Tesla Data Test

Author: Cheng Peng
Email: peng.cheng6479@gmail.com

Abstract

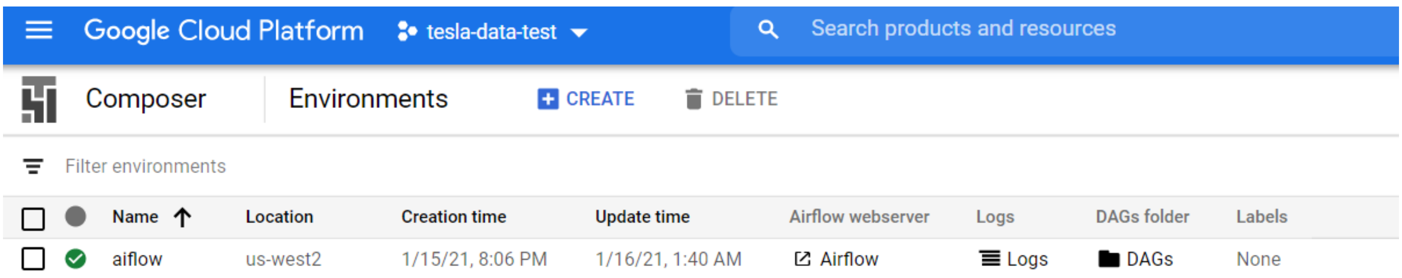
Powerwall is Tesla's top product which refers to a solar and grid energy management solution. It is an important part of Tesla's mission of accelerating the world's transition to sustainable energy. Powerwall is composed of Tesla battery and an intelligent system which linked to the grid and solar panel. The system will maintain house's electricity power at different situations and minimize the cost.

It is important to track and monitor each sold Powerwall product to make sure they are in good working conditions and avoid anomalies. In this test, three kinds of signal data will be acquired by a provided API for each site. And I am providing a solution in this article to transfer, manipulate, store, and analyze these signal data in order to monitor Powerwall products.

The solution includes 2 parts. First is data **ETL**, where I will build a data pipeline(a DAG procedure) and deploy it in **Airflow**. It will run with a scheduler for every minute and send API response data in **Google Cloud Storage** as three time-sequential csv files for each signal. Then I will set data transfer procedure to load these data from Storage into **Bigquery**. In second Part, I will build connection between **Jupyter Notebook** and Bigquery to get the data in real time and do the analytics job for anomalies detection. The result will be stored as error log and can be printed or exported. I will define 3 anomaly metrics and check them while reading streaming data.

Part I: ETL

Although we can directly call API and do the analytics right away, a scheduled pipeline can get the data automatically and store the historical data at the place we want. Airflow is an open-source tool and has all the functionalities, so it is my ideal choice. A normal usage in this situation is virtual machine + docker + airflow. But I choose **Google composer** as better way in terms of connivence. It provides a VM integrated with Airflow, only need to set up several configurations like node number, vm type, and OAuth. After that, I can see my airflow service is started as following:



Next step is to write a airflow dag which call API every minute, format the table and transfer them into Google Cloud Storage. The API provides 2 endpoints: get all the sites and get three signal at request moment for a single site. In order to tracking each site and get their continuous signal data, I build three table for each signal and columns are timestamp and all the sites, like this:

SITE_SM_batteryInstPower:

timestamp	siteID_1	siteID_2	siteID_3	…

SITE_SM_siteInstPower:

timestamp	siteID_1	siteID_2	siteID_3	…

SITE_SM_solarInstPower:

timestamp	siteID_1	siteID_2	siteID_3	…

To keep timestamp consistency, I build all the table structures and fill in one row data in one step. In this way, I can also avoid call API three times to fill all the tables. DAG code as below:

```
# to keep timestamp consistency and avoid repeated api calls, I generate dfs for three signals in one step
# one could also make this a function and run for each signal data in order to reduce the code volume
# so here's a trade-off
http = urllib3.PoolManager()
# define url path to call api: get all the sites
url1 = 'https://te-data-test.herokuapp.com/api/sites?toke[redacted]
req1 = http.request('GET', url1)
# transform data format
j1 = json5.loads(req1.data.decode('utf-8'))
df1 = pd.DataFrame(data = j1)
# create header for the tables of the three parameters
sites = list(df1["sites"])
header = ["timestamp"] + sites

SITE_SM_batteryInstPower = pd.DataFrame(columns = header)
SITE_SM_siteInstPower = pd.DataFrame(columns = header)
SITE_SM_solarInstPower = pd.DataFrame(columns = header)

# define tmp lists and call api to insert data, keep the same timestamp here for the three list
tmp_SITE_SM_batteryInstPower = [df1['timestamp'][1]]
tmp_SITE_SM_siteInstPower = [df1['timestamp'][1]]
tmp_SITE_SM_solarInstPower = [df1['timestamp'][1]]
```

```
for site in sites:
    url2 = 'https://te-data-test.herokuapp.com/api/signals?tok[redacted]+ str(site)
    req2 = http.request('GET', url2)
    j2 = json5.loads(req2.data.decode('utf-8'))
    df2 = pd.DataFrame(data = j2)
    if "SITE_SM_batteryInstPower" in j2["signals"] and j2["signals"]["SITE_SM_batteryInstPower"]:
        tmp_SITE_SM_batteryInstPower.append(j2["signals"]["SITE_SM_batteryInstPower"])
    else:
        tmp_SITE_SM_batteryInstPower.append(None)

    if "SITE_SM_siteInstPower" in j2["signals"] and j2["signals"]["SITE_SM_siteInstPower"]:
        tmp_SITE_SM_siteInstPower.append(j2["signals"]["SITE_SM_siteInstPower"])
    else:
        tmp_SITE_SM_siteInstPower.append(None)

    if "SITE_SM_solarInstPower" in j2["signals"] and j2["signals"]["SITE_SM_solarInstPower"]:
        tmp_SITE_SM_solarInstPower.append(j2["signals"]["SITE_SM_solarInstPower"])
    else:
        tmp_SITE_SM_solarInstPower.append(None)

SITE_SM_batteryInstPower.loc[0] = tmp_SITE_SM_batteryInstPower
SITE_SM_siteInstPower.loc[0] = tmp_SITE_SM_siteInstPower
SITE_SM_solarInstPower.loc[0] = tmp_SITE_SM_solarInstPower
```

Now I have three DataFrame for the three signal, and each of them has 1 row new data. Then I want to insert them(or create new csv file if this is pipeline's first run) into csv file at Storage. The following code will check and read cvs files in Storage, merge them will new data we just get, and upload them back to replace previous files

```
def read_storage_csv(bucket, path: str):
    blob = bucket.blob(path)
    byte_object = BytesIO()
    blob.download_to_file(byte_object)
    byte_object.seek(0)

    return byte_object

# define task template
def dataUpate(csv_name: str, folder_name: str, df: pd.DataFrame,
                 bucket_name = "signal-data-bucket", **kwargs):
    hook = GoogleCloudStorageHook()

    # cache prev data if exists in storage and merge with new generated df
    if hook.exists(bucket_name, object = '{}/{}.csv'.format(folder_name, csv_name)):
        storage_client = storage.Client()
        bucket = storage_client.get_bucket(bucket_name)
        f_download = read_storage_csv(bucket, path = '{}/{}.csv'.format(folder_name, csv_name))

        df_download = pd.read_csv(f_download).iloc[:, 1:]
        # append according to column names, lack value will filled with null
        # this is for situations when api returns diff sites numbers and columns might not match
        df = df_download.append(df, ignore_index=True)

    df.to_csv(csv_name)
```

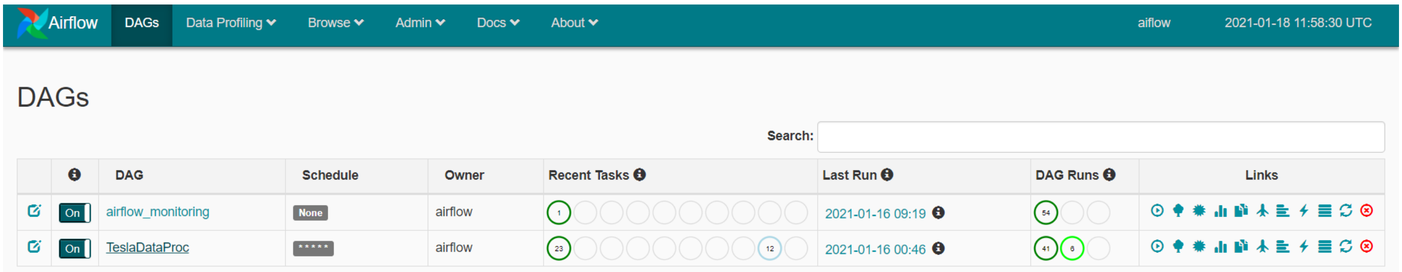
Here I use hook function from Google Cloud to achieve file transfer.
In the end, I defined a DAG with three tasks inside, which uploads the three generated csv to cloud. The DAG is configured to run every minute.

```
    hook.upload(bucket_name,
                object = '{}/{}.csv'.format(folder_name, csv_name),
                filename = csv_name,
                mime_type = 'text/csv')

dag = DAG('TeslaDataProc',
        default_args = default_args,
        schedule_interval = '* * * * *', # schedule run for every minute
        catchup = True)
# three parallel tasks in this dag
with dag:
    dummy_start_up = DummyOperator(
        task_id='All_jobs_start')

    dummy_shut_down = DummyOperator(
        task_id='All_jobs_end')

    batteryInstPower_task = PythonOperator(
        task_id = 'batteryInstPowerUpdate',
        python_callable = dataUpate,
        provide_context = True,
        op_kwargs = {'csv_name': 'SITE_SM_batteryInstPower.csv', 'folder_name': 'airflow',
                     'df': SITE_SM_batteryInstPower},
    )
```

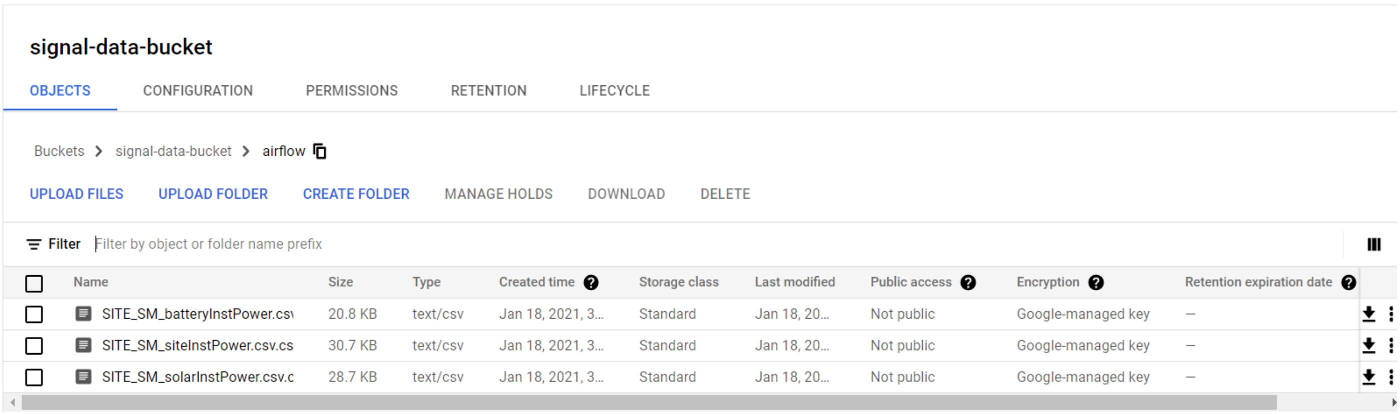
```
siteInstPower_task = PythonOperator(
    task_id = 'siteInstPowerUpdate',
    python_callable = dataUpate,
    provide_context = True,
    op_kwargs={'csv_name': 'SITE_SM_siteInstPower.csv', 'folder_name': 'airflow',
               'df': SITE_SM_siteInstPower},
)

solarInstPower_task = PythonOperator(
    task_id = 'solarInstPowerUpdate',
    python_callable = dataUpate,
    provide_context = True,
    op_kwargs = {'csv_name': 'SITE_SM_solarInstPower.csv', 'folder_name': 'airflow',
                 'df': SITE_SM_solarInstPower},
)
#parallel dependency and run
for task in (batteryInstPower_task, siteInstPower_task, solarInstPower_task):
    dummy_start_up >> task >> dummy_shut_down
```

One point to mention here is, dummy start and dummy shut down are necessary in the dependency, since I want the three tasks to run in parallel.

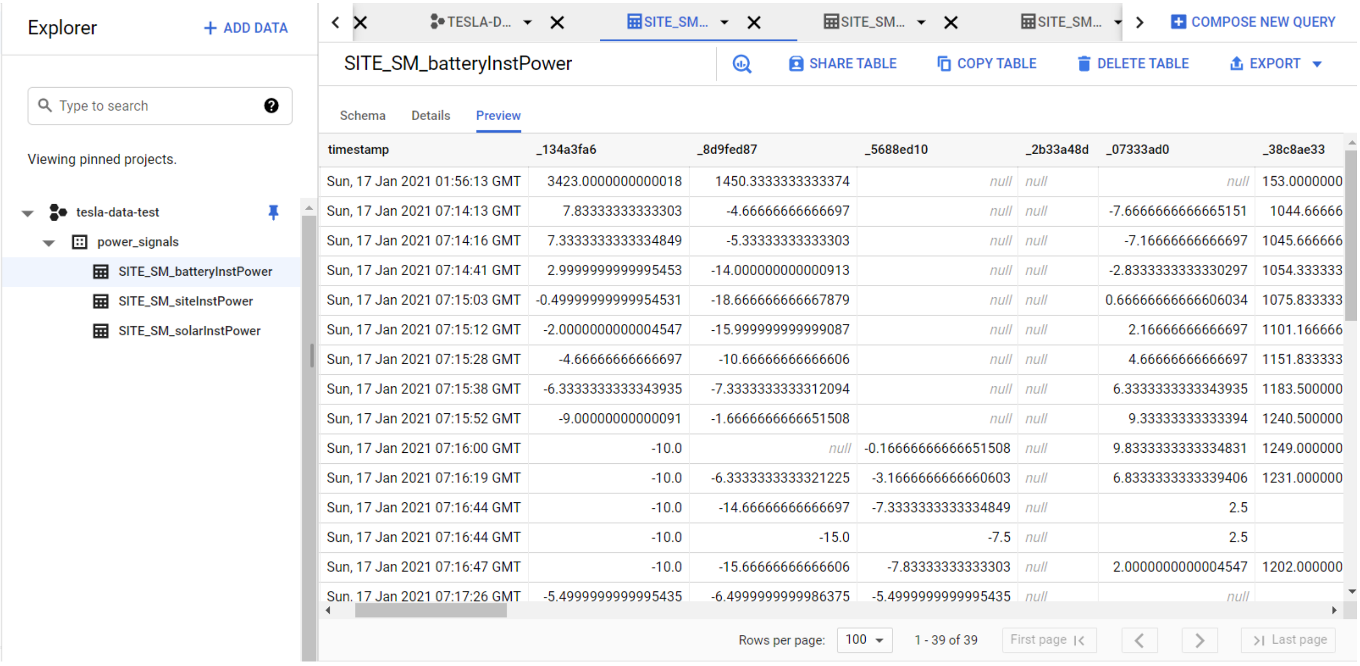
After deployed in airflow, I can see the DAG running well in web browser:



Waiting for several minutes, I can see the csv file on cloud:



Google provides easy connection and scheduled data transfer between Bigquery and Storage, so I want to skip here and directly show the data in Bigquery:



timestamp	_134a3fa6	_8d9fed87	_5688ed10	_2b33a48d	_07333ad0	_38c8ae33
Sun, 17 Jan 2021 01:56:13 GMT	3423.0000000000018	1450.3333333333374	null	null	null	153.0000000
Sun, 17 Jan 2021 07:14:13 GMT	7.83333333333303	-4.66666666666697	null	null	-7.6666666666665151	1044.66666
Sun, 17 Jan 2021 07:14:16 GMT	7.3333333333334849	-5.33333333333303	null	null	-7.16666666666697	1045.666666
Sun, 17 Jan 2021 07:14:41 GMT	2.9999999999995453	-14.000000000000913	null	null	-2.8333333333330297	1054.333333
Sun, 17 Jan 2021 07:15:03 GMT	-0.49999999999954531	-18.666666666667879	null	null	0.66666666666606034	1075.833333
Sun, 17 Jan 2021 07:15:12 GMT	-2.0000000000004547	-15.999999999999087	null	null	2.16666666666697	1101.166666
Sun, 17 Jan 2021 07:15:28 GMT	-4.66666666666697	-10.66666666666606	null	null	4.66666666666697	1151.833333
Sun, 17 Jan 2021 07:15:38 GMT	-6.3333333333343935	-7.3333333333312094	null	null	6.3333333333343935	1183.500000
Sun, 17 Jan 2021 07:15:52 GMT	-9.00000000000091	-1.6666666666651508	null	null	9.33333333333394	1240.500000
Sun, 17 Jan 2021 07:16:00 GMT	-10.0	null	-0.16666666666651508	null	9.8333333333334831	1249.000000
Sun, 17 Jan 2021 07:16:19 GMT	-10.0	-6.3333333333321225	-3.1666666666660603	null	6.8333333333339406	1231.000000
Sun, 17 Jan 2021 07:16:44 GMT	-10.0	-14.66666666666697	-7.3333333333334849	null	2.5	
Sun, 17 Jan 2021 07:16:44 GMT	-10.0	-15.0	-7.5	null	2.5	
Sun, 17 Jan 2021 07:16:47 GMT	-10.0	-15.66666666666606	-7.83333333333303	null	2.0000000000004547	1202.000000
Sun, 17 Jan 2021 07:17:26 GMT	-5.4999999999995435	-6.4999999999986375	-5.4999999999995435	null	null	

Now I get data on cloud that is keep updating by my pipeline, I can start my analytics part.

Part II: Analytics

An Industrial solution for this part might be a powerful BI tool like Tableau or Looker. Since I don't have their Licenses and also want to show more details about my logic(mainly by code), I choose Jupyter Notebook as my tool.

Setting up the connection from Jupyter Notebook to Bigquery is tricky but nothing worth to mention here. So, let's see the valuable parts.

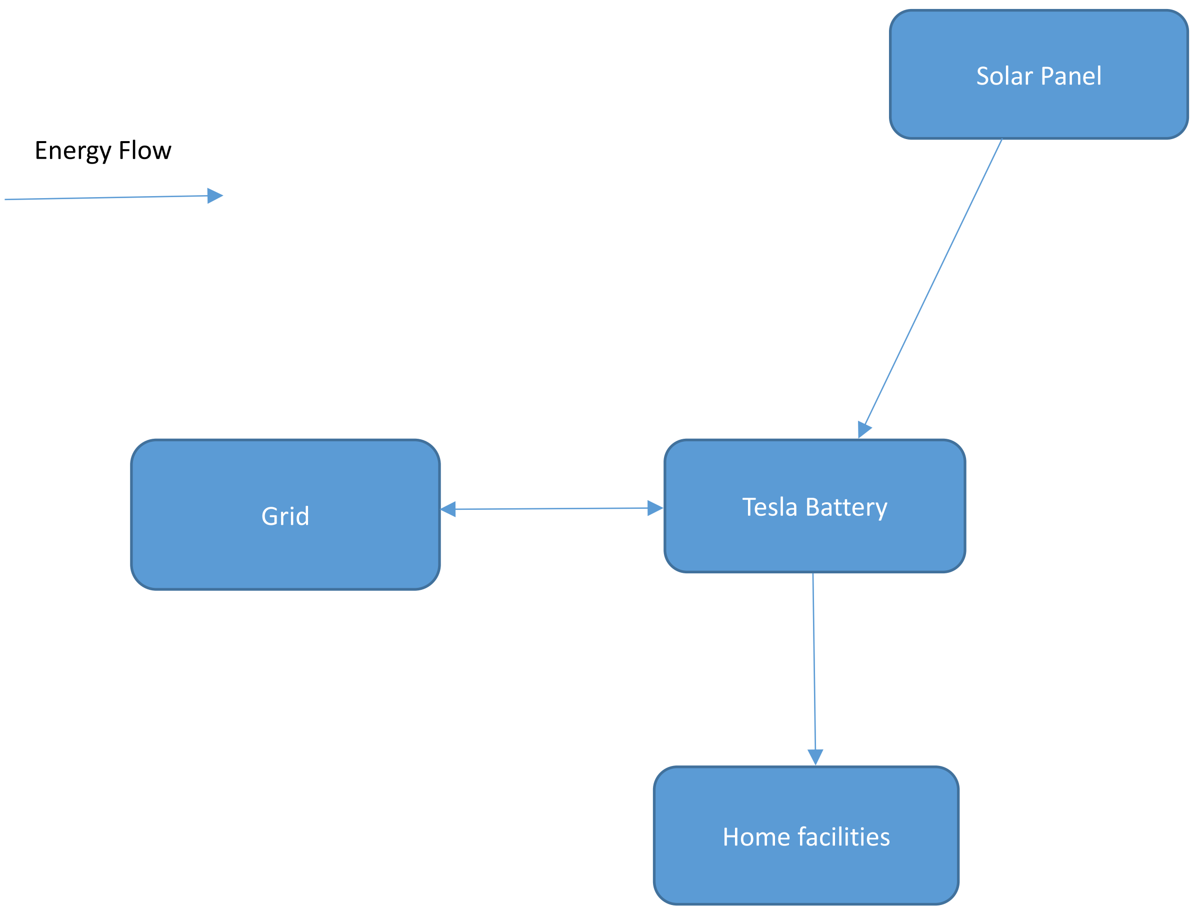
I build a monitor class to help me on the analytics. It uses sql query to get data from Bigquery. Each time it only gets one row according to the timestamp and update its own timestamp at the same time. This is for creating a simulation of data streaming and not getting itself overloaded.

```
# sql query for SITE_SM_siteInstPower
sql_siteInstPower = """
SELECT
     PARSE_TIMESTAMP("%a, %d %b %Y %k:%M:%S %Z", timestamp) AS timestamp,
"""\
+ siteID + \
"""
FROM
    `tesla-data-test.power_signals.SITE_SM_siteInstPower`
WHERE PARSE_TIMESTAMP("%a, %d %b %Y %k:%M:%S %Z",timestamp) > TIMESTAMP(@ts)
LIMIT 1
"""

# sql query for SITE_SM_solarInstPower
sql_solarInstPower = """
SELECT
     PARSE_TIMESTAMP("%a, %d %b %Y %k:%M:%S %Z", timestamp) AS timestamp,
"""\
+ siteID + \
"""
FROM
     `tesla-data-test.power_signals.SITE_SM_solarInstPower`
WHERE PARSE_TIMESTAMP("%a, %d %b %Y %k:%M:%S %Z",timestamp) > TIMESTAMP(@ts)
LIMIT 1
"""
```

```
#get data from bigquery
job_config = bigquery.QueryJobConfig(
    query_parameters=[
        bigquery.ScalarQueryParameter("ts", "STRING", self.ts),
    ]
)

df_batteryInstPower = client.query(sql_batteryInstPower, job_config).to_dataframe()
df_siteInstPower = client.query(sql_siteInstPower, job_config).to_dataframe()
df_solarInstPower = client.query(sql_solarInstPower, job_config).to_dataframe()

# when get empty df
if len(df_batteryInstPower) == 0 or len(df_siteInstPower) == 0 or len(df_solarInstPower) == 0:
    return float('-inf'), float('-inf'), float('-inf')

# store/update data
self.siteCollection[siteID][0].append((df_batteryInstPower.loc[0][0], df_batteryInstPower.loc[0][1]))
self.siteCollection[siteID][1].append((df_siteInstPower.loc[0][0], df_siteInstPower.loc[0][1]))
self.siteCollection[siteID][2].append((df_solarInstPower.loc[0][0], df_solarInstPower.loc[0][1]))
self.siteCollection[siteID][3] += df_batteryInstPower.loc[0][1] if df_batteryInstPower.loc[0][1] else 0

# update monitor's timestamp to latest
self.ts = str(df_batteryInstPower.loc[0][0])

return df_batteryInstPower.loc[0][1], df_siteInstPower.loc[0][1], df_solarInstPower.loc[0][1]
```

While reading the stream in data, the monitor will save the data in its own data structure and do the anomalies detections at same time. Why I still save the data since they already exist on cloud? Because first this will help me do one of the anomalies detections. Second, if we want to add more functionalities like plotting the trend, this will help.

Now let's talk about anomalies. In my assumption(well, I might be wrong since I didn't find enough documents about this), the model of Powerwall should be like this:



I can define three metrics for anomalies detection based on this model:

1. Solar Panel only export energy, thus when SITE_SM_solarInstPower is negative, it means solar panel is consuming, which might be an anomaly
   I name it as anomaly 1 and use following method to detect it.

```
def anomaliesDetection_1(self, solarInstPower_signal):
    if solarInstPower_signal and solarInstPower_signal < 0:
        return True
    return False
```

2. Battery has capacity. The SITE_SM_batteryInstPower shows and **net flow**(in or out) of the battery. If the Accumulated net flow is positive and larger than the capacity, it might be an anomaly. Knowing Powerwall's capacity is 13.5kwh, and based on the data I see in the API response, I set 135000 as the capacity. Following code is used to detect this anomaly, named as anomaly 2. (**data stored in the monitor class is used here**)

```
def anomaliesDetection_2(self, siteID):
    if self.siteCollection[siteID][3] > self.batteryCapacity:
        return True
    return False
```

3. Battery can export energy to grid. But when battery is not working(API returns null value), the energy exported to grid will be very suspicious. It might mean there are errors in battery signal data or grid signal data. Following code is to detect this anomaly 2

```
def anomaliesDetection_3(self, batteryInstPower_signal, siteInstPower):
    if not batteryInstPower_signal and siteInstPower:
        if siteInstPower < 0:
            return True
    return False
```

In addition to the above functions, I also integrated other 2 mechanisms in the monitor. First is monitor interruption. Users can use input to stop the monitor when it shows it's **enabled** to interrupt(which will be printed while running). This is for preventing interruption while the streaming or analytics is still on-going. Second is sleeping and waiting for data refresh. If Bigquery returns null dataframes, it means pipelines haven't refreshed the tables. So the monitor will sleep for 60s waiting for data to refresh. Code as below:

```
# monitor on single site
def startMonitor(self, siteID):
    # when switch is on, continiously ask for data from bigquery
    while self.switch:
        try:
            s = askChoice()
            if s:
                self.switch = False

        except func_timeout.exceptions.FunctionTimedOut as e:
            print(str(self.ts) + " interupt disabled")
            batteryInstPower_signal, siteInstPower, solarInstPower_signal = self.bigqueryStreaming(siteID)

            #if no data returns, monitor sleeps for 60s waiting for data source to refresh
            if batteryInstPower_signal + siteInstPower + solarInstPower_signal == float('-inf'):
                time.sleep(60)
                continue
```

The monitor will print detection log and instruction while running. The anomalies logs will also be stored and able to print or export. Outputs like this:

```
# monitor a site with anomaly 1
m = signalMonitor(sites)
m.startMonitor("_07333ad0")
```

```
2001-01-01 00:00:00+00:00 interupt disabled
2021-01-17 01:56:13+00:00 anomalies 1 detected, for details see error log
2021-01-17 01:56:13+00:00 monitor working...
2021-01-17 01:56:13+00:00 interupt enabled
2021-01-17 01:56:13+00:00 interupt disabled
2021-01-17 07:14:13+00:00 anomalies 1 detected, for details see error log
2021-01-17 07:14:13+00:00 monitor working...
2021-01-17 07:14:13+00:00 interupt enabled
wait for interruption is enabled, input any to interupt process: 1
```

```
for row in m.errLog:
    print(row)
```

```
2021-01-17 01:56:13+00:00: detected negative solarInstPower signal: -5.553000132242838at site: _07333ad0
2021-01-17 07:14:13+00:00: detected negative solarInstPower signal: -5.553000132242838at site: _07333ad0
```

```
# monitor a site with anomaly 1 and 2: need wait for around 30 logs for anomaly 2 appearing
m = signalMonitor(sites)
m.startMonitor("c8eb2d3d")
```

```
2001-01-01 00:00:00+00:00 interupt disabled
2021-01-17 01:56:13+00:00 anomalies 1 detected, for details see error log
2021-01-17 01:56:13+00:00 monitor working...
2021-01-17 01:56:13+00:00 interupt enabled
2021-01-17 01:56:13+00:00 interupt disabled
2021-01-17 07:14:13+00:00 anomalies 1 detected, for details see error log
2021-01-17 07:14:13+00:00 monitor working...
2021-01-17 07:14:13+00:00 interupt enabled
2021-01-17 07:14:13+00:00 interupt disabled
2021-01-17 07:14:16+00:00 anomalies 1 detected, for details see error log
2021-01-17 07:14:16+00:00 monitor working...
2021-01-17 07:14:16+00:00 interupt enabled
2021-01-17 07:14:16+00:00 interupt disabled
2021-01-17 07:14:41+00:00 anomalies 1 detected, for details see error log
2021-01-17 07:14:41+00:00 monitor working...
2021-01-17 07:14:41+00:00 interupt enabled
2021-01-17 07:14:41+00:00 interupt disabled
2021-01-17 07:15:03+00:00 anomalies 1 detected, for details see error log
2021-01-17 07:15:03+00:00 monitor working...
2021-01-17 07:15:03+00:00 interupt enabled
2021-01-17 07:15:03+00:00 interupt disabled
2021-01-17 07:15:12+00:00 anomalies 1 detected, for details see error log
2021-01-17 07:15:12+00:00 monitor working...
2021-01-17 07:15:12+00:00 interupt enabled
2021-01-17 07:15:12+00:00 interupt disabled
2021-01-17 07:15:28+00:00 anomalies 1 detected, for details see error log
2021-01-17 07:15:28+00:00 monitor working...
```

```
for row in m.errLog:
    print(row)
```

```
2021-01-17 01:56:13+00:00: detected negative solarInstPower signal: -3.3100000023841853at site: c8eb2d3d
2021-01-17 07:14:13+00:00: detected negative solarInstPower signal: -3.3100000023841853at site: c8eb2d3d
2021-01-17 07:14:16+00:00: detected negative solarInstPower signal: -3.277999997139005at site: c8eb2d3d
2021-01-17 07:14:41+00:00: detected negative solarInstPower signal: -3.0006666183471578at site: c8eb2d3d
2021-01-17 07:15:03+00:00: detected negative solarInstPower signal: -2.796999925375016at site: c8eb2d3d
2021-01-17 07:15:12+00:00: detected negative solarInstPower signal: -2.7529999276002206at site: c8eb2d3d
2021-01-17 07:15:28+00:00: detected negative solarInstPower signal: -2.538499942421878at site: c8eb2d3d
2021-01-17 07:15:38+00:00: detected negative solarInstPower signal: -2.538499942421878at site: c8eb2d3d
2021-01-17 07:15:52+00:00: detected negative solarInstPower signal: -2.538499942421878at site: c8eb2d3d
2021-01-17 07:16:00+00:00: detected negative solarInstPower signal: -2.538499942421878at site: c8eb2d3d
2021-01-17 07:16:00+00:00: battery total charge exceeded its capacity at site: c8eb2d3d
2021-01-17 07:16:19+00:00: detected negative solarInstPower signal: -2.538499942421878at site: c8eb2d3d
2021-01-17 07:16:19+00:00: battery total charge exceeded its capacity at site: c8eb2d3d
```

In the End

[redacted]

All the code will be attached with this doc. My GCP services will remain running for a week and Jupyter Notebook's connection is set well, so ideally it can run on any computer directly.