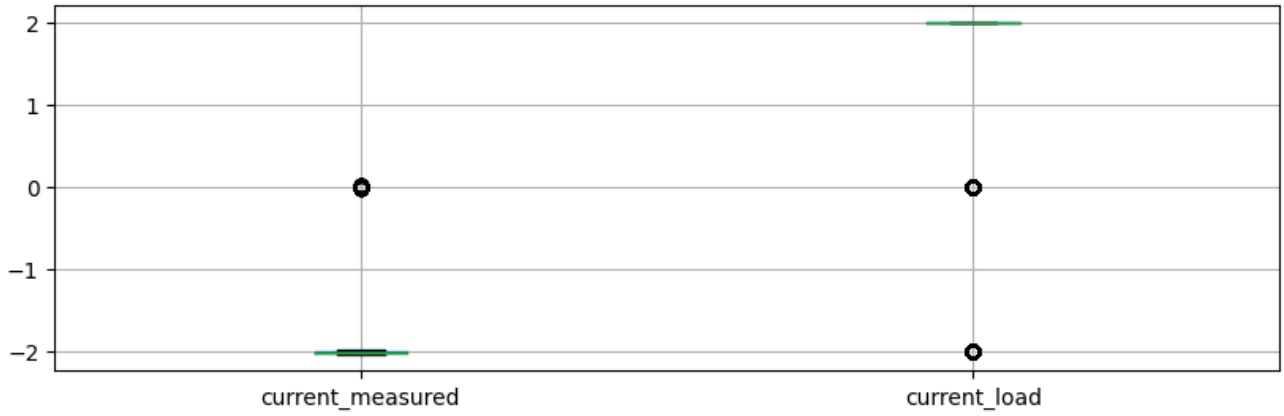
```
cycle                          23215.0
datetime                23 days 02:10:03
capacity                       0.315825
voltage_measured               0.278098
current_measured               0.021444
temperature_measured           5.850056
current_load                     0.0008
voltage_load                      0.308
time                           1542.688
flag                                2.0
dtype: object
```

In [108]:
```
fuel_cells_df[num_cols_1].boxplot(figsize = (10,5))
plt.show()
```



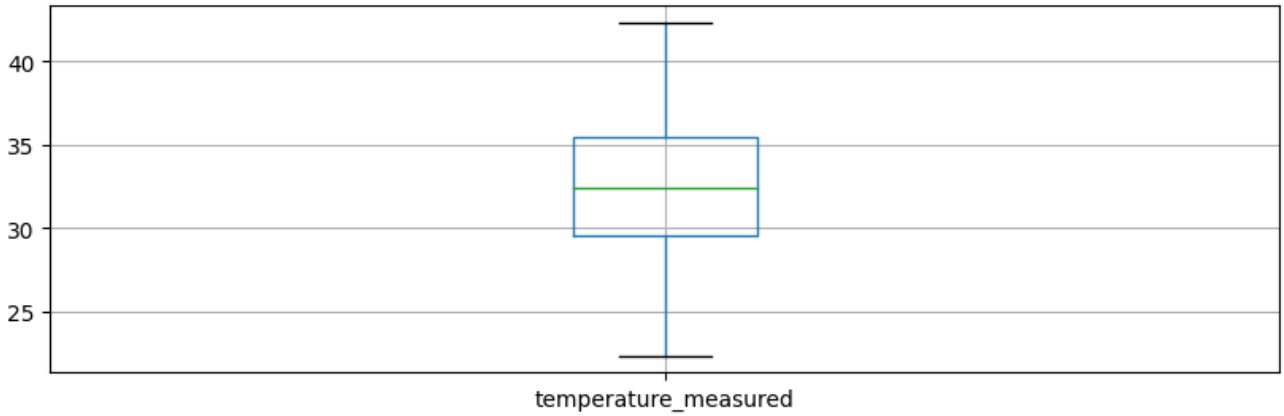
- 'voltage_measured','voltage_load' have more no. of outliers (below the lower whisker)

In [109]:
```
1 fuel_cells_df[num_cols_2].boxplot( figsize = (10,3))
2 plt.show()
```

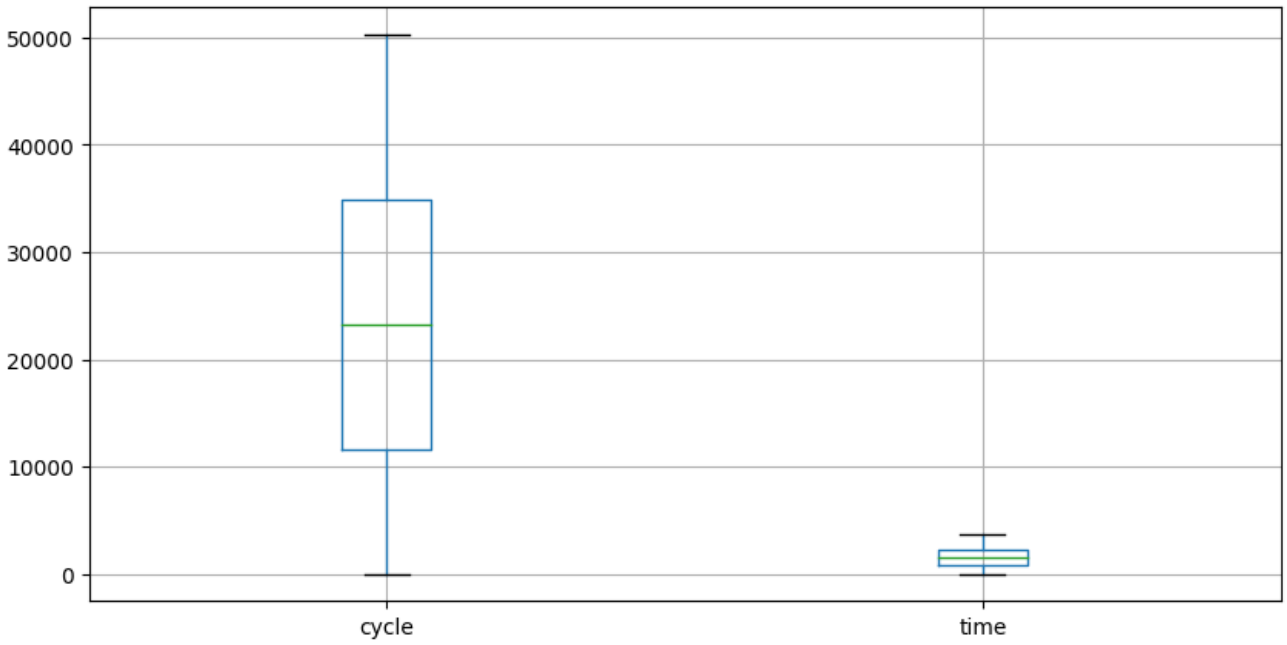


- As the dataponts are less, the IQR range is very small, but some outliers are present for 'current_measured','current_load'

In [110]:
```
1 fuel_cells_df[temp].boxplot(figsize = (10,3))
2 plt.show()
```



In [111]:
```
1 fuel_cells_df[num_cols_3].boxplot(figsize = (10,5))
2 plt.show()
```



- All other features are having very less outliers

In [112]:
```
num_cols = ['cycle','capacity', 'voltage_measured', 'current_measured',
       'temperature_measured', 'current_load', 'voltage_load', 'time']
```

In [113]:
```
fuel_cells_df.shape
```

Out[113]: (185721, 10)

In [114]:
```
fuel_cells_df1 = fuel_cells_df.copy()
```

Outliers treatment using IQR based method

In [115]:
```
outliers_mask = ((fuel_cells_df1[num_cols] < (Q1 - 1.5 * IQR)) | (fuel_cells_df1[num_
fuel_cells_df1 = fuel_cells_df1[~outliers_mask].reset_index(drop=True)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[115], line 1
----> 1 outliers_mask = ((fuel_cells_df1[num_cols] < (Q1 - 1.5 * IQR)) | (fuel_cells_df1
[num_cols] > (Q3 + 1.5 * IQR))).any(axis=1)
      2 fuel_cells_df1 = fuel_cells_df1[~outliers_mask].reset_index(drop=True)

File c:\Users\pmoff\OneDrive\Desktop\PWSkills\MLProjects\EV_E_Mobility\venv\lib\site-pac
kages\pandas\core\ops\common.py:81, in _unpack_zerodim_and_defer.<locals>.new_method(sel
f, other)
     77             return NotImplemented
     79 other = item_from_zerodim(other)
---> 81 return method(self, other)

File c:\Users\pmoff\OneDrive\Desktop\PWSkills\MLProjects\EV_E_Mobility\venv\lib\site-pac
kages\pandas\core\arraylike.py:48, in OpsMixin.__lt__(self, other)
     46 @unpack_zerodim_and_defer("__lt__")
     47 def __lt__(self, other):
---> 48     return self._cmp_method(other, operator.lt)

File c:\Users\pmoff\OneDrive\Desktop\PWSkills\MLProjects\EV_E_Mobility\venv\lib\site-pac
kages\pandas\core\frame.py:7442, in DataFrame._cmp_method(self, other, op)
   7439 def _cmp_method(self, other, op):
   7440     axis: Literal[1] = 1  # only relevant for Series other case
-> 7442     self, other = ops.align_method_FRAME(self, other, axis, flex=False, level=No
ne)
   7444     # See GH#4537 for discussion of scalar op behavior
   7445     new_data = self._dispatch_frame_op(other, op, axis=axis)

File c:\Users\pmoff\OneDrive\Desktop\PWSkills\MLProjects\EV_E_Mobility\venv\lib\site-pac
kages\pandas\core\ops\__init__.py:323, in align_method_FRAME(left, right, axis, flex, le
vel)
    321 if not flex:
    322     if not left.axes[axis].equals(right.index):
--> 323         raise ValueError(
    324             "Operands are not aligned. Do "
    325             "`left, right = left.align(right, axis=1, copy=False)` "
    326             "before operating."
    327         )
    329 left, right = left.align(
    330     right, join="outer", axis=axis, level=level, copy=False
    331 )
    332 right = _maybe_align_series_as_frame(left, right, axis)

ValueError: Operands are not aligned. Do `left, right = left.align(right, axis=1, copy=F
alse)` before operating.
```

In [58]:

```
1 outlier_filter = ((fuel_cells_df1[num_cols] < (Q1 - 1.5 * IQR)) | (fuel_cells_df1[num
2 fuel_cells_df1 = fuel_cells_df1[~outlier_filter].reset_index(drop=True)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[58], line 1
----> 1 outlier_filter = ((fuel_cells_df1[num_cols] < (Q1 - 1.5 * IQR)) | (fuel_cells_df
1[num_cols] > (Q3 + 1.5 * IQR))).any(axis=1)
      2 fuel_cells_df1 = fuel_cells_df1[~outlier_filter].reset_index(drop=True)

File c:\Users\pmoff\OneDrive\Desktop\PWSkills\MLProjects\EV_E_Mobility\venv\lib\site-pac
kages\pandas\core\ops\common.py:81, in _unpack_zerodim_and_defer.<locals>.new_method(sel
f, other)
     77             return NotImplemented
     79 other = item_from_zerodim(other)
---> 81 return method(self, other)

File c:\Users\pmoff\OneDrive\Desktop\PWSkills\MLProjects\EV_E_Mobility\venv\lib\site-pac
kages\pandas\core\arraylike.py:48, in OpsMixin.__lt__(self, other)
     46 @unpack_zerodim_and_defer("__lt__")
     47 def __lt__(self, other):
---> 48     return self._cmp_method(other, operator.lt)

File c:\Users\pmoff\OneDrive\Desktop\PWSkills\MLProjects\EV_E_Mobility\venv\lib\site-pac
kages\pandas\core\frame.py:7442, in DataFrame._cmp_method(self, other, op)
   7439 def _cmp_method(self, other, op):
   7440     axis: Literal[1] = 1  # only relevant for Series other case
-> 7442     self, other = ops.align_method_FRAME(self, other, axis, flex=False, level=No
ne)
   7444     # See GH#4537 for discussion of scalar op behavior
   7445     new_data = self._dispatch_frame_op(other, op, axis=axis)

File c:\Users\pmoff\OneDrive\Desktop\PWSkills\MLProjects\EV_E_Mobility\venv\lib\site-pac
kages\pandas\core\ops\__init__.py:323, in align_method_FRAME(left, right, axis, flex, le
vel)
    321 if not flex:
    322     if not left.axes[axis].equals(right.index):
--> 323         raise ValueError(
    324             "Operands are not aligned. Do "
    325             "`left, right = left.align(right, axis=1, copy=False)` "
    326             "before operating."
    327         )
    329 left, right = left.align(
    330     right, join="outer", axis=axis, level=level, copy=False
    331 )
    332 right = _maybe_align_series_as_frame(left, right, axis)

ValueError: Operands are not aligned. Do `left, right = left.align(right, axis=1, copy=F
alse)` before operating.
```

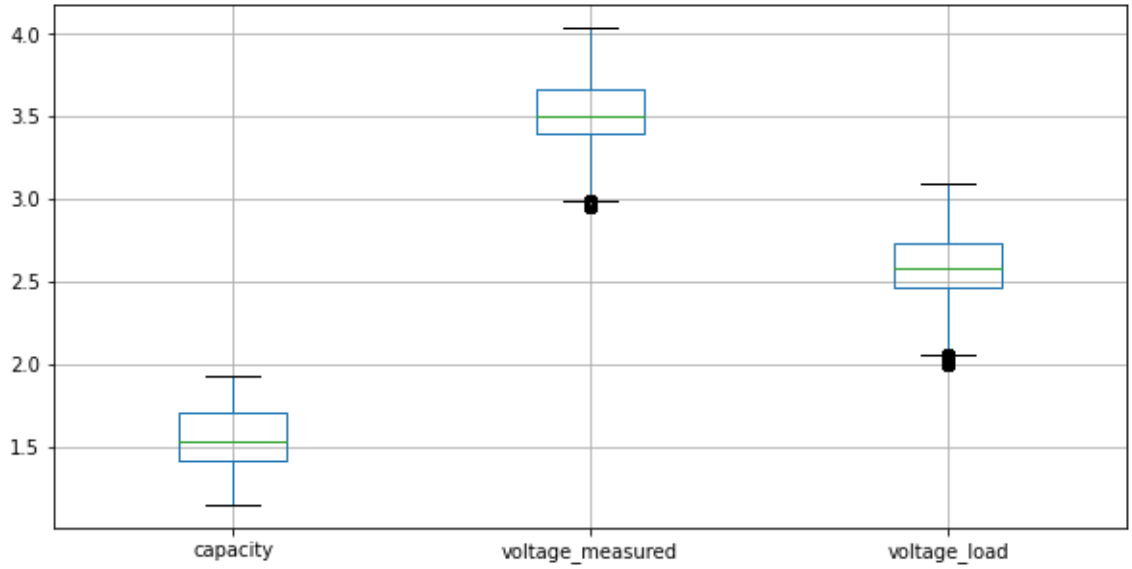
In [56]:
```
1 Outlier_filter = ~((fuel_cells_df1[num_cols] < (Q1 - 1.5*IQR)) | (fuel_cells_df1[num_
2 fuel_cells_df1 = fuel_cells_df1[Outlier_filter].reset_index(drop = True)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[56], line 1
----> 1 Outlier_filter = ~((fuel_cells_df1[num_cols] < (Q1 - 1.5*IQR)) | (fuel_cells_df1
[num_cols] > (Q3 +1.5*IQR))).any(axis = 1)
      2 fuel_cells_df1 = fuel_cells_df1[Outlier_filter].reset_index(drop = True)

File c:\Users\pmoff\OneDrive\Desktop\PWSkills\MLProjects\EV_E_Mobility\venv\lib\site-pac
kages\pandas\core\ops\common.py:81, in _unpack_zerodim_and_defer.<locals>.new_method(sel
f, other)
     77             return NotImplemented
     79 other = item_from_zerodim(other)
---> 81 return method(self, other)

File c:\Users\pmoff\OneDrive\Desktop\PWSkills\MLProjects\EV_E_Mobility\venv\lib\site-pac
kages\pandas\core\arraylike.py:48, in OpsMixin.__lt__(self, other)
     46 @unpack_zerodim_and_defer("__lt__")
     47 def __lt__(self, other):
---> 48     return self._cmp_method(other, operator.lt)

File c:\Users\pmoff\OneDrive\Desktop\PWSkills\MLProjects\EV_E_Mobility\venv\lib\site-pac
kages\pandas\core\frame.py:7442, in DataFrame._cmp_method(self, other, op)
   7439 def _cmp_method(self, other, op):
   7440     axis: Literal[1] = 1  # only relevant for Series other case
-> 7442     self, other = ops.align_method_FRAME(self, other, axis, flex=False, level=No
ne)
   7444     # See GH#4537 for discussion of scalar op behavior
   7445     new_data = self._dispatch_frame_op(other, op, axis=axis)

File c:\Users\pmoff\OneDrive\Desktop\PWSkills\MLProjects\EV_E_Mobility\venv\lib\site-pac
kages\pandas\core\ops\__init__.py:323, in align_method_FRAME(left, right, axis, flex, le
vel)
    321 if not flex:
    322     if not left.axes[axis].equals(right.index):
--> 323         raise ValueError(
    324             "Operands are not aligned. Do "
    325             "`left, right = left.align(right, axis=1, copy=False)` "
    326             "before operating."
    327         )
    329 left, right = left.align(
    330     right, join="outer", axis=axis, level=level, copy=False
    331 )
    332 right = _maybe_align_series_as_frame(left, right, axis)

ValueError: Operands are not aligned. Do `left, right = left.align(right, axis=1, copy=F
alse)` before operating.
```

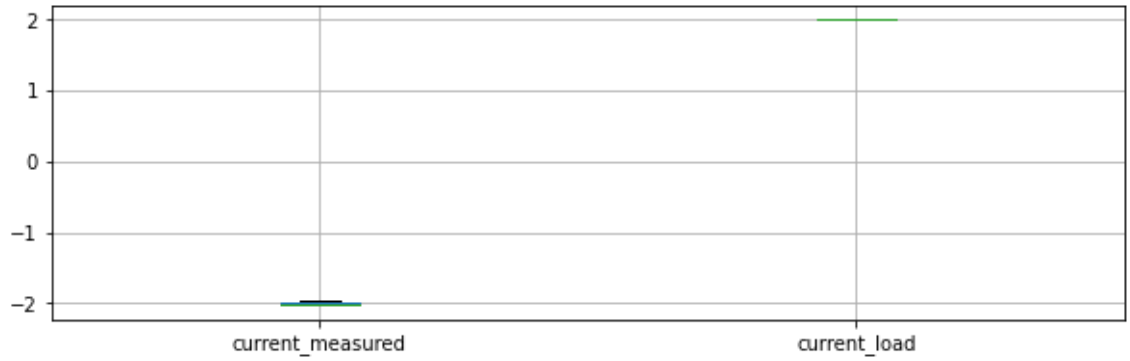
In [51]:
```
1 fuel_cells_df1.shape
```

Out[51]: (147955, 10)

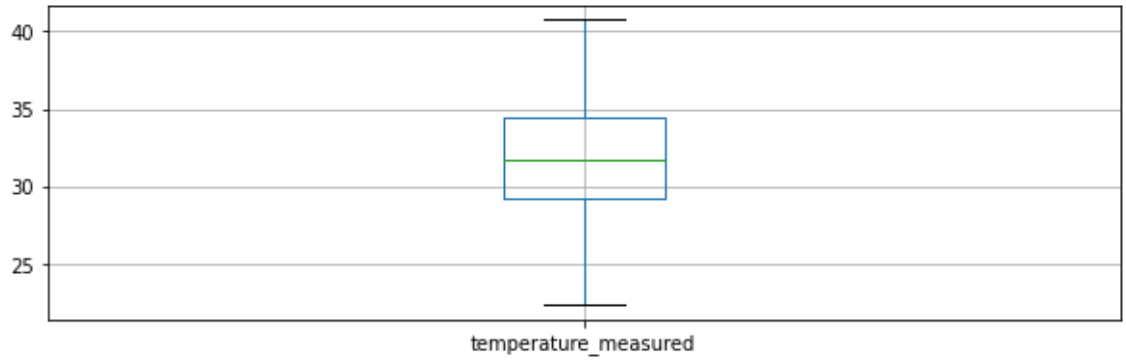
In [52]:
```
1 fuel_cells_df1[num_cols_1].boxplot(figsize = (10,5))
2 plt.show()
```



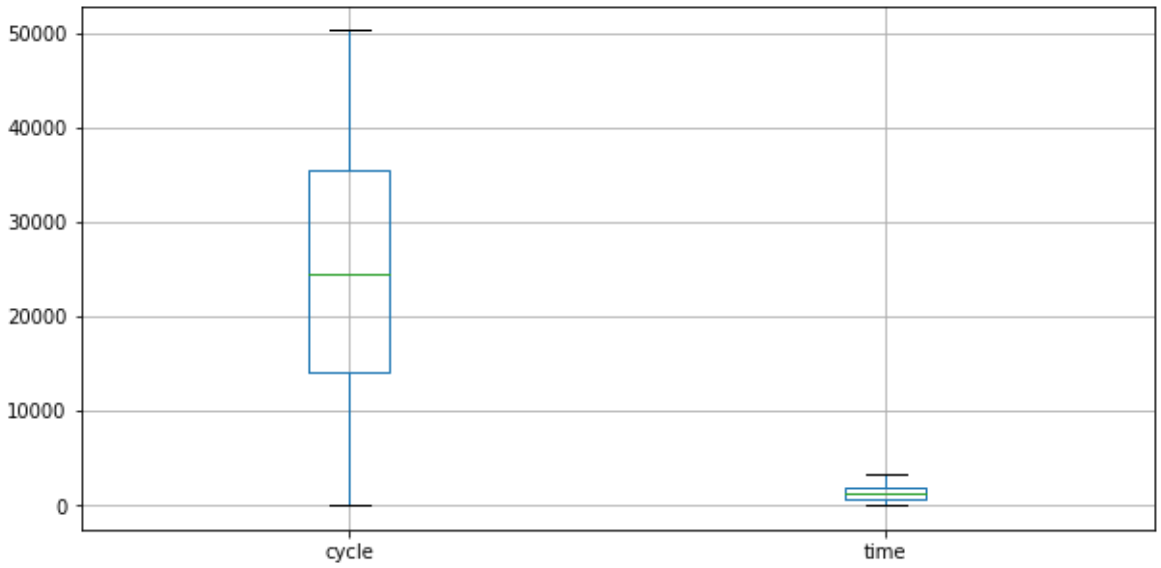
In [53]:
```
1 fuel_cells_df1[num_cols_2].boxplot(figsize = (10,3))
2 plt.show()
```



In [54]:
```
1 fuel_cells_df1[temp].boxplot(figsize = (10,3))
2 plt.show()
```



In [55]:
```
fuel_cells_df1[num_cols_3].boxplot(figsize = (10,5))
plt.show()
```



In [56]:
```
fuel_cells_df1.shape[0] - fuel_cells_df.shape[0]
```

Out[56]: -37766

- As we can see, total 37,766 datapoints are eliminated using IQR based method.
- This might contain some useful data as well, so it's better to treat/ detect outliers with some robust and SOTA methods of anomaly detection

In [57]:
```
# copy before mormalization and before datetime converson to int
fuel_cells_df_copy1 = fuel_cells_df.copy()
```

In [58]:
```
fuel_cells_df['datetime'] = fuel_cells_df['datetime'].astype('int64')
fuel_cells_df['datetime'].head()
# copy before mormalization and after datetime converson to int
fuel_cells_df_copy2 = fuel_cells_df.copy()
```

2. DBSCAN based anomaly detection

- As we know from above IQR method, that we have approx 37k outliers from 1.85 lac dataponts, I have carefully tried the hyperparameters (eps and minpts)
- I have tried with several values of eps and minpts, and got to a conclusion that,with minpts = 200 and eps = 0.5 we are getting the best results (compared the results with sampling (taking 10k datapoints (later)) and bootstrapping with replacement)

In [59]:
```
# DBSCAn on entire dataset
```

In [60]:
```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.cluster import DBSCAN
from sklearn import metrics

cols = ['cycle', 'datetime', 'capacity', 'voltage_measured', 'current_measured',
       'temperature_measured', 'current_load', 'voltage_load', 'time', 'flag']



std_scaler = StandardScaler().fit(fuel_cells_df_copy2)
std_df = std_scaler.transform(fuel_cells_df_copy2)

dbsc = DBSCAN(eps = .5, min_samples = 200).fit(std_df)


labels = dbsc.labels_




```

In [61]:
```
out_df = fuel_cells_df_copy2.copy()
out_df['label'] = dbsc.labels_
```

In [62]:
```
out_df['label'].value_counts()
```

Out[62]:
```
 7     41652
 1     39197
 4     37765
 9     31177
-1      8009
 6      5394
 3      5292
 0      5209
 5      4825
 2      4202
 10     1991
 8      1008
Name: label, dtype: int64
```

In [63]:
```
count_df = pd.DataFrame(data = out_df['label'].value_counts().values, columns = ['lab
count_df
```

Out[63]:

	label_counts
0	41652
1	39197
2	37765
3	31177
4	8009
5	5394
6	5292
7	5209
8	4825
9	4202
10	1991
11	1008

In [64]:
```
index_df = pd.DataFrame(data = out_df['label'].value_counts().index, columns = ['labe
index_df
```

Out[64]:

	label_index
0	7
1	1
2	4
3	9
4	-1
5	6
6	3
7	0
8	5
9	2
10	10
11	8

In [65]:
```
label_counts_df = pd.concat([index_df,count_df], axis = 1)
label_counts_df
```

Out[65]:

	label_index	label_counts
0	7	41652
1	1	39197
2	4	37765
3	9	31177
4	-1	8009
5	6	5394
6	3	5292
7	0	5209
8	5	4825
9	2	4202
10	10	1991
11	8	1008

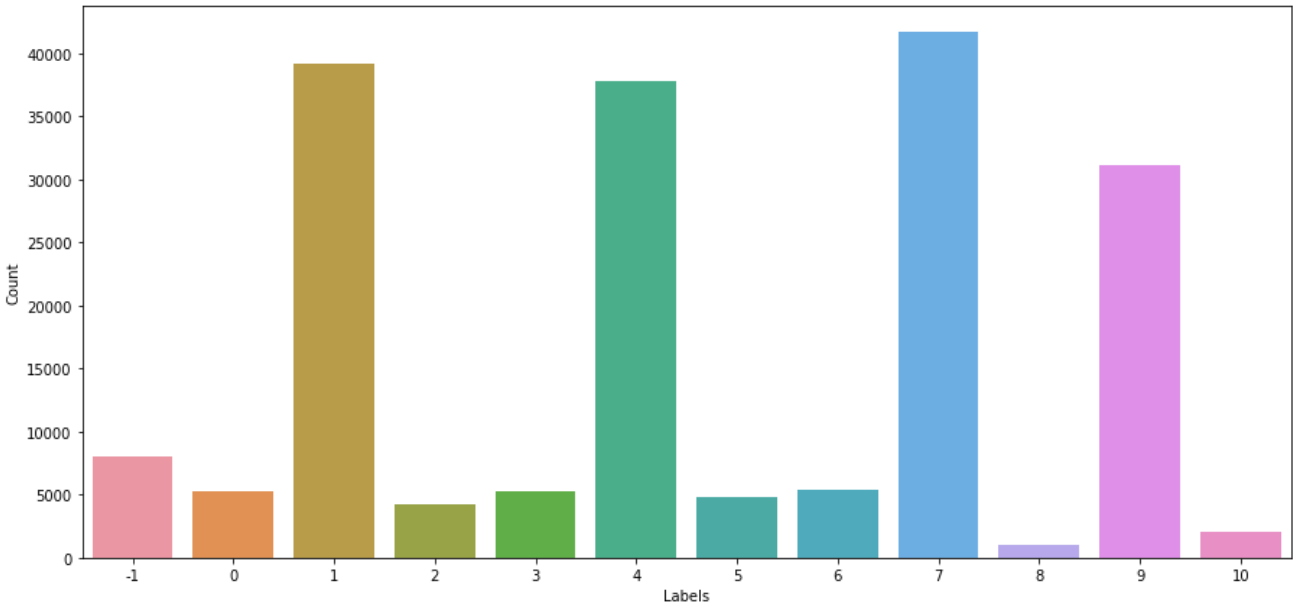
In [66]:
```
1 out_df
```

Out[66]:

	cycle	datetime	capacity	voltage_measured	current_measured	temperature_measured
0	1	1207149941000000000	1.856487	4.191492	-0.004902	24.330034
1	2	1207149941000000000	1.856487	4.190749	-0.001478	24.325993
2	3	1207149941000000000	1.856487	3.974871	-2.012528	24.389085
3	4	1207149941000000000	1.856487	3.951717	-2.013979	24.544752
4	5	1207149941000000000	1.856487	3.934352	-2.011144	24.731385
...	...	...	...	...	...	...
185716	34862	1219221439000000000	1.341051	3.443760	-0.002426	35.383979
185717	34863	1219221439000000000	1.341051	3.453271	-0.000981	35.179732
185718	34864	1219221439000000000	1.341051	3.461963	0.000209	34.977000
185719	34865	1219221439000000000	1.341051	3.469907	0.001516	34.785943
185720	34866	1219221439000000000	1.341051	3.477277	-0.001940	34.581660

185721 rows × 11 columns

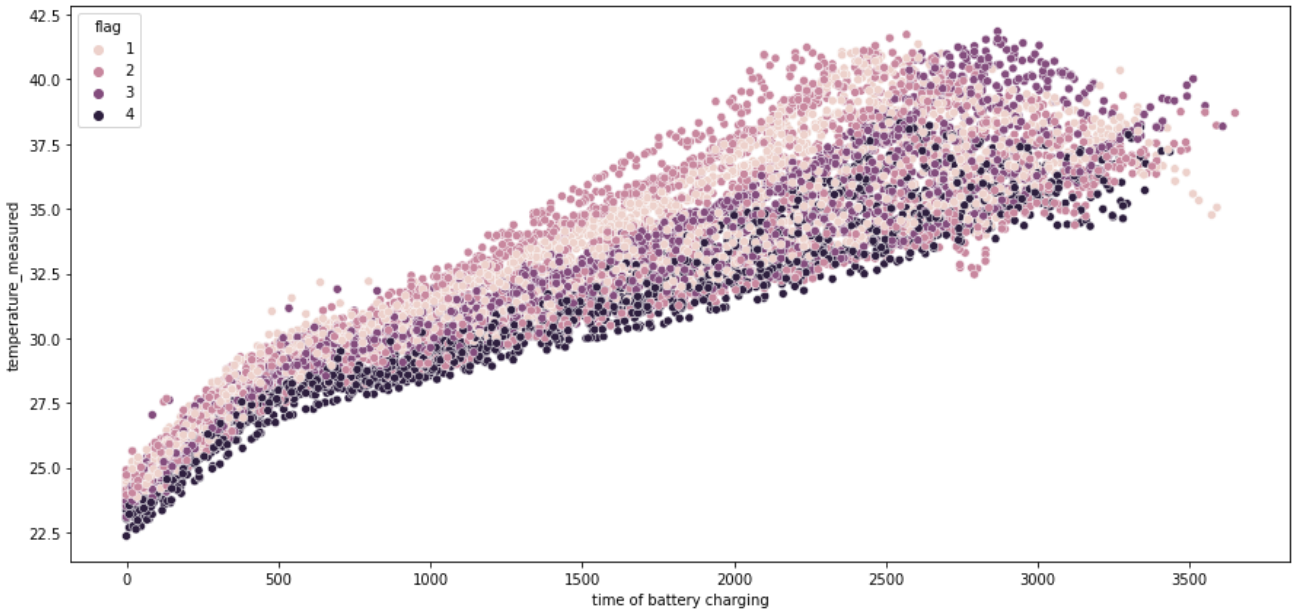
In [67]:
```
1 plt.figure(figsize=(15, 7))
2 sns.barplot(x =  'label_index', y = 'label_counts', data = label_counts_df)
3 plt.xlabel('Labels')
4 plt.ylabel('Count')
5 plt.show()
```



- As we can see, when all features are considered, the datapoints corresponding to -1 (noisy data) are 8009
- lets' further check with visualization and DBSCAN on other two features.
- It's running with very high time complexity and hence we have visualised on 10k samples.

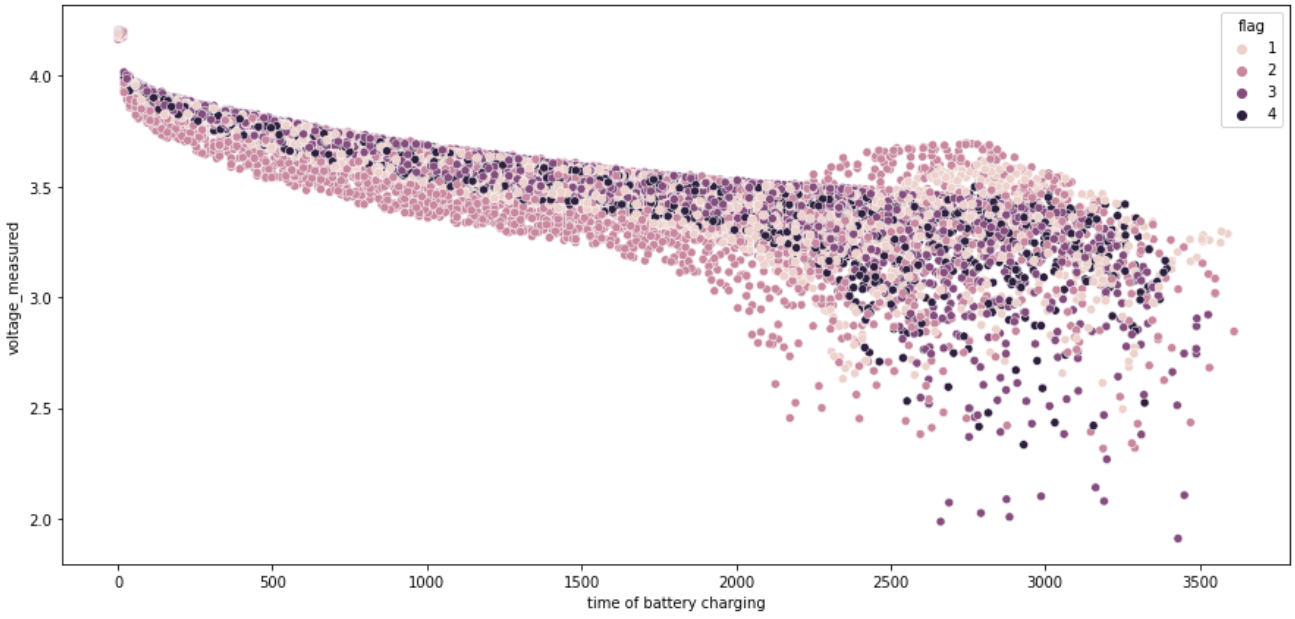
Visualization in lower dimention (2 features)

In [68]:
```
# Let's plot two features data now for viz
plt.figure(figsize=(15, 7))
sns.scatterplot(x = 'time', y = 'temperature_measured', hue ='flag', data = out_df.sa
plt.xlabel('time of battery charging')
plt.ylabel('temperature_measured')
plt.show()
```



- We can hardly see any outliers present when visualized with these two features.
- Lets cross check with some other two features

In [69]:
```
# Let's plot different two features data now for viz
plt.figure(figsize=(15, 7))
sns.scatterplot(x = 'time', y = 'voltage_measured', hue ='flag', data = out_df.sample
plt.xlabel('time of battery charging')
plt.ylabel('voltage_measured')
plt.show()
```



In [70]:
```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.cluster import DBSCAN
from sklearn import metrics


df = fuel_cells_df_copy2[['time','voltage_measured']].sample(n = 10000, replace = Tru

std_scaler = StandardScaler().fit(df)
std_df = std_scaler.transform(df)

dbsc = DBSCAN(eps = .5, min_samples = 200).fit(std_df)


labels = dbsc.labels_
```

In [71]:
```
out_df = df.copy()
out_df['label'] = dbsc.labels_
```

In [72]:
```
out_df['label'].value_counts()
```

Out[72]:
```
 0    9772
-1     228
Name: label, dtype: int64
```

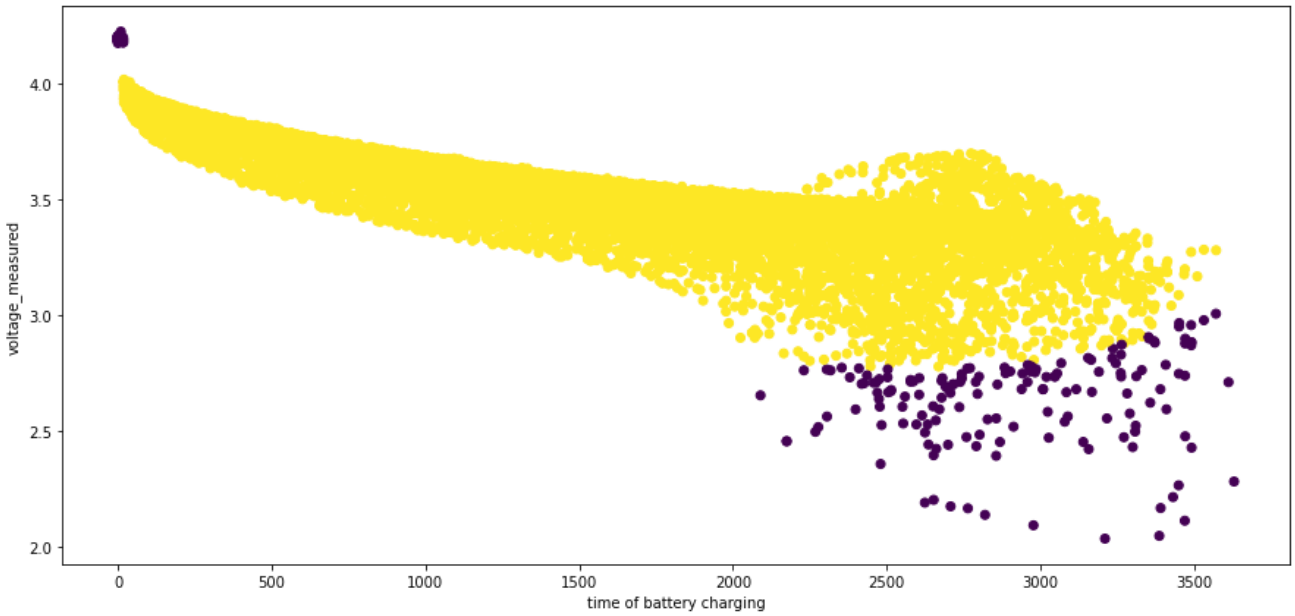
In [73]:
```
out_df
```

Out[73]:

	time	voltage_measured	label
92991	2201.890	2.976937	0
113051	2080.610	3.504083	0
9903	2754.484	3.391786	0
97634	422.453	3.579768	0
74564	1283.875	3.434059	0
...	...	...	...
147672	918.579	3.572996	0
38884	843.984	3.574765	0
116353	375.625	3.826459	0
85920	2427.141	2.927008	0
80506	2052.500	3.256455	0

10000 rows × 3 columns

In [74]:
```
plt.figure(figsize=(15, 7))
plt.scatter(out_df['time'], out_df['voltage_measured'], c=out_df['label'])
plt.xlabel('time of battery charging')
plt.ylabel('voltage_measured')
plt.show()
```



- With DBSCAn, we are clearly able to see the cluster for noise points in purple.

- The total no. of outliers are 8009

In [75]:
```
fuel_cells_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 185721 entries, 0 to 185720
Data columns (total 10 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   cycle                 185721 non-null  int64  
 1   datetime              185721 non-null  int64  
 2   capacity              185721 non-null  float64
 3   voltage_measured      185721 non-null  float64
 4   current_measured      185721 non-null  float64
 5   temperature_measured  185721 non-null  float64
 6   current_load          185721 non-null  float64
 7   voltage_load          185721 non-null  float64
 8   time                  185721 non-null  float64
 9   flag                  185721 non-null  int64  
dtypes: float64(7), int64(3)
memory usage: 15.6 MB
```

Minmax Scaling of orginal data:

In [76]:
```
cols = ['cycle', 'datetime', 'capacity', 'voltage_measured', 'current_measured',
       'temperature_measured', 'current_load', 'voltage_load', 'time', 'flag']
mm_scaler = MinMaxScaler()

fuel_cells_df[cols] = mm_scaler.fit_transform(fuel_cells_df[cols])
fuel_cells_df.head()
```

Out[76]:

	cycle	datetime	capacity	voltage_measured	current_measured	temperature_measured	current_load	volt
0	0.00000	0.0	0.797111	0.983242	0.990600	0.099077	0.49985	
1	0.00002	0.0	0.797111	0.982944	0.992276	0.098875	0.49985	
2	0.00004	0.0	0.797111	0.896465	0.008109	0.102032	0.00045	
3	0.00006	0.0	0.797111	0.887189	0.007399	0.109822	0.00045	
4	0.00008	0.0	0.797111	0.880233	0.008787	0.119162	0.00045	

3. Isolation Forest based Anomaly detection

The individual trees in an Isolation forest pick up a random feature and a random threshold and splits the data based on that. We expect the outliers to be found out at lower depth than normal points.

In [77]:
```
fuel_cells_df['flag'].unique()
```

Out[77]:
```
array([0.        , 0.33333333, 0.66666667, 1.        ])
```

In [78]:
```
from sklearn.ensemble import IsolationForest
import matplotlib.pyplot as plt

flag_df = pd.DataFrame()
flag_df_final = pd.DataFrame()
for j in fuel_cells_df['flag'].unique():
    contamination_arr = [0.01, 0.02, 0.03, 0.04]
    flag_df = fuel_cells_df[fuel_cells_df['flag'] == j]
    for i in contamination_arr:
        model = IsolationForest(n_estimators = 200, max_samples = 'auto', contaminati
        flag_df['anomaly_score'+'_'+str(i)] = model.fit_predict(flag_df)
        flag_df['scores'+'_'+str(i)] = model.decision_function(flag_df[flag_df.columr
    flag_df_final = pd.concat([flag_df,flag_df_final])
```

In [79]:
```
# flag_df
```

In [80]:
```
flag_df_final['anomaly_score_0.01'].value_counts(ascending= True)
```

Out[80]:
```
-1      1858
 1    183863
Name: anomaly_score_0.01, dtype: int64
```

In [81]:
```
for i in contamination_arr:
    print(f'Anomalies with contamination {i}:',len(flag_df_final[flag_df_final['anoma
```

```
Anomalies with contamination 0.01: 1858
Anomalies with contamination 0.02: 3716
Anomalies with contamination 0.03: 5573
Anomalies with contamination 0.04: 7431
```

In [82]:
```
## Data before normalization merging with new:

fuel_cells_df_final = pd.concat([fuel_cells_df_copy2, flag_df_final.drop(columns=fuel
```

In [83]:
```
df_final_copy = fuel_cells_df_final.copy()
```

In [84]:
```
fuel_cells_df_final
```

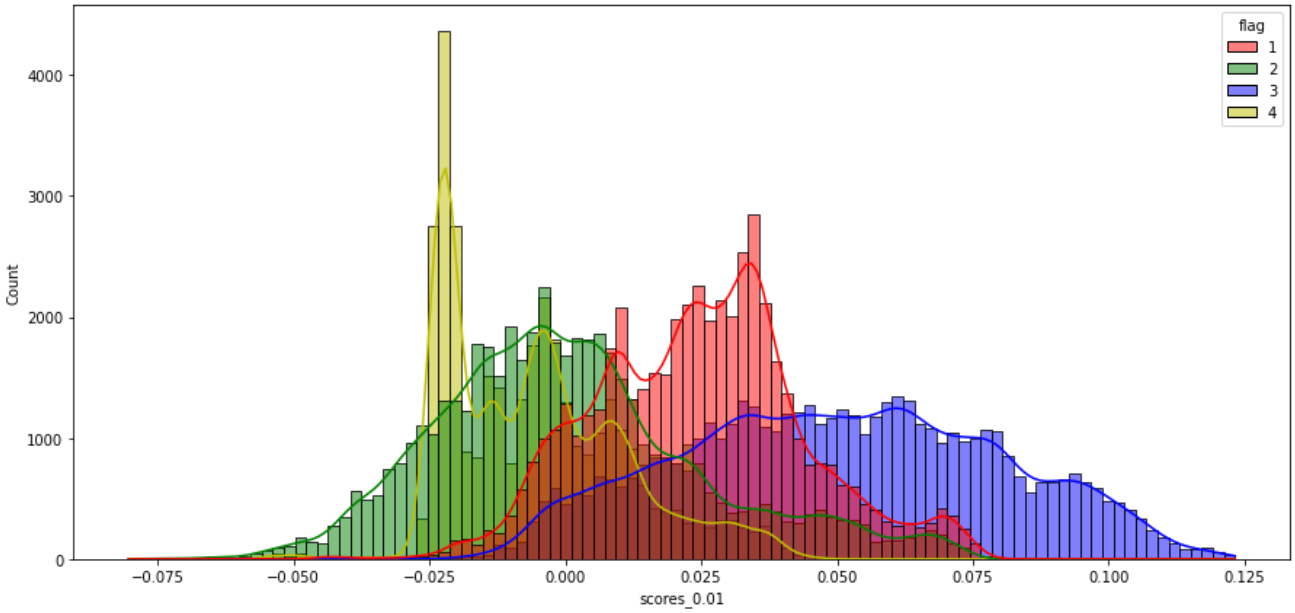
Out[84]:

	cycle	datetime	capacity	voltage_measured	current_measured	temperature_measured
0	1	1207149941000000000	1.856487	4.191492	-0.004902	24.330034
1	2	1207149941000000000	1.856487	4.190749	-0.001478	24.325993
2	3	1207149941000000000	1.856487	3.974871	-2.012528	24.389085
3	4	1207149941000000000	1.856487	3.951717	-2.013979	24.544752
4	5	1207149941000000000	1.856487	3.934352	-2.011144	24.731385
...	...	...	...	...	...	...
185716	34862	1219221439000000000	1.341051	3.443760	-0.002426	35.383979
185717	34863	1219221439000000000	1.341051	3.453271	-0.000981	35.179732
185718	34864	1219221439000000000	1.341051	3.461963	0.000209	34.977000
185719	34865	1219221439000000000	1.341051	3.469907	0.001516	34.785943
185720	34866	1219221439000000000	1.341051	3.477277	-0.001940	34.581660

185721 rows × 18 columns

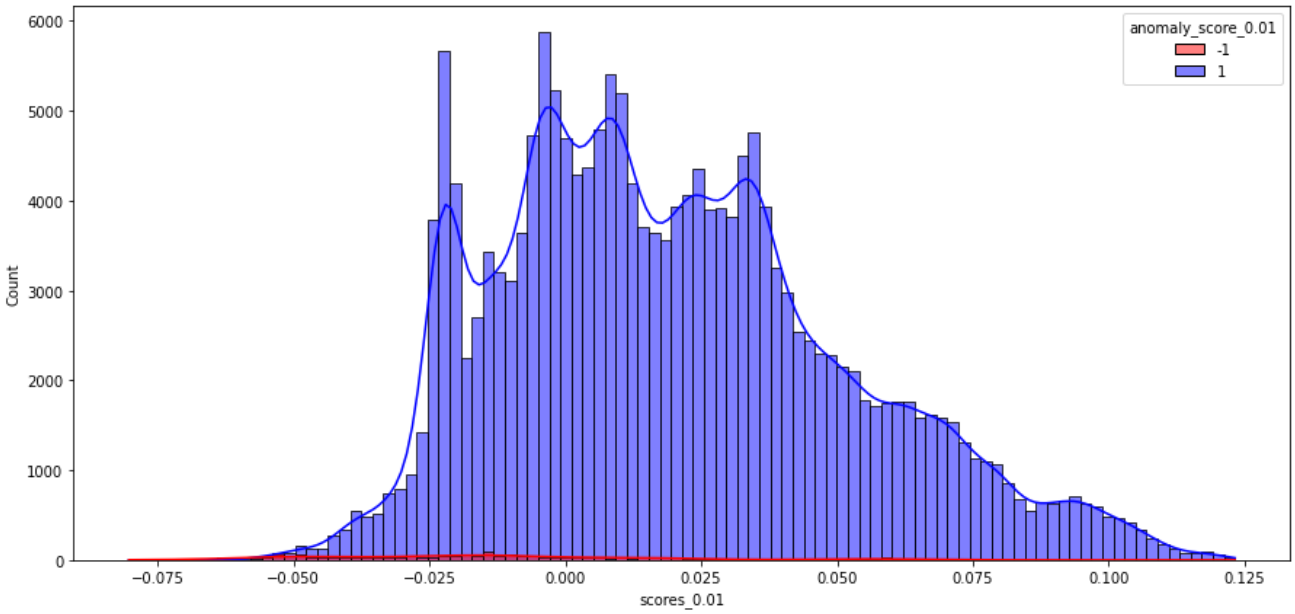
Plotting scores to visualize the scores given to the data

In [85]:
```
plt.figure(figsize=(15, 7))
sns.histplot(x = 'scores_0.01',hue = 'flag', palette=['r','g','b','y'], data = df_fir
fig.show()
```



- flag 2 is having maximum spread in the scores, which again confirms that Battery 2 is having max no. of outliers

In [86]:
```
plt.figure(figsize=(15, 7))
sns.histplot(x = 'scores_0.01',hue = 'anomaly_score_0.01', palette=['r','b'], data =
fig.show()
```



In [87]:
```
df_final_copy[df_final_copy['anomaly_score_0.01']==-1]['flag'].value_counts()
```

Out[87]:
```
1    503
2    503
3    503
4    349
Name: flag, dtype: int64
```

In [88]:
```
df_final_copy[df_final_copy['anomaly_score_0.04']==-1]['flag'].value_counts()
```

Out[88]:
```
1    2012
2    2012
3    2012
4    1395
Name: flag, dtype: int64
```

In [89]:
```
data = df_final_copy[df_final_copy['anomaly_score_0.01']==-1]
```

In [90]:
```
wd = df_final_copy.loc[(df_final_copy['anomaly_score_0.01'] == -1) & (df_final_copy[
len(wd)
```

Out[90]:
```
503
```

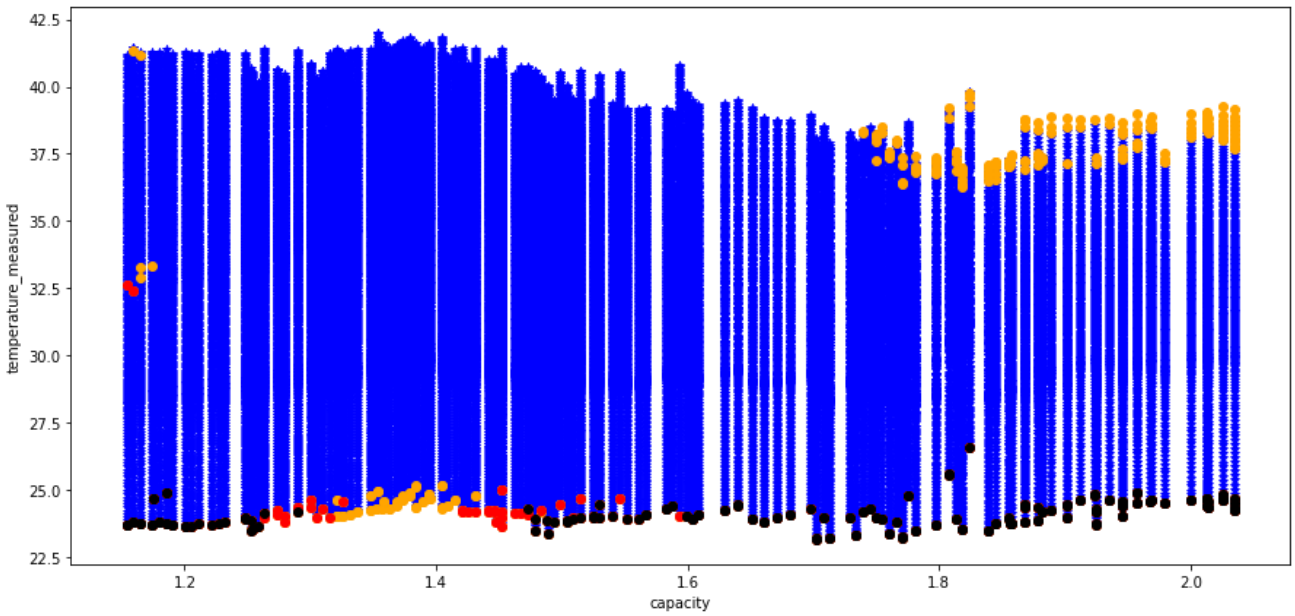
In [91]:
```
# Thresholding on scores ( < 0.04)

wd2 = df_final_copy.loc[(df_final_copy['anomaly_score_0.01'] == -1) & (df_final_copy|
len(wd2)
```

Out[91]:
```
243
```

In [92]:
```
# Thresholding on scores ( < 0.05)

wd3 = df_final_copy.loc[(df_final_copy['anomaly_score_0.01'] == -1) & (df_final_copy|
len(wd3)
```

Out[92]:
```
185
```

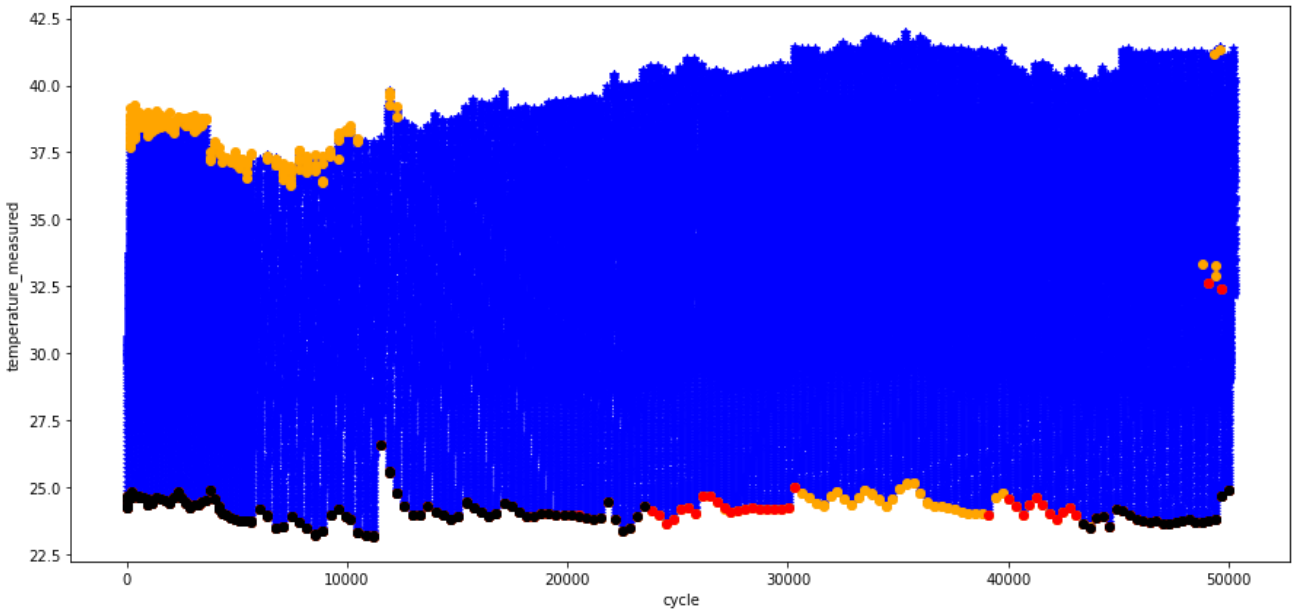
In [93]:
```
plt.figure(figsize=(15, 7))
plt.scatter(df_final_copy[(df_final_copy['anomaly_score_0.01'] == 1) & (df_final_copy
            df_final_copy[(df_final_copy['anomaly_score_0.01'] == 1) & (df_final_copy|

plt.scatter(wd['capacity'], wd['temperature_measured'], c = 'orange', marker = 'o')
plt.scatter(wd2['capacity'], wd2['temperature_measured'], c = 'red', marker = 'o')
plt.scatter(wd3['capacity'], wd3['temperature_measured'], c = 'black', marker = 'o')

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



- Batteries with much higher capacities and much lower capacities but having minimum temperature measured are having most critical number of outliers
- WHereas as the batteries with least temperature measured are having more number of outliers.

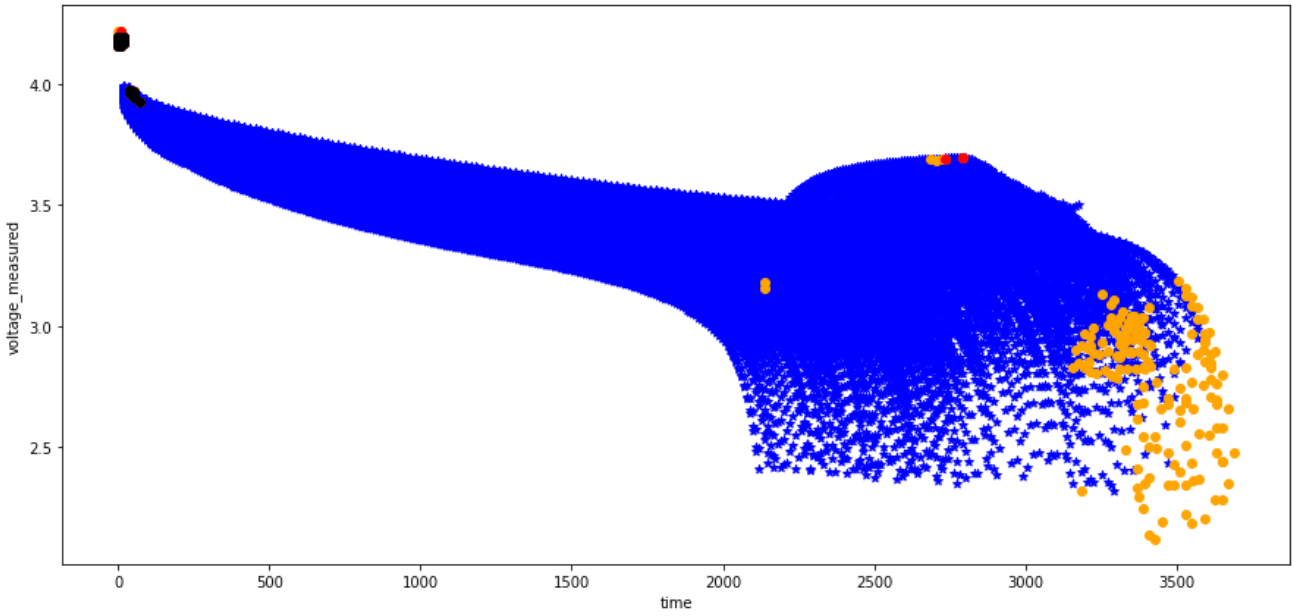
In [94]:

```
plt.figure(figsize=(15, 7))
plt.scatter(df_final_copy[(df_final_copy['anomaly_score_0.01'] == 1) & (df_final_copy
            df_final_copy[(df_final_copy['anomaly_score_0.01'] == 1) & (df_final_copy[

plt.scatter(wd['cycle'], wd['temperature_measured'], c = 'orange', marker = 'o')
plt.scatter(wd2['cycle'], wd2['temperature_measured'], c = 'red', marker = 'o')
plt.scatter(wd3['cycle'], wd3['temperature_measured'], c = 'black', marker = 'o')

plt.xlabel('cycle')
plt.ylabel('temperature_measured')
plt.show()
```



- The batteries with least temperature measured are having more number of outliers.

In [95]:
```
plt.figure(figsize=(15, 7))
plt.scatter(df_final_copy[(df_final_copy['anomaly_score_0.01'] == 1) & (df_final_copy
            df_final_copy[(df_final_copy['anomaly_score_0.01'] == 1) & (df_final_copy[

plt.scatter(wd['time'], wd['voltage_measured'], c = 'orange', marker = 'o')
plt.scatter(wd2['time'], wd2['voltage_measured'], c = 'red', marker = 'o')
plt.scatter(wd3['time'], wd3['voltage_measured'], c = 'black', marker = 'o')

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



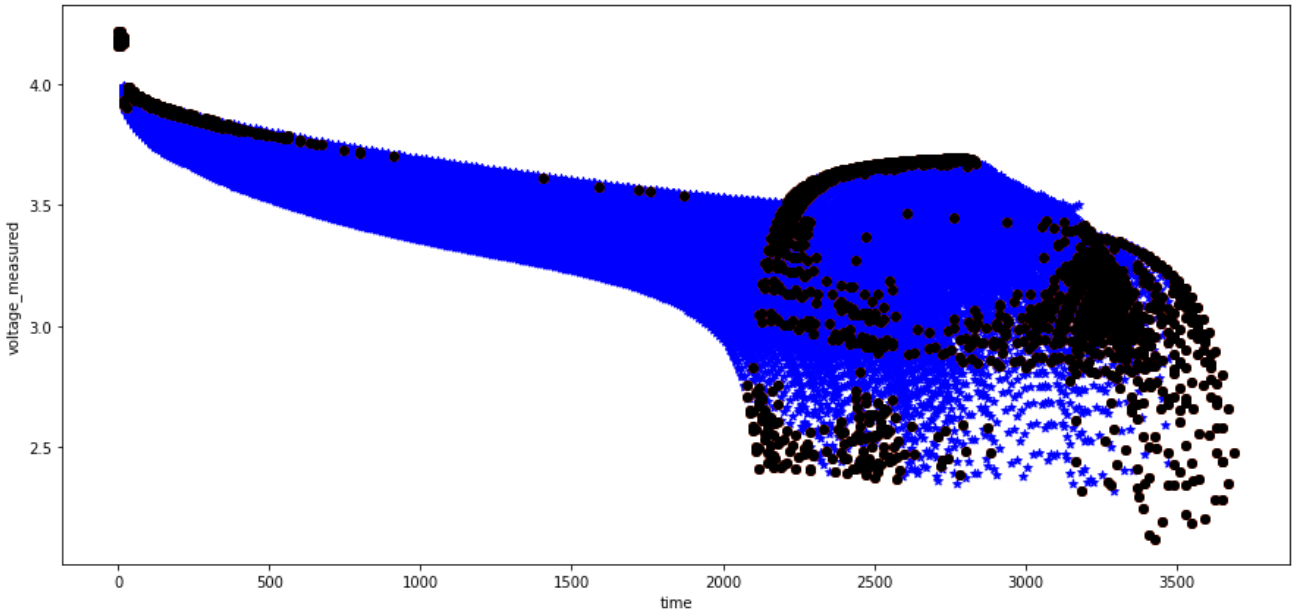
- Batteries datapoints which are producing maximum voltages with barely minimum charging time are ceratinly outliers/anamolies.
- Also, with least contamination, the Batteries datapoints which requires maximum time to charge ceratinly outliers/anamolies.

Anamolies with increased contamination %

In [96]:
```
# Testing with multiple values such as with  anomaly_score_0.04

wd = df_final_copy.loc[(df_final_copy['anomaly_score_0.04'] == -1) & (df_final_copy[
wd2 = df_final_copy.loc[(df_final_copy['anomaly_score_0.04'] == -1) & (df_final_copy[
wd3 = df_final_copy.loc[(df_final_copy['anomaly_score_0.04'] == -1) & (df_final_copy[

```

In [97]:

```
plt.figure(figsize=(15, 7))
plt.scatter(df_final_copy[(df_final_copy['anomaly_score_0.04'] == 1) & (df_final_copy
            df_final_copy[(df_final_copy['anomaly_score_0.04'] == 1) & (df_final_copy[

plt.scatter(wd['time'], wd['voltage_measured'], c = 'orange', marker = 'o')
plt.scatter(wd2['time'], wd2['voltage_measured'], c = 'red', marker = 'o')
plt.scatter(wd3['time'], wd3['voltage_measured'], c = 'black', marker = 'o')

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

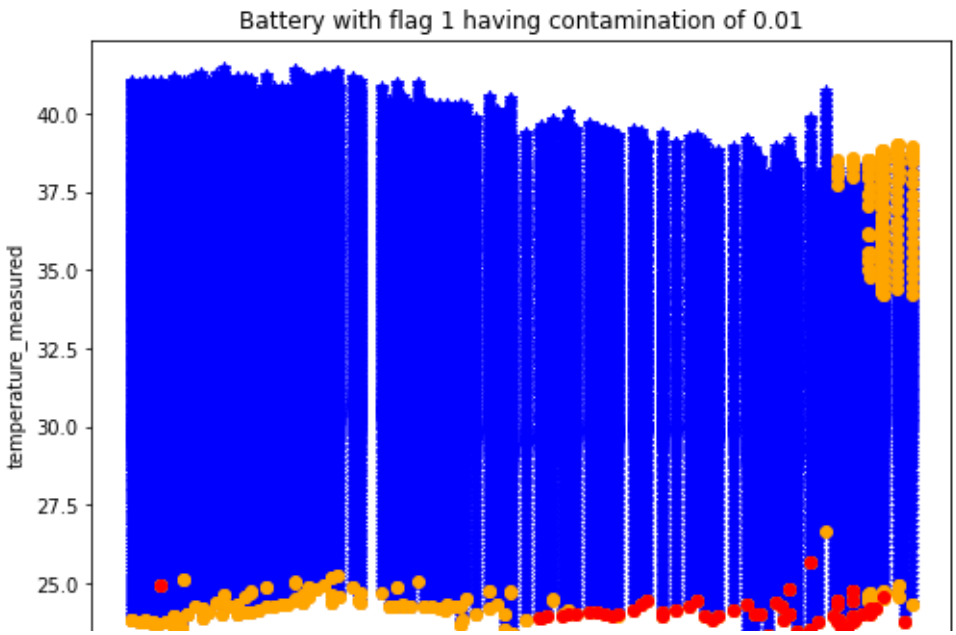


- As it's clearly visible , if we increase the contamination % of outliers to 4 % , it's able to classify the visually seen anamolies as outliers as compared to contamionation of 1 %
- Also, with max contamination of 4%, the Batteries datapoints which requires maximum time to charge and whose voltages are low are ceratinly outliers/anamolies.

In [98]:

```
flags_arr = [1,2,3,4]
contamination_arr = [0.01,0.02,0.03,0.04]
```

In [99]:
```
for i in flags_arr:
    for j in contamination_arr:
        wd = df_final_copy.loc[(df_final_copy['anomaly_score'+'_'+str(j)] == -1) & (
        wd2 = df_final_copy.loc[(df_final_copy['anomaly_score'+'_'+str(j)] == -1) & (
        wd3 = df_final_copy.loc[(df_final_copy['anomaly_score'+'_'+str(j)] == -1) & (

        plt.figure(figsize=(8, 6))
        plt.scatter(df_final_copy[(df_final_copy['anomaly_score'+'_'+str(j)] == 1) &
                    df_final_copy[(df_final_copy['anomaly_score'+'_'+str(j)] == 1) & (

        plt.scatter(wd['capacity'], wd['temperature_measured'], c = 'orange', marker
        plt.scatter(wd2['capacity'], wd2['temperature_measured'], c = 'red', marker =
        plt.scatter(wd3['capacity'], wd3['temperature_measured'], c = 'black', marker

        plt.xlabel('capacity')
        plt.ylabel('temperature_measured')
        plt.title('Battery with flag ' + str(i)+ ' having contamination of '+str(j))
        plt.show()
```



- The batteries with least temperature measured are having more number of outliers.
- Batteries with much higher capacities and much lower capacities but having minimum temperature measured are having most critical number of outliers
- WHereas as the batteries with least temperature measured are having more number of outliers.
- As we go on increase the contamination %, we are getting more number of datapoints classified as extreme outliers.

4. Local Outlier Factor(LOF) based Anomaly detection

In [100]:
```
lof_df = fuel_cells_df_copy2.copy()
```

In [101]:
```
1 fuel_cells_df_copy2
```

Out[101]:

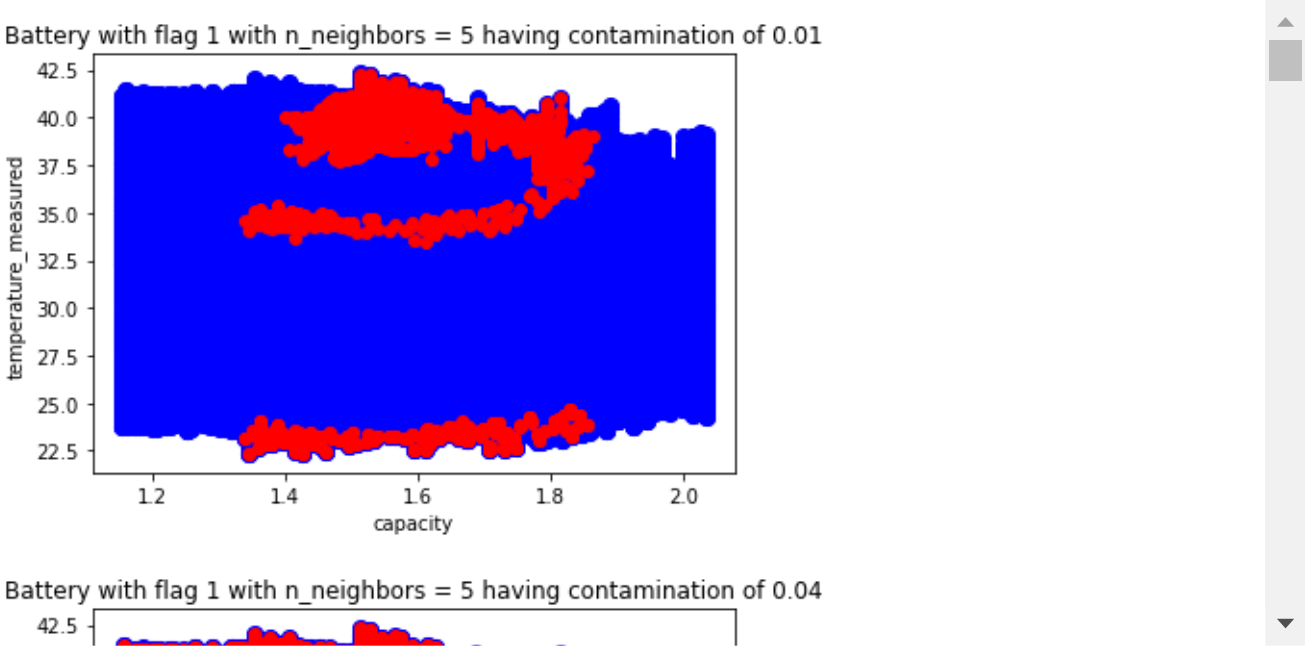
	cycle	datetime	capacity	voltage_measured	current_measured	temperature_measured
0	1	1207149941000000000	1.856487	4.191492	-0.004902	24.330034
1	2	1207149941000000000	1.856487	4.190749	-0.001478	24.325993
2	3	1207149941000000000	1.856487	3.974871	-2.012528	24.389085
3	4	1207149941000000000	1.856487	3.951717	-2.013979	24.544752
4	5	1207149941000000000	1.856487	3.934352	-2.011144	24.731385
...	...	...	...	...	...	...
185716	34862	1219221439000000000	1.341051	3.443760	-0.002426	35.383979
185717	34863	1219221439000000000	1.341051	3.453271	-0.000981	35.179732
185718	34864	1219221439000000000	1.341051	3.461963	0.000209	34.977000
185719	34865	1219221439000000000	1.341051	3.469907	0.001516	34.785943
185720	34866	1219221439000000000	1.341051	3.477277	-0.001940	34.581660

185721 rows × 10 columns

In [102]:
```
1 n_neighbors = [5, 20]
2 contamination_arr = [0.01,0.04]
3 flags_arr = [1,2,3,4]
```

In [103]:

```
from sklearn.neighbors import LocalOutlierFactor
for f in flags_arr:
    for n in n_neighbors:
        for c in contamination_arr:
            # model specification
            model1 = LocalOutlierFactor(n_neighbors = n, metric = "manhattan", contar
            # model fitting
            y_pred = model1.fit_predict(fuel_cells_df_copy2)
            # filter outlier index
            outlier_index = np.where(y_pred == -1) # negative values are outliers and
            # filter outlier values
            outlier_values = fuel_cells_df_copy2.iloc[outlier_index]
            # plot data
            plt.scatter(fuel_cells_df_copy2["capacity"], fuel_cells_df_copy2["tempera
            # plot outlier values
            plt.scatter(outlier_values["capacity"], outlier_values["temperature_measu
            plt.xlabel('capacity')
            plt.ylabel('temperature_measured')
            plt.title('Battery with flag ' + str(f)+ ' with n_neighbors = '+ str(n)+'
            plt.show()
```



- In LOF, the hypermeters used are n_neighbors = [5, 20] and contamination_ = [0.01,0.04].
- Almost 7429 datapoints are getting classified as outliers using maximum nearest neigbors (20) and with higesht contamination percentage of 4%.
- With least nearest neigbors and least contamination percentage, we are getting less no. of outliers specifically in the mid-battery capacity range (1.4-1.6) having least and maximum temperature measured. This can be informed to subject matter expert and thus can be validated before removal.

In [104]:

```
len(y_pred)
```

Out[104]: 185721

In [105]:
```
1 fuel_cells_df_copy2
```

Out[105]:

	cycle	datetime	capacity	voltage_measured	current_measured	temperature_measured
0	1	1207149941000000000	1.856487	4.191492	-0.004902	24.330034
1	2	1207149941000000000	1.856487	4.190749	-0.001478	24.325993
2	3	1207149941000000000	1.856487	3.974871	-2.012528	24.389085
3	4	1207149941000000000	1.856487	3.951717	-2.013979	24.544752
4	5	1207149941000000000	1.856487	3.934352	-2.011144	24.731385
...	...	...	...	...	...	...
185716	34862	1219221439000000000	1.341051	3.443760	-0.002426	35.383979
185717	34863	1219221439000000000	1.341051	3.453271	-0.000981	35.179732
185718	34864	1219221439000000000	1.341051	3.461963	0.000209	34.977000
185719	34865	1219221439000000000	1.341051	3.469907	0.001516	34.785943
185720	34866	1219221439000000000	1.341051	3.477277	-0.001940	34.581660

185721 rows × 10 columns

In [106]:
```
1 y_pred_df = pd.DataFrame(data=y_pred,columns=['y_pred'])
2 fuel_cells_df_copy2_final_lof = pd.concat([fuel_cells_df_copy2,y_pred_df],axis = 1)
3 fuel_cells_df_copy2_final_lof
```

Out[106]:

	cycle	datetime	capacity	voltage_measured	current_measured	temperature_measured
0	1	1207149941000000000	1.856487	4.191492	-0.004902	24.330034
1	2	1207149941000000000	1.856487	4.190749	-0.001478	24.325993
2	3	1207149941000000000	1.856487	3.974871	-2.012528	24.389085
3	4	1207149941000000000	1.856487	3.951717	-2.013979	24.544752
4	5	1207149941000000000	1.856487	3.934352	-2.011144	24.731385
...	...	...	...	...	...	...
185716	34862	1219221439000000000	1.341051	3.443760	-0.002426	35.383979
185717	34863	1219221439000000000	1.341051	3.453271	-0.000981	35.179732
185718	34864	1219221439000000000	1.341051	3.461963	0.000209	34.977000
185719	34865	1219221439000000000	1.341051	3.469907	0.001516	34.785943
185720	34866	1219221439000000000	1.341051	3.477277	-0.001940	34.581660

185721 rows × 11 columns

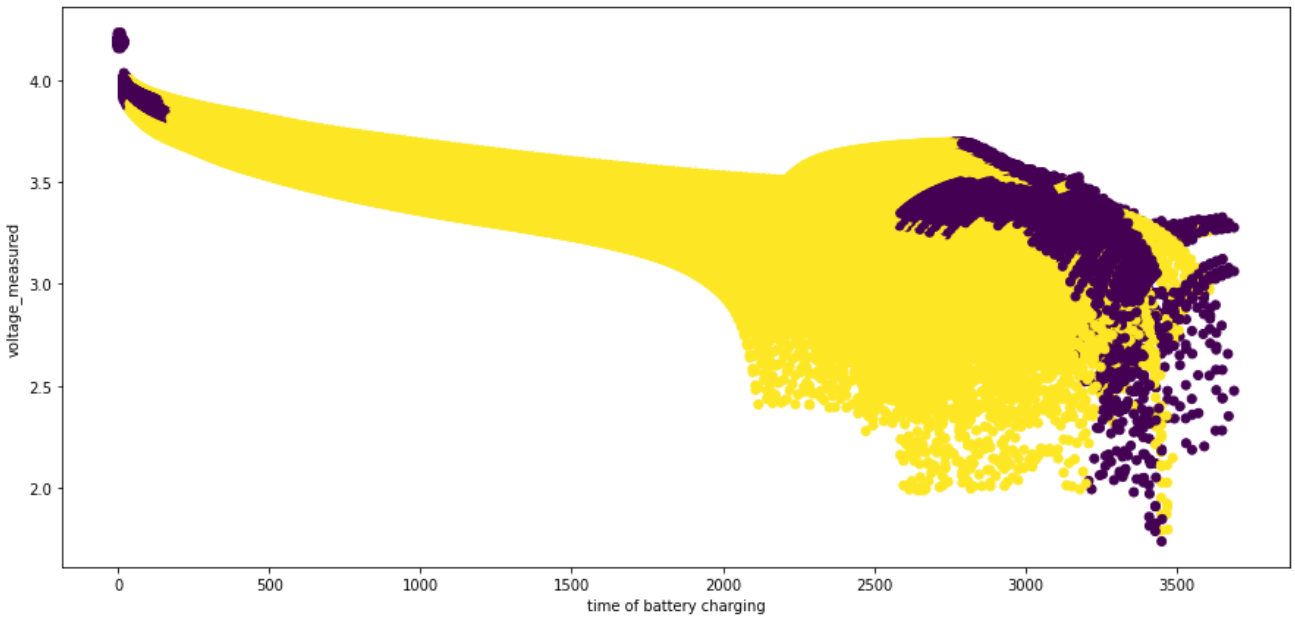
In [107]:
```
# No. of outliers when we are considering the highest contamoination i.e 4 %

fuel_cells_df_copy2_final_lof[fuel_cells_df_copy2_final_lof['y_pred'] == -1]
```

Out[107]:

	cycle	datetime	capacity	voltage_measured	current_measured	temperature_measured
0	1	1207149941000000000	1.856487	4.191492	-0.004902	24.330034
1	2	1207149941000000000	1.856487	4.190749	-0.001478	24.325993
2	3	1207149941000000000	1.856487	3.974871	-2.012528	24.389085
3	4	1207149941000000000	1.856487	3.951717	-2.013979	24.544752
193	194	1207149941000000000	1.856487	3.264121	-0.001532	35.021738
...	...	...	...	...	...	...
185716	34862	1219221439000000000	1.341051	3.443760	-0.002426	35.383979
185717	34863	1219221439000000000	1.341051	3.453271	-0.000981	35.179732
185718	34864	1219221439000000000	1.341051	3.461963	0.000209	34.977000
185719	34865	1219221439000000000	1.341051	3.469907	0.001516	34.785943
185720	34866	1219221439000000000	1.341051	3.477277	-0.001940	34.581660

7429 rows × 11 columns

In [108]:
```
plt.figure(figsize=(15, 7))
plt.scatter(fuel_cells_df_copy2_final_lof['time'], fuel_cells_df_copy2_final_lof['vol
plt.xlabel('time of battery charging')
plt.ylabel('voltage_measured')
plt.show()
```



- I have validated visually in lower dimension with above hyperparameters but the performance is less acuurate than Isolation Forest but more accurate than other methods.

5. Elliptical Envelope based Anomaly detection

In [109]:
```
eliptical_df = fuel_cells_df_copy2.copy()
```

In [110]:
```
from sklearn.preprocessing import StandardScaler
cols = ['cycle', 'datetime', 'capacity', 'voltage_measured', 'current_measured',
       'temperature_measured', 'current_load', 'voltage_load', 'time', 'flag']
scaler = StandardScaler()

eliptical_df[cols] = scaler.fit_transform(eliptical_df[cols])
eliptical_df.head()
```

Out[110]:

	cycle	datetime	capacity	voltage_measured	current_measured	temperature_measured	current_load
0	-1.678666	-1.424022	1.477315	2.758442	3.255533	-1.998389	-1.194938
1	-1.678595	-1.424022	1.477315	2.755491	3.261632	-1.999392	-1.194938
2	-1.678524	-1.424022	1.477315	1.897777	-0.320553	-1.983728	-2.823146
3	-1.678453	-1.424022	1.477315	1.805782	-0.323138	-1.945079	-2.823146
4	-1.678383	-1.424022	1.477315	1.736792	-0.318086	-1.898742	-2.823146

In [111]:
```
from sklearn.model_selection import train_test_split
```

In [112]:
```
X_train, X_test= train_test_split(eliptical_df, test_size=0.33, random_state=42)
```

In [113]:
```
# convert tarin dataframe to arrays
data = X_train[cols].values
```

In [114]:
```

from sklearn.covariance import EllipticEnvelope
# instantiate model
model1 = EllipticEnvelope(contamination = 0.1)
# fit model
model1.fit(data)
```

Out[114]:
```
EllipticEnvelope()
```

In [115]:
```
# convert test dataframe to arrays
# new data for prediction (data needs to be in arrays)
new_data = X_test[cols].values
```

In [116]:
```

# predict on new data
pred1 = model1.predict(new_data)
print(pred1)
```

```
[1 1 1 ... 1 1 1]
```

In [117]:
```
len(new_data)
```

Out[117]:
```
61288
```

In [118]:
```
len(pred1)
```

Out[118]:
```
61288
```

In [119]:
```
pred_df = pd.DataFrame(data=pred1,columns=['pred1'])
fuel_cells_df_copy2_final_eliptical = pd.concat([X_test,pred_df],axis = 1)

```

In [120]:
```
1 pred_df
```

Out[120]:

	pred1
0	1
1	1
2	1
3	1
4	1
...	...
61283	1
61284	1
61285	1
61286	1
61287	1

61288 rows × 1 columns

In [121]:
```
1 X_test
```

Out[121]:

	cycle	datetime	capacity	voltage_measured	current_measured	temperature_measured	current_l
94986	1.488209	0.026246	-1.686115	-0.469938	-0.319177	-0.111202	0.434
182194	0.541568	2.434538	-0.990579	-3.487875	-0.314516	1.460708	0.434
96713	1.610559	0.064419	-1.852381	0.757447	-0.314559	-1.444924	0.434
86446	0.883186	-0.156326	-1.028915	-0.553402	-0.320190	0.545350	0.434
134163	0.701254	-0.221676	-0.240105	0.502181	-0.281645	-0.370104	0.435
...	...	...	...	...	...	...	
170230	-0.306030	2.066050	-0.181811	0.419846	-0.314026	-0.877855	0.434
68294	-0.402805	-0.553487	-0.040528	-0.462780	-0.316967	0.421779	0.434
184683	0.717903	2.536516	-1.081689	-2.332737	-0.315089	1.169705	0.434
32289	0.608871	-0.244885	-0.660021	0.863215	-0.322651	-0.574446	0.434
88361	1.018856	-0.121309	-1.278001	-1.101655	-0.317462	1.188723	0.434

61288 rows × 10 columns

In [122]:
```
1 fuel_cells_df_copy2_final_eliptical[fuel_cells_df_copy2_final_eliptical['pred1'] == -
```

Out[122]: (5943, 11)

In [123]:
```
1 # predict on all data
2 all_data = eliptical_df[cols].values
3 pred2 = model1.predict(all_data)
4 print(pred2)
```

```
[ 1  1 -1 ... -1 -1 -1]
```

In [124]:
```
1 len(pred2)
```

Out[124]: 185721

In [125]:
```
len(all_data)
```

Out[125]: 185721

In [126]:
```
pred_df_all = pd.DataFrame(data=pred2,columns=['pred2'])
final_eliptical = pd.concat([fuel_cells_df_copy2,pred_df_all],axis = 1)
final_eliptical
```

Out[126]:

	cycle	datetime	capacity	voltage_measured	current_measured	temperature_measured
0	1	1207149941000000000	1.856487	4.191492	-0.004902	24.330034
1	2	1207149941000000000	1.856487	4.190749	-0.001478	24.325993
2	3	1207149941000000000	1.856487	3.974871	-2.012528	24.389085
3	4	1207149941000000000	1.856487	3.951717	-2.013979	24.544752
4	5	1207149941000000000	1.856487	3.934352	-2.011144	24.731385
...	...	...	...	...	...	...
185716	34862	1219221439000000000	1.341051	3.443760	-0.002426	35.383979
185717	34863	1219221439000000000	1.341051	3.453271	-0.000981	35.179732
185718	34864	1219221439000000000	1.341051	3.461963	0.000209	34.977000
185719	34865	1219221439000000000	1.341051	3.469907	0.001516	34.785943
185720	34866	1219221439000000000	1.341051	3.477277	-0.001940	34.581660

185721 rows × 11 columns

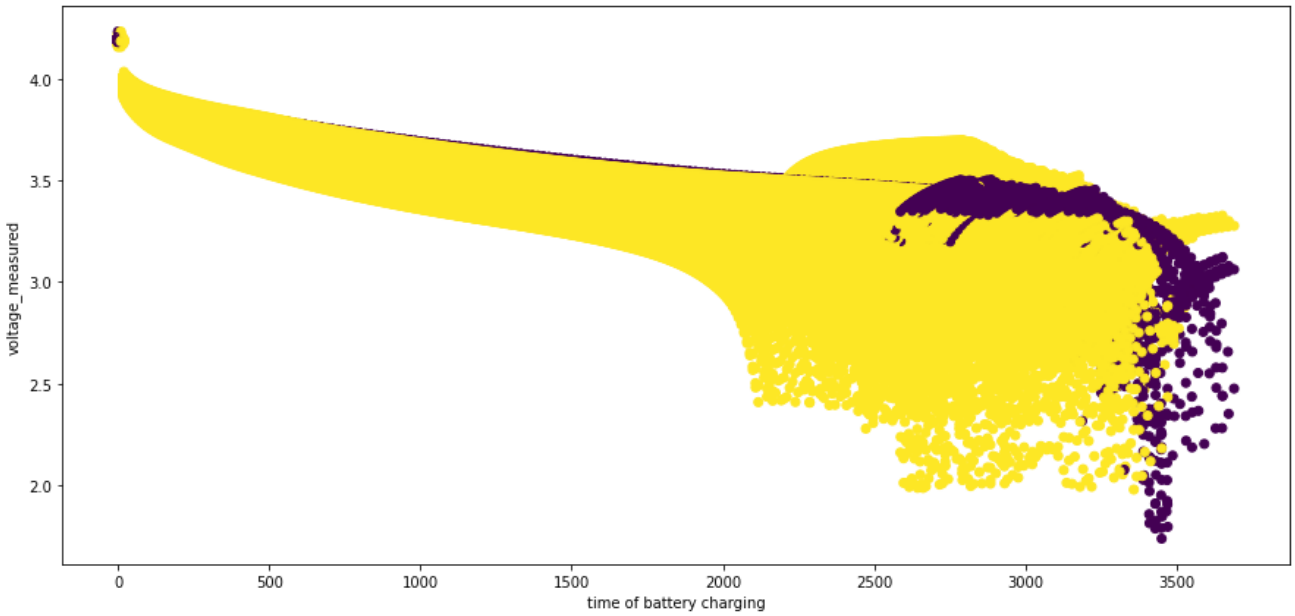
In [127]:
```
final_eliptical[final_eliptical['pred2'] == -1]
```

Out[127]:

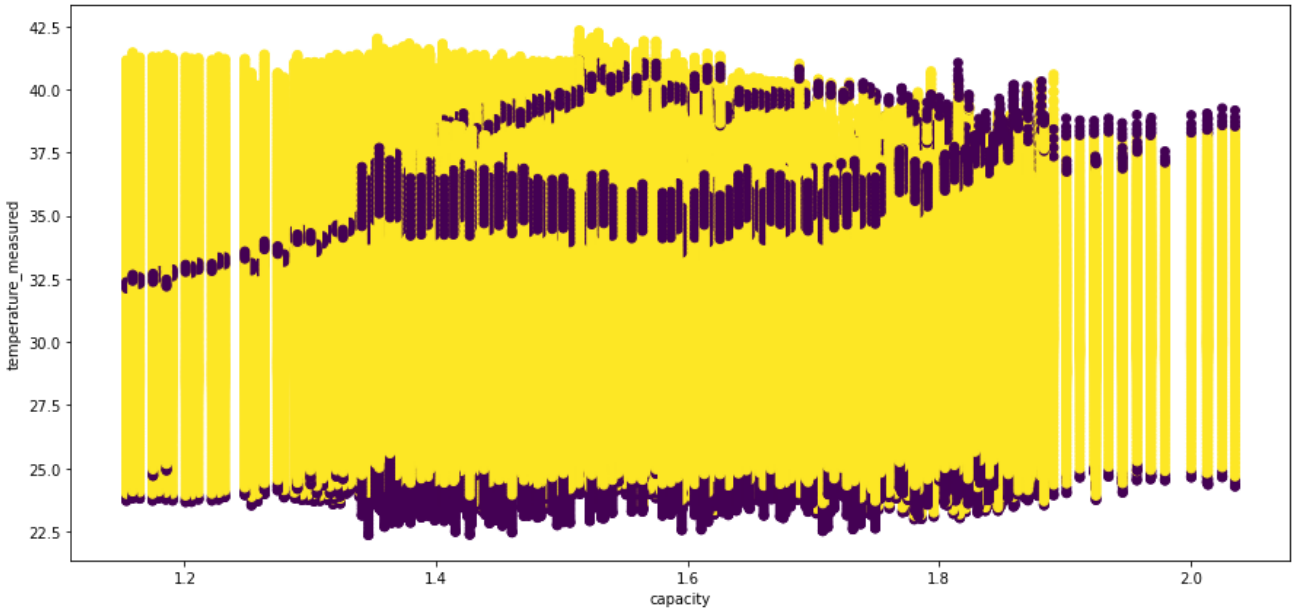
	cycle	datetime	capacity	voltage_measured	current_measured	temperature_measured
2	3	1207149941000000000	1.856487	3.974871	-2.012528	24.389085
3	4	1207149941000000000	1.856487	3.951717	-2.013979	24.544752
4	5	1207149941000000000	1.856487	3.934352	-2.011144	24.731385
5	6	1207149941000000000	1.856487	3.920058	-2.013007	24.909816
6	7	1207149941000000000	1.856487	3.907904	-2.014400	25.105884
...	...	...	...	...	...	...
185716	34862	1219221439000000000	1.341051	3.443760	-0.002426	35.383979
185717	34863	1219221439000000000	1.341051	3.453271	-0.000981	35.179732
185718	34864	1219221439000000000	1.341051	3.461963	0.000209	34.977000
185719	34865	1219221439000000000	1.341051	3.469907	0.001516	34.785943
185720	34866	1219221439000000000	1.341051	3.477277	-0.001940	34.581660

18387 rows × 11 columns

In [128]:
```
plt.figure(figsize=(15, 7))
plt.scatter(final_eliptical['time'], final_eliptical['voltage_measured'], c=final_eli
plt.xlabel('time of battery charging')
plt.ylabel('voltage_measured')
plt.show()
```



In [129]:
```
plt.figure(figsize=(15, 7))
plt.scatter(fuel_cells_df_copy2_final_lof['capacity'], fuel_cells_df_copy2_final_lof[
plt.xlabel('capacity')
plt.ylabel('temperature_measured')
plt.show()
```



Comparison of results from the above algorithms:

I have tried 5 methods for anamoly detection:

- 1. IQR based
- 2. DBSCAN (density based)
- 3. Isolation Forest ()
- 4. Local Outlier Factor (LOF)
- 5. Elliptical Envelope

1. IQR based

- As we have see, total **37,766 datapoints (20.13% data) are tagged as outliers** using IQR based method.
- This data is huge and might contain some useful information as well, so it's better to treat/ detect outliers with some robust and SOTA methods of anomaly detection. (worst performer)

2. DBSCAN (density based)

- With proper hit and trial for finding the best hyperparameters (eps and minpts) I got **eps = 0.5, min_samples = 200**. With this, I was able to detect **8009 datapoints as outliers**. Have validated visually in lower dimension with above hyperparameters.

3. Isolation Forest

- Anomalies with contamination **1%, 2%, 3%, 4% are 1858, 3716, 5573, 7431 respectievly**.
- Even with the least contamination %, we are able to get the exact no. of outliers which I have validated visually in lower dimension with different different set of features.
- However, **as per the subject matter expert we can still get it validated** if certain points are anamlalies or not, but we are getting good match.
- We can easily further classify the outliers into **good, average and extreme anaamloes.**
- As we go on increasing the contamination %, we are getting more number of datapoints classified as extreme outliers.

4. Local Outlier Factor (LOF)

- In LOF, the hypermeters used are n_neighbors = [5, 20] and contamination_ = [0.01,0.04].
- Almost **7429 datapoints are getting classified as outliers** using **maximum nearest neigbors (20) and with higesht contamination percentage of 4%.**
- I have validated visually in lower dimension with above hyperparameters but the performance is less acuurate than Isolation Forest but more accurate than other methods.
- With least nearest neigbors and least contamination percentage, we are getting less no. of outliers specifically in the mid-battery capacity range (1.4-1.6) having least and maximum temperature measured. This can be informed to subject matter expert and thus can be validated before removal.
- Along with extreme endpoints, the mid level inliers are also considered as outliers upon increasing contamination% upto 4% in bivariate analysis of temperature vs capacity

5. Elliptical Envelope

- Providing contamination as hyperparametr, we are able to get **18387 data points labelled (-1) i.e outliers.**
- Along with extreme endpoints, the mid level inliers are also considered as outliers even in lower contamination% of 1 % in bivariate analysis of temperature vs capacity

Final performance conclusions:

Isolation Forest >> Local Outlier Factor (LOF) > DBSCAN (density based) > Elliptical Envelope >> IQR based

Actionable Insights & Recommendations:

- Capacity of batteries decrease as no. of cycles of battery usage increases. Also **Capacity decreases over a period of time.**
- Battery B0006 (2) is having the largest capacity when it's new. Whereas B0018 and B0007 is having the least capacity when it's unused.

- Over a period of time, **B0006 shows much more deteriorating trend** than others , whereas **B0007 has the most efficient capacity**. Also, **Battery (B0006) with flag 2 has max no. of outliers**. This might be the primary reason for Battery 2's worst performance. **The classified anamolous datapoints should be highlighted** to the management after required **scrutiny from SMEs** and thus the team can take **proper action on improvement** with given feature's values.
- There hasn't been a continuos testing of charging of batteries throught the cycle. First three batteries (5,6,7) have been charged/tested synchronously whereas the 4th battery (B0018) is charged/tested post July 2008 with couple of breaks in between.
- The general trend is more or less the same which is as time of battery's cycle charging decreases over a period of time.
- We can clearly see that, **Battery (B0018) has been launched recently** and is having average capacity over time.
- Capacity of Battery B0002 deteriorates the most over time whereas for blue it's most efficient in terms of capacity over a period of time.
- As time to charge the batteries is increasing, the capacity is also increasing on an average for all four batteries
- **As time increases for batteries, the depreciation increases** because, temeperature_measured increases in every cycle of charging.
- Batteries with **much higher capacities and much lower capacities but having minimum temperature measured** are having most critical number of outliers. These can be checked with subject matter expert and then the **NASA team can work upon finding the exact root cause behind this behaviour**
- On an average, The batteries with **least temperature measured are having more number of outliers.**
- Batteries datapoints which are producing **maximum voltages with barely minimum charging time are ceratinly outliers/anamolies.**
- Also, with least contamination, the Batteries datapoints which requires maximum time to charge ceratinly outliers/anamolies.
- As it's clearly visible , if we increase the contamination % of outliers to 4 % , it's able to classify the visually seen anamolies as outliers as compared to contamionation of 1 %
- Also, with max contamination of 4%, the Batteries datapoints which **requires maximum time to charge and whose voltages are low are ceratinly outliers/anamolies.**
- As we go on **increase the contamination %**, we are getting **more number of datapoints classified as extreme outliers.**
- With least nearest neigbors and least contamination percentage, we are getting less no. of outliers specifically in the **mid-battery capacity range (1.4-1.6) having least and maximum temperature measured**. This can be informed to subject matter expert and thus can be validated before removal.

```
1
```

In [ ]:
```
1
```